

Univerzitet u Beogradu
Matematički fakultet

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

Zbirka programa

Beograd, 2015.

Predgovor

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

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

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

Autori

Sadržaj

1	Uvodni zadaci	3
1.1	Podela koda po datotekama	3
1.2	Algoritmi za rad sa bitovima	6
1.3	Rekurzija	12
1.4	Rešenja	19
2	Pokazivači	61
2.1	Pokazivačka aritmetika	61
2.2	Višedimenzioni nizovi	64
2.3	Dinamička alokacija memorije	69
2.4	Pokazivači na funkcije	72
2.5	Rešenja	74
3	Algoritmi pretrage i sortiranja	113
3.1	Pretraživanje	113
3.2	Sortiranje	118
3.3	Bibliotečke funkcije pretrage i sortiranja	128
3.4	Rešenja	134
4	Dinamičke strukture podataka	209
4.1	Liste	209
4.2	Stabla	225
4.3	Rešenja	235
5	Ispitni rokovi	345
5.1	Programiranje 2, praktični deo ispita, jun 2015.	345
5.2	Programiranje 2, praktični deo ispita, jul 2015.	347
5.3	Rešenja	349

Glava 1

Uvodni zadaci

1.1 Podela koda po datotekama

Zadatak 1.1 Napisati program za rad sa kompleksnim brojevima.

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

1 Uvodni zadaci

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

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

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

Test 1

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

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

        (1.00 - 3.00 i) + (-1.00 + 4.00 i) = (1.00 i)

        (1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)

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

[Rešenje 1.1]

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

Test 1

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

[Rešenje 1.2]

Zadatak 1.3 Napisati malu biblioteku za rad sa polinomima.

- (a) Definisati strukturu `Polinom` koja predstavlja polinom (stepena najviše 20). Struktura sadrži stepen i niz koeficijenata. Redosled navođenja koeficijenata u nizu treba da bude takav da na nultoj poziciji u nizu bude koeficijent uz slobodan član, na prvoj koeficijent uz prvi stepen, itd.

- (b) Napisati funkciju koja ispisuje polinom na standardni izlaz u što lepšem obliku.
- (c) Napisati funkciju koja učitava polinom sa standardnog ulaza.
- (d) Napisati funkciju za izračunavanje vrednosti polinoma u datoj tački koristeći Hornerov algoritam.
- (e) Napisati funkciju koja sabira dva polinoma.
- (f) Napisati funkciju koja množi dva polinoma.

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

Upotreba programa 1

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

[Rešenje 1.3]

Zadatak 1.4 Napraviti biblioteku za rad sa razlomcima.

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

1 Uvodni zadaci

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

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

Test 1

```

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

```

1.2 Algoritmi za rad sa bitovima

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

Test 1

```
Ulaz:    0x7F  
Izlaz:   000000000000000000000000000000001111111
```

Test 2

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

Test 3

```
Ulaz: 0x00FF00FF
Izlaz: 00000000111111110000000011111111
```

Test 4

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

Test

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

[Rešenje 1.5]

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

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

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

Test 1

```

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

Test 2

```

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

Test 3

```

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

Test 4

```

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

Test 4

```

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

[Rešenje 1.6]

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

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

Test 1

```

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

Test 2

```

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

1 Uvodni zadaci

Test 3

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

Test 4

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

Test 4

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

[Rešenje 1.7]

Zadatak 1.8 Napisati program za rad sa bitovima.

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

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

1 Uvodni zadaci

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

Test 1

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

[Rešenje 1.10]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.11]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.12]

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

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

Zadatak 1.14 Napisati funkciju koja na osnovu neoznačenog broja x formira nisku s koja sadrži heksadekadni zapis broja x , koristeći algoritam za brzo prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksadekadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

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

[Rešenje 1.14]

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

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

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

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

1.3 Rekurzija

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

Test 1

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

[Rešenje 1.17]

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

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

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

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

[Rešenje 1.18]

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

Test 1

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

[Rešenje 1.19]

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

$$F(0) = 0$$

$$F(1) = 1$$

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

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

Test 1

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

[Rešenje 1.20]

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

Test 1

```
|| Ulaz:   123
|| Izlaz:  6
```

Test 2

```
|| Ulaz:   23156
|| Izlaz:  17
```

Test 3

```
|| Ulaz:   1432
|| Izlaz:  10
```

Test 4

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

Test 5

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

[Rešenje 1.21]

1 Uvodni zadaci

Zadatak 1.22 Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva. Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju n ($0 < n \leq 100$) celobrojnog niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom.

Test 1

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

[Rešenje 1.22]

Zadatak 1.23 Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. Niz neće imati više od 256 elemenata, i njegovi elementi se unose sve do kraja ulaza.

Test 1

```
|| Ulaz:  3 2 1 4 21
|| Izlaz: 21
```

Test 2

```
|| Ulaz:  2 -1 0 -5 -10
|| Izlaz: 2
```

Test 3

```
|| Ulaz:  1 11 3 5 8 1
|| Izlaz: 11
```

Test 4

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

[Rešenje 1.23]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.24]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.25]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.26]

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

Test 1

```
|| Ulaz: 0x7F
|| Izlaz: 7
```

Test 2

```
|| Ulaz: 0x80
|| Izlaz: 1
```

Test 3

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

1 Uvodni zadaci

Test 4

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

[Rešenje 1.27]

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

Test 1

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

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.29]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[\[Rešenje 1.30\]](#)

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

Test

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

[\[Rešenje 1.31\]](#)

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

Test 1

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

[\[Rešenje 1.32\]](#)

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

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

1 Uvodni zadaci

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

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

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

Test 1

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

Test 2

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

[Rešenje 1.33]

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

Test 1

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

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

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

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

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

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

1.4 Rešenja

Rešenje 1.1

```

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

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

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

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

```

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

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

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

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

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

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

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

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



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

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

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

92    return z;
94 }

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

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

104    return z;
106 }

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

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

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

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

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

```

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

```

Rešenje 1.2

```

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

```

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

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

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

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

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

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

1 Uvodni zadaci

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

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

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

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

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

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

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

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

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

```

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

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

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

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

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

27 void ispisi_kompleksan_broj(KompleksanBroj z) ;

29 float realan_deo(KompleksanBroj z) ;

31 float imaginaran_deo(KompleksanBroj z);

33 float moduo(KompleksanBroj z);

35 KompleksanBroj konjugovan(KompleksanBroj z) ;

37 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2 ) ;

39 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2 ) ;

41 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2 ) ;

43 float argument(KompleksanBroj z) ;

45 /* Kraj zakljucanog dela */
    #endif

2  /*
    * Koristimo korektno definisanu biblioteku kompleksnih brojeva.

```

1 Uvodni zadaci

```

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

Rešenje 1.3

```

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

```

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

```

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

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

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

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

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

93        return r;
95    }

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

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

111        return r;
    }

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



```

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

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

        r = izvod(p);
127         for (i = 1; i < n; i++)
            r = izvod(&r);
129
131         return r;
133     }

```

```

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

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

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

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

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

```

1 Uvodni zadaci

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

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

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

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

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

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

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

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

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

```

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

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

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

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

46     /* Uspesno završavamo program */
    return 0;
58 }

```

Rešenje 1.5

```

#include <stdio.h>

2
/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
4 celog broja u memoriji. Bitove u zapisu broja treba da
    ispisujemo sa leva na desno, tj. od bita najvece tezine ka
6 bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost
    maske uzimamo vrednost ciji binarni zapis je takav da je bit
8 najvece tezine 1, a svi ostali nule. Nakon toga, u svakoj
    iteraciji cemo tu jedinicu pomerati u desno, kako bismo
10 ocitali naredni bit, gledano s leva na desno. Odgovarajuci
    karakter, ('0' ili '1'), ispisuje se na ekranu. Zbog
12 siftovanja maske u desno koja na pocetku ima najvisi bit
    postavljen na 1, neophodno je da maska bude neoznacena ceo
14 broj i da se siftovanjem u desno ova 1 ne bi smatrala znakom
    i prepisivala, vec da bi nam se svakim siftovanjem sa levog
16 kraja binarnog zapisa pojavljivale 0. */
void print_bits(unsigned x)
18 {

20     /* broj bitova celog broja */
    unsigned velicina = sizeof(unsigned) * 8;
22     /* maska koju cemo koristiti za "ocitavanje" bitova */

```

1 Uvodni zadaci

```
    unsigned maska;

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

28     putchar('\n');
29 }

30

32 int main()
33 {
34     int broj;
35     scanf("%x", &broj);
36     print_bits(broj);

38     return 0;
39 }
```

Rešenje 1.6

```
1  #include <stdio.h>

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

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

13     putchar('\n');
14 }

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

21
22     /* Formiramo masku 100000...0000000, koja služi za očitavanje
   bita najveće težine. U svakoj iteraciji maska se pomera u
   desno za 1 mesto, i očitavamo sledeći bit. Petlja se
   završava kada više nema jedinica tj. kada maska postane
   nula. */
27     unsigned maska = 1 << wl;
28     for (; maska != 0; maska >>= 1)
29         x & maska ? br++ : 1;

31 }
```

```

    return br;
33 }

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

```

```

1 #include <stdio.h>

3 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
   celog broja u memoriji */
5 void print_bits(int x)
6 {
7     /* broj bitova celog broja */
8     unsigned velicina = sizeof(int) * 8;
9
10    /* maska koju cemo koristiti za "ocitavanje" bitova */
11    unsigned maska;
12
13    /* Bitove u zapisu broja treba da ispisujemo sa leva na desno,
       tj. od bita najvece tezine ka bitu najmanje tezine. Iz tog
15    razloga, za pocetnu vrednost maske uzimamo vrednost ciji
       binarni zapis je takav da je bit najvece tezine 1, a svi
17    ostali nule. Nakon toga, u svakoj iteraciji cemo tu
       jedinicu pomerati u desno, kako bismo ocitali naredni bit,
19    gledano s leva na desno. Odgovarajuci karakter, ('0' ili
       '1'), ispisuje se na ekranu.
21
       Zbog siftovanja maske u desno koja na pocetku ima najvisi
23    bit postavljen na 1, neophodno je da maska bude neoznacena
       ceo broj i da se siftovanjem u desno ova 1 ne bi smatrala
25    znakom i prepisivala, vec da bi nam se svakim siftovanjem
       sa levog kraja binarnog zapisa pojavljivale 0. */
27
28    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
29        putchar(x & maska ? '1' : '0');
31
32    putchar('\n');
33 }
34
35 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
   x pomeranjem broja x */
36 int count_bits1(int x)
37 {
38     int br = 0;
39     unsigned wl = sizeof(int) * 8 - 1;

```

1 Uvodni zadaci

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

53  /* Formiramo masku 100000...0000000, koja služi za očitavanje
55     bita najveće težine. U svakoj iteraciji x se pomera u levo
57     za 1 mesto, i očitavamo sledeći bit. Petlja se završava
59     kada više nema jedinica tj. kada x postane nula. */
61  unsigned maska = 1 << wl;
63  for (; x != 0; x <<= 1)
65      x & maska ? br++ : 1;
67  return br;
}

int main()
{
    int x;
    scanf("%x", &x);
    printf("Broj jedinica u zapisu je %d.\n", count_bits1(x));

    return 0;
}
```

Rešenje 1.7

```
#include <stdio.h>

2  /* Funkcija vraca najveći neoznaceni broj sastavljen iz istih
4     bitova kao i x */
5  unsigned najveći(unsigned x)
6  {
7      unsigned velicina = sizeof(unsigned) * 8;
8
9      /* Formiramo masku 100000...0000000 */
10     unsigned maska = 1 << (velicina - 1);
11
12     /* Inicijalizujemo rezultat na 0 */
13     unsigned rezultat = 0;
14
15     /* Dokle god postoje jedinice u binarnoj reprezentaciji broja
16        x (tj. dokle god je x različit od nule) pomeramo ga ulevo. */
17     for (; x != 0; x <<= 1) {
18         /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
19            leva u rezultat */
20         if (x & maska) {
```

```

    rezultat >>= 1;
22     rezultat |= maska;
    }
24 }

26     return rezultat;
    }
28

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

    /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
36     for (; x != 0; x >>= 1) {
        /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
38         desna u rezultat */
        if (x & 1) {
40             rezultat <<= 1;
42             rezultat |= 1;
        }
44     }

46     return rezultat;
    }
48

/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
50   celog broja u memoriji */
void print_bits(int x)
52 {
    unsigned velicina = sizeof(int) * 8;
54     unsigned maska;

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

58     putchar('\n');
60 }

62 int main()
    {
64     int broj;
        scanf("%x", &broj);
66

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

        printf("Najmanji:\n");
70         print_bits(najmanji(broj));
72 
```

1 Uvodni zadaci

```
    return 0;
74 }
```

Rešenje 1.8

```
1  #include <stdio.h>

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

13        Na primer, za n=5 i p=10 formiramo masku oblika 1111 1111
15        1111 1111 1111 1000 0011 1111 To postizemo na sledeci
       nacin: ~0 1111 1111 1111 1111 1111 1111 1111 1111 (~0 << n)
17        1111 1111 1111 1111 1111 1111 1110 0000 ~(~0 << n) 0000
       0000 0000 0000 0000 0000 0001 1111 (~(~0 << n) << ( p-n+1))
19        0000 0000 0000 0000 0000 0111 1100 0000 ~(~0 << n) << (
       p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111 */
21    unsigned maska = ~(~(~0 << n) << (p - n + 1));

23    return x & maska;
   }

25

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

37        return x | maska;
   }

39

41

43  /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
   predstavlja n bitova pocev od pozicije p u binarnoj
   reprezentaciji broja x, pri cemu se pozicija broji sa desna
45  ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
   i broj 1010 1011 1100 1101 1110 1010 1110 0111 0000 0000 0000
```



```

47     0000 0000 0000 0000 1011 */
unsigned get_bits(unsigned x, unsigned n, unsigned p)
49 {
    /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a
51     ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
       0001 1111 */
53     unsigned maska = ~(~0 << n);

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

61
/* Funkcija vraca broj x kome su n bitova pocev od pozicije p
63 postavljani na vrednosti n bitova najnize tezine binarne
   reprezentacije broja y */
65 unsigned set_n_bits(unsigned x, unsigned n, unsigned p,
                       unsigned y)
67 {
    /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a
69     ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
       0001 1111 */
71     unsigned last_n_1 = ~(~0 << n);

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

79     /* x sa resetovanih n bita na pozicijama pocev od p */
    unsigned x_reset = x & middle_n_0;

81     /* y cijih je n bitova najnize tezine pomereni tako da stoje
       pocev od pozicije p. Ostali bitovi su nule. (y & last_n_1)
83     resetuje sve bitove osim najnizih n */
    unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);

85     return x_reset ^ y_shift_middle;
87 }
89

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

```

```
99  /* Operator ekskluzivno ili invertuje sve bitove gde je
    odgovarajuci bit maske 1. Ostali bitovi ostaju
101  nepromenjeni. */
    return maska ^ x;
103 }

105 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
107 void print_bits(int x)
109 {
    unsigned velicina = sizeof(int) * 8;
111    unsigned maska;

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

115    putchar('\n');
117 }

119

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

127

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

131

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

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

137

139     printf("y = %31u = ", y);
        print_bits(y);
        printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
141     print_bits(set_n_bits(broj, n, p, y));

143     printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
        print_bits(invert(broj, n, p));

145     return 0;
147 }
```

Rešenje 1.9

```

#include <stdio.h>

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

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

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

32   /* n puta ponavljamo rotaciju u desno za jedan bit. U svakoj
   iteraciji odredjujemo bit najmanje tezine broja x, zatm
34   tako odredjeni bit siftujemo u levo tako da najnizi bit
   dode do pozicije najviseg bita i nakon siftovanja x za 1 u
36   desno postavljamo x-ov najvisi bit na vrednost najnizeg
   bita. */
38   for (i = 0; i < n; i++) {
       last_bit = x & 1;
40       x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
   }
42

44   return x;
}

46
/* Verzija funkcije koja broj x rotira u desno za n mesta, gde
48   je x oznaceni broj */
int rotate_right_signed(int x, unsigned n)
50 {
   unsigned last_bit;

```

1 Uvodni zadaci

```
52  int i;

54

56  /* U svakoj iteraciji odredjujemo bit najmanje tezine tj.
57     last_bit. Kako je x oznacen ceo broj, tada se prilikom
58     siftovanja u desno vrsi aritmeticki sift i cuva se znak
59     broja. Iza tog razloga imamo dva slucaja u zavisnosti od
60     znaka od x. Nije dovoljno da se ova provera izvrši pre
61     petlje, jer rotiranjem u desno na mesto najviseg bita moze
62     doci i 0 i 1, nezavisno od pocetnog znaka x. */
63  for (i = 0; i < n; i++) {
64     last_bit = x & 1;

65     if (x < 0)
66         /* Siftovanjem u desno broja koji je negativan dobijamo 1
67            na najvisoj poziciji. Na primer ako je x 1010 1011
68            1100 1101 1110 0001 0010 001b (sa b oznacavamo u
69            primeru 1 ili 0 na najnižoj poziciji) last_bit je 0000
70            0000 0000 0000 0000 0000 000b nakon Siftovanja za
71            1 u desno 1101 0101 1110 0110 1111 0000 1001 0001 da
72            bismo najvisu 1 u x postavili na b nije dovoljno da ga
73            siftujemo na najvisu poziciju jer bi se time dobile 0,
74            a nama su potrebne 1 zbog bitovskog & zato prvo
75            komplementiramo, pa tek onda siftujemo ~last_bit <<
76            (sizeof(int)*8 -1) B000 0000 0000 0000 0000 0000 0000
77            0000 (B oznacava ~b ) i ponovo komplementiramo da bismo
78            imali b na najvisoj poziciji i sve 1 na ostalim
79            pozicijama ~(~last_bit << (sizeof(int)*8 -1)) b111 1111
80            1111 1111 1111 1111 1111 1111 */
81         x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
82     else
83         x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
84 }

85 return x;
86 }

87

88

89 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
90    celog broja u memoriji */
91 void print_bits(int x)
92 {
93     unsigned velicina = sizeof(int) * 8;
94     unsigned maska;
95     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
96         putchar(x & maska ? '1' : '0');
97
98     putchar('\n');
99 }

100

101 int main()
102 {
```

```

104 unsigned x, k;
    scanf("%x%x", &x, &k);
106 printf("x %36s = ", "");
    print_bits(x);
108 printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
    print_bits(rotate_left(x, k));
110
    printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
112 print_bits(rotate_right(x, k));
114
    printf("rotate_right_signed(%7u,%6u) = ", x, k);
    print_bits(rotate_right_signed(x, k));
116
    return 0;
118 }

```

Rešenje 1.10

```

#include <stdio.h>
2
/* Funkcija vraca vrednost cija je binarna reprezentacija slika
4 u ogledalu binarne reprezentacije broja x. Na primer za x
   cija binarna reprezentacija izgleda ovako
6 101010111100110111100100100100011 funkcija treba da vrati
   broj cija binarna reprezentacija izgleda:
8 11000100100001111011001111010101 */
unsigned mirror(unsigned x)
10 {
    unsigned najnizi_bit;
12 unsigned rezultat = 0;
14
    int i;
    /* Krecemo od najnizeg bita u zapisu broja x i dodajemo ga u
16 rezultat */
    for (i = 0; i < sizeof(x) * 8; i++) {
18         najnizi_bit = x & 1;
        x >>= 1;
20         /* Potiskujemo trenutni rezultat ka levom kraju. Tako svi
           prethodno postavljeni bitovi dobijaju vecu poziciju. Novi
           bit postavljamo na najnizu poziciju */
        rezultat <<= 1;
22         rezultat |= najnizi_bit;
24     }
    return rezultat;
26 }
28
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
30 celog broja u memoriji */
void print_bits(int x)
32 {

```

1 Uvodni zadaci

```
34 unsigned velicina = sizeof(int) * 8;
   unsigned maska;
36 for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
   putchar(x & maska ? '1' : '0');

38   putchar('\n');
40 }

42 int main()
43 {
44   int broj;
   scanf("%x", &broj);

46   /* Ispisujemo binarnu reprezentaciju unetog broja */
48   print_bits(broj);

50   /* Ispisujemo binarnu reprezentaciju broja dobijenog pozivom
   funkcije mirror */
52   print_bits(mirror(broj));

54   return 0;
}
```

Rešenje 1.11

```
#include <stdio.h>

2
/* Funkcija vraća 1 ukoliko je u binarnoj reprezentaciji broja n
4 broj jedinica veći od broja nula. U suprotnom funkcija vraća
0 */
6 int Broj01(unsigned int n)
7 {
8
9   int broj_nula, broj_jedinica;
10  unsigned int maska;

12  broj_nula = 0;
   broj_jedinica = 0;

14  /* Postavljamo masku tako da počinjemo sa analiziranjem bita
   najveće težine */
16  maska = 1 << (sizeof(unsigned int) * 4 - 1);

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

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

```

    broj_nula++;
28 }

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

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

40 int main()
42 {
    unsigned int n;
44
    /* Ucitavamo broj */
46     scanf("%u", &n);

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

```

Rešenje 1.12

```

#include <stdio.h>
2
int broj_parova(unsigned int x)
4 {
    int broj_parova;
    unsigned int maska;
6
    /* Postavljamo broj parova na 0 */
8     broj_parova = 0;

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

16

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

22         /* Proveravamo da li se na najmanjim pozicijama broj x

```

1 Uvodni zadaci

```

    nalazi 11 par */
24     if ((x & maska) == maska) {
        broj_parova++;
26     }

28     /* Pomeramo broj u desnu stranu tako da mozemo da ocitamo
        vrednost sledeceg para bitova */
30     x = x >> 1;
    }

32

34     return broj_parova;
36 }

38 int main()
39 {
40     unsigned int x;

42     /* Ucitavamo broj */
43     scanf("%u", &x);
44
45     /* Ispsujemo vrednost funkcije */
46     printf("%d\n", broj_parova(x));
47
48     return 0;
49 }
```

Rešenje 1.13

Rešenje 1.14

```

#include <stdio.h>

2
/*
4     Niska koju formiramo je duzine (sizeof(unsigned int)*8)/4 +1
    jer za svaku heksadekadnu cifru nam trebaju 4 binarne cifre i
6     jedna dodatna pozicija nam treba za terminirajucu nulu.

8     Prethodni izraz je identican sa sizeof(unsigned int)*2+1.

10    Na primer, ako je duzina unsigned int 4 bajta onda je
    MAX_DUZINA 9 */

12
13 #define MAX_DUZINA sizeof(unsigned int)*2 +1
14

15 void prevod(unsigned int x, char s[])
16 {
17
18 }
```



```
int i;
20 unsigned int maska;
int vrednost;

22 /* Heksadekadni zapis broja 15 je 000...0001111 - ovo nam
24 odgovara ako hocemo da citamo 4 uzastopne cifre */
maska = 15;

26
/*
28 Broj cemo citati od pozicije najmanje tezine ka poziciji
najvece tezine; npr. za broj
30 00000000001101000100001111010101 u prvom koraku cemo
procitati bitove: 0000000000110100010000111101<0101>
32 (bitove izdvojene sa <...>) u drugom koraku cemo procitati:
000000000011010001000011<1101>0101 u trecem koraku cemo
34 procitati: 00000000001101000100<0011>11010101 i tako redom

36 indeks i oznacava poziciju na koju smestamo vrednost

38 */
for (i = MAX_DUZINA - 2; i >= 0; i--) {
40 /* Vrednost izdvojene cifre */
vrednost = x & maska;

42
/* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter
44 dobijamo dodavanjem ASCII koda '0' Ako je vrednost iz
opsega od 10 do 15 odgovarajuci karakter dobijamo tako
46 sto prvo oduzmemo 10 (dobijamo vrednosti od 0 do 5) pa
dodamo ASCII kod 'A' (time dobijamo slova 'A', 'B', ...
48 'F') */
if (vrednost < 10) {
50 s[i] = vrednost + '0';
} else {
52 s[i] = vrednost - 10 + 'A';
}

54
/* Broj pomeramo za 4 bita u desnu stranu tako da mozemo da
56 procitamo sledecu cifru */
x = x >> 4;

58 }

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

62
int main()
64 {

66 unsigned int x;
char s[MAX_DUZINA];

68
/* Ucitavamo broj */
70 scanf("%u", &x);
```

```
72  /* Pozivamo funkciju */
    prevod(x, s);
74
76  /* Ispisujemo dobijenu nisku */
    printf("%s\n", s);
78
    return 0;
}
```

Rešenje 1.17

```
#include <stdio.h>

2
/* Iskomentarisan je deo koji se ispisuje svaki put kad se udje
4  u funkciju. Odkomentarisati pozive printf funkcije u obe
   funkcije da uocite razliku u broju rekurzivnih poziva obe
6  verzije. */

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

16  return x * stepen(x, k - 1);

18  /* Celo telo funkcije se moze ovako kratko zapisati return k
    == 0 ? 1 : x * stepen(x,k-1); */
20 }

22 /* Druga verzija prethodne funkcije. Obratiti paznju na
   efikasnost u odnosu na prvu verziju! */
24

26 /* Logaritamsko resenje je zasnovano na cinjenicama: -  $x^0 = 1$ ; -
    $x^k = x * (x^2)^{(k/2)}$ , za neparno k -  $x^k = (x^2)^{(k/2)}$ ,
28  za parno k

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

38  /* Ako je stepen paran */
    if ((k % 2) == 0)
```

```

40     return stepen2(x * x, k / 2);
    /* Inace (ukoliko je stepen neparan) */
42     return x * stepen2(x * x, k / 2);
}

44
45 int main()
46 {
47     int x, k;
48     scanf("%d%d", &x, &k);

50     printf("%d", stepen(2, 10));
    // printf("\n-----\n");
52     // printf("%d\n", stepen2(2, 10));
    return 0;
54 }

```

Rešenje 1.18

```

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

3

   #define MAX 100

5
   /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
7     (posrednu) rekurziju. */

9
   /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre
11  svoje definicije. Funkcija je mogla biti deklarirana i u telu
   funkcije paran. */

13
   unsigned neparan(unsigned n);

15
   /* Funkcija vraca 1 ako broj n ima paran broj cifara inace
17  vraca 0. */
   unsigned paran(unsigned n)
19  {
20      if (n >= 0 && n <= 9)
21          return 0;
22      else
23          return neparan(n / 10);
24  }

25
   /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace
27  vraca 0. */
   unsigned neparan(unsigned n)
29  {
30      if (n >= 0 && n <= 9)
31          return 1;
32      else
33          return paran(n / 10);

```

```

}
35
/* Glavna funkcija za testiranje */
37 int main()
{
39     int n;
    scanf("%d", &n);
41     printf("Uneti broj ima %sparan broj cifara\n",
            (paran(n) == 1 ? "" : "ne"));
43     return 0;
}

```

Rešenje 1.19

```

#include <stdio.h>
2
/* Repno-rekurzivna (eng. tail recursive) je ona funkcija
   Cije se telo završava rekurzivnim pozivom, pri čemu taj
   rekurzivni poziv ne učestvuje u nekom izrazu.
4
   Kod ovih funkcija se po završetku za tekuci rekurzivni
   poziv umesto skoka na adresu povratka skace na adresu
   povratka za prethodni poziv, odnosno za poziv na manjoj
   dubini. Time se stedi i prostor i vreme.
6
   Ovakve funkcije se mogu lako zameniti odgovarajućom
   iterativnom funkcijom, čime se smanjuje prostorna
   složenost algoritma. */
12
/* Pomoćna funkcija koja izračunava n! * result. Koristi
   repnu rekurziju. */
14
/* Result je argument u kom ćemo akumulirati do tada
   izračunatu vrednost faktoriijela. Kada završimo, tj. kada
   dodjemo do izlaza iz rekurzije potrebno je da vratimo
   result. */
18
int faktoriijelRepna(int n, int result)
{
20
22     if (n == 0)
        return result;
24
    return faktoriijelRepna(n - 1, n * result);
26
}

28 /* Sada želimo da se oslobodimo repne rekurzije koja postoji u
   funkciji faktoriijelRepna, koristeći algoritam sa predavanja.
30
   Najpre ćemo vrednost argumenta funkcije postaviti na vrednost
   koja bi se prosledjivala rekurzivnom pozivu i pomoću goto
   naredbe vratiti se na početak tela funkcije. */
32
34 int faktoriijelRepna_v1(int n, int result)
36 {
    pocetak:

```

```

38     if (n == 0)
39         return result;
40
41     result = n * result;
42     n = n - 1;
43     goto pocetak;
44 }
45
46 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
47    programerska praksa. Iskoristicemo prethodni medjukorak da
48    bismo dobili iterativno resenje bez bezuslovnih skokova. */
49 int faktorijelRepna_v2(int n, int result)
50 {
51     while (n != 0) {
52         result = n * result;
53         n = n - 1;
54     }
55
56     return result;
57 }
58
59 /* Nasim gore navedenim funkcijama pored n, mora da se salje i 1
60    za vrednost drugog argumenta u kome ce se akumulirati
61    rezultat. Funkcija faktorijel(n) je ovde radi udobnosti
62    korisnika, jer je sasvim prirodno da za faktorijel zahteva
63    samo 1 parametar. Funkcija faktorijel izracunava n!, tako Sto
64    odgovarajucoj gore navedenoj funkciji koja zaista racuna
65    faktorijel, salje ispravne argumente i vraca rezultat koju
66    joj ta funkcija vrati. Za testiranje, zameniti u telu
67    funkcije faktorijel poziv faktorijelRepna sa pozivom
68    faktorijelRepna_v1, a zatim sa pozivom funkcije
69    faktorijelRepna_v2. */
70 int faktorijel(int n)
71 {
72     return faktorijelRepna(n, 1);
73 }
74
75 /* Test program */
76 int main()
77 {
78     int n;
79
80     printf("Unesite n (<= 12): ");
81     scanf("%d", &n);
82     printf("%d! = %d\n", n, faktorijel(n));
83
84     return 0;
85 }

```

Rešenje 1.20

Rešenje 1.21

Rešenje 1.22

```
1  #include <stdio.h>
2  #define MAX_DIM 1000
3
4  /*
5   * Ako je n==0, onda je suma(a,0) = 0 Ako je n>0, onda je
6   * suma(a,n) = a[n-1] + suma(a,n-1) Suma celog niza je jednaka
7   * sumi prvih n-1 elementa uvecenoj za poslednji element celog
8   * niza. */
9
10 int sumaNiza(int *a, int n)
11 {
12     /* Ne stavljamo strogu jednakost n==0, za slucaj da korisnik
13     * prilikom prvog poziva, posalje negativan broj za velicinu
14     * niza. */
15     if (n <= 0)
16         return 0;
17
18     return a[n - 1] + sumaNiza(a, n - 1);
19 }
20
21 /*
22 * n==0, suma(a,0) = 0 n >0, suma(a,n) = a[0]+suma(a+1,n-1) Suma
23 * celog niza je jednaka zbiru prvog elementa niza i sume
24 * preostalih n-1 elementa. */
25
26 int sumaNiza2(int *a, int n)
27 {
28     if (n <= 0)
29         return 0;
30
31     return a[0] + sumaNiza2(a + 1, n - 1);
32 }
33
34 int main()
35 {
36     int a[MAX_DIM];
37     int n, i = 0;
38
39     /* Ucitavamo broj elemenata niza */
40     scanf("%d", &n);
41
42     /* Ucitavamo n elemenata niza. */
43     for (i = 0; i < n; i++)
44         scanf("%d", &a[i]);
45
46     printf("Suma elemenata je %d\n", sumaNiza(a, n));
47     // printf("Suma elemenata je %d\n", sumaNiza2(a, n));
48
49     return 0;
50 }
```

48 }

Rešenje 1.23

```

#include <stdio.h>
#define MAX_DIM 256

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

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

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

int main()
{
    int brojevi[MAX_DIM];
    int n;

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

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

```

Rešenje 1.24

```

#include <stdio.h>
#define MAX_DIM 256

int skalarno(int a[], int b[], int n)
{

```

```
6  /* Izlazak iz rekurzije */
   if (n == 0)
8     return 0;

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

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

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

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

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

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

32     return 0;
34 }
```

Rešenje 1.25

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

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

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

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

18     /* UCitavamo broj koji se trazi */
```



```

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

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

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

```

Rešenje 1.26

```

#include<stdio.h>
#define MAX_DIM 256

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

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

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

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

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

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

```

1 Uvodni zadaci

```
printf("ne\n");
36 return 0;
38 }
```

Rešenje 1.27

```
#include <stdio.h>

2
/* funkcija koja broji bitove svog argumenta */
/*
4 ako je x ==0, onda je count(x) = 0 inace count(x) =
6 najvisi_bit +count(x<<1)

8 Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
siftovanjem sa desne strane uvek dopisuju 0, argument x ce u
10 nekom rekurzivnom pozivu biti bas 0 i izacicemo iz rekurziije. */

12 int count(int x)
{
14 /* izlaz iz rekurziije */
if (x == 0)
16 return 0;

18 /* Dakle, neki bit je postavljen na 1. */
/* Proveravamo vrednost najviseg bita Kako za rekurzivni poziv
20 moramo slati siftovano x i x je oznacen ceo broj, onda ne
smemo koristiti siftovanje desno, jer funkciji moze biti
22 prosleden i negativan broj. Iz tog razloga, odlucujemo se
da proveramo najvisi, umesto najnizeg bita */
24 if (x & (1 << (sizeof(x) * 8 - 1)))
return 1 + count(x << 1);
26 /* Najvisi bit je 1. Sacekacemo da zavrshi poziv koji racuna
koliko ima jedinica u ostatku binarnog zapisa x i potom
28 uvecati taj rezultat za 1. */
else
30 /* Najvisi bit je 0. Stoga je broj jedinica u zapisu x isti
kao broj jedinica u zapisu broja x<<1, jer se siftovanjem
32 u levo sa desne stane dopisuju 0. */
return count(x << 1);
34

/* jednolinijska return naredba sa proverom i rekurzivnim
36 pozivom return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) +
count(x<<1); */
38 }

40 int main()
{
42 int x;
scanf("%x", &x);
44 printf("%d\n", count(x));
```

```
46     return 0;
}
```

Rešenje 1.29

```
#include<stdio.h>

2
/* Rekurzivna funkcija za odredjivanje najveće heksadekadne
4     cifre u broju */
int max_oktalna_cifra(unsigned x)
6 {
    /* izlazak iz rekurziije */
8     if (x == 0)
        return 0;
10    /* Odredjivanje poslednje heksadekadne cifre u broju */
    int poslednja_cifra = x & 7;
12    /* Odredjivanje maksimalne oktalne cifre u broju kada se iz
        njega izbrise poslednja oktalna cifra */
14    int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
    return poslednja_cifra >
16        max_bez_poslednje_cifre ? poslednja_cifra :
        max_bez_poslednje_cifre;
18 }

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

Rešenje 1.30

```
#include<stdio.h>

2
/* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u
4     broju */
int max_heksadekadna_cifra(unsigned x)
6 {
    /* Izlazak iz rekurziije */
8     if (x == 0)
        return 0;
10    /* Odredjivanje poslednje heksadekadne cifre u broju */
    int poslednja_cifra = x & 15;
12    /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
        njega izbrise poslednja heksadekadna cifra */
14 }
```

```
16     int max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);
    return poslednja_cifra >
        max_bez_poslednje_cifre ? poslednja_cifra :
18     max_bez_poslednje_cifre;
}

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

Rešenje 1.31

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

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

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

17
18     /* Ucitavamo nisku sa ulaza */
19     scanf("%s", s);

20
21     /* Odredjujemo duzinu niske */
22     n = strlen(s);

23
24     /* Ispisujemo na izlazu poruku da li je niska palindrom ili
25        nije */
26     if (palindrom(s, n))
27         printf("da\n");
28     else
29         printf("ne\n");
30
31     return 0;
32 }
```

Rešenje 1.32

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX_DUZINA_NIZA 50
4
5 void ispisiNiz(int a[], int n)
6 {
7     int i;
8
9     for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
11     printf("\n");
12 }
13
14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
15    prethodnih 1...n mesta */
16 int koriscen(int a[], int n, int x)
17 {
18     int i;
19     for (i = 1; i <= n; i++)
20         if (a[i] == x)
21             return 1;
22
23     return 0;
24 }
25
26 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[]
27    je niz u koji smesta permutacije m - oznacava da se na m-tu
28    poziciju u permutaciji smesta jedan od preostalih celih
29    brojeva n- je velicina skupa koji se permutuje Funkciju
30    pozivamo sa argumentom m=1 jer krecemo da formiramo
31    permutaciju od 1. pozicije i nikada ne koristimo a[0]. */
32 void permutacija(int a[], int m, int n)
33 {
34     int i;
35
36     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
37        broj premasila velicinu skupa, onda se svi brojevi vec
38        nalaze u permutaciji i ispisujemo permutaciju. */
39     if (m > n) {
40         ispisiNiz(a, n);
41         return;
42     }
43
44     /* Ideja: pronalazimo prvi broj koji mozemo da postavimo na
45        m-to mesto u nizu (broj koji se do sada nije pojavio u
46        permutaciji). Zatim, rekurzivno pronalazimo one permutacije
47        koje odgovaraju ovako postavljenom pocetku permutacije.
48        Kada to zavravimo, proveravamo da li postoji jos neki broj
49        koji moze da se stavi na m-to mesto u nizu (to se radi u
50        petlji). Ako ne postoji, funkcija je zavrSila sa radom.
```

```
51     Ukoliko takav broj postoji, onda ponovo pozivamo rekurzivno
52     pronalazenje odgovarajucih permutacija, ali sada sa
53     drugacije postavljenim prefiksom. */
54
55     for (i = 1; i <= n; i++) {
56         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do
57            m-1 pozicije, onda ga stavljamo na poziciju m i pozivamo
58            funkciju da napravi permutaciju za jedan vece duzine, tj.
59            m+1. Inace, nastavljamo dalje, trazeci broj koji se nije
60            pojavio do sada u permutaciji. */
61         if (!koriscen(a, m - 1, i)) {
62             a[m] = i;
63             /* Pozivamo ponovo funkciju da dopuni ostatak permutacije
64                posle upisivanja i na poziciju m. */
65             permutacija(a, m + 1, n);
66         }
67     }
68 }
69
70 int main(void)
71 {
72     int n;
73     int a[MAX_DUZINA_NIZA];
74
75     printf("Unesite duzinu permutacije: ");
76     scanf("%d", &n);
77     if (n < 0 || n >= MAX_DUZINA_NIZA) {
78         fprintf(stderr,
79             "Duzina permutacije mora biti broj veci od 0 i manji od %
80             d!\n",
81             MAX_DUZINA_NIZA);
82         exit(EXIT_FAILURE);
83     }
84
85     permutacija(a, 1, n);
86
87     exit(EXIT_SUCCESS);
88 }
```

Rešenje 1.33

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

```

11         && k < n) ? binomniKoeficijent(n - 1,
12                                     k - 1) +
13         binomniKoeficijent(n - 1, k) : 1;
14     }
15     /* Iterativno izracunavanje datog binomnog koeficijenta.
16
17     int binomniKoeficijent (int n, int k) { int i, j, b; for
18         (b=i=1, j=n; i<=k; b=b*j--/i++); return b; }
19
20     */
21
22     /* Prostim opaZanjem se uocava da se svaki element n-te
23     hipotenuze (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1
24     hipotenuze. Uz pomenute dve nove ivicne jedinice lako se
25     zakljucuje da ce suma elementa n-te hipotenuze biti tacno 2
26     puta veca. */
27     int sumaElemenataHipotenuze(int n)
28     {
29         return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
30     }
31
32     int main()
33     {
34         int n, k, i, d;
35
36
37         scanf("%d %d", &d, &n);
38
39         /* Ispisivanje Paskalovog trougla */
40         putchar('\n');
41         for (n = 0; n <= d; n++) {
42             for (i = 0; i < d - n; i++)
43                 printf(" ");
44             for (k = 0; k <= n; k++)
45                 printf("%4d", binomniKoeficijent(n, k));
46             putchar('\n');
47         }
48
49         if (n < 0) {
50             fprintf(stderr,
51                 "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
52             );
53             exit(EXIT_FAILURE);
54         }
55         printf("%d\n", sumaElemenataHipotenuze(n));
56
57         exit(EXIT_SUCCESS);
58     }

```


Glava 2

Pokazivači

2.1 Pokazivačka aritmetika

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

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

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

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

Zadatak 2.2 Dat je niz realnih brojeva dimenzije n .

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

2 Pokazivači

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

Test 1

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

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

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

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

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

Test 1

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

Test 2

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

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

Test 1

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

Test 2

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

Test 3

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

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

Test 1

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

Test 2

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

Test 3

```
Poziv: ./a.out ulaz.txt 2
(ne postoji datoteka ulaz.txt)
Izlaz: Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

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

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

Test 1

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

Test 2

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

Test 3

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

Test 3

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

2.2 Višedimenzioni nizovi

Zadatak 2.8 Data je kvadratna matrica dimenzije n .

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

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

Test 1

```

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

```

Test 2

```

Ulaz:  0
Izlaz: Greska: neodgovarajuca dimenzija matrice.

```

Zadatak 2.9 Date su dve kvadratne matrice istih dimenzija n .

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

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

Test 1

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

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

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

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

Test 1

```

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

```

Zadatak 2.11 Data je kvadratna matrica dimenzije n .

- Napisati funkciju koja određuje najveći element matrice na sporednoj dijagonali.
- Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.
- Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.
- Napisati funkciju koja određuje broj negativnih elemenata matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice čija se dimenzija n ($0 < n \leq 32$) zadaje kao argument komandne linije. Na standardni izlaz ispisati najveći element matrice na sporednoj dijagonali, indeks kolone koja sadrži najmanji element, indeks vrste koja sadrži najveći element i broj negativnih elemenata učitane matrice.

Milena: Izbegavala bih komentare koji ulaze u kod i na taj način narušavaju citljivost koda, kao što je to npr u funkciji `indeks_min` i `indeks_max`. Resenje 2.15 - izbacila bih napomenu iz komentara. Slicno mi se čini i za zadatak 2.17. Zadatak 2.17 - čini mi se da je resenje bez koriscenja biblioteckih funkcija visak? Zadatak 2.19 — izvuci komentare za učitaj i ispisi ispred funkcija, umesto što su

unutar funkcija. Zadatak 2.21 - cini mi se da komentari unutar funkcije izmeni narušavaju citljivost koda. Resenje 2.26 — izbaciti nasa slova iz komentara, izbaciti mozda napomenu sa pocetka jer je suvisna

Test 1

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

Test 2

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

Test 3

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

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

Test 1

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

Test 2

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

Test 3

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

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati

2.3 Dinamička alokacija memorije

dimenzije matrice n ($0 < n \leq 10$) i m ($0 < m \leq 10$), a zatim i elemente matrice (pozivom gore napisane funkcije). Na standardni izlaz spiralno ispisati elemente učitane matrice.

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

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

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

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

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

2.3 Dinamička alokacija memorije

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

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

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

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

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

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

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

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

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

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

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

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

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

Test 1

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

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

Test 1

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

Zadatak 2.21 Data je kvadratna realna matrica dimenzije n .

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

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

Test 1

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

Zadatak 2.22 Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci „slicice.txt“ se nalaze informacije o sličicama koje mu nedostaju u formatu: `redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`. Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronade ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. Napomena: za realokaciju memorije koristiti `realloc()` funkciju. **Jelena: treba dodati test primer.**

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

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

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

2.4 Pokazivači na funkcije

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

Test 1

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

Test 2

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

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

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

Test 1

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

Test 2

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

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

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

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

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

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

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

2.5 Rešenja

Rešenje 2.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 100
5
6  /* Funkcija obrće elemente niza koriscenjem indekse sintakse */
7  void obrni_niz_v1(int a[], int n)
8  {
9      int i, j;
10
11     for (i = 0, j = n - 1; i < j; i++, j--) {
12         int t = a[i];
13         a[i] = a[j];
14         a[j] = t;
15     }
16 }
```

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

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

62 }

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

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

Rešenje 2.2

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

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

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

14   return s;
}
16

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



```
{
20     double p = 1;

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

24     return p;
26 }

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

34     /* Ispitujemo da li se medju ostalim elementima niza nalazi
        najmanji */
36     for (i = 1; i < n; i++)
        if (a[i] < min)
38             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));
80    printf("proizvod = %5.3f\n", proizvod(a, n));
    printf("min = %5.3f\n", min(a, n));
82    printf("max = %5.3f\n", max(a, n));

84    return 0;
}
```

Rešenje 2.3

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

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

14    while (prvi < poslednji) {

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

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

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

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

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

```

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

```

Rešenje 2.4

```

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

```

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

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

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

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

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

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

42
/* koristeći pokazivačku sintaksu */

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

46
return 0;
48 }
```

Rešenje 2.5

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

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

```

    return 0;
12  return 1;
    }

14
int main(int argc, char **argv)
16 {
    int i, n = 0;

18
    /* Multi argument komandne linije je ime izvrsnog programa */
20    for (i = 1; i < argc; i++)
        if (palindrom(*(argv + i)))
22        n++;

24    printf("%d\n", n);
    return 0;
26 }

```

Rešenje 2.6

```

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

13
int main(int argc, char **argv)
15 {
    char rec[MAX_KARAKTERA];
    int br = 0, i = 0, n;
    FILE *in;

19
    /* Ako korisnik nije uneo trazene argumente, prijavljujemo
21     gresku */
    if (argc < 3) {
23         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
25         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
                argv[0]);
        exit(EXIT_FAILURE);
27     }

29
    /* Otvaramo datoteku sa imenom koje se zadaje kao prvi
31     argument komandne linije. */
    in = fopen(*(argv + 1), "r");

```

2 Pokazivači

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

Rešenje 2.7

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define MAX_KARAKTERA 100
5
6  /* Funkcija strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
8  {
9      int i;
10     for (i = 0; *(src + i); i++)
11         *(dest + i) = *(src + i);
12 }
13
14 /* Funkcija strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
16 {
17     int i;
18     for (i = 0; *(s + i) == *(t + i); i++)
19         if (*(s + i) == '\0')
20             return 0;
21     return *(s + i) - *(t + i);
22 }
23
24 /* Funkcija strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
```

```

    for (i = 0; *(s + i); i++);
29     return i;
    }

31
/* Funkcija ispituje da li je niska zadata drugim argumentom
33     funkcije sufiks niske zadate prvi argumentom funkcije */
int sufiks_niske(char *niska, char *sufiks)
35 {
    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)
    {
47         int i;
        if (duzina_niske(prefiks) <= duzina_niske(niska)) {
49             for (i = 0; i < duzina_niske(prefiks); i++)
                if (*(prefiks + i) != *(niska + i))
51                 return 0;
            return 1;
53         } else
            return 0;
55     }

57 int main(int argc, char **argv)
    {
59         /* Ako korisnik nije uneo trazene argumente, prijavljujemo
            gresku */
61         if (argc < 4) {
            printf("Greska: ");
63             printf("Nedovoljan broj argumenata komandne linije.\n");
            printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
65                 argv[0]);
            exit(EXIT_FAILURE);
67         }

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

73         in = fopen(*(argv + 1), "r");
        if (in == NULL) {
75             fprintf(stderr, "Greska: ");
            fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
77                 argv[1]);
            exit(EXIT_FAILURE);
79         }
    }

```

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

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

Rešenje 2.8

```
#include <stdio.h>
2  #include <math.h>
  #include <stdlib.h>
4
  #define MAX 100
6
  /* Deklarisemo funkcije koje cemo kasnije da definisemo */
8  double euklidska_norma(int M[][MAX], int n);
  int trag(int M[][MAX], int n);
10 int gornja_vandijagonalna_norma(int M[][MAX], int n);

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

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

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

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

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



```

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

    int tr = trag(A, n);
    printf("trag = %d\n", tr);
    54

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

    58 return 0;
    60 }

    62 /* Definisemo funkcije koju smo ranije deklarirali */

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

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

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

    82 return sqrt(norma);
    84 }

```

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

Rešenje 2.9

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
   standardnog ulaza */
7 void ucitaj_matricu(int m[][MAX], int n)
8 {
9     int i, j;
10
11     for (i = 0; i < n; i++)
12         for (j = 0; j < n; j++)
13             scanf("%d", &m[i][j]);
14 }
15
16 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
   standardni izlaz */
17 void ispisi_matricu(int m[][MAX], int n)
18 {
19     int i, j;
20
21     for (i = 0; i < n; i++) {
22         for (j = 0; j < n; j++)
23             printf("%d ", m[i][j]);
24         printf("\n");
25     }
26 }
27
28
29 /* Funkcija proverava da li su zadate kvadratne matrice a i b
   dimenzije n jednake */
30 int jednake_matrice(int a[][MAX], int b[][MAX], int n)
31 {
32     int i, j;
33
34     for (i = 0; i < n; i++)
35         for (j = 0; j < n; j++)
36             if (a[i][j] != b[i][j])
37                 return 0;
38     return 1;
39 }
```

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

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

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

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

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

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

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

114
    return 0;
116 }
```

Rešenje 2.10

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

4  #define MAX 64

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

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

```

20     return 1;
22 }

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

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

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

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

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

56     return 1;
57 }

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

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

```

```

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

```

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

```
176     printf("Refleksivnost: %s\n",
            refleksivnost(m, n) == 1 ? "da" : "ne");
178
180     printf("Simetricnost: %s\n",
            simetricnost(m, n) == 1 ? "da" : "ne");
182
184     printf("Tranzitivnost: %s\n",
            tranzitivnost(m, n) == 1 ? "da" : "ne");
186
188     printf("Refleksivno zatvorenje:\n");
189     ref_zatvorenje(m, n, pomocna);
190     pisi_matricu(pomocna, n);
191
192     printf("Simetricno zatvorenje:\n");
193     sim_zatvorenje(m, n, pomocna);
194     pisi_matricu(pomocna, n);
195
196     printf("Refleksivno-tranzitivno zatvorenje:\n");
197     tran_zatvorenje(m, n, pomocna);
198     pisi_matricu(pomocna, n);
199
200     /* Zatvaramo datoteku */
201     fclose(ulaz);
202
203     return 0;
204 }
```

Rešenje 2.11

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



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

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

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

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

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

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

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

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

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

Rešenje 2.12

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

4 #define MAX 32

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

```
10     int i, j;

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

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

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

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

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

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

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

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

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

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

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

53         }
54     }
55 }
```

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

Rešenje 2.13

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

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

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

Rešenje 2.14

Rešenje 2.15

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

```

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

26  /* Alociramo prostor za n celih brojeva */
28  if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
30      fprintf(stderr, "malloc(): ");
32      fprintf(stderr, "greska pri alokaciji memorije.\n");
34      exit(EXIT_FAILURE);
36  }

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

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

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

58  /* Oslobadjamo prostor */
60  free(p);

62  return 0;
64 }

```

Rešenje 2.16

```

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

```

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

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

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

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

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

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

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

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

58
        /* Završavamo program */
60      exit(EXIT_FAILURE);
62  }

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

68
```



```

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

72

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

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

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

       }

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

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

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

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

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

```

Rešenje 2.17

```

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

```

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

```

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

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

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

70  return 0;
}

```

Rešenje 2.18

```

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

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

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

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

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

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

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

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

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

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

```

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

44     exit(EXIT_FAILURE);
    }
46 }

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

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

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

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

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

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

70 return 0;
}
```

Rešenje 2.19

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

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

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

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

2 Pokazivači

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

Rešenje 2.20

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

Rešenje 2.21

```
#include <stdio.h>
2 #include <stdlib.h>
#include <math.h>
4
/* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni(float **a, int n)
{
8     int i, j;
10    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
12        /* Ako je indeks vrste manji od indeksa kolone */
            if (i < j)
14                /* element se nalazi iznad glavne dijagonale pa ga
                    polovimo */
16                a[i][j] /= 2;
            else
18                /* Ako je indeks vrste veci od indeksa kolone */
                if (i > j)
20                    /* element se nalazi ispod glavne dijagonale pa ga
                        dupliramo */
22                    a[i][j] *= 2;
}
24
/* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
26    sporedne dijagonale */
float zbir_ispod_sporedne_dijagonale(float **m, int n)
28 {
    int i, j;
30    float zbir = 0;
32    for (i = 0; i < n; i++)
```

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

40     return zbir;
}

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

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

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

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

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

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

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

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

```

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

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

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

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

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

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



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

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

142
    float zbir = zbir_ispod_sporedne_dijagonale(a, n);

144
    /* Pozivamo funkciju za modifikovanje elemenata */
    izmeni(a, n);

146
    /* Ispisujemo rezultat */
    printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
150    printf("je %.2f.\n", zbir);

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

154
    /* Oslobadjamo memoriju */
    oslobodi_memoriju(a, n);

156
    /* Zatvaramo datoteku */
    fclose(ulaz);

160
    /* i prekidamo sa izvršavanjem programa */
162    return 0;
}
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

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

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

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

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

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

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

52
54      /* Ako korisnik nije uneo žtraene argumente, prijavljujemo
        šgreku */
56      if (argc < 2) {
        printf("Greska: ");
58        printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
60          argv[0]);
        exit(EXIT_FAILURE);
```

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

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

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

122      exit(EXIT_SUCCESS);
}
```

Rešenje [2.27](#)

Rešenje [2.28](#)

Rešenje [2.29](#)

Glava 3

Algoritmi pretrage i sortiranja

3.1 Pretraživanje

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

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

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

Test 1

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

```
Izlaz:
```

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

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

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

[Rešenje 3.1]

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

Test 1

```
Ulaz: 11 2 5 6 8 10 11 23
```

```
Izlaz:
```

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

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

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

Test 2

```
Ulaz: 14 10 32 35 43 66 89 100
```

```
Izlaz:
```

```
Linearna pretraga  
Element se ne nalazi u nizu.
```

```
Binarna pretraga  
Element se ne nalazi u nizu.
```

```
Interpolaciona pretraga  
Element se ne nalazi u nizu.
```

[Rešenje 3.2]

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

Upotreba programa 1

```

Datoteka:
20140003 Marina Petrovic
20140012 Stefan Mitrovic
20140032 Dejan Popovic
20140049 Mirko Brankovic
20140076 Sonja Stevanovic
20140104 Ivan Popovic
20140187 Vlada Stankovic
20140234 Darko Brankovic
Interakcija programa:
Unesite indeks studenta cije informacije želite: 20140076
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
Unesite prezime studenta cije informacije želite: Popovic
Indeks: 20140032, Ime i prezime: Dejan Popovic

```

[Rešenje 3.3]

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

[Rešenje 3.4]

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

Test 1

```

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

```

Test 2

```

Poziv: ./a.out dat.txt
Datoteka:
12 53
2.342 34.1
-0.3 23
-1 2.1
123.5 756.12
Izlaz: -1 2.1

```

[Rešenje 3.5]

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

3 Algoritmi pretrage i sortiranja

Test 1

|| Izlaz: 1.57031

[Rešenje 3.6]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.7]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.8]

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

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

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

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

<p><i>Test 1</i></p> <pre> Ulaz: 10 Izlaz: 3 3</pre>	<p><i>Test 2</i></p> <pre> Ulaz: 4 Izlaz: 2 2</pre>
<p><i>Test 3</i></p> <pre> Ulaz: 17 Izlaz: 4 4</pre>	<p><i>Test 4</i></p> <pre> Ulaz: 1031 Izlaz: 10 10</pre>

[Rešenje 3.9]

**** Zadatak 3.10** U prvom kvadrantu dato je $1 \leq N \leq 10000$ duži svojim koordinatama (duži mogu da se seku, preklapaju, itd.). Napisati program koji pronalazi najmanji ugao $0 \leq \alpha \leq 90^\circ$, na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom α 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 $[0, 90^\circ]$.*

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

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

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

```

```
{"Ljubivoje", "Rsumovic"};
```

Test 1	Test 2
<pre> Ulaz: R Izlaz: Dusko Radovic</pre>	<pre> Ulaz: E Izlaz: Dobrica Eric</pre>

3.2 Sortiranje

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

Test 1	Test 2
<pre> Ulaz: 23 64 123 76 22 7 Izlaz: 1</pre>	<pre> Ulaz: 21 654 65 123 65 12 61 Izlaz: 0</pre>

[Rešenje 3.12]

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

Test 1	Test 2
<pre> Ulaz: anagram ramgana Izlaz: jesu</pre>	<pre> Ulaz: anagram anagrnm Izlaz: nisu</pre>

[Rešenje 3.13]

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

Test 1	Test 2
<pre> Ulaz: 4 23 5 2 4 6 7 34 6 4 5 Izlaz: 4</pre>	<pre> Ulaz: 2 4 6 2 6 7 99 1 Izlaz: 2</pre>

[Rešenje 3.14]

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

<i>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

[Rešenje 3.15]

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

<i>Test 1</i>	<i>Test 2</i>
Ulaz:	Ulaz:
Unesite elemente prvog niza:	Unesite elemente prvog niza:
3 6 7 11 14 35 0	1 4 7 0
Unesite elemente drugog niza:	Unesite elemente drugog niza:
3 5 8 0	9 11 23 54 75 0
Izlaz:	Izlaz:
3 3 5 6 7 8 11 14 35	1 4 7 9 11 23 54 75

[Rešenje 3.16]

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

Test 1

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

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

Izlazna datoteka (ceo-tok.txt):

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

[Rešenje 3.17]

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

Upotreba programa 1

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

Upotreba programa 2

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

Upotreba programa 3

```

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

```

[Rešenje 3.18]

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

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

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

Test 1

```

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

```

Test 2

```

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

```

[Rešenje 3.19]

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

Test 1

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

[Rešenje 3.20]

Zadatak 3.21 Definisana je struktura podataka

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

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

Test 1

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

Test 2

```
Poziv: ./a.out in.txt out.txt
Ulazna datoteka (in.out):
Milijana Maric 2009
Izlazna datoteka (out.txt):
Milijana Maric 2009
```

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

Test 1

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

[Rešenje 3.22]

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

Upotreba programa 1

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

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

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

      UKUPNO: 270.00 dinara.

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

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

      UKUPNO: 230.00 dinara.

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

Kraj rada kase!
```

[Rešenje 3.23]

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

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

Test 1

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

[Rešenje 3.24]

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

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

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

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

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.25]

3 Algoritmi pretrage i sortiranja

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

Test 1

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

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

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

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

3.3 Bibliotečke funkcije pretrage i sortiranja

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

Upotreba programa 1

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

Upotreba programa 2

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

[Rešenje 3.28]

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

Upotreba programa 1

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

[Rešenje 3.29]

Zadatak 3.30 Korišćenjem 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 standardnom izlazu.

Upotreba programa 1

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

[Rešenje 3.30]

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

[Rešenje 3.31]

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

Upotreba programa 1

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

[Rešenje 3.32]

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

[Rešenje 3.33]

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

Test 1

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

Test 2

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

[Rešenje 3.34]

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

Test 1

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

Test 2

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

[Rešenje 3.35]

Zadatak 3.36 Definisana je struktura:

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

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

Test 1

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

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

Upotreba programa 1

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

Upotreba programa 2

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

[Rešenje 3.37]

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

Upotreba programa 1

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

Upotreba programa 2

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

3.4 Rešenja

Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt
7    opcijom -lrt zbog funkcije clock_gettime() */
8
9 /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
10    njemu element x. Pretraga se vrši prostom iteracijom kroz niz.
11    Ako se element pronadje funkcija vraca indeks pozicije na
12    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;
30     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 vracamo -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 vracamo indeks levi (ili indeks
73         desni. Ova provera je neophodna, zato sto bismo inace
            prilikom izracunavanja srednji imali deljenje nulom. */
75         else if (a[levi] == a[desni])
            return levi;
        /* Racunamo srednji indeks */
        srednji =
79         levi +
            ((double) (x - a[levi]) / (a[desni] - a[levi])) *
81         (desni - levi);
        /* Napomena: Indeks srednji je uvek izmedju levi i desni,

```

3 Algoritmi pretrage i sortiranja

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

```

135     sekundama od Nove godine 1970. random()%100 daje brojeve
        izmedju 0 i 99 */
137     srandom(time(NULL));
    for (i = 0; i < n; i++)
139         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;

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

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

167     /* Interpolaciona pretraga */
    printf("Interpolaciona pretraga\n");
169     clock_gettime(CLOCK_REALTIME, &time5);
    i = interpolaciona_pretraga(a, n, x);
171     clock_gettime(CLOCK_REALTIME, &time6);
    if (i == -1)
173         printf("Element nije u nizu\n");
    else
175         printf("Element je u nizu na poziciji %d\n", i);
    printf("-----\n");
177

    /* Upisujemo podatke o izvrsavanju programa u log fajl */
179     if ((f = fopen("vremena.txt", "a")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
181         exit(EXIT_FAILURE);
    }

183

185     fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
    fprintf(f, "\tLinearna pretraga:%10ld ns\n",
        (time2.tv_sec - time1.tv_sec) * 1000000000 +

```

3 Algoritmi pretrage i sortiranja

```
187         time2.tv_nsec - time1.tv_nsec);
188     fprintf(f, "\tBinarna: %19ld ns\n",
189         (time4.tv_sec - time3.tv_sec) * 1000000000 +
190         time4.tv_nsec - time3.tv_nsec);
191     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
192         (time6.tv_sec - time5.tv_sec) * 1000000000 +
193         time6.tv_nsec - time5.tv_nsec);
194
195     /* Zatvaramo datoteku */
196     fclose(f);
197
198     return 0;
199 }
```

Rešenje 3.2

```
#include <stdio.h>

2
int lin_pretgraga_rek_sufiks(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)
        return 0;
    /* Pretraga ostatka niza */
    tmp = lin_pretgraga_rek_sufiks(a + 1, n - 1, x);
    return tmp < 0 ? tmp : tmp + 1;
}

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

28
int bin_pretgraga_rek(int a[], int l, int d, int x)
30
{
    int srednji;
    if (l > d)
        return -1;
    /* Srednja pozicija na kojoj se trazi vrednost x */
    srednji = (l + d) / 2;
```

```

36  /* Ako je sredisnji element trazeni */
    if (a[srednji] == x)
38      return srednji;
    /* Ako je trazeni broj veci od srednjeg, pretrazujemo desnu
40     polovinu niza */
    if (a[srednji] < x)
42      return bin_pretgraga_rek(a, srednji + 1, d, x);
    /* Ako je trazeni broj manji od srednjeg, pretrazujemo levu
44     polovinu niza */
    else
46      return bin_pretgraga_rek(a, l, srednji - 1, x);
}

48

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

66
67 #define MAX 1024
68
69 int main()
70 {
    int a[MAX];
72     int x;
    int i, indeks;
74
    /* Ucitavamo trazeni broj */
76     printf("Unesite trazeni broj: ");
    scanf("%d", &x);
78
    /* Ucitavamo elemente niza sve do kraja ulaza - ocekujemo da
80     korisnik pritisne CTRL+D za naznaku kraja */
    i = 0;
82     printf("Unesite sortiran niz elemenata: ");
    while (scanf("%d", &a[i]) == 1) {
84         i++;
    }
86
    /* Linearna pretraga */

```

3 Algoritmi pretrage i sortiranja

```
88     printf("Linearna pretraga\n");
      indeks = lin_pretgraga_rek_sufiks(a, i, x);
90     if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
92     else
        printf("Pozicija elementa je %d.\n", indeks);
94
      /* Binarna pretraga */
96     printf("Binarna pretraga\n");
      indeks = bin_pretgraga_rek(a, 0, i - 1, x);
98     if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
100    else
        printf("Pozicija elementa je %d.\n", indeks);
102
      /* Interpolaciona pretraga */
104     printf("Interpolaciona pretraga\n");
      indeks = interp_pretgraga_rek(a, 0, i - 1, x);
106     if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
108     else
        printf("Pozicija elementa je %d.\n", indeks);
110
      return 0;
112 }
```

Rešenje 3.3

```
#include <stdio.h>
2 #include <stdlib.h>
  #include <string.h>
4
  #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16

8 /* 0 svakom studentu imamo 3 informacije i njih objedinjujemo u
   strukturu kojom cemo predstavljati svakog studenta. */
10 typedef struct {
    /* Indeks mora biti tipa long jer su podaci u datoteci
12     preveliki za int, npr. 20140123 */
    long indeks;
14     char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
16 } Student;

18 /* Ucitani niz studenata ce biti sortiran prema indeksu, jer cemo
   ih, redom, kako citamo smestati u niz, a u datoteci su vec
20 smesteni sortirani rastuce prema broju indeksa. Iz tog
   razloga pretragu po indeksu cemo vrsiti binarnom pretragom,
22 dok pretragu po prezimenu moramo vrsiti linearno, jer nemamo
   garancije da postoji uredjenje po prezimenu. */
```



```

24  /* Funkcija trazi u sortiranom nizu studenata a[] duzine n
26     studenta sa indeksom x. Vraca indeks pozicije nadjenog clana
        niza ili -1, ako element nije pronadjen */
28  int binarna_pretraga(Student a[], int n, long x)
    {
30      int levi = 0;
32      int desni = n - 1;
34      int srednji;
        /* Dokle god je indeks levi levo od indeksa desni */
        while (levi <= desni) {
36          /* Racunamo srednji indeks */
            srednji = (levi + desni) / 2;
38          /* Ako je srednji element veci od x, tada se x mora nalaziti
                u levoj polovini niza */
            if (x < a[srednji].indeks)
40                desni = srednji - 1;
            /* Ako je srednji element manji od x, tada se x mora
42                nalaziti u desnoj polovini niza */
            else if (x > a[srednji].indeks)
44                levi = srednji + 1;
            else
46                /* Ako je srednji element jednak x, tada smo pronasli x na
                    poziciji srednji */
                    return srednji;
48        }
        /* Ako nije pronadjen vracamo -1 */
50        return -1;
52    }

54  /* Linearnom pretragom niza studenata trazimo prezime x */
    int linearna_pretraga(Student a[], int n, char x[])
56    {
        int i;
58        for (i = 0; i < n; i++)
            /* Poredimo prezime i-tog studenta i poslato x */
60            if (strcmp(a[i].prezime, x) == 0)
                return i;
62        return -1;
    }

64  /* Main funkcija mora imate argumente jer se ime datoteke dobija
        kao argument komandne linije */
66  int main(int argc, char *argv[])
    {
68        /* Ucitacemo redom sve studente iz datoteke u niz. */
        Student dosije[MAX_STUDENATA];
        FILE *fin = NULL;
72        int i;
        int br_studenata = 0;
74        long trazen_indeks = 0;
        char trazeno_prezime[MAX_DUZINA];

```

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

```

128  /* Rezultat linearne pretrage */
    if (i == -1)
130      printf("Ne postoji student sa prezimenom %s\n",
              trazeno_prezime);
    else
132      printf("Indeks: %ld, Ime i prezime: %s %s\n",
134              dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

136  return 0;
}

```

Rešenje 3.4

```

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

5  #define MAX_STUDENATA 128
   #define MAX_DUZINA 16

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

13
15 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
   long x)
{
17  /* Ako je indeks elementa na levom kraju veci od indeksa
   elementa na desnom kraju dela niza koji se pretrazuje, onda
19  zapravo pretrazujemo prazan deo niza. U praznom nizu nema
   elementa koji trazimo i zato vracamo -1 */
21  if (levi > desni)
   return -1;
23  /* Racunamo indeks srednjeg elementa */
   int srednji = (levi + desni) / 2;
25  /* Da li je srednji, bas onaj kog trazimo? */
   if (a[srednji].indeks == x) {
27     return srednji;
   }
29  /* Ako je trazeni indeks manji od indeksa srednjeg, onda
   potragu nastavljamo u levoj polovini niza jer znamo da je
31  niz sortiran po indeksu u rastucem poretku. */
   if (x < a[srednji].indeks)
33     return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
   /* Inace ga treba traziti u desnoj polovini */
35  else
   return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }

```

3 Algoritmi pretrage i sortiranja

```
39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
41     /* Ako je niz prazan, vracamo -1, jer ga ne mozemo naci */
42     if (n == 0)
43         return -1;
44     /* Kako trazimo prvog studenta sa trazanim prezimenom,
45        pocinjemo sa prvim studentom u nizu. */
46     if (strcmp(a[0].prezime, x) == 0)
47         return 0;
48     int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
49     return i >= 0 ? 1 + i : -1;
50 }
51
52 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
53 {
54     /* Ako je niz prazan, vracamo -1, jer ga ne mozemo naci */
55     if (n == 0)
56         return -1;
57     /* Kako trazimo poslednjeg studenta sa trazanim prezimenom,
58        pocinjemo sa poslednjim studentom u nizu. */
59     if (strcmp(a[n - 1].prezime, x) == 0)
60         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
62 }
63
64 /* Main funkcija mora imate argumente jer se ime datoteke dobija
65    kao argument komandne linije */
66 int main(int argc, char *argv[])
67 {
68     /* Ucitacemo redom sve studente iz datoteke u niz. */
69     Student dosije[MAX_STUDENATA];
70     FILE *fin = NULL;
71     int i;
72     int br_studenata = 0;
73     long trazeni_indeks = 0;
74     char trazeno_prezime[MAX_DUZINA];
75
76     /* Proveravamo da li nam je korisnik prilikom poziva prosledio
77        ime datoteke sa informacijama o studentima */
78     if (argc != 2) {
79         fprintf(stderr,
80             "Greska: Program se poziva sa %s ime_datoteke\n",
81             argv[0]);
82         exit(EXIT_FAILURE);
83     }
84
85     /* Otvaramo datoteku */
86     fin = fopen(argv[1], "r");
87     if (fin == NULL) {
88         fprintf(stderr,
89             "Neuspesno otvaranje datoteke %s za citanje\n",
90             argv[1]);
```

```

91     exit(EXIT_FAILURE);
92 }
93
94 /* Citamo sve dok imamo red sa informacijama o studentu */
95 i = 0;
96 while (1) {
97     if (i == MAX_STUDENATA)
98         break;
99     if (fscanf
100         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
101         dosije[i].prezime) != 3)
102         break;
103     i++;
104 }
105 br_studenata = i;
106
107 /* Nakon citanja datoteka nam vise nije neophodna i odmah je
108 zatvaramo */
109 fclose(fin);
110
111 /* Unos indeksa koji se binarno trazi u nizu */
112 printf("Unesite indeks studenta cije informacije zelite: ");
113 scanf("%ld", &trazen_indeks);
114 i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
115                                 trazen_indeks);
116
117 if (i == -1)
118     printf("Ne postoji student sa indeksom %ld\n",
119            trazen_indeks);
120 else
121     printf("Indeks: %ld, Ime i prezime: %s %s\n",
122            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
123
124 printf("Unesite prezime studenta cije informacije zelite: ");
125 scanf("%s", trazeno_prezime);
126 i = linearna_pretraga_rekurzivna(dosije, br_studenata,
127                                 trazeno_prezime);
128
129 if (i == -1)
130     printf("Ne postoji student sa prezimenom %s\n",
131            trazeno_prezime);
132 else
133     printf
134         ("Poslednji takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
135         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
136
137 return 0;
138 }

```

Rešenje 3.5

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  /* Struktura koja opisuje tacku u ravni */
7  typedef struct Tacka {
8      float x;
9      float y;
10 } Tacka;
11
12 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13    pocetka (0,0) */
14 float rastojanje(Tacka A)
15 {
16     return sqrt(A.x * A.x + A.y * A.y);
17 }
18
19 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u
20    nizu zadatih tacaka t dimenzije n */
21 Tacka najbliza_koordinatnom(Tacka t[], int n)
22 {
23     Tacka najbliza;
24     int i;
25     najbliza = t[0];
26     for (i = 1; i < n; i++) {
27         if (rastojanje(t[i]) < rastojanje(najbliza)) {
28             najbliza = t[i];
29         }
30     }
31     return najbliza;
32 }
33
34 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
35    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)
```

```
{
54     Tacka najbliza;
55     int i;
56     najbliza = t[0];
57     for (i = 1; i < n; i++) {
58         if (fabs(t[i].y) < fabs(najbliza.y)) {
59             najbliza = t[i];
60         }
61     }
62     return najbliza;
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     /* Ocekujemo da korisnik unese barem ime izvrsne verzije
73        programa i ime datoteke sa tackama */
74     if (argc < 2) {
75         fprintf(stderr,
76             "koriscenje programa: %s ime_datoteke\n", argv[0]);
77         return EXIT_FAILURE;
78     }

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

86     /* Sve dok ima tacaka u datoteci, smestamo ih u niz sa
87        tackama; i predstavlja indeks tekuce tacke */
88     i = 0;
89     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
90         i++;
91     }
92     n = i;

93     /* Proveravamo koji su dodatni argumenti komandne linije. Ako
94        nema dodatnih argumenata */
95     if (argc == 2)
96         /* Trazimo najblizu tacku u odnosu na koordinatni pocetak */
97         najbliza = najbliza_koordinatnom(tacke, n);
98     /* Inace proveravamo koji je dodatni argument. Ako je u
99        pitanju opcija -x */
100 }
```

```
106     else if (strcmp(argv[2], "-x") == 0)
        /* 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)
        /* 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    /* Zatvaramo datoteku */
    fclose(ulaz);

124    return 0;
126 }
```

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;
        /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
21         tacnosti, prekidamo pretragu */
        if (fabs(cos(s)) < EPS) {
23             break;
        }
25     /* Ako je nula u levom delu intervala, nastavljamo pretragu
        na intervalu [l, s] */
    }
```



```

27     if (cos(l) * cos(s) < 0)
        d = s;
29     else
        /* Inace, nastavljamo pretragu na intervalu [s, d] */
31         l = s;
    }

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

37     return 0;
}

```

Rešenje 3.7

```

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

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

26 #define MAX 256

28 int main()
30 {
    int niz[MAX];
32    int n = 0;

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

```

3 Algoritmi pretrage i sortiranja

```
        n++;

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

42      return 0;
}
```

Rešenje 3.8

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

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

26 #define MAX 256

28 int main()
30 {
    int niz[MAX];
32    int n = 0;

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

38

    /* Stampanje rezultata */
40    printf("%d\n", prvi_manji_od_nule(niz, n));
}
```

```

42     return 0;
}

```

Rešenje 3.9

```

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

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

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

int main()
{
    unsigned int x;

    /* Unos podatka */
    printf("Unesi pozitivan ceo broj: ");
    scanf("%u", &x);

    /* Provera da li je uneti broj pozitivan */
    if (x == 0) {

```

3 Algoritmi pretrage i sortiranja

```
    fprintf(stderr, "Logaritam od nule nije definisan\n");
48    exit(EXIT_FAILURE);
}

50    /* Ispis povratnih vrednosti funkcija */
52    printf("%u %u\n", logaritam_a(x), logaritam_b(x));
54    return 0;
}
```

Rešenje 3.12

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

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

12    /* U svakoj iteraciji ove petlje se pronalazi najmanji element
       medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
14    poziciju i, dok se element na poziciji i premesta na
       poziciju min, na kojoj se nalazio najmanji od gore
16    navedenih elemenata. */
    for (i = 0; i < n - 1; i++) {
18        /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
           nalazi najmanji od elemenata a[i],...,a[n-1]. */
20        min = i;
        for (j = i + 1; j < n; j++)
22            if (a[j] < a[min])
                min = j;
24        /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
           samo ako su (i) i min razliciti, inace je nepotrebno. */
26        if (min != i) {
            pom = a[i];
            a[i] = a[min];
            a[min] = pom;
28        }
30    }
32 }

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++)
41         if (a[i] - a[i - 1] < min)
42             min = a[i] - a[i - 1];
43     return min;
44 }

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

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

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

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

62     return 0;
63 }
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 {
9     int i, j, min;
10    char pom;
11    for (i = 0; i < n; i++) {
12        min = i;
13        for (j = i + 1; j < n; j++)
14            if (s[j] < s[min])
15                min = j;
16        if (min != i) {
17            pom = s[i];
18            s[i] = s[min];
19            s[min] = pom;
20        }
21    }
22 }
```

3 Algoritmi pretrage i sortiranja

```
24 /* Funkcija vraća: 1 - ako jesu anagrami; 0 - inace. */
int anagrami(char s[], char t[], int n_s, int n_t)
26 {
    int i, n;

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

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

38     n = n_s;

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

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

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

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

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

Rešenje 3.14

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

```

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

22 /* Funkcija za odredjivanje onog elementa sortiranog niza koji
    se najviše 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
            najviše 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 najviše puta pojavio u nizu */
40     return a[i_max_pojava];
}

42
int main()
44 {
    int a[MAX_DIM], i;
46
    /* Ucitavaju se elementi 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);

```

3 Algoritmi pretrage i sortiranja

```
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;
    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
/* Funkcija za binarnu pretragu niza. funkcija vraca: 1 - ako se
22 element x nalazi u nizu; 0 - inace. pretpostavlja 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;
        else if (a[srednji] > x)
32             desni = srednji - 1;
        else if (a[srednji] < x)
34             levi = srednji + 1;
    }
36     return 0;
38 }

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



```

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

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

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

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

68     return 0;
}

```

Rešenje 3.16

```

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

3

   int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
5       int dim3)
   {
7       int i = 0, j = 0, k = 0;
       /* U slucaju da je dimenzija treceg niza manja od neophodne,
9       funkcija vraca -1 */
       if (dim3 < dim1 + dim2)
11          return -1;

13       /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
          jednog od njih */
15       while (i < dim1 && j < dim2) {
           if (niz1[i] < niz2[j])
17               niz3[k++] = niz1[i++];
           else
19               niz3[k++] = niz2[j++];
       }
21       /* Ostatak prvog niza prepisujemo u treci */
       while (i < dim1)

```

3 Algoritmi pretrage i sortiranja

```
23     niz3[k++] = niz1[i++];

25     /* Ostatak drugog niza prepisujemo u treci */
    while (j < dim2)
27         niz3[k++] = niz2[j++];
    return dim1 + dim2;
29 }

31 int main()
{
33     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
    int i = 0, j = 0, k, dim3;

35

37     /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
        Pretpostavka je da na ulazu nece biti vise od MAX_DIM
        elemenata */
39     printf("Unesite elemente prvog niza: ");
    while (1) {
41         scanf("%d", &niz1[i]);
        if (niz1[i] == 0)
43             break;
        i++;
45     }
    printf("Unesite elemente drugog niza: ");
47     while (1) {
        scanf("%d", &niz2[j]);
49         if (niz2[j] == 0)
            break;
51         j++;
    }

53

55     /* Poziv trazene funkcije */
    dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

57     /* Ispis niza */
    for (k = 0; k < dim3; k++)
59         printf("%d ", niz3[k]);
    printf("\n");

61     return 0;
63 }
```

Rešenje 3.17

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

4

int main(int argc, char *argv[])
6 {
    FILE *fin1 = NULL, *fin2 = NULL;
```

```
8 FILE *fout = NULL;
   char ime1[11], ime2[11];
10 char prezime1[16], prezime2[16];
   int kraj1 = 0, kraj2 = 0;

12
   /* Ako nema dovoljno argumenata komandne linije */
14 if (argc < 3) {
       fprintf(stderr,
16         "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
       exit(EXIT_FAILURE);
18 }

20 /* Otvaramo datoteku zadatu prvim argumentom komandne linije */
   fin1 = fopen(argv[1], "r");
22 if (fin1 == NULL) {
       fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",
24         argv[1]);
       exit(EXIT_FAILURE);
26 }

28 /* Otvaramo datoteku zadatu drugim argumentom komandne linije */
   fin2 = fopen(argv[2], "r");
30 if (fin2 == NULL) {
       fprintf(stderr, "Neuspesno otvaranje datoteke %s\n",
32         argv[2]);
       exit(EXIT_FAILURE);
34 }

36 /* Otvaranje datoteke za upis rezultata */
   fout = fopen("ceo-tok.txt", "w");
38 if (fout == NULL) {
       fprintf(stderr,
40         "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
       exit(EXIT_FAILURE);
42 }

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

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

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

3 Algoritmi pretrage i sortiranja

```
60     kraj1 = 1;
61 } else {
62     /* Ime i prezime iz druge datoteke je leksikografski
63        ranije, upisujemo ga u izlaznu datoteku */
64     fprintf(fout, "%s %s\n", ime2, prezime2);
65     /* Citamo narednog studenta iz druge datoteke */
66     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
67         kraj2 = 1;
68 }
69 }
70
71 /* Ako smo iz prethodne petlje izašli zato što se doslo do
72    kraja druge datoteke, onda ima još imena u prvoj datoteci, i
73    prepisujemo ih, redom, jer su već sortirani po imenu. */
74 while (!kraj1) {
75     fprintf(fout, "%s %s\n", ime1, prezime1);
76     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
77         kraj1 = 1;
78 }
79
80 /* Ako smo iz prve petlje izašli zato što se doslo do kraja
81    prve datoteke, onda ima još imena u drugoj datoteci, i
82    prepisujemo ih, redom, jer su već sortirani po imenu. */
83 while (!kraj2) {
84     fprintf(fout, "%s %s\n", ime2, prezime2);
85     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
86         kraj2 = 1;
87 }
88
89 /* Zatvaramo datoteke */
90 fclose(fin1);
91 fclose(fin2);
92 fclose(fout);
93
94 return 0;
95 }
```

Rešenje 3.18

```
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 insertion sort(int a[], int n);
9 /* Bubble sort */
10 void bubblesort(int a[], int n);
11 /* Shell sort */
12 void shellsort(int a[], int n);
```

```

13 /* Merge sort */
void mergesort(int a[], int l, int r);
15 /* Quick sort */
void quicksort(int a[], int l, int r);
17
#endif

/* Datoteka sort.c */
2
#include "sort.h"
4
/* Funkcija sortira niz celih brojeva metodom sortiranja
6 izborom. Ideja algoritma je sledeca: U svakoj iteraciji
pronalazimo najmanji element i postavljamo ga na pocetak
8 niza. Dakle, u prvoj iteraciji, pronalazimo najmanji element,
i dovodimo ga na nulto mesto u nizu. U i-toj iteraciji
10 najmanjih i elemenata su vec na svojim pozicijama, pa od i+1
do n-1 elementa trazimo najmanji, koji dovodimo na i+1
12 poziciju. */
void selectionsort(int a[], int n)
14 {
    int i, j;
    int min;
    int pom;
16
    /* U svakoj iteraciji ove petlje se pronalazi najmanji element
20 medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
poziciju i, dok se element na poziciji i premesta na
22 poziciju min, na kojoj se nalazio najmanji od gore
navedenih elemenata. */
24 for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
26 nalazi najmanji od elemenata a[i],...,a[n-1]. */
    min = i;
    for (j = i + 1; j < n; j++)
        if (a[j] < a[min])
            min = j;
30
    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
32 samo ako su (i) i min razliciti, inace je nepotrebno. */
    if (min != i) {
        pom = a[i];
        a[i] = a[min];
        a[min] = pom;
34
    }
36
    }
38
}
40
}
42

/* Funkcija sortira niz celih brojeva metodom sortiranja
44 umetanjem. Ideja algoritma je sledeca: neka je na pocetku

```

3 Algoritmi pretrage i sortiranja

```
46     i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
47     sortirano. U i-toj iteraciji želimo da element a[i] umetnemo
48     na pravu poziciju medju prvih i elemenata tako da dobijemo
49     niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
50     element najpre uporedimo sa njegovim prvim levim susedom
51     (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i
52     niz a[0],a[1],...,a[i] je sortiran, pa mozemo preci na
53     sledecu iteraciju. Ako je a[i-1] vece, tada zamenjujemo a[i]
54     i a[i-1], a zatim proveravamo da li je potrebno dalje
55     potiskivanje elementa u levo, poredeci ga sa njegovim novim
56     levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
57     pravo mesto u nizu. */
58 void insertionsort(int a[], int n)
59 {
60     int i, j;
61
62     /* Na pocetku iteracije pretpostavljamo da je niz
63        a[0],...,a[i-1] sortiran */
64     for (i = 1; i < n; i++) {
65
66         /* U ovoj petlji redom potiskujemo element a[i] u levo
67            koliko je potrebno, dok ne zauzme pravo mesto, tako da
68            niz a[0],...,a[i] bude sortiran. Indeks j je trenutna
69            pozicija na kojoj se element koji umecemo nalazi. Petlja
70            se zavrшава ili kada element dodje do levog kraja (j==0)
71            ili dok ne naidjemo na element a[j-1] koji je manji od
72            a[j]. */
73         for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
74             int temp = a[j];
75             a[j] = a[j - 1];
76             a[j - 1] = temp;
77         }
78     }
79 }
80
81
82 /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
83    algoritma je sledeca: prolazimo kroz niz redom poredeci
84    susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
85    poretku. Ovim se najveći element poput mehurica istiskuje na
86    "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je
87    potrebno ovaj postupak ponoviti nad nizom a[0],...,a[n-2],
88    tj. nad prvih n-1 elemenata niza bez poslednjeg koji je
89    postavljen na pravu poziciju. Nakon toga se istu postupak
90    ponavlja nad sve kracim i kracim prefiksima niza, cime se
91    jedan po jedan istiskuju elementi na svoje prave pozicije. */
92 void bubblesort(int a[], int n)
93 {
94     int i, j;
95     int ind;
96
97     for (i = n, ind = 1; i > 1 && ind; i--)
```

```

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

108              /* Promenljiva ind registruje da je bilo premestanja.
              Samo u tom slučaju ima smisla ici na sledeću
110              iteraciju, jer ako nije bilo premestanja, znači da su
              svi elementi već u dobrom poretku, pa nema potrebe
112              prelaziti na kraci prefiks niza. Moglo je naravno i
              bez ovoga, algoritam bi radio ispravno, ali bi bio
114              manje efikasan, jer bi često nepotrebno vrsio mnoga
              upoređivanja, kada je već jasno da je sortiranje
116              završeno. */
              ind = 1;
118          }
    }

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

136    void shellsort(int a[], int n)
    {
138        int h = n / 2, i, j;
        while (h > 0) {
140            /* Insertion sort sa korakom h */
            for (i = h; i < n; i++) {
142                int temp = a[i];
                j = i;
144                while (j >= h && a[j - h] > temp) {
                    a[j] = a[j - h];
146                    j -= h;
                }
                a[j] = temp;
148            }
        }
    }

```

3 Algoritmi pretrage i sortiranja

```
150     h = h / 2;
151 }
152 }

154 #define MAX 1000000

156 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem.
157    Sortiranje se vrši od elementa na poziciji l do onog na
158    poziciji d. Na početku, da bismo dobili niz kompletno
159    sortirani, l mora biti 0, a d je jednako posljednjem validnom
160    indeksu u nizu. Funkcija niz podeli na dve polovine, levu i
161    desnu, koje zatim rekurzivno sortira. Od ova dva sortirana
162    podniza, dobijamo sortirani niz ucesljavanjem, tj.
163    istovremenim prolaskom kroz oba niza i izborom trenutnog
164    manjeg elementa koji se smesta u pomocni niz. Na kraju
165    algoritma, sortirani elementi su u pomocnom nizu, koji se
166    kopira u originalni niz. */
167 void mergesort(int a[], int l, int d)
168 {
169     int s;
170     static int b[MAX];          /* Pomocni niz */
171     int i, j, k;

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

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

177     /* Rekurzivni pozivi */
178     mergesort(a, l, s);
179     mergesort(a, s + 1, d);

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

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

193     /* U slucaju da se prethodna petlja završila izlaskom
194        promenljive j iz dopustenog opsega u pomocni niz
195        prepisujemo ostatak leve polovine niza */
```



```
202 while (i <= s)
    b[k++] = a[i++];
204
/* U slucaju da se prethodna petlja zavrсила izlaskom
206 promenljive i iz dopustenog opsega u pomocni niz
    prepisujemo ostatak desne polovine niza */
208 while (j <= d)
    b[k++] = a[j++];
210
/* Prepisujemo "ucesljani" niz u originalni niz */
212 for (k = 0, i = 1; i <= d; i++, k++)
    a[i] = b[k];
214 }

216 /* Funkcija menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
218 {
    int tmp = a[i];
220 a[i] = a[j];
    a[j] = tmp;
222 }

224
/* Funkcija sortira deo niza brojeva a izmedju pozicija l i r.
226 Njena ideja sortiranja je izbor jednog elementa niza, koga
    nazivamo pivot, koga cemo dovesti na svoje mesto. Posle ovog
228 koraka, svi elementi levo od njega bice manji, a svi desno
    bice veci od njega. Kako smo pivota doveli na svoje mesto, da
230 bismo imali kompletno sortiran niz, treba sortirati elemente
    levo (manje) od njega, i elemente desno (vece). Kako je
232 dimenzija ova dva podniza manja od dimenzije pocetgnom niza
    koji je trebalo sortirati, ovaj deo ce za nas uraditi
234 rekurzija. */
void quicksort(int a[], int l, int r)
236 {
    int i, pivot_position;
238
    /* Izlaz iz rekurzije -- prazan niz */
240 if (l >= r)
        return;
242

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

3 Algoritmi pretrage i sortiranja

```
254     if (a[i] < a[l])
255         swap(a, ++pivot_position, i);
256
257     /* Postavljamo pivot na svoje mesto */
258     swap(a, l, pivot_position);
259
260     /* Rekurzivno sortiramo elemente manje od pivota */
261     quicksort(a, l, pivot_position - 1);
262     /* Rekurzivno sortiramo elemente vece pivota */
263     quicksort(a, pivot_position + 1, r);
264 }
```

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

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

8
int main(int argc, char *argv[])
10 {
    /******
12     tip_sortiranja == 0 => selectionsort
        (podrazumevano)
14     tip_sortiranja == 1 => insertionsort
        -i opcija komandne linije
16     tip_sortiranja == 2 => bubblesort
        -b opcija komandne linije
18     tip_sortiranja == 3 => shellsort
        -s opcija komandne linije
20     tip_sortiranja == 4 => mergesort
        -m opcija komandne linije
22     tip_sortiranja == 5 => quicksort
        -q opcija komandne linije
24     *****/
    int tip_sortiranja = 0;
26     /******
        tip_niza == 0 => slucajno generisani nizovi
        (podrazumevano)
28     tip_niza == 1 => rastuce sortirani nizovi
        -r opcija komandne linije
30     tip_niza == 2 => opadajuce soritrani nizovi
        -o opcija komandne linije
32     *****/
34     int tip_niza = 0;

36     /* Dimenzija niza koji se sortira */
    int dimenzija;
38     int i;
    int niz[MAX];
40 }
```

```
42  /* Provera argumenata komandne linije */
43  if (argc < 2) {
44      fprintf(stderr,
45          "Program zahteva bar 2 argumenta komandne linije!\n");
46      exit(EXIT_FAILURE);
47  }
48
49  /* Ocitavamo opcije i argumente prilikom poziva programa */
50  for (i = 1; i < argc; i++) {
51      /* Ako je u pitanju opcija... */
52      if (argv[i][0] == '-') {
53          switch (argv[i][1]) {
54              case 'i':
55                  tip_sortiranja = 1;
56                  break;
57              case 'b':
58                  tip_sortiranja = 2;
59                  break;
60              case 's':
61                  tip_sortiranja = 3;
62                  break;
63              case 'm':
64                  tip_sortiranja = 4;
65                  break;
66              case 'q':
67                  tip_sortiranja = 5;
68                  break;
69              case 'r':
70                  tip_niza = 1;
71                  break;
72              case 'o':
73                  tip_niza = 2;
74                  break;
75              default:
76                  printf("Pogresna opcija -%c\n", argv[i][1]);
77                  return 1;
78                  break;
79          }
80      }
81      /* Ako je u pitanju argument, onda je to duzina niza koji
82       treba da se sortira */
83      else {
84          dimenzija = atoi(argv[i]);
85          if (dimenzija <= 0 || dimenzija > MAX) {
86              fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
87              exit(EXIT_FAILURE);
88          }
89      }
90  }
91
92  /* Elemente niza odredjujemo slucajno, ali vodeci racuna o
   tipu niza dobijenom iz komandni linije. srandom funkcija
```

3 Algoritmi pretrage i sortiranja

```

    obezbedjuje novi seed za pozivanje random funkcije, i kako
94     nas niz ne bi uvek isto izgledao seed smo postavili na
    tekuce vreme u sekundama od Nove godine 1970. random()%100
96     daje brojeve izmedju 0 i 99 */
srandom(time(NULL));
98     if (tip_niza == 0)
        for (i = 0; i < dimenzija; i++)
100         niz[i] = random();
    else if (tip_niza == 1)
102         for (i = 0; i < dimenzija; i++)
            niz[i] =
104             i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
    else
106         for (i = 0; i < dimenzija; i++)
            niz[i] =
108             i == 0 ? random() % 100 : niz[i - 1] - random() % 100;

110 /* Ispisujemo elemente niza */
112 /******
    Ovaj deo je iskomentarisan jer ne zelimo da se sledeci ispis
    nadje na izlazu. Njegova svrha je samo bila provera da li je
114     niz generisan u skladu sa opcijama komandne linije.

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

120

122 /* Sortiramo niz na odgovarajuci nacin */
    if (tip_sortiranja == 0)
124         selectionsort(niz, dimenzija);
    else if (tip_sortiranja == 1)
126         insertionsort(niz, dimenzija);
    else if (tip_sortiranja == 2)
128         bubblesort(niz, dimenzija);
    else if (tip_sortiranja == 3)
130         shellsort(niz, dimenzija);
    else if (tip_sortiranja == 4)
132         mergesort(niz, 0, dimenzija - 1);
    else
134         quicksort(niz, 0, dimenzija - 1);

136 /* Ispisujemo elemente niza */
138 /******
    Ovaj deo je iskomentarisan jer nismo zeleli da vreme potrebno
    za njegovo izvršavanje bude ukljuceno u vreme izmereno
140     programom time. Takodje, kako je svrha ovog programa da prikaze
    vremena razlicitih algoritama sortiranja, dimenzije nizova ce
142     biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
    od toliko elemenata. Ovaj deo je korisćen u razvoju programa
144     zarad testiranja korektnosti.
    *****/

```

```

146     printf("Sortiran niz je:\n");
        for (i = 0; i < dimenzija; i++)
148         printf("%d\n", niz[i]);
        *****/
150
152     return 0;
}

```

Rešenje 3.19

```

1  #include <stdio.h>
   #include <string.h>
3  #include <math.h>
   #include <stdlib.h>
5
   #define MAX_BR_TACAKA 128
7
   /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
        int x;
11     int y;
    } Tacka;
13
   /* Funkcija racuna rastojanje zadate tacke od koordinatnog
15     pocetka (0,0) */
   float rastojanje(Tacka A)
17 {
        return sqrt(A.x * A.x + A.y * A.y);
19 }
21
   /* Funkcija koja sortira niz tacaka po rastojanju od
        koordinatnog pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)
    {
25         int min, i, j;
        Tacka tmp;
27
        for (i = 0; i < n - 1; i++) {
29             min = i;
            for (j = i + 1; j < n; j++) {
31                 if (rastojanje(t[j]) < rastojanje(t[min])) {
                    min = j;
33                 }
            }
35             if (min != i) {
                tmp = t[i];
37                 t[i] = t[min];
                t[min] = tmp;
39             }
        }
    }
}

```

```
41 }  
  
43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */  
44 void sortiraj_po_x(Tacka t[], int n)  
45 {  
46     int min, i, j;  
47     Tacka tmp;  
  
49     for (i = 0; i < n - 1; i++) {  
50         min = i;  
51         for (j = i + 1; j < n; j++) {  
52             if (abs(t[j].x) < abs(t[min].x)) {  
53                 min = j;  
54             }  
55         }  
56         if (min != i) {  
57             tmp = t[i];  
58             t[i] = t[min];  
59             t[min] = tmp;  
60         }  
61     }  
62 }  
  
63 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */  
64 void sortiraj_po_y(Tacka t[], int n)  
65 {  
66     int min, i, j;  
67     Tacka tmp;  
  
69     for (i = 0; i < n - 1; i++) {  
70         min = i;  
71         for (j = i + 1; j < n; j++) {  
72             if (abs(t[j].y) < abs(t[min].y)) {  
73                 min = j;  
74             }  
75         }  
76         if (min != i) {  
77             tmp = t[i];  
78             t[i] = t[min];  
79             t[min] = tmp;  
80         }  
81     }  
82 }  
83 }  
  
85  
  
87 int main(int argc, char *argv[])  
88 {  
89     FILE *ulaz;  
90     FILE *izlaz;  
91     Tacka tacke[MAX_BR_TACAKA];  
92     int i, n;
```

```
93  /* Proveravamo broj argumenata komandne linije: ocekujemo ime
95  izvrsnog programa, opciju, ime ulazne datoteke i ime
    izlazne datoteke tj. ocekujemo 4 argumenta */
97  if (argc != 4) {
99      fprintf(stderr,
    "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
    return 0;
101 }

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

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

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

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

131
133 /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
    "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
    (karakter -) i dalje proveravamo sta je na sledecoj
135 poziciji tj. sta je argv[1][1] */

137 switch (argv[1][1]) {
    case 'x':
139     /* Ako je u pitanju karakter x, pozivamo funkciju za
        sortiranje po vrednosti x koordinate */
        sortiraj_po_x(tacke, n);
        break;
141     case 'y':
143     /* Ako je u pitanju karakter y, pozivamo funkciju za
```

3 Algoritmi pretrage i sortiranja

```
145     sortiranje po vrednosti y koordinate */
146     sortiraj_po_y(tacke, n);
147     break;
148 case 'o':
149     /* Ako je u pitanju karakter o, pozivamo funkciju za
150        sortiranje po udaljenosti od koordinatnog pocetka */
151     sortiraj_po_rastojanju(tacke, n);
152     break;
153 }

155 /* Upisujemo dobijeni niz u izlaznu datoteku */
156 for (i = 0; i < n; i++) {
157     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
158 }

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

163 /* Završavamo sa programom */
164 return 0;
165 }
```

Rešenje 3.20

```
#include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 1000
6 #define MAX_DUZINA 16
7
8 /* Struktura koja reprezentuje jednog gradjanina */
9 typedef struct gr {
10     char ime[MAX_DUZINA];
11     char prezime[MAX_DUZINA];
12 } Gradjanin;
13
14 /* Funkcija sortira niz gradjana rastuce po imenima */
15 void sort_ime(Gradjanin a[], int n)
16 {
17     int i, j;
18     int min;
19     Gradjanin pom;
20
21     for (i = 0; i < n - 1; i++) {
22         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
23            nalazi najmanji od elemenata a[i].ime,...,a[n-1].ime. */
24         min = i;
25         for (j = i + 1; j < n; j++)
26             if (strcmp(a[j].ime, a[min].ime) < 0)
```



```

    min = j;
28  /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
    samo ako su (i) i min razliciti, inace je nepotrebno. */
30  if (min != i) {
    pom = a[i];
32    a[i] = a[min];
    a[min] = pom;
34  }
  }
36 }

38 /* Funkcija sortira niz gradjana rastuce po prezimenima */
void sort_prezime(Gradjanin a[], int n)
40 {
    int i, j;
42    int min;
    Gradjanin pom;
44
    for (i = 0; i < n - 1; i++) {
46        /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
            nalazi najmanji od elemenata
48            a[i].prezime,...,a[n-1].prezime. */
        min = i;
50        for (j = i + 1; j < n; j++)
            if (strcmp(a[j].prezime, a[min].prezime) < 0)
52                min = j;
        /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
54            samo ako su (i) i min razliciti, inace je nepotrebno. */
        if (min != i) {
56            pom = a[i];
            a[i] = a[min];
58            a[min] = pom;
        }
60    }
}

62
/* Pretraga niza Gradjana */
64 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
{
66    int i;
    for (i = 0; i < n; i++)
68        if (strcmp(a[i].ime, x->ime) == 0
            && strcmp(a[i].prezime, x->prezime) == 0)
70            return i;
    return -1;
72 }

74
int main()
76 {
    Gradjanin spisak1[MAX], spisak2[MAX];
78    int isti_rbr = 0;
```

3 Algoritmi pretrage i sortiranja

```
int i, n;
80 FILE *fp = NULL;

82 /* Otvaranje datoteke */
if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
84     fprintf(stderr,
            "Neuspesno otvaranje datoteke biracki-spisak.txt.\n");
86     exit(EXIT_FAILURE);
}

88 /* Citanje sadrzaja */
90 for (i = 0;
        fscanf(fp, "%s %s", spisak1[i].ime,
92             spisak1[i].prezime) != EOF; i++)
    spisak2[i] = spisak1[i];
94 n = i;

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

98 sort_ime(spisak1, n);

100 /******
102 Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih nizova. Koriscen je samo u fazi testiranja programa.
104
    printf("Biracki spisak [uredjen prema imenima]:\n");
106     for(i=0; i<n; i++)
        printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
108 *****/

110 sort_prezime(spisak2, n);

112 /******
114 Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih nizova. Koriscen je samo u fazi testiranja programa.
116
    printf("Biracki spisak [uredjen prema prezimenima]:\n");
    for(i=0; i<n; i++)
118         printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
    *****/

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

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

```

132     isti_rbr++;
133     *****/
134     /* Ispis rezultata */
135     printf("%d\n", isti_rbr);
136
137     exit(EXIT_SUCCESS);
138 }

```

Rešenje 3.22

```

#include <stdio.h>
2 #include <string.h>
#include <ctype.h>
4
#define MAX_BR_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
34 /* Funkcija koja sortira reci po zadatom kriterijumu.
35    Informacija o duzini reci se mora proslediti zbog pravilnog
36    upravljanja memorijom */
37 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)
38 {
39     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
40         duzina_j, duzina_min;

```

```
char tmp[MAX_DUZINA_RECII];
42 for (i = 0; i < n - 1; i++) {
    min = i;
44     for (j = i; j < n; j++) {
        /* Prvo uporedjujemo broj suglasnika */
46         broj_suglasnika_j = broj_suglasnika(reci[j]);
        broj_suglasnika_min = broj_suglasnika(reci[min]);
48         if (broj_suglasnika_j < broj_suglasnika_min)
            min = j;
50         else if (broj_suglasnika_j == broj_suglasnika_min) {
            /* Zatim, reci imaju isti broj suglasnika uporedjujemo
52             duzine */
            duzina_j = strlen(reci[j]);
54             duzina_min = strlen(reci[min]);

            if (duzina_j < duzina_min)
                min = j;
58             else
                /* A ako reci imaju i isti broj suglasnika i iste
60                 duzine, uporedjujemo ih leksikografski */
                if (duzina_j == duzina_min
62                     && strcmp(reci[j], reci[min]) < 0)
                    min = j;
64         }
    }
66     if (min != i) {
        strcpy(tmp, reci[min]);
68         strcpy(reci[min], reci[i]);
        strcpy(reci[i], tmp);
70     }
72 }

74 int main()
{
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 pojedinacne 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 */
```

```

94     while (fscanf(ulaz, "%s", reci[i]) != EOF) {
        /* Proveravamo da li smo ucitali najvise dozvoljenih reci i
           ako jesmo, prekidamo učitavanje */
96         if (i == MAX_BR_RECI)
            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
       prosledjivanja niza reci */
106     sortiraj_reci(reci, n);
108
    /* Ispisujemo sortirani niz reci */
110     for (i = 0; i < n; i++) {
        printf("%s ", reci[i]);
112     }
    printf("\n");
114
    /* Zatvaramo datoteku */
116     fclose(ulaz);
118
    /* Završavamo sa programom */
    return 0;
120 }

```

Rešenje 3.23

```

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

5  #define MAX_ARTIKALA 100000

7  /* Struktura koja predstavlja jedan artikal */
   typedef struct art {
9     long kod;
    char naziv[20];
11    char proizvodjac[20];
    float cena;
13 } Artikal;

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

```

3 Algoritmi pretrage i sortiranja

```
21  /* Dokle god je indeks levi levo od indeksa desni */
23  while (levi <= desni) {
24      /* Racunamo sredisnji indeks */
25      int srednji = (levi + desni) / 2;
26      /* Ako je sredisnji element veci od x, tada se x mora
27       nalaziti u levoj polovini niza */
28      if (x < a[srednji].kod)
29          desni = srednji - 1;
30      /* Ako je sredisnji element manji od x, tada se x mora
31       nalaziti u desnoj polovini niza */
32      else if (x > a[srednji].kod)
33          levi = srednji + 1;
34      else
35          /* Ako je sredisnji element jednak x, tada smo pronasli x
36           na poziciji srednji */
37          return srednji;
38  }
39  /* Ako nije pronadjen vracamo -1 */
40  return -1;
41 }

43 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44 void selection_sort(Artikal a[], int n)
45 {
46     int i, j;
47     int min;
48     Artikal pom;
49
50     for (i = 0; i < n - 1; i++) {
51         min = i;
52         for (j = i + 1; j < n; j++)
53             if (a[j].kod < a[min].kod)
54                 min = j;
55         if (min != i) {
56             pom = a[i];
57             a[i] = a[min];
58             a[min] = pom;
59         }
60     }
61 }

63 int main()
64 {
65     Artikal asortiman[MAX_ARTIKALA];
66     long kod;
67     int i, n;
68     float racun;
69
70     FILE *fp = NULL;
71
72     /* Otvaranje datoteke */
```

```

73  if ((fp = fopen("artikli.txt", "r")) == NULL) {
74      fprintf(stderr,
75          "Neuspesno otvaranje datoteke artikli.txt.\n");
76      exit(EXIT_FAILURE);
77  }

78  /* Ucitavanje artikala */
79  i = 0;
81  while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
82      asortiman[i].naziv, asortiman[i].proizvodjac,
83      &asortiman[i].cena) == 4)
84      i++;
85
86  /* Zatvaranje datoteke */
87  fclose(fp);
88
89  n = i;

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 sortirani 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    selection_sort(asortiman, n);

106
107  /* Ispis stanja u prodavnici */
108  printf
109  ("Asortiman:\nKOD          Naziv artikla      Ime
110     proizvodjaca      Cena\n");
111  for (i = 0; i < n; i++)
112      printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113          asortiman[i].naziv, asortiman[i].proizvodjac,
114          asortiman[i].cena);
115
116  kod = 0;
117  while (1) {
118      printf("-----\n");
119      printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
120      printf("- Za nov racun unesite kod artikla!\n\n");
121      /* Unos bar koda provog artikla sledeceg kupca */
122      if (scanf("%ld", &kod) == EOF)
123          break;

```

3 Algoritmi pretrage i sortiranja

```
125     /* Trenutno racun novog kupca */
126     racun = 0;
127     /* Za sve artikle trenutnog kupca */
128     while (1) {
129         /* Nalazimo ih u nizu */
130         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
131             printf
132             ("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
133         } else {
134             printf("\tTrazili ste:\t%s %s %12.2f\n",
135                 asortiman[i].naziv, asortiman[i].proizvodjac,
136                 asortiman[i].cena);
137             /* I dodajemo na ukupan racun */
138             racun += asortiman[i].cena;
139         }
140         /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0
141            ako on nema vise artikla */
142         printf("Unesite kod artikla [ili 0 za prekid]: \t");
143         scanf("%ld", &kod);
144         if (kod == 0)
145             break;
146     }
147     /* Stampanje ukupnog racuna trenutnog kupca */
148     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
149 }
150
151 printf("Kraj rada kase!\n");
152
153 exit(EXIT_SUCCESS);
154 }
```

Rešenje 3.24

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 500
6
7 /* Struktura koja nam je neophodna za sve informacije o
8    pojedinačnom studentu */
9 typedef struct {
10     char ime[20];
11     char prezime[25];
12     int prisustvo;
13     int zadaci;
14 } Student;
15
16 /* Funkcija kojom sortiramo niz struktura po prezimenu
17    leksikografski rastuće */
18 void sort_ime_leksikografski(Student niz[], int n)
```



```

{
20     int i, j;
    int min;
22     Student pom;

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

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

38 /* Funkcija kojom sortiramo niz struktura po ukupnom broju
    uradjenih zadataka opadajuće, ukoliko neki studenti imaju
40     isti broj uradjenih zadataka sortiraju se po dužini imena
    rastuće. */
42 void sort_zadatke_pa_imena(Student niz[], int n)
{
44     int i, j;
    int max;
46     Student pom;
    for (i = 0; i < n - 1; i++) {
48         max = i;
        for (j = i + 1; j < n; j++)
50             if (niz[j].zadaci > niz[max].zadaci)
                max = j;
52             else if (niz[j].zadaci == niz[max].zadaci
                        && strlen(niz[j].ime) < strlen(niz[max].ime))
54                 max = j;
        if (max != i) {
56             pom = niz[max];
            niz[max] = niz[i];
58             niz[i] = pom;
        }
60     }
}

62 /* Funkcija kojom sortiramo niz struktura po broju casova na
    kojima su bili opadajuće, a ukoliko * neki studenti imaju
64     isti broj casova, sortiraju se opadajuće po broju uradjenih
    zadataka, * a ukoliko se i po broju zadataka poklapaju
66     sortirati ih po prezimenu opadajuće. */
68 void sort_prisustvo_pa_zadatke_pa_prezimana(Student niz[], int n)
{
70     int i, j;

```

```
int max;
Student pom;
72 for (i = 0; i < n - 1; i++) {
74     max = i;
    for (j = i + 1; j < n; j++)
76         if (niz[j].prisustvo > niz[max].prisustvo)
            max = j;
78     else if (niz[j].prisustvo == niz[max].prisustvo
              && niz[j].zadaci > niz[max].zadaci)
80         max = j;
    else if (niz[j].prisustvo == niz[max].prisustvo
              && niz[j].zadaci == niz[max].zadaci
              && strcmp(niz[j].prezime, niz[max].prezime) > 0)
84         max = j;
    if (max != i) {
86         pom = niz[max];
        niz[max] = niz[i];
88         niz[i] = pom;
    }
90 }
92 }

94 int main(int argc, char *argv[])
96 {
    Student praktikum[MAX];
98     int i, br_studenata = 0;

100     FILE *fp = NULL;

102     /* Otvaranje datoteke za citanje */
    if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
104         fprintf(stderr,
            "Neupesno otvaranje datoteke aktivnost.txt.\n");
106         exit(EXIT_FAILURE);
    }

108     /* Ucitavanje sadrzaja */
    for (i = 0;
110         fscanf(fp, "%s%s%d%d", praktikum[i].ime,
            praktikum[i].prezime, &praktikum[i].prisustvo,
            &praktikum[i].zadaci) != EOF; i++);
114     /* Zatvaranje datoteke */
    fclose(fp);
116     br_studenata = i;

118     /* Kreiramo prvi spisak studenata na kom su sortirani
        leksikografski po imenu rastuce */
    sort_ime_leksikografski(praktikum, br_studenata);
120     /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat1.txt", "w")) == NULL) {
122
```

```

124     fprintf(stderr, "Neuspješno otvaranje datoteke dat1.txt.\n");
        exit(EXIT_FAILURE);
    }
126     /* Upis niza u datoteku */
    fprintf
128     (fp,
        "Studenti sortirani po imenu leksikografski rastuce:\n");
130     for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
132                praktikum[i].prezime, praktikum[i].prisustvo,
                praktikum[i].zadaci);
134     /* Zatvaranje datoteke */
    fclose(fp);

136
138     /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
        opadajuće, ukoliko neki studenti imaju isti broj uradjenih
        zadataka sortiraju se po dužini imena rastuce. */
140     sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
142     if ((fp = fopen("dat2.txt", "w")) == NULL) {
        fprintf(stderr, "Neuspješno otvaranje datoteke dat2.txt.\n");
144         exit(EXIT_FAILURE);
    }
146     /* Upis niza u datoteku */
    fprintf(fp,
148            "Studenti sortirani po broju zadataka opadajuće,\n");
    fprintf(fp, "pa po dužini imena rastuce:\n");
150     for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
152                praktikum[i].prezime, praktikum[i].prisustvo,
                praktikum[i].zadaci);
154     /* Zatvaranje datoteke */
    fclose(fp);

156
158     /* Na trecem spisku su sortirani po broju casova na kojima su
        bili opadajuće, a ukoliko neki studenti imaju isti broj
        casova, sortiraju se opadajuće po broju uradjenih zadataka,
160         a ukoliko se i po broju zadataka poklapaju sortirati ih po
        prezimenu opadajuće. */
162     sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum,
                                                br_studenata);
164     /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat3.txt", "w")) == NULL) {
166         fprintf(stderr, "Neuspješno otvaranje datoteke dat3.txt.\n");
        exit(EXIT_FAILURE);
168     }
    /* Upis niza u datoteku */
170     fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
    fprintf(fp, "pa po broju zadataka,\n");
172     fprintf(fp, "pa po prezimenima leksikografski opadajuće,\n");
    for (i = 0; i < br_studenata; i++)
174         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,

```

3 Algoritmi pretrage i sortiranja

```
176         praktikum[i].prezime, praktikum[i].prisustvo,
           praktikum[i].zadaci);
178     /* Zatvaranje datoteke */
    fclose(fp);
180     return 0;
}
```

Rešenje 3.25

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

6 /* Struktura koja opisuje jednu pesmu */
typedef struct {
8     char *izvodjac;
    char *naslov;
10    int broj_gledanja;
} Pesma;
12

/* Funkcija za uporedjivanje pesama po broju gledanosti
14    (potrebna za rad qsort funkcije) */
int uporedi_gledanost(const void *pp1, const void *pp2)
16 {
    Pesma *p1 = (Pesma *) pp1;
18    Pesma *p2 = (Pesma *) pp2;

20    return p2->broj_gledanja - p1->broj_gledanja;
}
22

/* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
24    qsort funkcije) */
int uporedi_naslove(const void *pp1, const void *pp2)
26 {
    Pesma *p1 = (Pesma *) pp1;
28    Pesma *p2 = (Pesma *) pp2;

30    return strcmp(p1->naslov, p2->naslov);
}
32

/* Funkcija za uporedjivanje pesama po izvodjaku (potrebna za
34    rad qsort funkcije) */
int uporedi_izvodjace(const void *pp1, const void *pp2)
36 {
    Pesma *p1 = (Pesma *) pp1;
38    Pesma *p2 = (Pesma *) pp2;

40    return strcmp(p1->izvodjac, p2->izvodjac);
}
```

```
42
43
44 int main(int argc, char *argv[])
45 {
46     FILE *ulaz;
47     Pjesma *pjesme;                                /* Pokazivac na deo memorije za
48                                                    cuvanje pjesama */
49     int alocirano_za_pjesme;                        /* Broj mesta alociranih za
50                                                    pjesme */
51     int i;                                          /* Redni broj pjesme cije se
52                                                    informacije citaju */
53     int n;                                          /* Ukupan broj pjesama */
54     int j, k;
55     char c;
56     int alocirano;                                /* Broj mesta alociranih za
57                                                    propratne informacije o
58                                                    pjesmama */
59
60     int broj_gledanja;
61
62     /* Pripremamo datoteku za citanje */
63     ulaz = fopen("pjesme_bez_pretpostavki.txt", "r");
64     if (ulaz == NULL) {
65         printf("Greska pri otvaranju ulazne datoteke!\n");
66         return 0;
67     }
68
69     /* Citamo informacije o pjesmama */
70     pjesme = NULL;
71     alocirano_za_pjesme = 0;
72     i = 0;
73
74     while (1) {
75
76         /* Proveravamo da li smo stigli do kraja datoteke */
77         c = fgetc(ulaz);
78         if (c == EOF) {
79             /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
80                sadrzaja za citanje */
81             break;
82         } else {
83             /* Ako nismo, vracamo procitani karakter nazad */
84             ungetc(c, ulaz);
85         }
86
87         /* Proveravamo da li imamo dovoljno memorije za citanje nove
88            pjesme */
89         if (alocirano_za_pjesme == i) {
90
91             /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
92                alociramo novih KORAK mesta */
93             alocirano_za_pjesme += KORAK;
```

```
94     pesme =
95         (Pesma *) realloc(pesme,
96                             alocirano_za_pesme * sizeof(Pesma));

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

104         /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
105         for (k = 0; k < i; k++) {
106             free(pesme[k].izvodjac);
107             free(pesme[k].naslov);
108         }
109         free(pesme);
110         return 0;
111     }

112 }

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

118 j = 0;                                /* Oznacava poziciju na koju
119                                         treba smestiti procitani
120                                         karakter */

121 alocirano = 0;                          /* Oznacava broj alociranih
122                                         mesta */
123 pesme[i].izvodjac = NULL;               /* Memorija koju mozemo
124                                         koristiti za smestanje
125                                         procitanih karaktera */

126 /* Sve dok ne stignemo do prve beline u liniji - beline koja
127 se nalazi nakon imena izvodjaca - citamo karaktere iz
128 datoteke */
129 while ((c = fgetc(ulaz)) != ' ') {

132     /* Proveravamo da li imamo dovoljno memorije za smestanje
133     procitanog karaktera */
134     if (j == alocirano) {

136         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
137         alociramo novih KORAK mesta */
138         alocirano += KORAK;
139         pesme[i].izvodjac =
140             (char *) realloc(pesme[i].izvodjac,
141                             alocirano * sizeof(char));

142         /* Proveravamo da li je nova alokacija uspesna */
143         if (pesme[i].izvodjac == NULL) {
144             /* Ako nije... */
```

```

146         /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
147         for (k = 0; k < i; k++) {
148             free(pesme[k].izvodjac);
149             free(pesme[k].naslov);
150         }
151         free(pesme);
152         /* I prekidamo sa izvršavanjem programa */
153         return 0;
154     }
155
156     }
157     /* Ako imamo dovoljno memorije, smestamo procitani
158        karakter */
159     pesme[i].izvodjac[j] = c;
160     j++;
161     /* I nastavljamo sa citanjem */
162 }
163
164 /* Upisujemo terminirajucu nulu na kraju reci */
165 pesme[i].izvodjac[j] = '\0';
166
167
168 /* Citamo - */
169 fgetc(ulaz);
170
171 /* Citamo razmak */
172 fgetc(ulaz);
173
174
175 /* Citamo naslov pesme */
176 j = 0;                                     /* Oznacava poziciju na koju
177                                           treba smestiti procitani
178                                           karakter */
179
180 alocirano = 0;                             /* Oznacava broj alociranih
181                                           mesta */
182
183 pesme[i].naslov = NULL;                    /* Memorija koju mozemo
184                                           koristiti za smestanje
185                                           procitanih karaktera */
186
187 /* Sve dok ne stignemo do zareza - zareza koji se nalazi
188    nakon naslova pesme - citamo karaktere iz datoteke */
189
190 while ((c = fgetc(ulaz)) != ',') {
191     /* Proveravamo da li imamo dovoljno memorije za smestanje
192        procitanog karaktera */
193     if (j == alocirano) {
194         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
195            alociramo novih KORAK mesta */
196         alocirano += KORAK;
197         pesme[i].naslov =
198             (char *) realloc(pesme[i].naslov,
199                             alocirano * sizeof(char));

```

```
198      /* Proveravamo da li je nova alokacija uspesna */
200      if (pesme[i].naslov == NULL) {
202          /* Ako nije... */
204          /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
206          for (k = 0; k < i; k++) {
208              free(pesme[k].izvodjac);
210              free(pesme[k].naslov);
212          }
214          free(pesme[i].izvodjac);
216          free(pesme);
218
220          /* I prekidamo izvršavanje programa */
222          return 0;
224      }
226
228      /* Ako imamo dovoljno memorije, smestamo procitani
230      karakter */
232      pesme[i].naslov[j] = c;
234      j++;
236      /* I nastavljamo dalje sa citanjem */
238      }
240      /* Upisujemo terminirajucu nulu na kraju reci */
242      pesme[i].naslov[j] = '\0';
244
246      /* Citamo razmak */
248      fgetc(ulaz);
250
252      /* Citamo broj gledanja */
254
256      broj_gledanja = 0;
258
260      /* Sve dok ne stignemo do znaka za novi red - kraja linije -
262      citamo karaktere iz datoteke */
264      while ((c = fgetc(ulaz)) != '\n') {
266          broj_gledanja = broj_gledanja * 10 + (c - '0');
268      }
270      pesme[i].broj_gledanja = broj_gledanja;
272
274      /* Prelazimo na citanje sledece pesme */
276      i++;
278
280      }
282
284      /* Cuvamo informaciju o broju pesama koje smo procitali */
286      n = i;
288
290      /* Zatvaramo datoteku jer nam nece vise trebati */
292      fclose(ulaz);
```



```

250  /* Analiziramo argumente komandne linije */
    if (argc == 1) {
252
        /* Nema dodatnih opcija - sortiramo po broju gledanja */
254      qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
    } else {
256
        if (argc == 2 && strcmp(argv[1], "-n") == 0) {
258          /* Sortiramo po naslovu */
          qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
        } else {
260          if (argc == 2 && strcmp(argv[1], "-i") == 0) {
262            /* Sortiramo po izvodjacu */
            qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
          } else {
264            printf("Nedozvoljeni argumenti!\n");
            free(pesme);
266            return 0;
          }
268        }
    }
270 }

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

278 /* Oslobadjamo memoriju */
    for (i = 0; i < n; i++) {
280      free(pesme[i].izvodjac);
      free(pesme[i].naslov);
282    }
    free(pesme);
284

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

```

Rešenje 3.28

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
   /* Funkcija poredi dva cela broja */
9  int compare_int(const void *a, const void *b)
   {

```

3 Algoritmi pretrage i sortiranja

```
11  /* Konvertujemo void pokazivace u int pokazivace koje zatim
13     dereferenciramo, dobijamo int-ove koje oduzimamo i razliku
        vracamo. */

15  /* Zbog moguceg prekoracenja opsega celih brojeva necemo ih
        oduzimati return *((int *)a) - *((int *)b); */

17
19  int b1 = *((int *) a);
    int b2 = *((int *) b);

21  if (b1 > b2)
        return 1;
23  else if (b1 < b2)
        /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
25      return -1;
    else
27      return 0;
}

29
int compare_int_desc(const void *a, const void *b)
31 {
    /* Za obrnuti poredak mozemo samo oduzimati a od b */
33    /* return *((int *)b) - *((int *)a); */

35    /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
        funkcija */
37    return -compare_int(a, b);
}

39
/* Test program */
41 int main()
{
43     size_t n;
    int i, x;
45     int a[MAX], *p = NULL;

47     /* Unosimo dimenziju */
    printf("Uneti dimenziju niza: ");
49     scanf("%ld", &n);
    if (n > MAX)
51         n = MAX;

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

57
    /* Sortiramo niz celih brojeva */
59     qsort(a, n, sizeof(int), &compare_int);

61
    /* Prikazujemo sortirani niz */
    printf("Sortirani niz u rastucem poretku:\n");
```

```

63     for (i = 0; i < n; i++)
        printf("%d ", a[i]);
65     putchar('\n');

67     /* Pretrazivanje niza */
    /* Vrednost koju cemo traziti u nizu */
69     printf("Uneti element koji se trazi u nizu: ");
    scanf("%d", &x);

71

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

79

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

87     return 0;
89 }

```

Rešenje 3.29

```

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

6  #define MAX 100

8  /* Funkcija racuna broj delilaca broja x */
int no_of_deviders(int x)
10 {
    int i;
12     int br;

14     /* Ako je argument negativan broj menjamo mu znak */
    if (x < 0)
        x = -x;
16     if (x == 0)
        return 0;
18     if (x == 1)
        return 1;
20     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */

```

3 Algoritmi pretrage i sortiranja

```
22  br = 2;
    /* Sve dok je */
24  for (i = 2; i < sqrt(x); i++)
    if (x % i == 0)
26      /* Ako i deli x onda su delioci: i, x/i */
        br += 2;
28  /* Ako je broj bas kvadrat, onda smo iz petlje izasli kada je
    i bilo bas jednako korenu od x, tada x ima jos jednog
30  delioca */
    if (i * i == x)
32        br++;
34  return br;
}

36
/* Funkcija poredjenja dva cela broja po broju delilaca */
38 int compare_no_deviders(const void *a, const void *b)
{
40     int ak = *(int *) a;
    int bk = *(int *) b;
42     int n_d_a = no_of_deviders(ak);
    int n_d_b = no_of_deviders(bk);
44
    if (n_d_a > n_d_b)
46         return 1;
    else if (n_d_a < n_d_b)
48         return -1;
    else
50         return 0;
}

52
/* 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
72     /* Sortiramo niz celih brojeva prema broju delilaca */
    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>
#include <stdlib.h>
#include <string.h>
#include <search.h>
#define MAX_NISKI 1000
#define MAX_DUZINA 30

8  /******
   Niz nizova karaktera ovog potpisa
10 char niske[3][4];
   se moze graficki predstaviti ovako:
12 -----
   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 | |
14 -----

   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu.
16 Za svaku je rezervisano po 4 karaktera ukljucujuci \0.
   Druga rec sa nalazi na adresi koja je za 4 veca od prve reci,
18 a za 4 manja od adrese na kojoj se nalazi treca rec.
   Adresa i-te reci je niske[i] i ona je tipa char*.

20
   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
22 elemenata koji trebaju biti uporedjeni, (npr. pri porecenju
   prve i poslednje reci, pokazivac a ce pokazivati na slovo 'a',
24 a pokazivac b na slovo 'f') kastujemo ih na char*, i pozivamo
   funkciju strcmp nad njima.
26 *****/
int poredi_leksikografski(const void *a, const void *b)
28 {
    return strcmp((char *) a, (char *) b);
30 }

32 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
int poredi_duzine(const void *a, const void *b)
34 {
    return strlen((char *) a) - strlen((char *) b);
36 }
38
int main()

```

```
40 {
41     int i;
42     size_t n;
43     FILE *fp = NULL;
44     char niske[MAX_NISKI][MAX_DUZINA];
45     char *p = NULL;
46     char x[MAX_DUZINA];
47
48     /* Otvaranje datoteke */
49     if ((fp = fopen("niske.txt", "r")) == NULL) {
50         fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
51         exit(EXIT_FAILURE);
52     }
53
54     /* Citanje sadrzaja datoteke */
55     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
56
57     /* Zatvaranje datoteke */
58     fclose(fp);
59     n = i;
60
61     /* Sortiramo niske leksikografski, tako sto biblioteckoj
62        funkciji qsort prosledjujemo funkciju kojom se zadaje
63        kriterijum poredjenja 2 niske po duzini */
64     qsort(niske, n, MAX_DUZINA * sizeof(char),
65          &poredi_leksikografski);
66
67     printf("Leksikografski sortirane niske:\n");
68     for (i = 0; i < n; i++)
69         printf("%s ", niske[i]);
70     printf("\n");
71
72     /* Unosimo trazeni nisku */
73     printf("Uneti trazenu nisku: ");
74     scanf("%s", x);
75
76     /* Binarna pretraga */
77     /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
78        jer nam je niz sortiran leksikografski. */
79     p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
80                &poredi_leksikografski);
81
82     if (p != NULL)
83         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84                p, (p - (char *) niske) / MAX_DUZINA);
85     else
86         printf("Niska nije pronadjena u nizu\n");
87
88     /* Linearna pretraga */
89     /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90        jer nam je niz sortiran leksikografski. */
91     p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
```

```

92         &poredi_leksikografski);
94     if (p != NULL)
95         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
96             p, (p - (char *) niske) / MAX_DUZINA);
97     else
98         printf("Niska nije pronadjena u nizu\n");
100     /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
101        funkciju poredjenja */
102     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
104     printf("Niske sortirane po duzini:\n");
105     for (i = 0; i < n; i++)
106         printf("%s ", niske[i]);
107     printf("\n");
108     exit(EXIT_SUCCESS);
110 }

```

Rešenje 3.31

```

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

8 /*****
Niz pokazivaca na karaktere ovog potpisa
char *niske[3];
posle alokacije u main-u se moze graficki predstaviti ovako:
12  -----
| X | -----> | a | b | c | \0|
14  -----
| Y | -----> | d | e | \0|
16  =====
| Z | -----> | f | g | h | \0|
18  -----

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

22
Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
24 elemenata koji trebaju biti upoređeni (recimo adresu od X
i adresu od Z), i kako su X i Z tipa char*, onda a i b su
26 tipa char**, pa ih tako moramo i kastovati. Da bi uporedili
leksikografski elemente X i Z, moramo uporediti stringove
28 na koje oni pokazuju, pa zato u sledecoj funkciji pozivamo
strcmp() nad onim na sta pokazuju a i b, kastovani na

```

```
30   odgovarajuci tip.
31   *****/
32   int poredi_leksikografski(const void *a, const void *b)
33   {
34       return strcmp(*(char **) a, *(char **) b);
35   }
36
37   /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
38      leksikografski, vec po duzini */
39   int poredi_duzine(const void *a, const void *b)
40   {
41       return strlen(*(char **) a) - strlen(*(char **) b);
42   }
43
44   /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje
45      na element u nizu sa kojim se poredi, pa njega treba
46      kastovati na char** i dereferencirati, (videti obrazlozenje
47      za prvu funkciju u ovom zadatku, a pokazivac a pokazuje na
48      element koji se trazi. U main funkciji je to x, koji je tipa
49      char*, tako da pokazivac a ovde samo kastujemo i ne
50      dereferenciramo. */
51   int poredi_leksikografski_b(const void *a, const void *b)
52   {
53       return strcmp((char *) a, *(char **) b);
54   }
55
56   int main()
57   {
58       int i;
59       size_t n;
60       FILE *fp = NULL;
61       char *niske[MAX_NISKI];
62       char **p = NULL;
63       char x[MAX_DUZINA];
64
65       /* Otvaranje datoteke */
66       if ((fp = fopen("niske.txt", "r")) == NULL) {
67           fprintf(stderr, "Neuspesno otvaranje datoteke niske.txt.\n");
68           exit(EXIT_FAILURE);
69       }
70
71       /* Citanje sadrzaja datoteke */
72       i = 0;
73       while (fscanf(fp, "%s", x) != EOF) {
74           /* Alociranje dovoljne memorije za i-tu nisku */
75           if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
76               fprintf(stderr, "Greska pri alociranju niske\n");
77               exit(EXIT_FAILURE);
78           }
79           /* Kopiranje procitane niske na svoje mesto */
80           strcpy(niske[i], x);
81           i++;
82       }
```



```

82     }

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

86     n = i;

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

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

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

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

108    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
110            *p, p - niske);
    else
112        printf("Niska nije pronadjena u nizu\n");

114    /* Linearna pretraga */
/* Prosledjujemo pokazivac na funkciju poredi_leksikografski
116    jer nam je niz sortiran leksikografski. */
p = lfind(x, niske, &n, sizeof(char *),
118            &poredi_leksikografski_b);

120    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
122            *p, p - niske);
    else
124        printf("Niska nije pronadjena u nizu\n");

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

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

```

```
134      /* Oslobadjanje zauzete memorije */
136      for (i = 0; i < n; i++)
          free(niske[i]);
138
139      exit(EXIT_SUCCESS);
140  }
```

Rešenje 3.32

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

6 #define MAX 500

8 /* Struktura koja nam je neophodna za sve informacije o
   pojedinacnom studentu */
10 typedef struct {
    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;
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;
43 }
44
45 /* Funkcija za poredjenje koje ce porediti samo po prezimenu
46     Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
47     a drugi parametar ce biti element niza cije se prezime
48     poredi. */
49 int compare_zalinearnaprezimena(const void *a, const void *b)
50 {
51     char *prezime = (char *) a;
52     Student *s = (Student *) b;
53     return strcmp(prezime, s->prezime);
54 }
55
56 int main(int argc, char *argv[])
57 {
58     Student kolokvijum[MAX];
59     int i;
60     size_t br_studenata = 0;
61     Student *nadjen = NULL;
62     FILE *fp = NULL;
63     int bodovi;
64     char prezime[21];
65
66     /* Ako je program pozvan sa nedovoljnim brojem argumenata
67         informisemo korisnika kako se program koristi i prekidamo
68         izvršavanje. */
69     if (argc < 2) {
70         fprintf(stderr,
71             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
72             argv[0]);
73         exit(EXIT_FAILURE);
74     }
75
76     /* Otvaranje datoteke */
77     if ((fp = fopen(argv[1], "r")) == NULL) {
78         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
79         exit(EXIT_FAILURE);
80     }
81
82     /* Ucitavanje sadrzaja */
83     for (i = 0;
84         fscanf(fp, "%s%d", kolokvijum[i].ime,
85             kolokvijum[i].prezime,
86             &kolokvijum[i].bodovi) != EOF; i++);
87
88     /* Zatvaranje datoteke */
89     fclose(fp);
90     br_studenata = i;
91
92     /* Sortiramo niz studenata po broju bodova, pa unutar grupe
```

3 Algoritmi pretrage i sortiranja

```
94     studenata sa istim brojem bodova sortiranje se vrši po
      prezimenu */
96     qsort(kolokvijum, br_studenata, sizeof(Student), &compare);

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

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

108     nadjen =
110         bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
                  &compare_za_bsearch);

112     if (nadjen != NULL)
114         printf
            ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
116             nadjen->ime, nadjen->prezime, nadjen->bodovi);
      else
118         printf("Nema studenta sa unetim brojem bodova\n");

120     /* Pretraga po prezimenu se mora vršiti linearnom pretragom
      jer nam je niz sortiran po bodovima, globalno gledano. */
122     printf("Unesite prezime: ");
      scanf("%s", prezime);

124     nadjen =
126         lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
                &compare_za_linearna_prezime);

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

136     return 0;
}
```

Rešenje 3.33

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

```

5  #define MAX 128

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

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

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

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

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

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

39   printf("Unesite drugu nisku: ");
   scanf("%s", t);

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

47   return 0;
49 }

```

Rešenje 3.34

```

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

```

```
4
#define MAX 10
6
#define MAX_DUZINA 32

8
/* Funkcija poredi dve niske (stringove) */
int uporedi_niske(const void *pa, const void *pb)
10
{
    return strcmp((char *) pa, (char *) pb);
12
}

14
int main()
{
16
    int i, n;
    char S[MAX][MAX_DUZINA];

18

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

22

    /* Unos niza niski */
    printf("Unesite niske:\n");
    for (i = 0; i < n; i++)
24
        scanf("%s", S[i]);

26

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

30

    /******
32
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih niski. Koriscen je samo u fazi testiranja programa.

34

    printf("Sortirane niske su:\n");
    for(i = 0; i < n; i++)
36
        printf("%s ", S[i]);

38
    *****/

40
    /* Ako postoje dve iste niske u nizu, onda ce one nakon
    sortiranja niza biti jedna do druge */
    for (i = 0; i < n - 1; i++)
42
        if (strcmp(S[i], S[i + 1]) == 0) {
44
            printf("ima\n");
            return 0;
46
        }

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

Rešenje 3.35

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4
5  #define MAX 21
6
7  /* Struktura koja predstavlja jednog studenta */
8  typedef struct student {
9      char nalog[8];
10     char ime[MAX];
11     char prezime[MAX];
12     int poeni;
13 } Student;
14
15
16 /* Funkcija poredi studente prema broju poena, rastuce */
17 int uporedi_poeni(const void *a, const void *b)
18 {
19     Student s = *(Student *) a;
20     Student t = *(Student *) b;
21     return s.poeni - t.poeni;
22 }
23
24
25 /* Funkcija poredi studente prvo prema godini, zatim prema smeru
26    i na kraju prema indeksu */
27 int uporedi_nalog(const void *a, const void *b)
28 {
29     Student s = *(Student *) a;
30     Student t = *(Student *) b;
31     /* Za svakog studenta iz naloga izdvajamo godinu upisa, smer i
32        broj indeksa */
33     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
34     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
35     char smer1 = s.nalog[1];
36     char smer2 = t.nalog[1];
37     int indeks1 =
38         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
39         s.nalog[6] - '0';
40     int indeks2 =
41         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
42         t.nalog[6] - '0';
43     if (godina1 != godina2)
44         return godina1 - godina2;
45     else if (smer1 != smer2)
46         return smer1 - smer2;
47     else
48         return indeks1 - indeks2;
49 }
50
51 int uporedi_bsearch(const void *a, const void *b)
52 {
```

```
53  /* Nalog studenta koji se trazi */
    char *nalog = (char *) a;
55  /* Kljuc pretrage */
    Student s = *(Student *) b;

57
    int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
59  int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
    char smer1 = nalog[1];
61  char smer2 = s.nalog[1];
    int indeks1 =
63      (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
        '0';
65  int indeks2 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
67      s.nalog[6] - '0';
    if (godina1 != godina2)
69        return godina1 - godina2;
    else if (smer1 != smer2)
71        return smer1 - smer2;
    else
73        return indeks1 - indeks2;
}

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

83
    /* Ako je broj argumenata komandne linije razlicit i od 2 i od
85     3, korisnik nije ispravno pozvao program i prijavljujemo
        gresku: */
87  if (argc != 2 && argc != 3) {
        fprintf(stderr,
89             "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\n"
        );
        exit(EXIT_FAILURE);
91  }

93  /* Otvaranje datoteke za citanje */
    in = fopen("studenti.txt", "r");
95  if (in == NULL) {
        fprintf(stderr,
97             "Greska prilikom otvarnja datoteke studenti.txt!\n");
        exit(EXIT_FAILURE);
99  }

101  /* Otvaranje datoteke za pisanje */
    out = fopen("izlaz.txt", "w");
103  if (out == NULL) {
```



```
105     fprintf(stderr,
106         "Greska prilikom otvaranja datoteke izlaz.txt!\n");
107     exit(EXIT_FAILURE);
108 }
109
110 /* Ucitavamo studente iz ulazne datoteke sve do njenog kraja */
111 while (fscanf
112     (in, "%s %s %s %d", niz_studenata[i].nalog,
113      niz_studenata[i].ime, niz_studenata[i].prezime,
114      &niz_studenata[i].poeni) != EOF)
115     i++;
116
117 br_studenata = i;
118
119 /* Ako je student uneo opciju -p, vrsi se sortiranje po
120 poenima */
121 if (strcmp(argv[1], "-p") == 0)
122     qsort(niz_studenata, br_studenata, sizeof(Student),
123         &uporedi_poeni);
124
125 /* A ako je uneo opciju -n, vrsi se sortiranje po nalogu */
126 else if (strcmp(argv[1], "-n") == 0)
127     qsort(niz_studenata, br_studenata, sizeof(Student),
128         &uporedi_nalog);
129
130 /* Sortirani studenti se ispisuju u izlaznu datoteku */
131 for (i = 0; i < br_studenata; i++)
132     fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
133         niz_studenata[i].ime, niz_studenata[i].prezime,
134         niz_studenata[i].poeni);
135
136 /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
137 studenta... */
138 if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
139     strcpy(nalog_trazeni, argv[2]);
140
141     /* ... pronalazi se student sa tim nalogom... */
142     nadjen =
143         (Student *) bsearch(nalog_trazeni, niz_studenata,
144             br_studenata, sizeof(Student),
145             &uporedi_bsearch);
146
147     if (nadjen == NULL)
148         printf("Nije nadjen!\n");
149     else
150         printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
151             nadjen->prezime, nadjen->poeni);
152 }
153
154 /* Zatvaranje datoteka */
155 fclose(in);
156 fclose(out);
```

```
    return 0;
157 }
```

Rešenje 3.37

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

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

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

18 /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
20 int zbir_vrste(int **a, int v, int m)
{
22     int i, zbir = 0;

24     for (i = 0; i < m; i++) {
        zbir += a[v][i];
26     }
    return zbir;
28 }

30 /* Funkcija koja sortira vrste matrice na osnovu zbira
   koriscenjem selection algoritma */
32 void sortiraj_vrste(int **a, int n, int m)
{
34     int i, j, min;

36     for (i = 0; i < n - 1; i++) {
        min = i;
38         for (j = i + 1; j < n; j++) {
            if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
40                 min = j;
            }
42         }
        if (min != i) {
44             int *tmp;
            tmp = a[i];
            a[i] = a[min];
46             a[min] = tmp;
        }
    }
}
```

```

    a[min] = tmp;
48 }
    }
50 }

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

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

66 /* Funkcija koja alokira memoriju za matricu dimenzija nxm */
68 int **alociraj_memoriju(int n, int m)
{
70     int i, j;
    int **a;

72     a = (int **) malloc(n * sizeof(int *));
74     if (a == NULL) {
        fprintf(stderr, "Problem sa alokacijom memorije!\n");
76         exit(EXIT_FAILURE);
    }

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

84         /* Proveravamo da li je doslo do greske prilikom alokacije */
        if (a[i] == NULL) {
86             /* Ako jeste, ispisujemo poruku */
            fprintf(stderr, "Problem sa alokacijom memorije!\n");
88
            /* I oslobadjamo memoriju zauzetu do ovog koraka */
90             for (j = 0; j < i; j++) {
                free(a[j]);
92             }
            free(a);
94             exit(EXIT_FAILURE);
        }
96     }

98     return a;

```

```
100 }
101
102 /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
103    nxm */
104 void oslobodi_memoriju(int **a, int n, int m)
105 {
106     int i;
107
108     for (i = 0; i < n; i++) {
109         free(a[i]);
110     }
111     free(a);
112 }
113
114
115 int main(int argc, char *argv[])
116 {
117     int **a;
118     int n, m;
119
120     /* Citamo dimenzije matrice */
121     printf("Unesite dimenzije matrice: ");
122     scanf("%d %d", &n, &m);
123
124     /* Alociramo memoriju */
125     a = alociraj_memoriju(n, m);
126
127     /* Ucitavamo elemente matrice */
128     ucitaj_matricu(a, n, m);
129
130     /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */
131     sortiraj_vrste(a, n, m);
132
133     /* Ispisujemo rezultujucu matricu */
134     printf("Sortirana matrica je:\n");
135     ispisi_matricu(a, n, m);
136
137     /* Oslobadjamo memoriju */
138     oslobodi_memoriju(a, n, m);
139
140     /* I prekidamo sa izvršavanjem programa */
141     return 0;
142 }
```

Glava 4

Dinamičke strukture podataka

4.1 Liste

Zadatak 4.1 Napisati biblioteku za rad sa jednostruko povezanom listom, čiji elementi sadrže cele brojeve. Ona treba da sadrži sledeće:

- (a) Definirati strukturu `Cvor` koja predstavlja čvor liste.
- (b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vraća adresu novo kreiranog čvora.
- (c) Napisati funkciju koja dodaje novi elemenat na početak liste.
- (d) Napisati funkciju koja dodaje novi elemenat na kraj liste.
- (e) Napisati funkciju koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (f) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama `[,]` i međusobno razdvojene zapetama.
- (g) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.
- (h) Napisati verziju prethodne funkcije za pretraživanje tako da se pretražuje neopadajuće uređena lista.

- (i) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.
- (j) Napisati verziju prethodne funkcije za brisanje elemenata tako da se brišu elementi neopadajuće uređene liste.
- (k) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Napomena: Sve funkcije za rad sa listom implementirati iterativno.

Potom napisati glavni 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).

- (a) U glavnom programu se učitani celi brojevi dodaju na početak liste. Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. Unosi se ceo broj koji se traži u unetoj listi, i na ekran se ispisuje rezultat pretrage.
- (b) U glavnom programu se učitani celi brojevi dodaju na kraj liste. Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.
- (c) U glavnom programu se učitani celi brojevi dodaju u listu tako da je elementi budu uređeni u neopadajućem poretku. Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. Unosi ceo broj koji se traži u unetoj listi, i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. *Napomena: Prilikom pretraživanja i brisanja elementa liste koristiti činjenicu da je lista uređena.*

Upotreba programa a) 1

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

  Trazeni broj 17 je u listi!
```

Upotreba programa a) 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 23 14 35
  Unosite element koji se trazi u listi: 8
Izlaz:
  Lista: []
  Lista: [23]
  Lista: [14, 23]
  Lista: [35, 14, 23]

  Element nije u listi!
```

Upotreba programa a) 3

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

  Element nije u listi!
```

4 Dinamičke strukture podataka

Upotreba programa b) 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
  Unosite element koji se briše iz liste: 3
Izlaz:
  Lista: []
  Lista: [2]
  Lista: [2, 3]
  Lista: [2, 3, 14]
  Lista: [2, 3, 14, 5]
  Lista: [2, 3, 14, 5, 3]
  Lista: [2, 3, 14, 5, 3, 3]
  Lista: [2, 3, 14, 5, 3, 3, 17]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1, 9]
  Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1, 9, 3]

  Lista nakon brisanja: [2, 14, 5, 17, 1, 9]
```

Upotreba programa b) 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 23 14 35
  Unosite element koji se briše iz liste: 3
Izlaz:
  Lista: []
  Lista: [23]
  Lista: [23, 14]
  Lista: [23, 14, 35]

  Lista nakon brisanja: [23, 14, 35]
```

Upotreba programa b) 3

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D):
  Unosite element koji se briše iz liste: 12
Izlaz:
  Lista: []

  Lista nakon brisanja: []
```


Upotreba programa c) 1

```

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

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

```

Upotreba programa c) 2

```

Poziv: ./a.out
Ulaz:
  Unosite elemente liste (za kraj unesite CTRL+D): 23 14 35
  Unosite element koji se trazi u listi: 8
  Unosite element koji se brise iz liste: 3
Izlaz:
  Lista: []
  Lista: [23]
  Lista: [14, 23]
  Lista: [14, 23, 35]

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

```

Upotreba programa c) 3

```

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

  Element nije u listi!
  Lista nakon brisanja: []

```

[Rešenje 4.1]

Zadatak 4.2 Prethodni zadatak uraditi tako da sve funkcije za rad sa listama budu rekurzivne. *Napomena:* Koristiti iste upotrebe programa iz zadatka

4 Dinamičke strukture podataka

4.1. Uputstvo: U slučaju da je u rekurzivnom pozivu funkcija koje dodaju nove elemente u listu došlo do greške pri alokaciji, funkcija treba da vrati 1 višem rekurzivnom pozivu koji tu informaciju prosleđuje u rekurzivni poziv iznad, i tako sve dok se ta informacija ne vrati u *main*. Ako je poziv funkcije u glavnom programu vratio 0, onda nije bilo greške. Ukoliko je vraćeno 1, dogodila se greška, i tad treba iz *main* funkcije pristupiti pravom početku liste i osloboditi je celu.

[Rešenje 4.2]

Zadatak 4.3 Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva, koja ima iste funkcionalnosti poput navedenih u zadatku 4.1. Dopuniti biblioteku sa

- (a) Funkcijom koja iz prosledene liste briše element na koju pokazuje pokazivač koji se funkciji šalje kao argument.

- (b) Funkcijom koja ispisuje sadržaj liste od poslednjeg elementa ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. *Napomena: Koristiti iste **main** programe i upotrebe programa iz zadatka 4.1. Main programe dopuniti na kraju i pozivom funkcije koja ispisuje listu u nazad.*

[Rešenje 4.3]

Zadatak 4.4 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke `dat.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

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

Test 2

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

Test 3

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

Test 3

```
Datoteka:
{(2 - 14) / (23 + 11)} * (2 + 13)
Izlaz:
Zagrade nisu ispravno uparene.
```

Test 4

```
Datoteka je prazna.
Izlaz:
Zagrade su ispravno uparene.
```

Test 5

```
Datoteka ne postoji.
Izlaz:
Greska prilikom otvaranja datoteke izraz.txt!
```

[Rešenje 4.4]

Zadatak 4.5 Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. Uputstvo: za rešavanje problema koristiti stek implementiran preko listi čiji su čvorovi HTML etikete.

Test 1

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

Test 2

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

Test 3

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

Test 4

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

Test 5

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

Test 6

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

[Rešenje 4.5]

Zadatak 4.6 Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike redom, kako su se prijavljivali. Službenik evidentira korisničke jmbg brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Zakazivanje termina korisnicima 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 obrađuje zahtev tako što ga dopisuje na kraj datoteke **izvestaj.txt**, u novom redu, JMBG i zahtev usluženog korisnika. Posle svakog 5 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. *Uputstvo: Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

Upotreba programa 1

Sluzbenik evidentira korisnicke zahteve unosnjem njihovog JMBG broja i opisa potrebne usluge:

Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG: 4567890123456
Opis problema: Zatvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 5678901234567
Opis problema: Podizanje kredita

Novi zahtev [CTRL+D za kraj]
JMBG:
Sledeci je korisnik sa JMBG brojem: 1234567890123
sa zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG brojem: 2345678901234
sa zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 3456789012345
sa zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG brojem: 4567890123456
sa zahtevom: Zatvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 5678901234567
sa zahtevom: Podizanje kredita
Da li ga vracate na kraj reda? [Da/Ne] Da

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 1234567890123
sa zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 3456789012345
sa zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG brojem: 5678901234567
sa zahtevom: Podizanje kredita
Da li ga vracate na kraj reda? [Da/Ne] Ne

izvestaj.txt:
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 4567890123456 Zahtev: Zatvaranje racuna
JMBG: 1234567890123 Zahtev: Otvaranje racuna
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 5678901234567 Zahtev: Podizanje kredita

[Rešenje 4.6]

Zadatak 4.7 Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etiketete smestati u listu, a za formiranje liste koristiti strukturu:

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

Test 1

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

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

Test 2

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

Test 3

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

Test 4

```
|| Poziv: ./a.out datoteka.html
|| Datoteka.html je prazna.
|| Izlaz:
```

[Rešenje 4.7]

Zadatak 4.8 Milena: malo me muci u ovom zadatku sto nema neki smisao. Naime, ako se samo vrsi učitavanje iz datoteka i ispisivanje, onda su ove liste zapravo visak jer isti rezultat moze da se dobije i bez koriscenja listi. Zato mi fali da program uradi nesto sto ne bi mogao da uradi bez koriscenja listi, npr da na osnovu unetog broja ispisuje svaki n-ti broj rezultujuce liste pa to u nekoj petlji da korisnik moze da ispisuje za razlicite unete n ili tako nesto...

Napisati program koji objedinjuje dve sortirane liste. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. *Napomena: Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1*

Test 1

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

Test 2

```
|| Poziv: ./a.out
|| Izlaz:
||   Greska! Program se poziva sa: ./a.out dat
||   .txt dat2.txt!
```

Test 3

```
|| Poziv: ./a.out dat1.txt
|| Izlaz:
||   Greska! Program se poziva sa: ./a.out dat1
||   .txt dat2.txt!
```

Test 4

```
|| Poziv: ./a.out dat1.txt dat2.txt
|| dat1.txt:
|| 2 4 6 10 15
|| dat2.txt ne postoji
|| Izlaz:
||   Greska prilikom otvaranja datoteke dat2.
||   txt.
```

Test 5

```
|| Poziv: ./a.out dat1.txt dat2.txt
|| dat1.txt ne postoji
|| dat2.txt: 2 4 6 10 15
|| Izlaz:
||   Greska prilikom otvaranja datoteke dat1.
||   txt.
```


Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz:
  2 4 6 10 15
```

[Rešenje 4.8]

Zadatak 4.9 Napisati funkciju koja dodaje na početak liste studenata novog studenta za koga su nam poznati su nam broj indeksa, ime i prezime. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. U glavnom programu učitati u listu sve podatke o studentima iz datoteke čije se ime zadaje kao argument komandne linije. U svakom redu datoteke nalaze se podaci o studentu i to broj indeksa, ime i prezime. Nakon toga se sa standardnog ulaza unose, jedan po jedan, indksi studenata koji se traže u učitanjoj listi. Posle svakog unetog indeksa, program ispisuje poruku da ili ne, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Nakon završenog pretraživanja rekurzivnom funkcijom osloboditi memoriju koju je data lista zauzimala. *Uputstvo: Pretpostaviti da je 10 karaktera dovoljno za zapis indeks i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

Test 1

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

Test 2

```
|| Poziv: ./a.out
|| Izlaz:
||      Greska! Program se poziva sa: ./a.out studenti.txt!
```

Test 5

```
|| Poziv: ./a.out studenti.txt
|| studenti.txt ne postoji
|| Izlaz:
||      Greska prilikom otvaranja datoteke studenti.txt
```

Test 5

```
|| Poziv: ./a.out studenti.txt
|| studenti.txt:          Ulaz:          Izlaz:
|| prazna                 3/2014          ne
||                        235/2008         ne
||                        41/2009         ne
```

[Rešenje 4.9]

Zadatak 4.10 Neka su date dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajućí raspoređene elemente lista L1 i L2 (prvi element iz L1, prvi element iz L2, drugi element L1, drugi element L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove rasporediti u jednu listu. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. Milena: Sta ako je neka lista duza? To precizirati. I ovde me muci sto nedostaje neki smisao zadatku, nesto sto ne bi moglo da se uradi da nismo koristili liste.

Test 1

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

Test 2

```
|| Poziv: ./a.out
|| Izlaz:
||      Greska! Program se poziva sa: ./a.out dat
||      .txt dat2.txt!
```

Test 3

```
|| Poziv: ./a.out dat1.txt
|| Izlaz:
||      Greska! Program se poziva sa: ./a.out dat1
||      .txt dat2.txt!
```

Test 4

```
|| Poziv: ./a.out dat1.txt dat2.txt
|| dat1.txt: 2 4 6 10 15
|| dat2.txt ne postoji
|| Izlaz:
||      Greska prilikom otvaranja datoteke dat2.
||      txt.
```

Test 5

```
|| Poziv: ./a.out dat1.txt dat2.txt
|| dat1.txt ne postoji
|| dat2.txt: 2 4 6 10 15
|| Izlaz:
||      Greska prilikom otvaranja datoteke dat1.
||      txt.
```

Test 6

```
|| Poziv: ./a.out dat1.txt dat2.txt
|| dat1.txt prazna
|| dat2.txt: 2 4 6 10 15
|| Izlaz:
||      2 4 6 10 15
```

Zadatak 4.11 Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

(a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

(b) Napisati funkciju koja u jednom prolazu kroz 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...

Test 1

```
Poziv: ./a.out
brojevi.txt:
  43 12 15 16 4 2 8
Izlaz:
Rezultat.txt :
  12 15 16
```

Test 2

```
Poziv: ./a.out
brojevi.txt ne postoji
Izlaz:
Rezultat.txt:
  Greska prilikom otvaranja datoteke dat2.
  txt.
```

Test 3

```
Poziv: ./a.out
brojevi.txt prazna
Izlaz:
Rezultat.txt je prazna.
```

Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt:
  2 4 6 10 15
Izlaz:
  2 4 6 10 15
```

Zadatak 4.12 Grupa od n plesača na kostimima imaju brojeve od 1 do n , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k -ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n , k ($k < n$) se učitavaju sa standardnog ulaza.

Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. *Uputstvo: Pri implementaciji koristiti dvostruko*

povezanu kružnu listu.

```

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

```

Zadatak 4.13 Grupa od n plesača na kostimima imaju brojeve od 1 do n , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k -ti plesač. Odbrojavanje se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k -ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n , k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. *Uputstvo: Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

```

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

```

```

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

```

4.2 Stabla

Zadatak 4.14 Napisati program za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.

- (b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alokira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.
- (c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.
- (d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.
- (e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- (f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

Test 1

```

Poziv: ./a.out
Ulaz:
Unesite brojeve (CRL+D za kraj unosa): 7 2 1 9 32 18
Izlaz:
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuce stablo: 1 2 9 18 32

```

Test 2

```

Poziv: ./a.out
Ulaz:
Unesite brojeve (CRL+D za kraj unosa): 8 -2 6 13 24 -3
Izlaz:
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuce stablo: -3 -2 6 8 13 24

```

[Rešenje 4.14]

Zadatak 4.15 Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski prema rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku *Nedostaje ime ulazne datoteke!*. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

Test 1

```

Poziv: ./a.out test.txt
Datoteka test.txt:
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUka
Izlaz:
jabuka: 2
programiranje: 2
racunar: 1
sunce: 3
utorak: 1

Najcesca rec: sunce (pojavljuje se 3 puta)

```

Test 2

```
Poziv: ./a.out suma.txt
Datoteka suma.txt:
    lipa zova hrast ZOVA breza LIPA
Izlaz:
    breza: 1
    hrast: 1
    lipa: 2
    zova: 2

    Najcesca rec: lipa (pojavljuje se 2 puta)
```

Test 3

```
Poziv: ./a.out
Izlaz:
    Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

Zadatak 4.16 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. **Pera Peric** 064/123-4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči **KRAJ**, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Upotreba programa 1

```
Poziv: ./a.out
Datoteka imenik.txt:
    Pera Peric 011/3240-987
    Marko Maric 064/1234-987
    Mirko Maric 011/589-333
    Sanja Savkovic 063/321-098
    Zika Zikic 021/759-858
Ulaz:
    Unesite ime datoteke: imenik.txt
    Unesite ime i prezime: Pera Peric
Izlaz:
    Broj je: 011/3240-987
Ulaz:
    Unesite ime i prezime: Marko Markovic
Izlaz:
    Broj nije u imeniku!
Ulaz:
    Unesite ime i prezime: KRAJ
```


[Rešenje 4.16]

Zadatak 4.17 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niz najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

Test 1

```

Poziv: ./a.out
Datoteka prijemni.txt:
  Marko Markovic 45.4 12.3 11
  Milan Jevremovic 35.2 1.3 9
  Maja Agic 60 19 20
  Nadica Zec 54.2 10 15.8
  Jovana Milic 23.3 2 5.6
Izlaz:
  1. Maja Agic 60.0 19.0 20.0 99.0
  2. Nadica Zec 54.2 10.0 15.8 80.0
  3. Marko Markovic 45.4 12.3 11.0 68.7
  4. Milan Jevremovic 35.2 1.3 9.0 45.5
  -----
  5. Jovana Milic 23.3 2.0 5.6 30.9

```

[Rešenje 4.17]

* **Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadanog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i

u formatu DD.MM.YYYY.

Upotreba programa 1

```
Poziv: a.out
Datoteka rodjendani.txt:
  Marko Markovic 12.12.1990.
  Milan Jevremovic 04.06.1989.
  Maja Agic 23.04.2000.
  Nadica Zec 01.01.1993.
  Jovana Milic 05.05.1990.
Ulaz:
  Unesite datum: 23.04.
Izlaz:
  Slavljenik: Maja Agic
Ulaz:
  Unesite datum: 01.01.
Izlaz:
  Slavljenik: Nadica Zec
Ulaz:
  Unesite datum: 01.05.
Izlaz:
  Slavljenik: Jovana Milic 05.05.
Ulaz:
  Unesite datum: CTRL+D
```

[Rešenje 4.18]

Zadatak 4.19 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. *Napomena: Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka `stabla.h` predstavljati popis funkcija biblioteke, a datoteka `stabla.c` njihove implementacije. Programe koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 10 5 15 3 2 4 30 12 14 13 0
  Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0
Izlaz:
  Stabla jesu identicna.
```

Test 2

```

Poziv: ./a.out
Ulaz:
  Prvo stablo: 10 5 15 4 3 2 30 12 14 13 0
  Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0
Izlaz:
  Stabla nisu identicna.

```

[Rešenje 4.19]

*** Zadatak 4.20** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

Test 1

```

Poziv: ./a.out
Ulaz:
  Prvo stablo: 1 7 8 9 2 2
  Drugo stablo: 3 9 6 11 1
Izlaz:
  Unija: 1 1 2 2 3 6 7 8 9 9 11
  Presek: 1 9
  Razlika: 2 2 7 8

```

[Rešenje 4.20]

Zadatak 4.21 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

Test 1

```

Poziv: ./a.out
Ulaz:
  n: 7
  a: 1 11 8 6 37 25 30
Izlaz:
  1 6 8 11 25 30 37

```

Test 2

```
Poziv: ./a.out
Ulaz:
  n: 55
Izlaz:
  Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

Zadatak 4.22 Dato je binarno pretraživačko stablo celih brojeva.

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

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

Test 2

```

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

```

[Rešenje 4.22]

Zadatak 4.23 Dato je binarno pretraživačko stablo celih brojeva.

- Napisati funkciju koja pronalazi čvor u stablu sa maksimalnim proizvodom vrednosti iz desnog podstabla.
- Napisati funkciju koja pronalazi čvor u stablu sa najmanjom sumom vrednosti iz levog podstabla.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

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

Test 1

```

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

```

Zadatak 4.24 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  0.nivo: 10
  1.nivo: 5 15
  2.nivo: 3 12 30
  3.nivo: 2 4 14
  4.nivo: 13
```

[Rešenje 4.24]

* **Zadatak 4.25** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 14 30 1 0
Izlaz:
  Stabla su slicna kao u ogledalu.
```

Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 20 15 0
Izlaz:
  Stabla nisu slicna kao u ogledalu.
```

Zadatak 4.26 AVL-stablo je binarno stablo pretrage kod koga apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 2 11 16 1 13
Izlaz:
  7
```

Test 2

```
Poziv: ./a.out
Ulaz:
  16 30 40 24 10 18 45 22
Izlaz:
  6
```

[Rešenje 4.26]

Zadatak 4.27 Binarno stablo se naziva HEAP ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva HEAP. Napisati zatim i glavni program koji kreira stablo kao na slici ..., poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

Test 1

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

[Rešenje 4.27]

4.3 Rešenja

Rešenje 4.1

```
1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste */
   typedef struct cvor {
6     /* Podatak koji cvor sadrzi */
       int vrednost;
8     /* Pokazivac na sledeci cvor liste */
       struct cvor *sledeci;
10  } Cvor;

12  Cvor* napravi_cvor(int broj);

14  void oslobodi_listu(Cvor** adresa_glave) ;

16  void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) ;

18  void dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj);

20  Cvor * pronadji_poslednji (Cvor * glava);

22  void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

24  Cvor * pronadji_mesto_umetanja(Cvor * glava, int broj ) ;

26  void dodaj_iza(Cvor * tekuci, Cvor * novi) ;
```

```
28 void dodaj_sortirano(Cvor ** adresa_glave, int broj) ;
30 Cvor * pretrazi_listu(Cvor * glava, int broj) ;
32 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj);
34 void obrisi_element(Cvor ** adresa_glave, int broj);
36 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
37 broj);
38 void ispisi_listu(Cvor * glava);
40
41 #endif
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost
6     novog cvora inicijalizuje na broj, dok pokazivac na
7     sledeci cvor u novom cvoru postavlja na NULL.
8     Funkcija vraća pokazivac na novokreirani cvor ili NULL
9     ako alokacija nije uspesno izvršena. */
10 Cvor * napravi_cvor(int broj)
11 {
12     Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
13     if( novi == NULL )
14         return NULL;
15
16     /* Inicijalizacija polja u novom cvoru */
17     novi->vrednost = broj;
18     novi->sledeci = NULL;
19     return novi;
20 }
21
22 /* Funkcija oslobadja dinamičku memoriju zauzetu za elemente
23     liste čiji se početni cvor nalazi na adresi adresa_glave. */
24 void oslobodi_listu(Cvor ** adresa_glave)
25 {
26     Cvor *pomocni = NULL;
27
28     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
29     while (*adresa_glave != NULL)
30     {
31         /* Potrebno je prvo zapamtiti adresu sledećeg elementa i
32            onda osloboditi element koji predstavlja glavu liste */
33         pomocni = (*adresa_glave)->sledeci;
34         free(*adresa_glave);
35         /* Sledeći element je nova glava liste */
36         *adresa_glave = pomocni;
37     }
38 }
```



```

37     }
38 }
39
40 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
41 i ukoliko alokacija nije bila uspesna, oslobadja se sva
42 prethodno zauzeta memorija za listu ciji pocetni cvor se
43 nalazi na adresi adresa_glave.*/
44 void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
45 {
46     /* Ukoliko je novi NULL */
47     if ( novi == NULL )
48     {
49         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
50
51         /* Oslobadjamo dinamicki alociranu memoriju i
52            prekidamo program */
53         oslobodi_listu(adresa_glave);
54         exit(EXIT_FAILURE);
55     }
56 }
57
58 /* Funkcija dodaje novi cvor na pocetak liste.
59 Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje
60 ga na pocetak */
61 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
62 {
63     /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
64        alokaciji */
65     Cvor *novi = napravi_cvor(broj);
66     prover_i_alokaciju(adresa_glave, novi);
67
68     /* Uvezujemo novi cvor na pocetak */
69     novi->sledeci = *adresa_glave;
70     /* Nov cvor je sada nova glava liste */
71     *adresa_glave = novi;
72 }
73
74
75
76 /* Funkcija pronalazi i vraca pokazivac na poslednji element
77 liste, ili NULL ukoliko je lista prazna. */
78 Cvor * pronadji_poslednji (Cvor * glava)
79 {
80     /* Prazna lista nema ni poslednji cvor i u tom slucaju
81        vracamo NULL.*/
82     if( glava == NULL)
83         return NULL;
84
85     /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
86        pomeramo pokazivac glava na taj sledeci element. Kada
87        izađemo iz petlje, glava ce pokazivati na element liste
88        koji nema sledeceg, tj, poslednji element liste je. Zato

```

```
89     vracamo vrednost pokazivaca glava.  
    Pokazivac glava je argument funkcije i njegove promene nece  
91     se odraziti na vrednost pokazivaca glava u pozivajucoj  
    funkciji. */  
93     while (glava->sledeci != NULL)  
        glava = glava->sledeci;  
95  
    return glava;  
97 }  
  
99 /* Funkcija dodaje novi cvor na kraj liste. */  
void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)  
101 {  
    Cvor *novi = napravi_cvor(broj);  
103     prover_i_alokaciju(adresa_glave, novi);  
  
105     /* U slucaju prazne liste */  
    if (*adresa_glave == NULL)  
107     {  
        /* Glava nove liste je upravo novi cvor i ujedno i cela  
109         lista. Azuriramo vrednost na koju pokazuje adresa_glave i  
        tako azuriramo i pokazivacku promenljivu u pozivajucoj  
111         funkciji. */  
        *adresa_glave = novi;  
113         return;  
    }  
115  
    /* Ako lista nije prazna, pronalazimo poslednji element */  
117     Cvor * poslednji = pronadji_poslednji(*adresa_glave);  
  
119     /* Dodajemo novi cvor na kraj preusmeravanjem pokazivaca */  
    poslednji->sledeci = novi;  
121 }  
  
123  
125 /* Pomocna funkcija pronalazi cvor u listi iza koga treba  
    umetnuti nov element sa vrednoscu broj.*/  
Cvor * pronadji_mesto_umetanja(Cvor * glava, int broj )  
127 {  
    /*Ako je lista prazna onda nema takvog mesta i vracamo NULL */  
129     if(glava == NULL)  
        return NULL;  
131  
    /* Krecemo se kroz listu sve dok se ne dodje do elementa ciji  
133     je sledeci element veci ili jednak od novog elementa, ili  
    dok se ne dodje do poslednjeg elementa.  
135  
    Zbog lenjog izracunavanja izraza u C-u prvi deo konjukcije  
137     mora biti provera da li smo dosli do poslednjeg elementa  
    liste pre nego sto proverimo vrednost njegovog sledeceg  
139     elementa, jer u slucaju poslednjeg, sledeci ne postoji,  
    pa ni vrednost.*/
```

```
141 while (glava->sledeci != NULL
    && glava->sledeci->vrednost < broj)
143     glava = glava->sledeci;

145 /* Iz petlje smo mogli izaci jer smo dosli do poslednjeg
    elementa ili smo se zaustavili ranije na elementu ciji
147     sledeci ima vrednost vecu od broj */
    return glava;
149 }

151 void dodaj_iza(Cvor * tekuci, Cvor * novi)
153 {
    /* Novi element dodajemo iza tekuceg elementa */
155     novi->sledeci = tekuci->sledeci;
    tekuci->sledeci = novi;
157 }

159 /* Funkcija dodaje novi element u sortiranu listu
    tako da nova lista ostane sortirana.*/
161 void dodaj_sortirano(Cvor ** adresa_glave, int broj)
    {
163     /* U slucaju prazne liste glava nove liste je novi cvor */
    if ( *adresa_glave == NULL ) {
165         Cvor *novi = napravi_cvor(broj);
        /* Proveravamo da li je doslo do greske prilikom
167             alokacije memorije */
        proveri_alokaciju(adresa_glave, novi);
169         *adresa_glave = novi;
        return;
171     }

173     /* Lista nije prazna*/
    /* Ako je broj manji ili jednak vrednosti u glavi liste,
175         onda ga dodajemo na pocetak liste */
    if ( (*adresa_glave)->vrednost >= broj ) {
177         dodaj_na_pocetak_liste(adresa_glave, broj);
        return;
179     }

181     /* U slucaju da je glava liste element manji od novog elementa,
        tada pronalazimo element liste iza koga treba da se umetne
183         nov broj */
    Cvor * pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);

185     Cvor *novi = napravi_cvor(broj);
    proveri_alokaciju(adresa_glave, novi);
187

189     /* Uvezujemo novi cvor iza pomocnog */
    dodaj_iza(pomocni, novi);
191 }
```

```
193 /* Funkcija trazi u listi element cija je vrednost jednaka datom
195 broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
    broj ili NULL u slucaju da takav element ne postoji u listi.*/
Cvor * pretrazi_listu(Cvor * glava, int broj)
197 {
    for ( ; glava != NULL; glava = glava->sledeci)
199         if (glava->vrednost == broj)
                return glava;
201
    /* Nema trazenog broja u listi i vracamo NULL*/
203     return NULL;
    }
205

207 /* Funkcija trazi u listi element cija je vrednost jednaka datom
209 broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
    broj ili NULL u slucaju da takav element ne postoji u listi.
    Funkcija se u pretrazi oslanja na cinjenicu da je lista koja
211 se pretrazuje neopadajuće sortirana. */
Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj)
213 {
    /* U konjukciji koja cini uslov ostanka u petlji,
215     bitan je redosled! */
    for ( ; glava != NULL && glava->vrednost <= broj ;
217         glava = glava->sledeci)
        if (glava->vrednost == broj)
219             return glava;

221     /* Nema trazenog broja u listi i vracamo NULL*/
    return NULL;
223 }
225

227 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
    Funkcija azurira pokazivac na glavu liste, koji može biti
    promenjen u slucaju da se obrise stara glava. */
229 void obrisi_element(Cvor ** adresa_glave, int broj)
    {
231         Cvor *tekuci = NULL;
        Cvor *pomocni = NULL;
233

235         /* Brisemo sa pocetka liste sve cvorove koji su jednaki datom
            broju, i azuriramo pokazivac na glavu */
        while (*adresa_glave != NULL
237             && (*adresa_glave)->vrednost == broj)
            {
239                 /* Sacuvamo adresu repa liste pre oslobadjanja glave */
                pomocni = (*adresa_glave)->sledeci;
                free(*adresa_glave);
                *adresa_glave = pomocni;
241            }
243    }
```

```

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

249  /* Od ovog trenutka se u svakom koraku nalazimo
    na tekucem cvoru koji je razlicit od trazenog
251  broja (kao i svi levo od njega). Poredimo
    vrednost sledeceg cvora (ako postoji) sa trazanim
253  brojem i brisemo ga ako je jednak, a prelazimo na
    sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
255  dok ne dodjemo do poslednjeg cvora. */
    tekuci = *adresa_glave;
257  while (tekuci->sledeci != NULL)
    if (tekuci->sledeci->vrednost == broj)
259      {
        /* tekuci->sledeci treba obrisati,
        zbog toga sacuvamo njegovu adresu u pomocni */
        pomocni = tekuci->sledeci;
261      /* Tekucem preusmerimo pokazivac sledeci
        tako sto preskakemo njegovog trenutnog sledeceg.
        Njegov novi sledeci ce biti sledeci od ovog koga
263      brisemo. */
        tekuci->sledeci = pomocni->sledeci;
        /* Sada mozemo da oslobodimo cvor sa vrednoscu broj. */
265      free(pomocni);
      }
267  else
      {
271      /* Ne treba brisati sledeceg. Prelazimo na sledeci. */
        tekuci = tekuci->sledeci;
273      }
275  return;
277 }

279
281 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
    neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
283  moze biti promenjen ukoliko se obrisu stara glava liste. */
    void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
285  broj)
    {
287      Cvor *tekuci = NULL ;
        Cvor *pomocni = NULL ;
289
        /* Brisemo sa pocetka liste sve eventualne cvorove koji su
        jednaki datom broju, i azuriramo pokazivac na glavu */
291      while (*adresa_glave != NULL
        && (*adresa_glave)->vrednost == broj)
293      {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
295      pomocni = (*adresa_glave)->sledeci;

```

```
297     free(*adresa_glave);
298     *adresa_glave = pomocni;
299 }

301 /* Ako je nakon toga lista ostala prazna ili glava liste sadrzi
302    vrednost koja je veca od broja, kako je lista sortirana
303    rastuce nema potrebe broj traziti u repu liste i zato
304    prekidamo funkciju */
305 if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
306     return;
307
308 /* Od ovog trenutka se u svakom koraku nalazimo u tekucem cvoru
309    cija vrednost je manja od trazenog broja kao i svi levo od
310    njega. Poredimo vrednost sledeceg cvora, ako postoji, sa
311    trazenim brojem i brisemo ga ako je jednak, a prelazimo na
312    sledeci cvor ako je razlicit. Ovaj postupak ponavljamo dok
313    ne dodjemo do poslednjeg cvora ili prvog cvora cija vrednost
314    je veca od trazenog broja. */
315 tekuci = *adresa_glave;
316 while (tekuci->sledeci != NULL
317        && tekuci->sledeci->vrednost <= broj)
318     if (tekuci->sledeci->vrednost == broj)
319     {
320         /* tekuci->sledeci treba obrisati, zbog toga cuvamo
321            njegovu adresu u pomocni */
322         pomocni = tekuci->sledeci;
323         /* Tekucem preusmerimo pokazivac sledeci tako sto
324            preskakemo njegovog trenutnog sledeceg. Njegov novi
325            sledeci ce biti sledeci od ovog koga brisemo. */
326         tekuci->sledeci = tekuci->sledeci->sledeci;
327         /* Sada mozemo da oslobodimo cvor sa vrednoscu broj */
328         free(pomocni);
329     }
330     else
331     {
332         /* Ne treba brisati sledeceg, jer je manji od trazenog i
333            prelazimo na sledeci */
334         tekuci = tekuci->sledeci;
335     }
336     return;
337 }

338
339 /* Funkcija prikazuje elemente liste pocev od glave ka kraju
340    liste. Ne saljemo joj adresu promenljive koja cuva glavu
341    liste, jer ova funkcija nece menjati listu, pa nema ni
342    potrebe da azuriza pokazivac na glavu liste iz pozivajuce
343    funkcije. */
344 void ispisi_listu(Cvor * glava)
345 {
346     putchar('[');
347     for ( ; glava != NULL; glava = glava->sledeci)
```

```

349     {
351         printf("%d", glava->vrednost);
353         if( glava->sledeci != NULL )
            printf(", ");
355     }
    printf("]\n");
}

```

```

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

4
int main()
6 {
    /* Lista je na pocetku prazna. */
8     Cvor * glava = NULL;
    Cvor * trazeni = NULL;
10     int broj;

12     /* Testiramo dodavanje na pocetak*/
    printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
14     printf("\n\tLista: ");
    ispisi_listu(glava);

16     while(scanf("%d",&broj)>0)
18     {
        dodaj_na_pocetak_liste(&glava, broj);
20         printf("\n\tLista: ");
        ispisi_listu(glava);
22     }

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

26     trazeni = pretrazi_listu(glava, broj);
28     if(trazeni == NULL)
        printf("Element NIJE u listi!\n");
30     else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
32     oslobodi_listu(&glava);

34     return 0;
36 }

```

```

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

4
int main()

```

```
6 {
    Cvor * glava = NULL;
8     int broj;

10     /* Testiramo dodavanje na kraj liste */
    printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
12     printf("\n\tLista: ");
    ispisi_listu(glava);

14     while (scanf("%d",&broj) > 0)
16     {
        dodaj_na_kraj_liste(&glava, broj);
18         printf("\n\tLista: ");
        ispisi_listu(glava);
20     }

22     printf("\nUnesite element koji se brise iz liste: ");
    scanf("%d", &broj);

24     /* Brisemo elemente iz liste cije polje vrednost je jednako
26         broju procitanom sa ulaza */
    obrisi_element(&glava, broj);

28     printf("Lista nakon brisanja: ");
30     ispisi_listu(glava);

32     oslobodi_listu(&glava);

34     return 0;
}
```

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

5
6
7 int main() {
    Cvor * glava = NULL;
    Cvor * trazeni = NULL;
9     int broj;

11     /* Testiramo dodavanje u listu tako da ona bude neopadajuće
        uredjena */
13     printf("Unosite elemente liste (za kraj unesite CTRL+D)\n");
    printf("\n\tLista: ");
15     ispisi_listu(glava);

17     while (scanf("%d",&broj)>0)
    {
19         dodaj_sortirano(&glava, broj);
        printf("\n\tLista: ");
21         ispisi_listu(glava);
    }
```



```

    }
23
    printf("\nUnesite element koji se trazi u listi: ");
25    scanf("%d", &broj);

27    trazeni = pretrazi_sortiranu_listu(glava, broj);
    if(trazeni == NULL)
29        printf("Element NIJE u listi!\n");
    else
31        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

33
    /* Brisemo elemente iz liste cije polje vrednost je jednako
35        broju procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
37    scanf("%d", &broj);

39    obrisi_element_sortirane_liste(&glava, broj);

41    printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

43
    oslobodi_listu(&glava);

45    return 0;
47 }

```

Rešenje 4.2

```

1  #ifndef _LISTA_H
   #define _LISTA_H
3
   /* Biblioteka rekurzivnih funkcije za rad
5       sa jednostruko povezanom listom celih brojeva */

7  /* Struktura kojom je predstavljen cvor liste */
   typedef struct cvor {
9      /* Podatak koji cvor sadrzi */
       int vrednost;
11     /* Pokazivac na sledeci cvor liste */
       struct cvor *sledeci;
13 } Cvor;

15 Cvor* napravi_cvor(int broj);

17 void oslobodi_listu(Cvor** adresa_glave) ;

19 int dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj);

21 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

```

```
23 int dodaj_sortirano(Cvor ** adresa_glave, int broj) ;
25 Cvor * pretrazi_listu(Cvor * glava, int broj) ;
27 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj);
29 void obrisi_element(Cvor ** adresa_glave, int broj);
31 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
    broj);
33 void ispisi_listu(Cvor * glava);
35 void ispisi_elemente(Cvor * glava);
37 #endif

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

/* Pomocna funkcija koja kreira cvor. Funkcija vrednost
   novog cvora inicijalizuje na broj, dok pokazivac na
   sledeci cvor u novom cvoru postavlja na NULL.
   Funkcija vraća pokazivac na novokreirani cvor ili NULL
   ako alokacija nije uspesno izvršena. */
Cvor * napravi_cvor(int broj)
{
    Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
    if( novi == NULL )
        return NULL;

    /* Inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
    novi->sledeci = NULL;
    return novi;
}

/* Funkcija oslobadja dinamičku memoriju zauzetu za elemente
   liste čiji se početni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave)
{
    /* Lista je već prazna */
    if( *adresa_glave == NULL )
        return;

    /* Ako lista nije prazna, treba osloboditi memoriju. Pre
       nego oslobodimo memoriju za glavu liste, moramo osloboditi
       rep liste. */
    oslobodi_listu( &(*adresa_glave)->sledeci);
    /* Nakon oslobodjenog repa, oslobadjamo i glavu liste */
    free(*adresa_glave);
}
```

```

36  /* Azuriramo glavu u pozivajucoj funkciji tako da odgovara
    praznoj listi */
38  *adresa_glave = NULL;
    }
40
42  /* Funkcija dodaje novi cvor na pocetak liste.
    Kreira novi cvor koriscenjem funkcije napravi_cvor() i
44  uvezuje ga na pocetak. Funkcija vraca 1 ukoliko je doslo do
    greske pri alokaciji, inace vraca 0. */
46  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
    {
48      /* Kreiramo nov cvor i proverimo da li je bilo greske pri
        alokaciji */
50      Cvor *novi = napravi_cvor(broj);
        if( novi == NULL)
52          return 1;

54      /* Uvezujemo novi cvor na pocetak */
        novi->sledeci = *adresa_glave;
56      /* Nov cvor je sada nova glava liste */
        *adresa_glave = novi;
58      return 0;
    }
60
62  /* Funkcija dodaje novi cvor na kraj liste.
    Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
    nov broj se dodaje u rep liste u rekurzivnom pozivu. U slucaju
64  da je u rekurzivnom pozivu doslo do greske pri alokaciji,
    funkcija vraca 1 visem rekurzivnom pozivu koji tu informaciju
66  vraca u rekurzivni poziv iznad, sve dok se ne vrati u main.
    Ako je funkcija vratila 0, onda nije bilo greske. Tek je iz
68  main funkcije moguće pristupiti pravom pocetku liste i
    osloboditi je celu, ako ima potrebe. */
70  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
    {
72      if (*adresa_glave == NULL)
        {
74          /* Glava liste je upravo novi cvor i ujedno i cela lista.*/
            Cvor *novi = napravi_cvor(broj);
76            if( novi == NULL)
                return 1;

78            /* Azuriramo vrednost na koju pokazuje adresa_glave i tako
                azuriramo i pokazivacku promenljivu u pozivajucoj
                funkciji. */
80            *adresa_glave = novi;
            return 0;
82        }
84    }

86  /* Ako lista nije prazna, nov element se dodaje u rep liste. */
    return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);

```

```
88 }

90 /* Funkcija dodaje novi element u rastuce sortiranu listu tako da
91    nova lista ostane sortirana. Vraca 0, ako je alokacija novog
92    cvora prosla bez greske, inace vraca 1 da bi ta vrednost bila
93    propagirala nazad do prvog poziva. */
94 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
95 {
96     /* u slucaju prazne liste adresa_glave nove liste je upravo
97        novi cvor */
98     if (*adresa_glave == NULL)
99     {
100         Cvor *novi = napravi_cvor(broj);
101         if( novi == NULL)
102             return 1;
103
104         *adresa_glave = novi;
105         return 0 ;
106     }
107
108     /* Lista nije prazna*/
109     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda
110        u sustini dodajemo na pocetak liste. */
111     if ((*adresa_glave)->vrednost >= broj )
112     {
113         /* Vracamo informaciju o uspesnosti alokacije */
114         return dodaj_na_pocetak_liste(adresa_glave, broj);
115     }
116
117     /* Inace, element treba dodati u rep, tako da rep i sa novim
118        elementom bude sortirana lista */
119     return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
120 }

122
124 /* Funkcija trazi u listi element cija je vrednost jednaka datom
125    broju. Vraca pokazivac na cvor liste u kome je sadržan traženi
126    broj ili NULL u slucaju da takav element ne postoji u listi.*/
127 Cvor * pretrazi_listu(Cvor * glava, int broj)
128 {
129     /* U praznoj listi ga sigurno nema */
130     if(glava == NULL )
131         return NULL;
132
133     /* Ako glava liste sadrži traženi broj */
134     if(glava->vrednost == broj )
135         return glava;
136
137     /* Ako nije nijedna od prethodnih situacija, pretragu
138        nastavljamo u repu */
139     return pretrazi_listu(glava->sledeci, broj);
```

```
140 }
142
144 /* Funkcija trazi u listi element cija je vrednost jednaka datom
146 broju. Funkcija se u pretrazi oslanja na cinjenicu da je lista
148 koja se pretrazuje neopadajuće sortirana. Vraca pokazivac na
    cvor liste u kome je sadržan traženi broj ili NULL u slučaju
    da takav element ne postoji u listi. */
Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj)
150 {
    /* U praznoj listi ga sigurno nema */
152     if(glava == NULL || glava->vrednost > broj)
        return NULL;

154     /* Ako glava liste sadrži traženi broj */
156     if(glava->vrednost == broj )
        return glava;

158     /* Ako nije nijedna od prethodnih situacija, pretragu
160 nastavljamo u repu */
    return pretrazi_listu(glava->sledeci, broj);
162 }

164
166 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
    Funkcija azurira pokazivac na glavu liste, koji može biti
    promenjen u slučaju da se obriše stara glava liste. */
168 void obrisi_element(Cvor ** adresa_glave, int broj)
    {
170     /* Ako je lista prazna nema šta da se brise, vracamo se iz
        funkcije. */
172     if( *adresa_glave == NULL)
        return ;

174     /* Pre nego proverimo situaciju sa glavom liste, obrisacemo sve
176 cvorove iz repa koji imaju vrednost bas broj */
    obrisi_element(&(*adresa_glave)->sledeci, broj);

178     /* Preostaje da proverimo da li glavu treba obrisati. */
180     if ( (*adresa_glave)->vrednost == broj )
        {
182         /* Cvor koji treba da se obriše */
        Cvor* pomocni = *adresa_glave;
184         /* Azuriramo pokazivac na glavu da pokazuje na sledeci u
            listi i brisemo element koji je bio glava liste. */
186         *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
188     }
    }
190 }
```

```
192 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
193     oslanjajuci se na cinjenicu da je prosledjena lista sortirana
194     neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
195     može biti promenjen ukoliko se obrise stara glava liste. */
196 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
    broj)
197 {
198     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
199     veca od broja, kako je lista sortirana rastuce nema potrebe
200     broj traziti u repu liste i zato prekidamo funkciju */
201     if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
202         return;
203
204     /* Pre nego proverimo situaciju sa glavom liste, obrisacemo
205     sve cvorove iz repa koji imaju vrednost bas broj */
206     obrisi_element(&(*adresa_glave)->sledeci, broj);
207
208     /* Preostaje da proverimo da li glavu treba obrisati. */
209     if ( (*adresa_glave)->vrednost == broj )
210     {
211         /* Cvor koji treba da se obrise */
212         Cvor* pomocni = *adresa_glave;
213         /* Azuriramo pokazivac na glavu da pokazuje na sledeci u
214         listi i brisemo element koji je bio glava liste. */
215         *adresa_glave = (*adresa_glave)->sledeci;
216         free(pomocni);
217     }
218 }
219
220 /* Funkcija ispisuje samo elemente liste razdvojene zapetama */
221 void ispisi_elemente(Cvor * glava)
222 {
223     /* Prazna lista */
224     if(glava == NULL)
225         return;
226
227     /* Ispisujemo element u glavi liste */
228     printf(" %d", glava->vrednost);
229     /* Rekurzivni poziv za ispis svega ostalo */
230     ispisi_elemente(glava->sledeci);
231 }
232
233 /* Funkcija prikazuje elemente liste pocev od glave ka kraju
234     liste. Ne saljemo joj adresu promenljive koja cuva glavu
235     liste, jer ova funkcija neće menjati listu, pa nema ni potrebe
236     da azuriza pokazivac iz pozivajuće funkcije. */
237 void ispisi_listu(Cvor * glava)
238 {
239     putchar('[');
240     ispisi_elemente(glava);
241     putchar(']');
```

```

244     putchar('\n');
    }

1  /* Rekurzivne funkcije za rad sa jednostruko povezanom listom */
   #include <stdio.h>
3  #include <stdlib.h>
   #include "lista.h"

5
   int main()
7  {
   /* Lista je prazna na pocetku. */
9  Cvor *glava = NULL;
   Cvor *trazeni = NULL;
11  int broj;

13  /* Testiramo dodavanje na pocetak*/
   printf("Unosite elemente liste! (za kraj unesite CTRL+D)\n");
15  while(scanf("%d",&broj)>0)
   {
17      /* Ako je funkcija vratila 1 onda je bilo greske pri
         alokaciji memorije za nov cvor. Listu moramo osloboditi
         pre napustanja programa. */
19      if ( dodaj_na_pocetak_liste(&glava, broj) == 1)
         {
21          fprintf(stderr,"Neuspela alokacija za cvor %d\n",broj);
23          oslobodi_listu(&glava);
           exit(EXIT_FAILURE);
25      }
27      printf("\n\tLista: ");
           ispisi_listu(glava);
29
           ispisi_listu(glava);

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

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

41      oslobodi_listu(&glava);

43      return 0;
   }

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

```

```
4  int main()
6  {
    Cvor * glava = NULL;
8   int broj;

10  /* Testiramo dodavanje na kraj liste */
    printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
12  printf("\n\tLista: ");
    ispisi_listu(glava);

14  while(scanf("%d",&broj) > 0)
16  {
        /* Ako je funkcija vratila 1 onda je bilo greske pri
18         alokaciji memorije za nov cvor. Listu moramo osloboditi
            pre napustanja programa. */
20         if ( dodaj_na_kraj_liste(&glava, broj) == 1)
            {
22             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
                oslobodi_listu(&glava);
24             exit(EXIT_FAILURE);
            }
26         printf("\n\tLista: ");
            ispisi_listu(glava);
28     }

30     printf("\nUnesite element koji se brise iz liste: ");
        scanf("%d", &broj);
32

34     /* Brisemo elemente iz liste cije polje vrednost je jednako
        broju procitanom sa ulaza */
        obrisi_element(&glava, broj);
36

38     printf("Lista nakon brisanja: ");
        ispisi_listu(glava);

40     oslobodi_listu(&glava);

42     return 0;
    }
```

```
1  #include <stdio.h>
    #include <stdlib.h>
3  #include "lista.h"

5

6  int main() {
7      Cvor * glava = NULL;
        Cvor * trazeni = NULL;
9      int broj;

11     /* Testiramo dodavanje u listu tako da ona bude neopadajuca
```



```

13     uredjena */
14     printf("Unosite elemente liste (za kraj unesite CTRL+D)\n");
15     printf("\n\tLista: ");
16     ispisi_listu(glava);
17
18     while(scanf("%d",&broj)>0)
19     {
20         /* Ako je funkcija vratila 1 onda je bilo greske pri
21            alokaciji memorije za nov cvor. Listu moramo osloboditi
22            pre napustanja programa. */
23         if ( dodaj_sortirano(&glava, broj) == 1)
24         {
25             fprintf(stderr,"Neuspela alokacija za cvor %d\n",broj);
26             oslobodi_listu(&glava);
27             exit(EXIT_FAILURE);
28         }
29         printf("\n\tLista: ");
30         ispisi_listu(glava);
31     }
32
33     printf("\nUnosite element koji se trazi u listi: ");
34     scanf("%d", &broj);
35
36     trazeni = pretrazi_sortiranu_listu(glava, broj);
37     if(trazeni == NULL)
38         printf("Element NIJE u listi!\n");
39     else
40         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
41
42     /* Brisemo elemente iz liste cije polje vrednost je jednako
43        broju procitanom sa ulaza */
44     printf("\nUnosite element koji se brise iz liste: ");
45     scanf("%d", &broj);
46
47     obrisi_element_sortirane_liste(&glava, broj);
48
49     printf("Lista nakon brisanja: ");
50     ispisi_listu(glava);
51
52     oslobodi_listu(&glava);
53
54     return 0;
55 }

```

Rešenje 4.3

```

1  #ifndef _LISTA_H
2  #define _LISTA_H
3
4  /* Struktura kojom je predstavljen cvor liste */

```

```
5 typedef struct cvor{
    int vrednost;
7     struct cvor *sledeci;
    struct cvor *prethodni;
9 } Cvor;

11 Cvor* napravi_cvor(int broj);

13 void oslobodi_listu(Cvor** adresa_glave) ;

15 void prover_i_alokaciju(Cvor** adresa_glave, Cvor* novi) ;

17 void dodaj_na_pocetak_liste(Cvor** adresa_glave, int broj);

19 Cvor * pronadji_poslednji (Cvor * glava);

21 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

23 Cvor * pronadji_mesto_umetanja(Cvor * glava, int broj ) ;

25 void dodaj_iza(Cvor * tekuci, Cvor * novi) ;

27 void dodaj_sortirano(Cvor ** adresa_glave, int broj) ;

29 Cvor * pretrazi_listu(Cvor * glava, int broj) ;

31 Cvor * pretrazi_sortiranu_listu(Cvor * glava, int broj);

33 void obrisi_tekuci(Cvor** adresa_glave, Cvor* tekuci);

35 void obrisi_element(Cvor ** adresa_glave, int broj);

37 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
    broj);

39 void ispisi_listu(Cvor * glava);

41 void ispisi_listu_u_nazad(Cvor* glava) ;

43 #endif
```

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

4
/* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
6 cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
    novom cvoru postavlja na NULL. Funkcija vraca pokazivac na
8 novokreirani cvor ili NULL ako alokacija nije uspesno
    izvrшена. */
10 Cvor *napravi_cvor(int broj)
    {
```

```

12     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
13     if (novi == NULL)
14         return NULL;

16     /* Inicijalizacija polja u novom cvoru. */
17     novi->vrednost = broj;
18     novi->sledeci = NULL;
19     return novi;
20 }

22 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
23    liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
24 void oslobodi_listu(Cvor ** adresa_glave)
25 {
26     Cvor *pomocni = NULL;

28     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
29     while (*adresa_glave != NULL) {
30         /* Potrebno je prvo zapamtiti adresu sledeceg elementa i
31            onda osloboditi element koji predstavlja glavu liste*/
32         pomocni = (*adresa_glave)->sledeci;
33         free(*adresa_glave);
34         /* Sledeci element je nova glava liste */
35         *adresa_glave = pomocni;
36     }
37 }

38 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
39    i ukoliko alokacija nije bila uspesna, oslobadja se sva
40    prethodno zauzeta memorija za listu ciji pocetni cvor se
41    nalazi na adresi adresa_glave. */
42 void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
43 {
44     if (novi == NULL) {
45         fprintf(stderr, "Neuspela alokacija za nov cvor\n");

47         /* Oslobadjamo dinamicki alociranu memoriju i prekidamo
48            program */
49         oslobodi_listu(adresa_glave);
50         exit(EXIT_FAILURE);
51     }
52 }

54

56 /* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
57    koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.*/
58 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
59 {
60     Cvor *novi = napravi_cvor(broj);
61     prover_i_alokaciju(adresa_glave, novi);

62     /* Sledbenik novog cvora je glava stare liste */

```

```
64     novi->sledeci = *adresa_glave;
65     /* Ako stara lista nije bila prazna, onda prethodni od glave
66        treba da bude nov cvor. */
67     if (*adresa_glave != NULL)
68         (*adresa_glave)->prethodni = novi;
69     /* azuriramo pokazivac na glavu u pozivajucoj funkciji jer
70        je novi od sada glava liste */
71     *adresa_glave = novi;
72 }

74
75 /* Funkcija pronalazi i vraca pokazivac na poslednji element
76    liste, ili NULL kao je lista prazna */
77 Cvor *pronadji_poslednji(Cvor * glava)
78 {
79     /*
80      * ako je lista prazna, nema ni poslednjeg cvor i u tom
81      * slucaju vracamo NULL.
82      */
83     if (glava == NULL)
84         return NULL;

85     /*
86      * Sve dok glava ne pokazuje na cvor koji nema sledeceg,
87      * pomeramo pokazivac glava na taj sledeci element. Kada
88      * izađemo iz petlje, glava ce pokazivati na element liste
89      * koji nema sledeceg, tj, poslednji element liste je. Zato
90      * vracamo vrednost pokazivaca glava. glava je argument
91      * funkcije i njegove promene nece se odraziti na vrednost
92      * pokazivaca glava u pozivajucoj funkciji.
93      */
94     while (glava->sledeci != NULL)
95         glava = glava->sledeci;

96     return glava;
97 }

100
101 /*
102  * Funkcija nov cvor dodaje na kraj liste.
103  */
104 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
105 {
106     Cvor *novi = napravi_cvor(broj);
107     /*
108      * Proveravamo da li je doslo do greske prilikom alokacije
109      * memorije
110      */
111     prover_i_alokaciju(adresa_glave, novi);

112     /*
113      * ako je lista u koju dodajemo prazna. Nov cvor je jedini
```

```
116     * cvor u novoj listi i time je i glava nove liste.
117     */
118     if (*adresa_glave == NULL) {
119         *adresa_glave = novi;
120         return;
121     }
122
123     /*
124     * Ako lista nije prazna, pronalazimo poslednji element
125     * liste
126     */
127     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
128
129     /*
130     * tada uvezujemo nov cvor na kraj, tako sto mu azuriramo
131     * pokazivac na prethodni da pokazuje na poslednjeg.
132     * Sledeci od poslednjeg treba da bude nov cvor.
133     */
134     poslednji->sledeci = novi;
135     novi->prethodni = poslednji;
136 }
137
138
139
140
141 /*
142 * Pomocna funkcija pronalazi cvor u listi iza koga treba
143 * umetnuti nov element sa vrednoscu broj.
144 */
145 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
146 {
147     /*
148     * Ako je lista prazna onda nema takvog mesta i vracamo NULL
149     */
150     if (glava == NULL)
151         return NULL;
152
153     /*
154     * Krecemo se kroz listu sve dok se ne dodje do elementa
155     * ciji je sledeci element veci ili jednak od novog
156     * elementa, ili dok se ne dodje do poslednjeg elementa.
157     *
158     * Zbog lenjog izracunavanja izraza u C-u prvi deo
159     * konjukcije mora biti provera da li smo dosli do
160     * poslednjeg elementa liste pre nego sto proverimo vrednost
161     * njegovog sledeceg elementa, jer u slucaju poslednjeg,
162     * sledeci ne postoji, pa ni vrednost.
163     */
164     while (glava->sledeci != NULL
165            && glava->sledeci->vrednost < broj)
166         glava = glava->sledeci;
```

```
168     /*
170     * Iz petlje smo mogli izaci jer smo dosli do poslednjeg
172     * elementa ili smo se zaustavili ranije na elementu ciji
174     * sledeci ima vrednost vecu od broj
176     */
178     return glava;
180 }
182
184 void dodaj_iza(Cvor * tekuci, Cvor * novi)
186 {
188     /*
190     * Novi element dodajemo iza tekuceg elementa
192     */
194     novi->sledeci = tekuci->sledeci;
196     novi->prethodni = tekuci;
198
199     /*
200     * Ako tekuci ima sledeceg, onda upravo dodajemo njemu
202     * prethodnika i potrebno je i njemu da postavimo pokazivace
204     * na ispravne adrese
206     */
208     if (tekuci->sledeci != NULL)
210         tekuci->sledeci->prethodni = novi;
212     tekuci->sledeci = novi;
214 }
216
218 /*
219 * Funkcija dodaje u listu nov cvor na odgovarajuce mesto, tako
220 * sto pronalazi cvor u listi iza kod treba uvezati nov cvor.
222 */
224 void dodaj_sortirano(Cvor ** adresa_glave, int broj)
226 {
228     /*
230     * Ako je lista prazna, glava nove liste je novi cvor
232     */
234     if (*adresa_glave == NULL) {
236         Cvor *novi = napravi_cvor(broj);
238         prover_i_alokaciju(adresa_glave, novi);
240         *adresa_glave = novi;
242         return;
244     }
246
247     /*
248     * Lista nije prazna
250     */
252     /*
253     * Ukoliko je vrednost glave liste veka od nove vrednosti
254     * onda nov cvor treba staviti na pocetak liste.
256     */
258 }
```

```

220     */
221     if ((*adresa_glave)->vrednost >= broj) {
222         dodaj_na_pocetak_liste(adresa_glave, broj);
223         return;
224     }

226     Cvor *novi = napravi_cvor(broj);
227     prover_i_alokaciju(adresa_glave, novi);

228     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
229     /*
230     * Uvezujemo novi cvor iza pomocnog
231     */
232     dodaj_iza(pomocni, novi);
233 }

234

235 /*
236 * Funkcija trazi u listi element cija je vrednost jednaka datom
237 * broju. Vraca pokazivac na cvor liste u kome je sadržan
238 * traženi broj ili NULL u slučaju da takav element ne postoji u
239 * listi.
240 */
241 Cvor *pretrazi_listu(Cvor * glava, int broj)
242 {
243     for (; glava != NULL; glava = glava->sledeci)
244         if (glava->vrednost == broj)
245             return glava;

246     /*
247     * Nema traženog broja u listi i vraćamo NULL
248     */
249     return NULL;
250 }

251

252 /*
253 * Funkcija trazi u listi element cija je vrednost jednaka datom
254 * broju. Funkcija se u pretrazi oslanja na činjenicu da je lista
255 * koja se pretražuje neopadajuće sortirana. Vraca pokazivac na
256 * cvor liste u kome je sadržan traženi broj ili NULL u slučaju da
257 * takav element ne postoji u listi.
258 */
259 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
260 {
261     /*
262     * U konjukciji koja cini uslov ostanka u petlji, bitan je
263     * redosled!
264     */
265     for (; glava != NULL && glava->vrednost <= broj;
266           glava = glava->sledeci)
267         if (glava->vrednost == broj)
268             return glava;
269 }

```

```
272     /*
273     * Nema trazenog broja u listi i vracamo NULL
274     */
275     return NULL;
276 }
277
278 /*
279 * Funkcija brise u listi na koju pokazuje pokazivac glava onaj
280 * cvor na koji pokazuje pokazivac tekuci. Obratiti paznju da je
281 * kod dvostruke liste ovo mnogo lakse uraditi jer cvor tekuci
282 * sadrzi pokazivace na svog sledbenika i prethodnika u listi.
283 * Pre nego sto fizicki obrisemo tekuci obavezno moramo azurirati
284 * sve pokazivace sledbenika i prethodnika.
285 */
286 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
287 {
288     /*
289     * Ako je tekuci NULL pokazivac nema sta da se brise.
290     */
291     if (tekuci == NULL)
292         return;
293
294     /*
295     * Ako postoji prethodnik od tekuceg onda se postavlja da
296     * njegov sledeci bude sledeci od tekuceg.
297     */
298     if (tekuci->prethodni != NULL)
299         tekuci->prethodni->sledeci = tekuci->sledeci;
300
301     /*
302     * Ako postoji sledbenik tekuceg (cvora koji bismo obrisali)
303     * onda njegov prethodnik treba da bude prethodnik tekuceg.
304     */
305     if (tekuci->sledeci != NULL)
306         tekuci->sledeci->prethodni = tekuci->prethodni;
307
308     /*
309     * Ako je glava element koji se brise, glava nove liste ce
310     * biti sledbenik od stare glave.
311     */
312     if (tekuci == *adresa_glave)
313         *adresa_glave = tekuci->sledeci;
314
315     /*
316     * Oslobadjamo dinamicki alociran prostor za cvor tekuci.
317     */
318     free(tekuci);
319 }
320
321
```



```

324 /*
325  * Funkcija brise iz liste sve cvorove koji sadrže dati broj.
326  * Funkcija azurira pokazivac na glavu liste, koji može biti
327  * promenjen u slučaju da se obriše stara glava.
328  */
329 void obrisi_element(Cvor ** adresa_glave, int broj)
330 {
331     Cvor *tekuci = *adresa_glave;
332
333     while ((tekuci =
334             pretrazi_listu(*adresa_glave, broj)) != NULL)
335         obrisi_tekuci(adresa_glave, tekuci);
336 }
337
338 /*
339  * Funkcija brise iz liste sve cvorove koji sadrže dati broj,
340  * oslanjajući se na činjenicu da je prosledjena lista sortirana
341  * neopadajuće. Funkcija azurira pokazivac na glavu liste, koji
342  * može biti promenjen ukoliko se obriše stara glava liste.
343  */
344 void obrisi_element_sortirane_liste(Cvor ** adresa_glave, int
345                                     broj)
346 {
347     Cvor *tekuci = *adresa_glave;
348
349     while ((tekuci =
350             pretrazi_sortiranu_listu(*adresa_glave,
351                                     broj)) != NULL)
352         obrisi_tekuci(adresa_glave, tekuci);
353 }
354
355 /*
356  * Funkcija prikazuje elemente liste počev od glave ka kraju
357  * liste. Ne saljemo joj adresu promenljive koja čuva glavu
358  * liste, jer ova funkcija neće menjati listu, pa nema ni
359  * potrebe da azurira pokazivac na glavu liste iz pozivajuće
360  * funkcije.
361  */
362 void ispisi_listu(Cvor * glava)
363 {
364     putchar('[');
365     for (; glava != NULL; glava = glava->sledeci) {
366         printf("%d", glava->vrednost);
367         if (glava->sledeci != NULL)
368             printf(", ");
369     }
370
371     printf("]\n");
372 }
373
374 }

```

```
376 /*
377  * Funkcija prikazuje elemente liste pocev od kraja ka glavi
378  * liste. Kod dvostruko povezane to je jako jednostavno jer
379  * svaki cvor ima pokazivac na prethodni element u listi.
380  */
381 void ispisi_listu_u_nazad(Cvor * glava)
382 {
383     putchar('[');
384     if (glava == NULL) {
385         printf("]\n");
386         return;
387     }
388
389     glava = pronadji_poslednji(glava);
390
391     for (; glava != NULL; glava = glava->prethodni) {
392         printf("%d", glava->vrednost);
393         if (glava->prethodni != NULL)
394             printf(", ");
395     }
396     printf("]\n");
397 }

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  int main()
6  {
7      /*
8       * Lista je na pocetku prazna.
9       */
10     Cvor *glava = NULL;
11     Cvor *trazeni = NULL;
12     int broj;
13
14     /*
15      * Testiramo dodavanje na pocetak
16      */
17     printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
18     printf("\n\tLista: ");
19     ispisi_listu(glava);
20
21     while (scanf("%d", &broj) > 0) {
22         dodaj_na_pocetak_liste(&glava, broj);
23         printf("\n\tLista: ");
24         ispisi_listu(glava);
25     }
26
27     printf("\nUnesite element koji se trazi u listi: ");
28     scanf("%d", &broj);
29 }
```

```

    trazen_i = pretrazi_listu(glava, broj);
31  if (trazen_i == NULL)
    printf("Element NIJE u listi!\n");
33  else
    printf("Trazeni broj %d je u listi!\n",
35         trazen_i->vrednost);

    printf("\nLista ispisana u nazad: ");
    ispisi_listu_u_nazad(glava);
39
    oslobodi_listu(&glava);
41
    return 0;
43 }

```

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  int main()
   {
7      Cvor *glava = NULL;
       int broj;

9
       /*
11      * Testiramo dodavanje na kraj liste
       */
13     printf("Unesite elemente liste (za kraj unesite CTRL+D)\n");
       printf("\n\tLista: ");
15     ispisi_listu(glava);

17     while (scanf("%d", &broj) > 0) {
         dodaj_na_kraj_liste(&glava, broj);
19         printf("\n\tLista: ");
         ispisi_listu(glava);
21     }

23     printf("\nUnesite element koji se brise iz liste: ");
       scanf("%d", &broj);
25

       /*
27       * Brisemo elemente iz liste cije polje vrednost je jednako
       * broju procitanom sa ulaza
       */
29     obrisi_element(&glava, broj);

31
       printf("Lista nakon brisanja: ");
33     ispisi_listu(glava);

35     printf("\nLista ispisana u nazad: ");
       ispisi_listu_u_nazad(glava);
37

```

```
39     oslobodi_listu(&glava);
41     return 0;
}
```

```
#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"
4
6 int main()
{
8     Cvor *glava = NULL;
    Cvor *trazeni = NULL;
10     int broj;

12     /*
    * Testiramo dodavanje u listu tako da ona bude neopadajuće
14     * uredjena
    */
16     printf
        ("\nUnosite elemente liste (za kraj unesite CTRL+D)\n");
18     printf("\n\tLista: ");
    ispisi_listu(glava);
20
    while (scanf("%d", &broj) > 0) {
22         dodaj_sortirano(&glava, broj);
        printf("\n\tLista: ");
24         ispisi_listu(glava);
    }

26     printf("\nUnesite element koji se traži u listi: ");
28     scanf("%d", &broj);

30     trazeni = pretraži_sortiranu_listu(glava, broj);
    if (trazeni == NULL)
32         printf("Element NIJE u listi!\n");
    else
34         printf("Traženi broj %d je u listi!\n",
            trazeni->vrednost);
36

38     /*
    * Brisemo elemente iz liste čije polje vrednost je jednako
40     * broju procitanom sa ulaza
    */
42     printf("\nUnesite element koji se briše iz liste: ");
    scanf("%d", &broj);
44
    obrisi_element_sortirane_liste(&glava, broj);
46
```

```

48     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

50     printf("\nLista ispisana u nazad: ");
    ispisi_listu_u_nazad(glava);

52     oslobodi_listu(&glava);

54     return 0;
56 }

```

Rešenje 4.4

```

1  #include<stdio.h>
   #include<stdlib.h>
3
   typedef struct cvor {
5     char z;
     struct cvor *sledeci;
7 } Cvor;

9 int main()
   {
11     Cvor *stek = NULL;
     FILE *in = NULL;
13     char pom;
     Cvor *tmp = NULL;

15     in = fopen("dat.txt", "r");
17     if (in == NULL) {
         fprintf(stderr,
19             "Greska prilikom otvaranja datoteke dat.txt!\n");
         exit(EXIT_FAILURE);
21     }

23     while ((pom = fgetc(in)) != EOF) {
         /* Ako je učitana otvorena zagrada stavljamo je na stek */
25         if (pom == '(' || pom == '{' || pom == '[') {
             Cvor *tmp = (Cvor *) malloc(sizeof(Cvor));
27             if (tmp == NULL) {
                 fprintf(stderr, "Greska prilikom alokacije memorije!\n");
29                 return 1;
             }
31             tmp->z = pom;
             tmp->sledeci = stek;
33             stek = tmp;
         }
35         /* Ako je učitana zatvorena zagrada proveravamo da stek nije
            prazan i da li se na vrhu steka nalazi njegova
37         odgovarajuca otvorena zagrada. */
         else {

```

```

39     if (pom == ')') || pom == '}' || pom == ']') {
40         if (stek != NULL && ((stek->z == '(' && pom == ')')
41                               || (stek->z == '{' && pom == '}')
42                               || (stek->z == '[' && pom == ']'))) {
43             /* Sa vrha steka uklanjamo otvorenu zagradu. */
44             Cvor *tmp = stek->sledeci;
45             free(stek);
46             stek = tmp;
47         } else {
48             /* Zagrade u aritmetickom izrazu nisu ispravno
49                uparene. */
50             break;
51         }
52     }
53 }
54 }
55
56 /* Ako je na kraju stek prazan i procitali smo datoteku,
57    zagrade su ispravno uparene, u suprotnom, nisu. */
58 if (stek == NULL && pom == EOF)
59     printf("Zagrade su ispravno uparene.\n");
60 else {
61     printf("Zagrade nisu ispravno uparene.\n");
62
63     while (stek != NULL) {
64         /* Oslobadjamo memoriju koja je ostala na steku, u slucaju
65            neispravnog uparivanja. */
66         Cvor *tmp = stek->sledeci;
67         free(stek);
68         stek = tmp;
69     }
70 }
71
72 fclose(in);
73 return 0;
74 }

```

Rešenje 4.5

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 #define MAX 100
7
8 #define OTVORENA 1
9 #define ZATVORENA 2
10
11 #define VAN_ETIKETE 0
12 #define PROCITANO_MANJE 1

```

```

14 #define U_ETIKETI 2
16 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete
   i pokazivac na sledeci cvor. */
18 typedef struct cvor {
   char etiketa[MAX];
20   struct cvor *sledeci;
} Cvor;
22 /* Funkcija kreira novi cvor, upisuje u njega etiketu i
   vraca njegovu adresu ili NULL ako alokacija nije bila
24   uspesna. */
Cvor *napravi_cvor(char *etiketa)
26 {
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
28   if (novi == NULL)
       return NULL;
30
   /* Inicijalizacija polja u novom cvoru */
32   if (strlen(etiketa) >= MAX) {
       fprintf(stderr,
34         "Etiketa koju pokusavamo staviti na stek preduga,
   bice skracena .\n");
36       etiketa[MAX - 1] = '\0';
   }
   strcpy(novi->etiketa, etiketa);
   novi->sledeci = NULL;
40   return novi;
}
42
/* Funkcija prazni stek */
44 void oslobodi_stek(Cvor ** adresa_vrha)
{
46   Cvor *pomocni;
   while (*adresa_vrha != NULL) {
48       pomocni = *adresa_vrha;
       *adresa_vrha = (*adresa_vrha)->sledeci;
50       free(pomocni);
   }
52 }
54 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
   i ukoliko alokacija nije bila uspesna, oslobadja se sva
56   prethodno zauzeta memorija za listu cija pocetni cvor se
   nalazi na adresi adresa_vrha. */
58 void prover_i_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
{
60   if (novi == NULL) {
       fprintf(stderr, "Neuspela alokacija za nov cvor\n");
62       oslobodi_stek(adresa_vrha);
       exit(EXIT_FAILURE);
64   }

```

```

    }
66
    /* Funkcija postavlja na vrh steka novu etiketu. */
68 void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
    {
69     Cvor *novi = napravi_cvor(etiketa);
    proveri_alokaciju(adresa_vrha, novi);
72     novi->sledeci = *adresa_vrha;
    *adresa_vrha = novi;
74 }

76 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
    pokazivac razlicit od NULL, tada u niz karaktera na koji on
78     pokazuje upisuje ime etikete koja je upravo skinuta sa steka
    dok u suprotnom ne radi nista. Funkcija vraca 0 ako je stek
80     prazan (pa samim tim nije bilo moguće skinuti vrednost sa
    steka) ili 1 u suprotnom. */
82 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
    {
84     Cvor *pomocni;

86     /* Pokušavamo da skinemo vrednost sa vrha praznog steka i
        imamo gresku. */
88     if (*adresa_vrha == NULL)
        return 0;

90     /* Ako adresa na koju želimo da smestamo etiketu nije NULL
        kopiramo tamo etiketu sa vrha steka. */
92     if (etiketa != NULL)
        strcpy(etiketa, (*adresa_vrha)->etiketa);

94     /* Skidamo element sa vrha steka. */
    pomocni = *adresa_vrha;
98     *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);

100     return 1;
102 }

104 /* Funkcija vraca pokazivac na string koji sadrži etiketu na
    vrhu steka. Ukoliko je stek prazan, vraca NULL. */
106 char *vrh_steka(Cvor * vrh)
    {
108     if (vrh == NULL)
        return NULL;
110     return vrh->etiketa;
    }

112 /* Funkcija prikazuje stek počev od vrha prema dnu. */
114 void prikazi_stek(Cvor * vrh)
    {
116     for (; vrh != NULL; vrh = vrh->sledeci)
```



```

    printf("<%s>\n", vrh->etiketa);
118 }

120 /* Funkcija iz fajla na koji pokazuje f cita sledecu etiketu, i
    njeno ime upisuje u niz na koji pokazuje pokazivac etiketa.
122 Funkcija vraca EOF u slucaju da se dodje do kraja fajla pre
    nego sto se procita etiketa, vraca OTVORENA ako je procitana
124 otvorena etiketa, odnosno ZATVORENA ako je procitana
    zatvorena etiketa. */
126 int uzmi_etiketu(FILE * f, char *etiketa)
    {
128     int c;
    int i = 0;
130
132     /* Stanje predstavlja informaciju dokle smo stali sa citanjem
    etikete inicijalno smo ga postavili na vrednost VAN_ETIKETE
134 jer jos uvek nismo poceli da citamo. Tip predstavlja
    informaciju o tipu etikete uzima vrednosti OTVORENA ili
136 ZATVORENA. */
    int stanje = VAN_ETIKETE;
138 int tip;

140     /* HTML je neosetljiv na razliku izmedju malih i velikih
    slova. U HTML-u etikete BODY i body imaju isto znacenje,
142 dok to u C-u ne vazi. Zato cemo sve etikete prevoditi u
    zapis samo malim slovima. */
144 while ((c = fgetc(f)) != EOF) {
    switch (stanje) {
146     case VAN_ETIKETE:
        if (c == '<')
148             stanje = PROCITANO_MANJE;
        break;
150     case PROCITANO_MANJE:
        if (c == '/') {
152             /* Citamo zatvarac */
            tip = ZATVORENA;
154         } else {
            if (isalpha(c)) {
156                 /* Citamo otvarac */
                tip = OTVORENA;
158                 etiketa[i++] = tolower(c);
            }
160         }
        /* Sada citamo etiketu i zato menjamo stanje. */
162         stanje = U_ETIKETI;
        break;
164     case U_ETIKETI:
        if (isalpha(c) && i < MAX - 1) {
166             /* Ako je procitani karakter slovo i nismo premasili
                maksimalnu dozvoljenu duzinu za etiketu, smestamo
168             procitani karakter u etiketu. */

```

```
    etiketa[i++] = tolower(c);
170 } else {
    stanje = VAN_ETIKETE;
172 /* U suprotnom, prestajemo sa citanjem etikete i menjamo
    stanje. */
174 etiketa[i] = '\\0';
    /* Završili smo sa citanjem etikete i vraćamo tip
176 etikete koju smo procitali, a ona nam je sacuvana u
    nisci etiketa. */
178 return tip;
    }
180 break;
    }
182 }

184 /* Dosli smo do kraja datoteke pre nego sto smo završili sa
    citanjem etikete, stoga imamo gresku i vraćamo EOF. */
186 return EOF;
    }

188 int main(int argc, char **argv)
190 {
    /* Stek nam je prazan na pocetku. */
192 Cvor *vrh = NULL;
    char etiketa[MAX];
194 int tip;
    /* Na pocetku su nam etikete upare, jer nismo nijednu jos
196 procitali. */
    int uparene = 1;
198 FILE *f = NULL;
    /* Ime datoteke dobijamo iz komandne linije. */
200 if (argc < 2) {
    fprintf(stderr, "Koriszenje: %s ime_html_datoteke\\n",
202 argv[0]);
    exit(0);
204 }

206 /* Otvaramo datoteku. */
    if ((f = fopen(argv[1], "r")) == NULL) {
208 fprintf(stderr, "fopen() greska!\\n");
    exit(1);
210 }
    /* Dokle god ima etiketa, uzimamo ih jednu po jednu sa ulaza. */
212 while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
    /* Ako je otvorena etiketa, dodajemo je na stek. Izuzetak su
214 etikete <br>, <hr> i <meta> koje nemaju sadrzaj, tako da
    ih nije potrebno zatvoriti. NAPOMENA: U html-u postoje
216 jos neke etikete koje nemaju sadrzaj (npr link).
    Pretpostavimo da njih nema u dokumentu, zbog
218 jednostavnosti. */
    if (tip == OTVORENA) {
220 if (strcmp(etiketa, "br") != 0
```

```

222     && strcmp(etiketa, "hr") != 0
223     && strcmp(etiketa, "meta") != 0)
224     potisni_na_stek(&vrh, etiketa);
225 }
226 /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti
227 da je u pitanju zatvaranje etikete koja je poslednja
228 otvorena, a jos uvek nije zatvorena. Ova etiketa se mora
229 nalaziti na vrhu steka. Ako je taj uslov ispunjen, tada
230 je skidamo sa steka, jer je zatvorena. U suprotnom,
231 obavestavamo korisnika da etikete nisu pravilno uparene. */
232 else if (tip == ZATVORENA) {
233     if (vrh_steka(vrh) != NULL
234         && strcmp(vrh_steka(vrh), etiketa) == 0)
235         skini_sa_steka(&vrh, NULL);
236     else {
237         printf(vrh_steka(vrh) !=
238             NULL ? "Etikete nisu pravilno uparene\n(nadjena\
239 etiketa </%s> a poslednja otvorena etiketa je </%s>)\n" : "Etikete
240 nisu pravilno uparene\n(nadjena etiketa </%s> koja nije\
241 otvorena)\n", etiketa, vrh_steka(vrh));
242         uparene = 0;
243         break;
244     }
245 }
246 /* Zatvaramo datoteku. */
247 fclose(f);
248 /* Ako nismo pronasli pogresno uparivanje do sada, stek treba
249 da bude prazan. Ako nije, tada znaci da postoje jos neke
250 etikete koje su otvorene ali nisu bile zatvorene. */
251 if (uparene) {
252     if (vrh_steka(vrh) == NULL)
253         printf("Etikete su pravilno uparene!\n");
254     else
255         printf("Etikete nisu pravilno uparene\n(etiketa </%s>
256 nije zatvorena)\n", vrh_steka(vrh));
257 }
258 /* Oslobadjamo memoriju alociranu za stek, ukoliko vec nije. */
259 oslobodi_stek(&vrh);
260 return 0;
261 }

```

Rešenje 4.6

```

1  #ifndef _RED_H
2  #define _RED_H
3
4  #include <stdio.h>
5  #include <stdlib.h>

```

```

7  #define MAX 1000
   #define JMBG_DUZINA 14
9
11 /* Struktura kojom predstavljamo zahtev korisnika, obuhvata JMBG
   korisnika i opis njegovog zahteva. */
13 typedef struct {
   char jmbg[JMBG_DUZINA];
   char opis[MAX];
15 } Zahtev;

17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
   korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
   Zahtev nalog;
   struct cvor *sledeci;
21 } Cvor;

23 Cvor * napravi_cvor( Zahtev * zahtev);

25 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

27 void prover_i_alokaciju(Cvor ** adresa_pocetka,
29                        Cvor ** adresa_kraja, Cvor* novi) ;

31 void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev* zahtev);

33 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
35                  Zahtev *zahtev);

37 Zahtev * pocetak_reda(Cvor * pocetak) ;

39 void prikazi_red(Cvor * pocetak);

41 #endif

```

```

1  #include "red.h"

3  /* Funkcija kreira novi cvor, inicijalizuje polje nalog na
   zahtev sa poslate adrese i vraca adresu novog cvora ili NULL
5  ako je doslo do greske pri alokaciji. Ako je doslo do greske,
   trebalo bi osloboditi ceo red. Ostavljamo da to uradi funkcija
7  koja je pozvala funkciju napravi_cvor, a gresku signaliziramo
   saljuci joj NULL. Funkciji se prosledjuje pokazivac na zahtev
9  koji treba smestiti u nov cvor zbog smestanja manjeg podatka
   na sistemski stek. Pokazivac na strukturu Zahtev je manje
11 velicine u bajtovima(B) u odnosu na strukturu Zahtev. */
Cvor *napravi_cvor(Zahtev * zahtev)
13 {
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
15   if (novi == NULL)
       return NULL;

```

```
17     novi->nalog = *zahtev;
18     novi->sledeci = NULL;
19     return novi;
20 }
21
22 /* Funkcija prazni red */
23 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
24 {
25     Cvor * pomocni = NULL;
26
27     while (*pocetak != NULL) {
28         pomocni = *pocetak;
29         *pocetak = (*pocetak)->sledeci;
30         free(pomocni);
31     }
32     *kraj = NULL;
33 }
34
35 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
36 i ukoliko alokacija nije bila uspesna, oslobadja se sva
37 prethodno zauzeta memorija za listu cija pocetni cvor se
38 nalazi na adresi adresa_pocetka. */
39 void prover_i_alokaciju(Cvor ** adresa_pocetka,
40                         Cvor ** adresa_kraja, Cvor * novi)
41 {
42     if (novi == NULL) {
43         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
44         oslobodi_red(adresa_pocetka, adresa_kraja);
45         exit(EXIT_FAILURE);
46     }
47 }
48
49
50 /* Funkcija dodaje na kraj reda novi fajl. */
51 void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
52                 Zahtev * zahtev)
53 {
54     Cvor * novi = napravi_cvor(zahtev);
55     prover_i_alokaciju(adresa_pocetka, adresa_kraja, novi);
56
57     /* U red se uvek dodaje na kraj, ali zbog postojanja
58 pokazivaca na kraj, dodavanje na kraj je podjednako
59 efikasno kao dodavanje na pocetak. */
60 if (*adresa_kraja != NULL) {
61     (*adresa_kraja)->sledeci = novi;
62     *adresa_kraja = novi;
63 } else {
64     /* Ako je red bio ranije prazan */
65     *adresa_pocetka = novi;
66     *adresa_kraja = novi;
67 }
```

```
69 }
71 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji
73 argument pokazivac razlicit od NULL, tada se u strukturu na
75 koju on pokazuje upisuje zahtev koji je upravo skinut sa reda
77 dok u suprotnom ne upisuje nista. Funkcija vraca 0 ako je red
79 bio prazan ili 1 u suprotnom. */
81 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
83 Zahtev * zahtev)
85 {
87     Cvor *pomocni = NULL;
89     if (*adresa_pocetka == NULL)
91         return 0;
93     if (zahtev != NULL)
95         *zahtev = (*adresa_pocetka)->nalog;
97     pomocni = *adresa_pocetka;
99     *adresa_pocetka = (*adresa_pocetka)->sledeci;
101     free(pomocni);
103     if (*adresa_pocetka == NULL)
105         *adresa_kraja = NULL;
107     return 1;
109 }
111 /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev
113 korisnika na pocetku reda. Ukoliko je red prazan, vraca NULL.
115 */
117 Zahtev * pocetak_reda(Cvor * pocetak)
119 {
121     if (pocetak == NULL)
123         return NULL;
125     return &(pocetak->nalog);
127 }
129 /* Funkcija prikazuje red. */
131 void prikazi_red(Cvor * pocetak)
133 {
135     for (; pocetak != NULL; pocetak = pocetak->sledeci)
137         printf("%s %s\n",
139             (pocetak->nalog).jmbg, (pocetak->nalog).opis);
141     printf("\n");
143 }
145
147 #include <stdio.h>
149 #include <stdlib.h>
```

```

3  #include <string.h>
   #include "red.h"
5
   #define VREME_ZA_PAUZU 5
7
9  int main(int argc, char **argv)
   {
11     /* Red je prazan. */
       Cvor *pocetak = NULL, *kraj = NULL;
       Zahtev nov_zahtev;
13     Zahtev *sledeci = NULL;
       char odgovor[3];
15     int broj_usluzenih = 0;
       FILE *izlaz = fopen("izvestaj.txt", "a");
17
       if (izlaz == NULL) {
19         fprintf(stderr, "Neuspesno otvaranje datoteke \
izvestaj.txt\n");
21         exit(EXIT_FAILURE);
       }
23
       /* Sluzbenik evidentira korisnicke zahteve. */
25     printf("Sluzbenik evidentira korisnicke zahteve unosnjem \
njihovog JMBG broja i opisa potrebne usluge:\n[CTRL+D za \
kraj]\n");
27
29     /* Neophodan je poziv funkcije getchar da bi se i nov red
       nakon JMBG broja procitao i da bi fgets nakon toga
       procitala ispravan red sa opisom zahteva. */
31     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
       while (scanf("%s", nov_zahtev.jmbg) != EOF) {
33         getchar();
35         printf("\tOpis problema: ");
         fgets(nov_zahtev.opis, MAX - 1, stdin);
37         /* Ako je poslednji karakter nov red, eliminisemo ga. */
         if (nov_zahtev.opis[strlen(nov_zahtev.opis)-1] == '\n')
39             nov_zahtev.opis[strlen(nov_zahtev.opis)-1] = '\0';
         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
41         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
       }
43
       /* Dokle god ima korisnika u redu, usluujemo ih. */
45     while (1) {
         sledeci = pocetak_reda(pocetak);
47         /* Ako nema nikog vise u redu. */
         if (sledeci == NULL)
49             break;

51         printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
            sledeci->jmbg);
53         printf("sa zahtevom: %s\n", sledeci->opis);

```

```

55     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
57     broj_usluzenih++;
59     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
    scanf("%s", odgovor);
61
    if (strcmp(odgovor, "Da") == 0)
63         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
    else
65         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
                    nov_zahtev.jmbg, nov_zahtev.opis);
67
        if (broj_usluzenih == VREME_ZA_PAUZU) {
69             printf("\nDa li je kraj smene? [Da/Ne] ");
            scanf("%s", odgovor);
71
            if (strcmp(odgovor, "Da") == 0)
73                 break;
            else
75                 broj_usluzenih = 0;
        }
77     }
79     fclose(izlaz);
81     /* Oslobođamo red ukoliko je službenik prekinuo sa radom
        možda je bilo još neusluženih korisnika. */
83     oslobodi_red(&pocetak, &kraj);
85     return 0;
}

```

```

/* Alternativno rešenje bez korišćenja funkcije za skidanje
2     zahteva sa pocetak reda. */

4     #include <stdio.h>
    #include <stdlib.h>
6     #include <string.h>
    #include "red.h"
8
    #define VREME_ZA_PAUZU 5
10
    int main(int argc, char **argv)
12    {
        /* Red je prazan. */
14        Cvor *pocetak = NULL, *kraj = NULL;
        Zahtev nov_zahtev;
16        Zahtev *sledeci = NULL;
        char odgovor[3];
18        int broj_usluzenih = 0;
        FILE *izlaz = fopen("izvestaj.txt", "a");

```



```
20     if (izlaz == NULL) {
22         fprintf(stderr, "Neuspesno otvaranje datoteke \
izvestaj.txt\n");
24         exit(EXIT_FAILURE);
25     }
26
27     /* Sluzbenik evidentira korisnicke zahteve. */
28     printf("Sluzbenik evidentira korisnicke zahteve unosnjem \
njihovog JMBG broja i opisa potrebne usluge:\n");
29     /* Neophodan je poziv funkcije getchar da bi se i nov red
30     nakon JMBG broja procitao i da bi fgets nakon toga
31     procitala ispravan red sa opisom zahteva. */
32     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
33     while (scanf("%s", nov_zahtev.jmbg) != EOF) {
34         getchar();
35         printf("\tOpis problema: ");
36         fgets(nov_zahtev.opis, MAX - 1, stdin);
37         /* Ako je poslednji karakter nov red, eliminisemo ga. */
38         if (nov_zahtev.opis[strlen(nov_zahtev.opis)-1] == '\n')
39             nov_zahtev.opis[strlen(nov_zahtev.opis)-1] = '\0';
40         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
41         printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
42     }
43
44     /* Dokle god ima korisnika u redu, usluujemo ih. */
45     while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
46         printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
47             nov_zahtev.jmbg);
48         printf("sa zahtevom: %s\n", nov_zahtev.opis);
49
50         broj_usluzenih++;
51
52         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
53         scanf("%s", odgovor);
54
55         if (strcmp(odgovor, "Da") == 0)
56             dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
57         else
58             fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
59                 nov_zahtev.jmbg, nov_zahtev.opis);
60
61         if (broj_usluzenih == VREME_ZA_PAUZU) {
62             printf("\nDa li je kraj smene? [Da/Ne] ");
63             scanf("%s", odgovor);
64
65             if (strcmp(odgovor, "Da") == 0)
66                 break;
67             else
68                 broj_usluzenih = 0;
69         }
70     }
71 }
```

```
72     fclose(izlaz);
74
76     /* Oslobadjamo red ukoliko je sluzbenik prekinuo sa radom
       mozda je bilo jos neusluzenih korisnika. */
78     oslobodi_red(&pocetak, &kraj);
79
80     return 0;
81 }
```

Rešenje 4.7

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #define MAX_DUZINA 20
5
6  typedef struct _Element {
7      unsigned broj_pojavljivanja;
8      char etiketa[20];
9      struct _Element *sledeci;
10 } Element;
11
12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi
13    cvor ili NULL ako alokacija nije uspesno izvrшена. */
14 Element *napravi_cvor(unsigned br, char *etiketa)
15 {
16     Element *novi = (Element *) malloc(sizeof(Element));
17     if (novi == NULL)
18         return NULL;
19
20     novi->broj_pojavljivanja = br;
21     strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
23     return novi;
24 }
25
26 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
27    liste. */
28 void oslobodi_listu(Element ** glava)
29 {
30     Element *pomocni = NULL;
31
32     while (*glava != NULL) {
33         pomocni = (*glava)->sledeci;
34         free(*glava);
35         *glava = pomocni;
36     }
37 }
38
39 /* Funkcija proverava uspesnost alokacije memorije za cvor novi
```

```

41     i ukoliko alokacija nije bila uspesna, oslobadja se sva
    prethodno zauzeta memorija za listu cija pocetni cvor se
    nalazi na adresi glava. */
43 void provera_alokacije(Element * novi, Element ** glava)
    {
45     if (novi == NULL) {
        fprintf(stderr, "malloc() greska u funkciji
47     napravi_cvor()!\n");
        oslobodi_listu(glava);
49     exit(EXIT_FAILURE);
    }
51 }

53 /* Funkcija dodaje novi cvor na pocetak liste. */
void dodaj_na_pocetak_liste(Element ** glava, unsigned br,
55     char *etiketa)
    {
57     Element *novi = napravi_cvor(br, etiketa);
    provera_alokacije(novi, glava);
59     novi->sledeci = *glava;
    *glava = novi;
61 }

63 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu.
    (NULL u suprotnom) */
65 Element *pretrazi_listu(Element * glava, char etiketa[])
    {
67     Element *tekuci;
    for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
69         if (strcmp(tekuci->etiketa, etiketa) == 0)
            return tekuci;
71     return NULL;
    }

73
75 /* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Element * glava)
    {
77     for (; glava != NULL; glava = glava->sledeci)
        printf("%s - %u\n", glava->etiketa,
79         glava->broj_pojavljivanja);
    }

81
83 int main(int argc, char **argv)
    {
    if (argc != 2) {
85         fprintf(stderr, "Greska! Program se poziva sa: ./a.out
    datoteka.html!\n");
87         exit(EXIT_FAILURE);
    }

89
    FILE *in = NULL;
91     in = fopen(argv[1], "r");

```

```
93     if (in == NULL) {
94         fprintf(stderr,
95             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
96         exit(EXIT_FAILURE);
97     }
98
99     char c;
100     int i = 0;
101     char a[MAX_DUZINA];
102
103     Element *glava = NULL;
104     Element *trazeni = NULL;
105
106     while ((c = fgetc(in)) != EOF) {
107         if (c == '<') {
108             /* Citamo zatvarac */
109             if ((c = fgetc(in)) == '/') {
110                 i = 0;
111                 while ((c = fgetc(in)) != '>')
112                     a[i++] = c;
113             }
114
115             /* Citamo otvarac */
116             else {
117                 i = 0;
118                 a[i++] = c;
119                 while ((c = fgetc(in)) != ' ' && c != '>')
120                     a[i++] = c;
121             }
122             a[i] = '\0';
123
124             /* Ispitujemo da li medju do sada formiranim cvorovima
125              postoji cvor sa ucitanom etiketom. Ukoliko ne postoji,
126              dodajemo novi cvor za ucitanu etiketu (broj
127              pojavljivanja postavljamo na 1), inace uvecavamo broj
128              pojavljivanja. */
129             trazeni = pretrazi_listu(glava, a);
130             if (trazeni == NULL)
131                 dodaj_na_pocetak_liste(&glava, 1, a);
132             else
133                 trazeni->broj_pojavljivanja++;
134         }
135     }
136
137     ispisi_listu(glava);
138     oslobodi_listu(&glava);
139
140     fclose(in);
141     return 0;
142 }
```

Rešenje 4.8

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "601/lista.h"
4
5  Cvor *objedini(Cvor ** glava1, Cvor ** glava2)
6  {
7      Cvor *l3 = NULL;
8      Cvor **tek = &l3;
9
10     if (*glava1 == NULL && *glava2 == NULL)
11         return NULL;
12
13     /* Ako je prva lista prazna, onda je rezultat druga lista. */
14     if (*glava1 == NULL)
15         return *glava2;
16
17     /* Ako je druga lista prazna, onda je rezultat prva lista. */
18     if (*glava2 == NULL)
19         return *glava1;
20
21     /* l3 pokazuje na pocetak nove liste, tj. na manji od brojeva
22        sadrzanih u cvorovima na koje pokazuju glava1 i glava2. */
23     l3 = ((*glava1)->vrednost < (*glava2)->vrednost) ? *glava1 :
24         *glava2;
25
26     while (*glava1 != NULL && *glava2 != NULL) {
27         if ((*glava1)->vrednost < (*glava2)->vrednost) {
28             *tek = *glava1;
29             *glava1 = (*glava1)->sledeci;
30         } else {
31             *tek = *glava2;
32             *glava2 = (*glava2)->sledeci;
33         }
34         tek = &((*tek)->sledeci);
35     }
36
37     /* Ukoliko smo izašli iz petlje zato što smo stigli do kraja
38        prve liste onda na rezultujuću listu nadovezujemo ostatak
39        druge liste. */
40     if (*glava1 == NULL)
41         *tek = *glava2;
42
43     else if (*glava2 == NULL)
44         *tek = *glava1;
45
46     return l3;
47 }
48
49 int main(int argc, char **argv)
```

```
51 {  
52     if (argc != 3) {  
53         fprintf(stderr,  
54             "Greska! Program se poziva sa: ./a.out dat1.txt dat2.txt  
55             !\n");  
56         exit(EXIT_FAILURE);  
57     }  
58  
59     FILE *in1 = NULL;  
60     in1 = fopen(argv[1], "r");  
61     if (in1 == NULL) {  
62         fprintf(stderr,  
63             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);  
64         exit(EXIT_FAILURE);  
65     }  
66  
67     FILE *in2 = NULL;  
68     in2 = fopen(argv[2], "r");  
69     if (in2 == NULL) {  
70         fprintf(stderr,  
71             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);  
72         exit(EXIT_FAILURE);  
73     }  
74  
75     int broj;  
76     Cvor *glava1 = NULL;  
77     Cvor *glava2 = NULL;  
78     Cvor *l3 = NULL;  
79  
80     /* Ucitavamo liste. */  
81     while (fscanf(in1, "%d", &broj) != EOF)  
82         dodaj_na_kraj_liste(&glava1, broj);  
83     while (fscanf(in2, "%d", &broj) != EOF)  
84         dodaj_na_kraj_liste(&glava2, broj);  
85  
86     /* Objedinjujemo ih u jednu listu. */  
87     l3 = objedini(&glava1, &glava2);  
88  
89     /* Ispisujemo rezultujuću listu. */  
90     ispisi_listu(l3);  
91     oslobodi_listu(&l3);  
92  
93     fclose(in1);  
94     fclose(in2);  
95     return 0;  
96 }
```

Rešenje 4.9

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

```

3  #include<string.h>

5  #define MAX_INDEKS 11
   #define MAX_IME_PREZIME 21

7
   typedef struct _Cvor {
9     char broj_indeksa[MAX_INDEKS];
       char ime[MAX_IME_PREZIME];
11    char prezime[MAX_IME_PREZIME];
       struct _Cvor *sledeci;
13 } Cvor;

15 /* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac
   na nov cvor ili NULL ukoliko alokacija nije prosla. */
17 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
   {
19     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
21         return NULL;
       strcpy(novi->broj_indeksa, broj_indeksa);
23     strcpy(novi->ime, ime);
       strcpy(novi->prezime, prezime);
25     novi->sledeci = NULL;
       return novi;
27 }

29 /* Funkcija oslobadja memoriju zauzetu za elemente liste. */
   void oslobodi_listu(Cvor ** glava)
31 {
       if (*glava == NULL)
33         return;
       oslobodi_listu(&(*glava)->sledeci);
35     free(*glava);
       *glava = NULL;
37 }

39 void prover_i_alokaciju(Cvor ** glava, Cvor * novi)
   {
41     if (novi == NULL) {
         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
43         oslobodi_listu(glava);
         exit(EXIT_FAILURE);
45     }
   }

47
   /* Funkcija dodaje novi cvor na pocetak liste. */
49 void dodaj_na_pocetak_liste(Cvor ** glava, char *broj_indeksa,
       char *ime, char *prezime)
51 {
       Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
53     prover_i_alokaciju(glava, novi);
       novi->sledeci = *glava;

```

```

55     *glava = novi;
56 }
57
58 void ispisi_listu(Cvor * glava)
59 {
60     for (; glava != NULL; glava = glava->sledeci)
61         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
62                glava->prezime);
63 }
64
65 /* Funkcija vraća cvor koji kao vrednost sadrži traženu etiketu,
66    u suprotnom vraća NULL. */
67 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
68 {
69     if (glava == NULL)
70         return NULL;
71     if (!strcmp(glava->broj_indeksa, broj_indeksa))
72         return glava;
73     return pretrazi_listu(glava->sledeci, broj_indeksa);
74 }
75
76 int main(int argc, char **argv)
77 {
78     if (argc != 2) {
79         fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
80 student1.txt!\n");
81         exit(EXIT_FAILURE);
82     }
83
84     FILE *in = NULL;
85     in = fopen(argv[1], "r");
86     if (in == NULL) {
87         fprintf(stderr,
88                 "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
89         exit(EXIT_FAILURE);
90     }
91
92     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
93     char broj_indeksa[MAX_INDEKS];
94     Cvor *glava = NULL;
95     Cvor *trazeni = NULL;
96
97     /* Učitavamo listu sa standardnog ulaza. */
98     while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) !=
99            EOF)
100         dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);
101
102     while (scanf("%s", broj_indeksa) != EOF) {
103         trazeni = pretrazi_listu(glava, broj_indeksa);
104         if (trazeni == NULL)
105             printf("ne\n");
106         else

```



```
107     printf("da: %s %s\n", trazen->ime, trazen->prezime);
108 }
109
110     oslobodi_listu(&glava);
111     fclose(in);
112     return 0;
113 }
```

Rešenje 4.10

Rešenje 4.11

Rešenje 4.12

Rešenje 4.13

Rešenje 4.14

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4      /* a) Struktura kojom se predstavlja cvor binarnog
5         pretrazivackog stabla */
6      typedef struct cvor {
7
8          int broj;
9
10         struct cvor *levo;
11
12         struct cvor *desno;
13     } Cvor;
14
15
16     /* b) Funkcija koja alocira memoriju za novi cvor stabla,
17        inicijalizuje polja strukture i vraca pokazivac na novi
18        cvor */
19     Cvor * napravi_cvor(int broj)
20     {
21
22         /* Alociramo memoriju */
23         Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
24
25         if (novi == NULL)
26
27         return NULL;
28
29     }
```

```
31      /* Inicijalizujemo polja novog cvora. */
      novi->broj = broj;
33  novi->levo = NULL;
35  novi->desno = NULL;
37
      /* Vracamo adresu novog cvora. */
39  return novi;
41 }
43
45  /* Funkcija koja proverava uspesnost kreiranja novog cvora
      stabla */
47 void proveri_alokaciju(Cvor * novi_cvor)
{
49
      /* Ukoliko je cvor neuspesno kreiran */
51  if (novi_cvor == NULL) {
53
          /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
              programa
55  */
          fprintf(stderr, "Malloc greska za novi cvor!\n");
57  exit(EXIT_FAILURE);
59  }
61 }
63
65  /* c) Funkcija koja dodaje zadati broj u stablo */
67 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
{
69
71      /* Ako je stablo prazno */
      if (*adresa_korena == NULL) {
73
75          /* Kreiramo novi cvor */
          Cvor * novi = napravi_cvor(broj);
77  proveri_alokaciju(novi);
79
81      /* I proglašavamo ga korenom stabla */
```

```

83     *adresa_korena = novi;
85 return;
87 }
89
90     /* U suprotnom trazimo odgovarajucu poziciju za zadati
91        broj */
92
93     /* Ako je zadata vrednost manja od vrednosti korena */
94     if (broj < (*adresa_korena)->broj)
95
96         /* Dodajemo broj u levo podstablo */
97         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
98
99     else
100
101         /* Inace, broj je veci (ili jednak) od vrednosti u
102            korenu pa ga
103            dodajemo u desno podstablo */
104         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
105
106 }
107
108
109 /*
110 d) Funkcija koja proverava da li se zadati broj nalazi
111    u stablu
112 */
113
114 Cvor * pretrazi_stablo(Cvor * koren, int broj)
115 {
116
117     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u
118        njemu */
119     if (koren == NULL)
120
121         return NULL;
122
123
124     /* Ako je trazena vrednost sadrazana u korenu */
125     if (koren->broj == broj) {
126
127         /* Prekidamo pretragu */
128         return koren;
129     }
130
131
132
133     /* Inace, ako je broj manji od vrednosti sadrzane u korenu
```

```
135     */
136     if (broj < koren->broj)
137
138         /* Pretragu nastavljamo u levom podstablu */
139         return pretrazi_stablo(koren->levo, broj);
140
141     else
142
143         /* U suprotnom, pretragu nastavljamo u desnom podstablu */
144         return pretrazi_stablo(koren->desno, broj);
145 }
146
147
148
149 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost
150    u stablu */
151 Cvor * pronadji_najmanji(Cvor * koren)
152 {
153
154     /* Ako je stablo prazno, prekidamo pretragu */
155     if (koren == NULL)
156
157     return NULL;
158
159
160     /* Vrednosti koje su manje od vrednosti u korenu stabla
161        nalaze se levo od njega */
162
163     /* Ako je koren cvor koji nema levo podstablo, onda on
164        sadrzi najmanju vrednost */
165     if (koren->levo == NULL)
166
167     return koren;
168
169
170     /* Inace, pretragu treba nastaviti u levom podstablu */
171     return pronadji_najmanji(koren->levo);
172 }
173
174
175
176
177 /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u
178    stablu
179    */
180 Cvor * pronadji_najveci(Cvor * koren)
181 {
182
183     /* Ako je stablo prazno, prekidamo pretragu */
184     if (koren == NULL)
```

```
187 return NULL;
189 /* Vrednosti koje su vece od vrednosti u korenu stabla
191    nalaze se desno od njega */
193 /* Ako je koren cvor koji nema desno podstablo, onda on
195    sadrzi najveću vrednost */
197 if (koren->desno == NULL)
199 return koren;
201 /* Inace, pretragu treba nastaviti u desnom podstablu */
203 return pronadji_najveci(koren->desno);
205
207 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati
209    broj */
211 void obrisi_element(Cvor ** adresa_korena, int broj)
213 {
215     Cvor * pomocni_cvor = NULL;
217     /* ako je stablo prazno, brisanje nije primenljivo pa
219        mozemo
221        prekinuti rad funkcije */
223     if (*adresa_korena == NULL)
225 return;
227 /* Ako je vrednost koju treba obrisati manja od vrednosti
229    u korenu stabla,
231    ona se eventualno nalazi u levom
233    podstablu,
235    pa treba rekursivno primeniti postupak na
237    levo
239    podstablo. Koren ovako modifikovanog stabla je
241    nepromenjen. */
243     if (broj < (*adresa_korena)->broj) {
245 obrisi_element(&(*adresa_korena)->levo, broj);
247 return;
249 }
251 }
```

```
239         /* Ako je vrednost koju treba obrisati veca od vrednosti u
241         korenu stabla,
243         ona se eventualno nalazi u
245         desnom
247         podstablu
249         pa treba rekurzivno primeniti
251         postupak na
253         desno
255         podstablo. Koren ovako
257         modifikovanog stabla
259         je
261         nepromenjen. */
263         if ((*adresa_korena)->broj < broj) {
265             obrisi_element(&(*adresa_korena)->desno, broj);
267         return;
269     }

271     /* Slede podslucajevi vezani za slucaj kada je vrednost
273     u
275     korenu jednaka broju koji se brise (tj. slucaj
277     kada
279     treba obrisati koren) */

281     /* Ako koren nema sinova, tada se on prosto brise, i
283     rezultat je prazno stablo (vracamo NULL) */
285     if ((*adresa_korena)->levo == NULL
287         &&(*adresa_korena)->desno == NULL) {
289         free(*adresa_korena);
291         *adresa_korena = NULL;
293         return;
295     }

297     /* Ako koren ima samo levog sina, tada se brisanje vrši
299     tako sto obrisemo koren, a novi koren postaje levi sin */
301     if ((*adresa_korena)->levo != NULL
303         &&(*adresa_korena)->desno == NULL) {
305         pomocni_cvor = (*adresa_korena)->levo;
```

```
291 free(*adresa_korena);
293 *adresa_korena = pomocni_cvor;
295 return;
297 }

299     /* Ako koren ima samo desnog sina, tada se brisanje vrši
301        tako sto obrisemo koren, a novi koren postaje desni sin
303        */
305     if ((*adresa_korena)->desno != NULL
306         &&(*adresa_korena)->levo == NULL) {
307         pomocni_cvor = (*adresa_korena)->desno;
309         free(*adresa_korena);
311         *adresa_korena = pomocni_cvor;
313         return;
315     }

317     /* Slučaj kada koren ima oba sina. Tada se brisanje vrši
319        na sledeći način:
321        - najpre se potraži sledbenik
322          korena (u smislu poretka) u stablu. To je upravo po
323          vrednosti najmanji cvor u desnom podstablu.
324        On se može
325        pronaći npr. funkcijom pronadji_najmanji().
326        Nakon
327          toga se u koren smesti vrednost tog cvora, a
328          u
329          taj
330             cvor
331          se smesti vrednost korena (tj. broj koji
332          se
333             briše).
334
335        - Onda se prosto rekurzivno pozove
336          funkcija
337          za brisanje
338          na desno podstablo. S obzirom
339          da u njemu
340          treba
341          obrisati
```

```
    najmanji element, a on
343         zasigurno ima
    najvise
345     jednog potomka, jasno je da ce
        njegovo
347     brisanje biti
        obavljeno na
349     jedan od
        jednostavnijih
351     nacina koji su
        gore opisani. */
353     pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);

355     (*adresa_korena)->broj = pomocni_cvor->broj;

357     pomocni_cvor->broj = broj;

359     obrisi_element(&(*adresa_korena)->desno, broj);

361 }

363

365     /* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
        postablo - Koren - Desno podstablo ) */
367 void ispisi_stablo_infiksno(Cvor * koren)
{
369     /* Ako stablo nije prazno */
371     if (koren != NULL) {

373         /* Prvo ispisujemo sve cvorove levo od korena */
        ispisi_stablo_infiksno(koren->levo);
375

377         /* Ispisujemo vrednost u korenu */
        printf("%d ", koren->broj);
379

381         /* Na kraju ispisujemo cvorove desno od korena */
        ispisi_stablo_infiksno(koren->desno);
383     }
385 }

387 }

389

391     /* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren
        - Levo
393     podstablo - Desno podstablo ) */
```



```
void ispisi_stablo_prefiksno(Cvor * koren)
395 {
397     /* Ako stablo nije prazno */
399     if (koren != NULL) {
401         /* Prvo ispisujemo vrednost u korenu */
403         printf("%d ", koren->broj);
405         /* Ispisujemo sve cvorove levo od korena */
407         ispisi_stablo_prefiksno(koren->levo);
409         /* Na kraju ispisujemo sve cvorove desno od korena */
411         ispisi_stablo_prefiksno(koren->desno);
413     }
415 }
417
419 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
    podstablo - Desno postablo - Koren) */
421 void ispisi_stablo_postfiksno(Cvor * koren)
423 {
425     /* Ako stablo nije prazno */
427     if (koren != NULL) {
429         /* Prvo ispisujemo sve cvorove levo od korena */
431         ispisi_stablo_postfiksno(koren->levo);
433         /* Ispisujemo sve cvorove desno od korena */
435         ispisi_stablo_postfiksno(koren->desno);
437         /* Na kraju ispisujemo vrednost u korenu */
439         printf("%d ", koren->broj);
441     }
443 }
445
```

```
447  /*
448  k) Funkcija koja oslobadja memoriju zauzetu stablom.
449  */
451  void oslobodi_stablo(Cvor ** adresa_korena)
452  {
453      /* Ako je stablo prazno, nepotrebno je oslobadjati
454      memoriju */
455      if (*adresa_korena == NULL)
456      {
457          return;
458      }
459
460      /* U suprotnom rekurzivno oslobadjamo memoriju koje
461      zauzima najpre
462      levo, a zatim i desno podstablo */
463      oslobodi_stablo(&(*adresa_korena)->levo);
464
465      oslobodi_stablo(&(*adresa_korena)->desno);
466
467      /* Oslobadjamo memoriju koju zauzima koren */
468      free(*adresa_korena);
469
470      /* I proglašavamo stablo praznim */
471      *adresa_korena = NULL;
472  }
473
474
475
476
477
478
479  int main()
480  {
481
482      Cvor * koren;
483
484      int n;
485
486      Cvor * trazeni_cvor;
487
488      /* Proglasavamo stablo praznim */
489      koren = NULL;
490
491      /* Dodajemo vrednosti u stablo */
492      printf("Unesite brojeve (CTRL+D za kraj unosa): ");
493
494      while (scanf("%d", &n) != EOF) {
```

```
499 dodaj_u_stablo(&koren, n);
501 }
503     /* Generisemo trazene ispise: */
504     printf("\nInfiksni ispis: ");
505     ispisi_stablo_infiksno(koren);
507     printf("\nPrefiksni ispis: ");
509     ispisi_stablo_prefiksno(koren);
511     printf("\nPostfiksni ispis: ");
513     ispisi_stablo_postfiksno(koren);
515
517     /* Demonstriramo rad funkcije za pretragu */
518     printf("\nTrazi se broj: ");
521     scanf("%d", &n);
523     trazeni_cvor = pretrazi_stablo(koren, n);
525     if (trazeni_cvor == NULL)
527         printf("Broj se ne nalazi u stablu!\n");
529     else
531         printf("Broj se nalazi u stablu!\n");
533
535     /* Demonstriramo rad funkcije za brisanje */
536     printf("Briše se broj: ");
537     scanf("%d", &n);
539     obrisi_element(&koren, n);
541     printf("Rezultujuće stablo: ");
543     ispisi_stablo_infiksno(koren);
545     printf("\n");
547
549     /* Oslobadjamo memoriju zauzetu stablom */
```

```
551         oslobodi_stablo(&koren);  
  
553         /* Prekidamo sa programom */  
555         return 0;  
  
557     }
```

Rešenje 4.15

```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  #include <string.h>  
4  #include <ctype.h>  
5  
6  #define MAX 50  
7  
8      /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen  
9      broj  
10     pojavljivanja i redom pokazivace na levo i desno  
11     podstablo */  
12     typedef struct cvor {  
13  
14         char *rec;  
15  
16         int brojac;  
17  
18         struct cvor *levo;  
19  
20         struct cvor *desno;  
21     } Cvor;  
22  
23  
24     /* Funkcija koja kreira novi cvora stabla */  
25     Cvor * napravi_cvor(char *rec)  
26     {  
27  
28         /* Alociramo memoriju za novi cvor */  
29         Cvor * novi_cvor = (Cvor *) malloc(sizeof(Cvor));  
30  
31         if (novi_cvor == NULL)  
32  
33             return NULL;  
34  
35         /* Alociramo memoriju za zadatu rec: potrebno je  
36         rezervisati  
37         memoriju za svaki karakter reci
```

```
41     ukljucujuci i terminirajucu nulu */
    novi_cvor->rec =
43     (char *) malloc((strlen(rec) + 1) * sizeof(char));

45 if (novi_cvor->rec == NULL) {
47     free(novi_cvor);
49     return NULL;
51 }

53     /* Inicijalizujemo polja u novom cvoru */
55     strcpy(novi_cvor->rec, rec);

57     novi_cvor->brojac = 1;
59     novi_cvor->levo = NULL;
61     novi_cvor->desno = NULL;

63     /* Vracamo adresu novog cvora */
65     return novi_cvor;
67 }

69

71     /* Funkcija koja proverava uspesnost kreiranja novog cvora
       stabla */
73 void proveri_alokaciju(Cvor * novi_cvor)
74 {
75
77     /* Ukoliko je cvor neuspesno kreiran */
78     if (novi_cvor == NULL) {
79
81         /* Ispisuje se odgovarajuca poruka i prekida
           izvršavanje programa */
82         fprintf(stderr, "Malloc greska za novi cvor!\n");
83
84         exit(EXIT_FAILURE);
85     }
87 }
88
89
91     /* Funkcija koja dodaje novu rec u stablo. */
```

```
93 void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
94 {
95
96     /* Ako je stablo prazno */
97     if (*adresa_korena == NULL) {
98
99         /* Kreiramo novi cvor */
100         Cvor * novi = napravi_cvor(rec);
101
102         proveri_alokaciju(novi);
103
104         /* i proglašavamo ga korenom stabla */
105         *adresa_korena = novi;
106
107         return;
108     }
109
110     /* U suprotnom trazimo odgovarajucu poziciju za novu rec */
111
112     /* Ako je rec leksikografski manju od reci u korenu
113        ubacujemo
114        je u levo podstablo */
115     if (strcmp(rec, (*adresa_korena)->rec) < 0)
116
117         dodaj_u_stablo(&(*adresa_korena)->levo, rec);
118
119     else
120
121         /* Ako je rec leksikografski veca od reci u korenu
122            ubacujemo je u desno podstablo */
123         if (strcmp(rec, (*adresa_korena)->rec) > 0)
124
125             dodaj_u_stablo(&(*adresa_korena)->desno, rec);
126
127     else
128
129         /* Ako je rec jednaka reci u korenu, uvecavamo njen
130            broj pojavljivanja */
131         (*adresa_korena)->brojac++;
132
133     }
134
135     /* Funkcija koja oslobadja memoriju zauzetu stablom */
136     void oslobodi_stablo(Cvor ** adresa_korena)
137     {
```

```
145      /* Ako je stablo prazno, nepotrebno je oslobadjati
146      memoriju */
147      if (*adresa_korena == NULL)
148
149  return;
150
151
152      /* Inace ... */
153      /* Oslobadjamo memoriju zauzetu levim podstablom */
154      oslobodi_stablo(&(*adresa_korena)->levo);
155
156      /* Oslobadjamo memoriju zauzetu desnim podstablom */
157      oslobodi_stablo(&(*adresa_korena)->desno);
158
159
160      /* Oslobadjamo memoriju zauzetu korenom */
161      free((*adresa_korena)->rec);
162
163  free(*adresa_korena);
164
165
166      /* Proglasavamo stablo praznim */
167      *adresa_korena = NULL;
168
169  }
170
171
172
173
174
175      /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju
176      rec (rec
177      sa najvećim brojem pojavljivanja) */
178      Cvor * najdi_najfrekventniju_rec(Cvor * koren)
179  {
180
181
182
183  Cvor * max, *max_levo, *max_desno;
184
185
186      /* Ako je stablo prazno, prekidamo sa pretragom */
187      if (koren == NULL)
188
189  return NULL;
190
191
192      /* Pronalazimo najfrekventniju reci u levom podstablu */
193      max_levo = najdi_najfrekventniju_rec(koren->levo);
194
195
196      /* Pronalazimo najfrekventniju reci u desnom podstablu */
```

```
197     max_desno = najdi_najfrekventniju_rec(koren->desno);
199
200     /* Trazimo maksimum vrednosti pojavljivanja reci iz
201     levog
202         podstabla, korena i desnog podstabla */
203     max = koren;
204
205     if (max_levo != NULL && max_levo->brojac > max->brojac)
206
207     max = max_levo;
208
209     if (max_desno != NULL && max_desno->brojac > max->brojac)
210
211     max = max_desno;
212
213     /* Vracamo adresu cvora sa najvećim brojacem */
214     return max;
215 }
216
217
218
219
220
221     /* Funkcija koja ispisuje reci iz stabla u leksikografskom
222     poretku
223     pracene brojem pojavljivanja */
224 void prikazi_stablo(Cvor * koren)
225 {
226
227
228
229     /* Ako je stablo prazno, završavamo sa ispisom */
230     if (koren == NULL)
231
232     return;
233
234
235     /* Zbog leksikografskog poretka, prvo ispisujemo sve reci
236     iz
237     levog podstabla */
238     prikazi_stablo(koren->levo);
239
240     /* Zatim ispisujemo rec iz korena */
241     printf("%s: %d\n", koren->rec, koren->brojac);
242
243     /* I nastavljamo sa ispisom reci iz desnog podstabla */
244     prikazi_stablo(koren->desno);
245 }
246
247
```



```
249      /* Funkcija ucitava sledecu rec iz zadate datoteke i upisuje
251      je
u niz rec. Maksimalna duzina reci je odredjena
253      argumentom max.
Funkcija vraca EOF ako nema vise reci
255      ili 0 u suprotnom.
Rec je niz malih ili velikih slova. */
257 int procitaj_rec(FILE * f, char rec[], int max)
{
259
      /* karakter koji citamo */
261 int c;

      /* indeks pozicije na koju se smesta procitani karakter */
263 int i = 0;
265

      /* Sve dok ima mesta za jos jedan karakter u nizu
i dokle
267      god nismo stigli do kraja datoteke... */
269 while (i < max - 1 && (c = fgetc(f)) != EOF) {
271
      /* Proveravamo da li je procitani karakter slovo */
273 if (isalpha(c))

      /* Ako jeste, smestamo ga u niz - pritom vrsimo
konverziju u mala slova jer program treba da bude
275      neosetljiv na
277      razliku izmedju malih i velikih
279      slova */
      rec[i++] = tolower(c);
281
      else
283
      /* Ako nije, proveravamo da li smo procitali barem
285      jedno
slovo nove rece */
287      /* Ako jesmo prekidamo sa citanjem */
      if (i > 0)
289 break;
291

      /* U suprotnom idemo na sledecu iteraciju */
293 }
295

      /* Dodajemo na rec terminirajucu nulu */
297 rec[i] = '\0';
299
```

```
301      /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */
302      return i > 0 ? 0 : EOF;
303  }
304
305
306
307
308
309  int main(int argc, char **argv)
310  {
311      Cvor * koren = NULL, *max;
312
313      FILE * f;
314
315      char rec[MAX];
316
317
318      /* Proveravamo da li je navedeno ime datoteke prilikom
319       pokretanja programa */
320      if (argc < 2) {
321
322          fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
323
324          exit(EXIT_FAILURE);
325      }
326
327
328
329      /* Otvaramo datoteku iz koje citamo reci */
330      if ((f = fopen(argv[1], "r")) == NULL) {
331
332          fprintf(stderr, "fopen() greska pri otvaranju %s\n",
333                  argv[1]);
334
335          exit(EXIT_FAILURE);
336      }
337
338
339
340
341      /* Ucitavamo reci iz datoteke i smestamo u binarno stablo
342       pretrage.
343      */
344      while (procitaj_rec(f, rec, MAX) != EOF)
345          dodaj_u_stablo(&koren, rec);
346
347
348      /* Posto smo zavrшили sa citanjem reci zatvaramo datoteku */
349      fclose(f);
350
351  }
```

```

353     /* Prikazujemo sve reci iz teksta i brojeve njihovih
        pojavljivanja. */
355     prikazi_stablo(koren);

357     /* Pronalazimo najfrekventniju rec */
359     max = najdi_najfrekventniju_rec(koren);

361     /* Ako takve reci nema... */
363     if (max == NULL)

365         /* Ispisujemo odgovarajuće obavestenje */
        printf("U tekstu nema reci!\n");

367     else

369         /* Inace, ispisujemo broj pojavljivanja reci */
        printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
            max->rec,
373 max->brojac);

375

377     /* Oslobadjamo dinamicki alociran prostor za stablo */
        oslobodi_stablo(&koren);

379

381     /* Završavamo sa programom */
        return 0;

383 }

```

Rešenje 4.16

```

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

6  #define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
8  #define MAX_IME_I_PREZIME 100

10     /* Struktura kojom se opisuje cvor stabla: sadrzi ime i
        prezime,
12     broj telefona i redom pokazivace na levo i
        desno podstablo */
14     typedef struct cvor {

16     char ime_i_prezime[MAX_IME_I_PREZIME];

```

```
18 char telefon[MAX_CIFARA];
20 struct cvor *levo;
22 struct cvor *desno;
24 } Cvor;
26
27 /* Funkcija koja kreira novi cvora stabla */
28 Cvor * napravi_cvor(char *ime_i_prezime, char *telefon)
29 {
30     /* Alociramo memoriju za novi cvor */
31     Cvor * novi_cvor = (Cvor *) malloc(sizeof(Cvor));
32
33     if (novi_cvor == NULL)
34         return NULL;
35
36     /* Inicijalizujemo polja u novom cvoru */
37     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
38
39     strcpy(novi_cvor->telefon, telefon);
40
41     novi_cvor->levo = NULL;
42     novi_cvor->desno = NULL;
43
44     /* Vracamo adresu novog cvora */
45     return novi_cvor;
46 }
47
48 /* Funkcija koja proverava uspesnost kreiranja novog cvora
49    stabla */
50 void proveri_alokaciju(Cvor * novi_cvor)
51 {
52     /* Ukoliko je cvor neuspesno kreiran */
53     if (novi_cvor == NULL) {
54
55         /* Ispisuje se odgovarajuca poruka i prekida
56            izvorsavanje
57            programa */
58         fprintf(stderr, "Malloc greska za novi cvor!\n");
59     }
60 }
```

```
exit(EXIT_FAILURE);
70 }
72 }
74
76
78     /* Funkcija koja dodaje novu osobu i njen broj telefona u
       stablo. */
79 void
80 dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
81 char *telefon)
82 {
83     /* Ako je stablo prazno */
84     if (*adresa_korena == NULL) {
85         /* Kreiramo novi cvor */
86         Cvor * novi = napravi_cvor(ime_i_prezime, telefon);
87         prover_i_alokaciju(novi);
88         /* i proglašavamo ga korenom stabla */
89         *adresa_korena = novi;
90     }
91     return;
92 }
93
94     /* U suprotnom trazimo odgovarajucu poziciju za novi unos */
95     /* Kako pretragu treba vrsiti po imenu i prezimenu,
96     stablo
97     treba da bude pretrazivacko po ovom polju */
98     /* Ako je zadato ime i prezime leksikografski manje od
99     imena i
100     prezimena sadrzanog u korenu, podatke dodajemo
101     u levo
102     podstablo */
103     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
104         < 0)
105     dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
106     telefon);
107     else
108         /* Ako je zadato ime i prezime leksikografski vece od
109         imena
```

```
122     i prezimena sadrzanog u korenu, podatke
        kodajemo u desno
124     podstablo */
        if (strcmp
126             (ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)

dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
128 telefon);
130 }
132
134     /* Funkcija koja oslobadja memoriju zauzetu stablom */
136 void oslobodi_stablo(Cvor ** adresa_korena)
{
138
140     /* Ako je stablo prazno, nepotrebno je oslobadjati
        memoriju */
142     if (*adresa_korena == NULL)

return;
144
146     /* Inace ... */
    /* Oslobadjamo memoriju zauzetu levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);
148
    /* Oslobadjamo memoriju zauzetu desnim podstablom */
    oslobodi_stablo(&(*adresa_korena)->desno);
150
    /* Oslobadjamo memoriju zauzetu korenom */
    free(*adresa_korena);
152
154     /* Proglasavamo stablo praznim */
    *adresa_korena = NULL;
156
158 }
160
162     /* Funkcija koja ispisuje imenik u leksikografskom poretku */
    /* Napomena: ova funkcija nije trazena u zadatku ali se
164     moze
    koristiti za proveru da li je stablo lepo kreirano
    ili ne */
166 void prikazi_stablo(Cvor * koren)
168 {
170
    /* Ako je stablo prazno, završavamo sa ispisom */
    if (koren == NULL)
172
```

```

return;
174
    /* Zbog leksikografskog poretka, prvo ispisujemo podatke
176       iz
    levog podstabla */
178     prikazi_stablo(koren->levo);

180     /* Zatim ispisujemo podatke iz korena */
    printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
182

184     /* I nastavljamo sa ispisom podataka iz desnog podstabla */
    prikazi_stablo(koren->desno);

186 }

188

190     /* Funkcija ucitava sledeci kontakt iz zadate datoteke i
    upisuje
192     ime i prezime i broj telefona u odgovarajuce
    nizove.
194     Maksimalna duzina imena i prezimena odredjena je
    konstantom
196     MAX_IME_PREZIME, a maksimalna duzina broja
    telefona
198     konstantom MAX_CIFARA. Funkcija vraca EOF ako
    nema vise
200     kontakata ili 0 u suprotnom. */
    int procitaj_kontakt(FILE * f, char *ime_i_prezime,
202
    char *telefon)
204 {
206     int c;
208     int i = 0;

210     /* Linije datoteke koje obradjujemo su formata Ime
    Prezime
212     BrojTelefona */
    /* Preskacemo eventualne praznine sa pocetka linije
214     datoteke */
    while ((c = fgetc(f)) != EOF && isspace(c));
216

218     /* Prvo procitano slovo upisujemo u ime i prezime */
    if (!feof(f))

220     ime_i_prezime[i++] = c;

222     /* Naznaka kraja citanja imena i prezimena ce biti pojava
    prve
224     cifre, tako da cemo citanje forsirati sve dok ne

```

```

    naidjemo na
226 cifru. Pri tom cemo voditi racuna da li
    ima dovoljno mesta za
228 smestanje procitanog karaktera i
    da slucajno ne dodjemo do
230 kraja datoteke */
    while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
232
233 if (!isdigit(c))
234 ime_i_prezime[i++] = c;
236
    else if (i > 0)
238
239 break;
240 }
242
    /* Upisujemo terminirajucu nulu na mesto poslednjeg
244 procitanog
    blanko karaktera */
246 ime_i_prezime[--i] = '\0';
248
    /* I pocinjemo sa citanjem broja telefona */
    i = 0;
250
    /* Upisujemo cifru koju smo vec procitali */
252 telefon[i++] = c;
254
    /* I citamo peostale cifre - naznaka kraja ce nam biti
    pojava
256 karaktera cije prisustvo nije dozvoljeno u
    broju telefona */
258 while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
260 if (c == '/' || c == '-' || isdigit(c))
262 telefon[i++] = c;
264
    else
266 break;
268
    /* Upisujemo terminirajucu nulu */
    telefon[i] = '\0';
270
    /* Vracamo 0 ako smo procitali kontakt, EOF u suprotnom */
272 return !feof(f) ? 0 : EOF;
274 }
276
```



```
278      /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
279         prezimenom */
280      Cvor * pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
281      {
282
283         /* Ako je imenik prazan, završavamo sa pretragom */
284         if (koren == NULL)
285
286             return NULL;
287
288         /* Ako je trazeno ime i prezime sadržano u korenu,
289            takodje
290            završavamo sa pretragom */
291         if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
292
293             return koren;
294
295         /* Ako je zadato ime i prezime leksikografski manje od
296            vrednosti u korenu pretragu nastavljamo levo */
297         if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
298
299             return pretrazi_imenik(koren->levo, ime_i_prezime);
300
301         else
302
303             /* u suprotnom, pretragu nastavljamo desno */
304             return pretrazi_imenik(koren->desno, ime_i_prezime);
305
306     }
307
308
309
310     int main(int argc, char **argv)
311     {
312
313         char ime_datoteke[MAX_IME_DATOTEKE];
314
315         Cvor * koren = NULL;
316
317         Cvor * trazeni;
318
319         FILE * f;
320
321         char ime_i_prezime[MAX_IME_I_PREZIME];
322
323         char telefon[MAX_CIFARA];
324
325         char c;
326
327         int i;
328
329     }
```

```
330      /* Ucitavamo ime datoteke i pripremamo je za citanje */
331      printf("Unesite ime datoteke: ");
332
333      scanf("%s", ime_datoteke);
334
335      if ((f = fopen(ime_datoteke, "r")) == NULL) {
336
337          fprintf(stderr, "fopen() greska prilikom otvaranja
338          %s\n", ime_datoteke);
339
340          exit(EXIT_FAILURE);
341      }
342
343
344      /* Ucitavamo podatke iz datoteke i smestamo kontakte u
345      binarno
346      stablo pretrage. */
347      while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
348
349          dodaj_u_stablo(&koren, ime_i_prezime, telefon);
350
351
352      /* Posto smo zavrшили sa citanjem podataka zatvaramo
353      datoteku */
354      fclose(f);
355
356
357      /* Omogucavamo pretragu imenika */
358      while (1) {
359
360          /* Ucitavamo ime i prezime */
361          printf("Unesite ime i prezime: ");
362
363          i = 0;
364
365          while ((c = getchar()) != '\n')
366
367              ime_i_prezime[i++] = c;
368
369          ime_i_prezime[i] = '\0';
370
371          /* Ako je korisnik uneo naznaku za kraj pretrage,
372          obustavljamo funkcionalnost */
373          if (strcmp(ime_i_prezime, "KRAJ") == 0)
374
375              break;
376
377          /* Inace, ispisujemo rezultat pretrage */
378          trazeni = pretrazi_imenik(koren, ime_i_prezime);
379
380      }
```

```

382   if (trazeni == NULL)
383   {
384       printf("Broj nije u imeniku!\n");
385   }
386   else
387   {
388       printf("Broj je: %s \n", trazeni->telefon);
389   }
390
391   /* Oslobadjamo memoriju zauzetu imenikom */
392   oslobodi_stablo(&koren);
393
394   /* Završavamo sa programom */
395   return 0;
396 }

```

Rešenje 4.17

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4
5  #define MAX 51
6
7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8   studenta, ukupan uspsih, uspeh iz matematike, uspeh iz
9   maternjeg jezika i redom pokazivace na levo i desno podstablo
10  */
11  typedef struct cvor_stabla {
12      char ime[MAX];
13      char prezime[MAX];
14      double uspeh;
15      double matematika;
16      double jezik;
17      struct cvor_stabla *levo;
18      struct cvor_stabla *desno;
19  } Cvor;
20
21  /* Funkcija kojom se kreira cvor stabla */
22  Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
23                     double matematika, double jezik)
24  {
25
26      /* Alociramo memoriju za novi cvor */
27      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
28      if (novi == NULL)
29          return NULL;

```

```
30      /* Inicijalizujemo polja strukture */
31      strcpy(novi->ime, ime);
32      strcpy(novi->prezime, prezime);
33      novi->uspeh = uspeh;
34      novi->matematika = matematika;
35      novi->jezik = jezik;
36      novi->levo = NULL;
37      novi->desno = NULL;
38
39      /* Vracamo adresu kreiranog cvora */
40      return novi;
41  }
42
43  /* Funkcija kojom se proverava uspesnost alociranja memorije */
44  void proveri_alokaciju(Cvor * novi_cvor)
45  {
46
47      /* Ako alokacije nije uspesna */
48      if (novi_cvor == NULL) {
49          /* Ispisujemo poruku i prekidamo sa izvršavanjem */
50          fprintf(stderr, "Malloc greska za novi cvor!\n");
51          exit(EXIT_FAILURE);
52      }
53  }
54
55  /* Funkcija kojom se oslobadja memorija zauzeta stablom */
56  void oslobodi_stablo(Cvor ** koren)
57  {
58
59      /* Ako je stablo prazno, nema potrebe za oslobadjanjem
60       * memorije */
61      if (*koren == NULL)
62          return;
63
64      /* oslobadjamo memoriju zauzetu levim podstablom */
65      oslobodi_stablo(&(*koren)->levo);
66
67      /* oslobadjamo memoriju zauzetu desnim podstablom */
68      oslobodi_stablo(&(*koren)->desno);
69
70      /* oslobadjamo memoriju zauzetu korenom */
71      free(*koren);
72
73      /* proglašavamo stablo praznim */
74      *koren = NULL;
75  }
76
77  /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
78  void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
79                     double uspeh, double matematika,
```

```

82         double jezik)
83     {
84
85         /* Ako je stablo prazno */
86         if (*koren == NULL) {
87             /* Kreiramo novi cvor */
88             Cvor *novi =
89                 napravi_cvor(ime, prezime, uspeh, matematika, jezik);
90             prover_i_alokaciju(novi);
91
92             /* I proglašavamo ga korenom stabla */
93             *koren = novi;
94
95             return;
96         }
97
98         /* Inace, dodajemo cvor u stablo tako da bude sortiran po
99            ukupnom broju poena */
100        if (uspeh + matematika + jezik >
101            (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
102            dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
103                           matematika, jezik);
104        else
105            dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
106                           matematika, jezik);
107    }
108
109    /* Funkcija ispisuje sadrzaj stabla - ukoliko je vrednost
110       argumenta polozili jednaka 0 ispisuju se informacije o
111       ucenicima koji nisu polozili prijemni, a ako je vrednost
112       argumenta razlicita od nule, ispisuju se informacije o
113       ucenicima koji su polozili prijemni */
114    void stampaj(Cvor * koren, int polozili)
115    {
116
117        /* Stablo je prazno - prekidamo sa ispisom */
118        if (koren == NULL)
119            return;
120
121        /* Stampamo informacije iz levog podstabla */
122        stampaj(koren->levo, polozili);
123
124        /* Stampamo informacije iz korenog cvora */
125        if (polozili && koren->matematika + koren->jezik >= 10)
126            printf("%s %s %.11f %.11f %.11f %.11f\n", koren->ime,
127                koren->prezime, koren->uspeh, koren->matematika,
128                koren->jezik,
129                koren->uspeh + koren->matematika + koren->jezik);
130        else if (!polozili && koren->matematika + koren->jezik < 10)
131            printf("%s %s %.11f %.11f %.11f %.11f\n", koren->ime,
132                koren->prezime, koren->uspeh, koren->matematika,

```

```
134         koren->jezik,
135         koren->uspeh + koren->matematika + koren->jezik);
136
137     /* Stampamo informacije iz desnog podstabla */
138     stampaj(koren->desno, položili);
139 }
140
141 /* Funkcija koja određuje koliko studenata nije položilo
142    prijemni ispit */
143 int nisu_položili(Cvor * koren)
144 {
145
146     /* Ako je stablo prazno, broj onih koji nisu položili je 0 */
147     if (koren == NULL)
148         return 0;
149
150     /* Pretragu vršimo i u levom i u desnom podstablu - ako uslov
151        za polaganje nije ispunjen za koreni cvor, broj studenata
152        uvećavamo za 1 */
153     if (koren->matematika + koren->jezik < 10)
154         return 1 + nisu_položili(koren->levo) +
155             nisu_položili(koren->desno);
156
157     return nisu_položili(koren->levo) +
158         nisu_položili(koren->desno);
159 }
160
161 int main(int argc, char **argv)
162 {
163
164     FILE *in;
165     Cvor *koren;
166     char ime[MAX], prezime[MAX];
167     double uspeh, matematika, jezik;
168
169     /* Otvaramo datoteku sa rezultatima sa prijemnog za citanje */
170     in = fopen("prijemni.txt", "r");
171     if (in == NULL) {
172         fprintf(stderr, "Greska prilikom citanja podataka!\n");
173         exit(EXIT_FAILURE);
174     }
175
176     /* Citamo podatke i dodajemo ih u stablo */
177     koren = NULL;
178     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
179                  &matematika, &jezik) != EOF) {
180         dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
181                      jezik);
182     }
183
184     /* Zatvaramo datoteku */
```

```

186     fclose(in);

188     /* Stampamo prvo podatke o ucenicima koji su polozili prijemni
        */
190     stampaj(koren, 1);

192     /* Liniju iscrivamo samo ako postoje ucenici koji nisu
        polozili prijemni */
194     if (nisu_polozili(koren) != 0)
        printf("-----\n");

196     /* Stampamo podatke o ucenicima koji nisu polozili prijemni */
198     stampaj(koren, 0);

200     /* Oslobadjamo memoriju zauzetu stablom */
        oslobodi_stablo(&koren);

202     /* Završavamo sa programom */
204     return 0;

206 }

```

Rešenje 4.18

```

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

5  #define MAX_NISKA 51
6  #define MAX_DATUM 3

7
8  /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i
9     prezime osobe, dan, mesec i godinu rođenja i redom
10     pokazuje na levo i desno podstablo */
11 typedef struct cvor_stabla {
12     char ime[MAX_NISKA];
13     char prezime[MAX_NISKA];
14     int dan;
15     int mesec;
16     int godina;
17     struct cvor_stabla *levo;
18     struct cvor_stabla *desno;
19 } Cvor;

21 /* Funkcija koja kreira novi cvor */
22 Cvor *napravi_cvor(char ime[], char prezime[], int dan,
23                   int mesec, int godina)
24 {
25
26     /* Alociramo memoriju */
27     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

```

```
    if (novi == NULL)
29         return NULL;

31     /* Inicijalizujemo polja strukture */
    strcpy(novi->ime, ime);
33     strcpy(novi->prezime, prezime);
    novi->dan = dan;
35     novi->mesec = mesec;
    novi->godina = godina;
37     novi->levo = NULL;
    novi->desno = NULL;

39     /* Vracamo adresu novog cvora */
41     return novi;
}

43

/* Funkcija koja proverava uspesnost alokacije */
45 void proveri_alokaciju(Cvor * novi_cvor)
{
47
    /* Ako memorija nije uspesno alocirana */
49     if (novi_cvor == NULL) {
        /* Ispisujemo poruku i prekidamo izvršavanje programa */
51         fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
53     }
}

55

/* Funkcija koja oslobadja memoriju zauzetu stablom */
57 void oslobodi_stablo(Cvor ** koren)
{
59
    /* Stablo je prazno */
61     if (*koren == NULL)
        return;

63
    /* Oslobadjamo memoriju zauzetu levim podstablom (ako postoji)
65     */
    if ((*koren)->levo)
67         oslobodi_stablo(&(*koren)->levo);

69
    /* Oslobadjamo memoriju zauzetu desnim podstablom (ako
    postoji) */
71     if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);

73
    /* Oslobadjamo memoriju zauzetu korenom */
75     free(*koren);

77
    /* Proglasavamo stablo praznim */
    *koren = NULL;
79 }
```



```

81  /* Funkcija koja dodaje novi cvor u stablo - stablo treba da
      bude uredjeno po datumu */
83  void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
      int dan, int mesec, int godina)
85  {

87      /* Ako je stablo prazno */
      if (*koren == NULL) {

89          /* Kreiramo novi cvor */
          Cvor *novi_cvor =
          napravi_cvor(ime, prezime, dan, mesec, godina);
          proveri_alokaciju(novi_cvor);

93          /* I proglašavamo ga korenom */
          *koren = novi_cvor;

97          return;
99      }

101     /* Kako se ne unosi godina za pretragu, stablo uredjujemo samo
          po mesecu (i danu u okviru istog meseca) */
103     if (mesec < (*koren)->mesec)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
105                     godina);
        else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
107         dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
                        godina);
        else
109         dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
                        godina);
111     }

113     /* Funkcija vrši pretragu stabla i vraća cvor sa traženim
        datumom (null ako takav ne postoji). u promenljivu pom ce
        biti smesten prvi datum (dan i mesec) veci od traženog datuma
        (null ako takav ne postoji)

117     */

119     Cvor *pretrazi(Cvor * koren, int dan, int mesec)
121     {

123         /* Stablo je prazno, obustavljamo pretragu */
        if (koren == NULL)
            return NULL;

125         /* Nasli smo traženi datum u stablu */
        if (koren->dan == dan && koren->mesec == mesec)
            return koren;

127         /* Ako je mesec traženog datuma manji od meseca sadržanog u
            korenu ili ako su meseci isti ali je dan traženog datuma
129
131

```

```

133     manji od aktuelnog datuma, pretražujemo levo podstablo -
135     pre toga svakako proveravamo da li leva grana postoji - ako
137     ne postoji treba da vratimo prvi sledeći, a to je bas
139     vrednost uocenog korena */
141     if (mesec < koren->mesec
143         || (mesec == koren->mesec && dan < koren->dan)) {
145         if (koren->levo == NULL)
147             return koren;
149         else
151             return pretrazi(koren->levo, dan, mesec);
153     }
155
157     /* inace, nastavljamo pretragu u desnom delu */
159     return pretrazi(koren->desno, dan, mesec);
161 }
163
165 int main(int argc, char **argv)
167 {
169     FILE *in;
171     Cvor *koren;
173     Cvor *slavljenik;
175     char ime[MAX_NISKA], prezime[MAX_NISKA];
177     int dan, mesec, godina;
179
181     /* Proveravamo da li je zadato ime ulazne datoteke */
183     if (argc < 2) {
185         /* Ako nije, ispisujemo poruku i prekidamo sa izvršavanjem
187         programa */
189         printf("Nedostaje ime ulazne datoteke!\n");
191         return 0;
193     }
195
197     /* Inace, pripremamo datoteku za citanje */
199     in = fopen(argv[1], "r");
201     if (in == NULL) {
203         fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
205         exit(EXIT_FAILURE);
207     }
209
211     /* I popunjavamo podacima stablo */
213     koren = NULL;
215     while (fscanf
217         (in, "%s %s %d.%d.%d.", ime, prezime, &dan, &mesec,
219         &godina) != EOF)
221         dodaj_u_stablo(&koren, ime, prezime, dan, mesec, godina);
223
225     /* I zatvaramo datoteku */
227     fclose(in);
229
231     /* Omogucavamo pretragu podataka */
233     while (1) {

```

```

185  /* Ucitavamo novi datum */
    printf("Unesite datum: ");
187  if (scanf("%d.%d.", &dan, &mesec) == EOF)
        break;
189
    /* Pretražujemo stablo */
191  slavljenik = pretrazi(koren, dan, mesec);

193  /* Ispisujemo pronadjene podatke */
    if (slavljenik == NULL) {
195      printf("Nema podataka o ovim ni o sledecem rođendanu.\n");
        continue;
197  }

199  /* Slučaj kada smo pronasli prave podatke */
    if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
201      printf("Slavljenik: %s %s\n", slavljenik->ime,
                slavljenik->prezime);
203      continue;
    }

205  /* Slučaj kada smo pronasli podatke o prvom sledecem
207      rođendanu */
    printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
209      slavljenik->prezime, slavljenik->dan,
        slavljenik->mesec);
211 }

213 /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);
215
    /* Prekidamo sa izvršavanjem programa */
217 return 0;
}

```

Rešenje 4.19

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1

4      /* Struktura kojom se predstavlja cvor binarnog
        pretraživackog stabla
6      */
        typedef struct cvor {
8
10         int broj;
        struct cvor *levo, *desno;
12     } Cvor;

```

```
14
16      /* Funkcija koja alocira memoriju za novi cvor stabla,
18         inicijalizuje polja strukture i vraća pokazivac na novi
         cvor */
20      Cvor * napravi_cvor(int broj);
22
23      /* Funkcija koja proverava uspesnost kreiranja novog cvora
         stabla. */
24      void proveri_alokaciju(Cvor * novi_cvor);
26
27      /* Funkcija koja dodaje zadati broj u stablo */
28      void dodaj_u_stablo(Cvor ** adresa_korena, int broj);
30
31      /* Funkcija koja proverava da li se zadati broj nalazi u
         stablu */
32      Cvor * pretrazi_stablo(Cvor * koren, int broj);
34
35      /* Funkcija koja pronalazi cvor koji sadrži najmanju
         vrednost
36      u stablu */
37      Cvor * pronadji_najmanji(Cvor * koren);
40
41      /* Funkcija koja pronalazi cvor koji sadrži najveći vrednost
         u stablu
42      */
43      Cvor * pronadji_najveci(Cvor * koren);
46
47      /* Funkcija koja briše cvor stabla koji sadrži zadati broj */
48      void obrisi_element(Cvor ** adresa_korena, int broj);
50
51      /* Funkcija koja ispisuje sadržaj stabla u infiksnoj
         notaciji (levo
52      podstablo - koren - desno podstablo) */
53      void prikazi_stablo(Cvor * koren);
56
57      /* Funkcija koja oslobadja memoriju zauzetu stablom */
58      void oslobodi_stablo(Cvor ** adresa_korena);
60
61      #endif                                     /*
         */
```

```

#include <stdio.h>
#include <stdlib.h>
#include "stabla.h"
/* Funkcija kojom se kreira novi cvor stabla koji sadrzi zadatu
   vrednost */
Cvor *napravi_cvor(int broj)
{
    /* Alociramo memoriju za novi cvor */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;
    /* Inicijalizujemo polja cvora */
    novi->broj = broj;
    novi->levo = NULL;
    novi->desno = NULL;
    /* Vracamo adresu novog cvora */
    return novi;
}

/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
void prover_i_alokaciju(Cvor * novi_cvor)
{
    /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvršavanje
           programa */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
    }
}

/* Funkcija koja dodaje novi broj u stablo. */
void dodaj_u_stablo(Cvor ** koren, int broj)
{
    /* Ako je stablo prazno */
    if (*koren == NULL) {
        /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
        /* i proglašavamo ga korenom stabla */
        *koren = novi;
        return;
    }
    /* U suprotnom trazimo odgovarajucu poziciju za novi broj */
    /* Ako je broj manji od vrednosti sadrzane u korenu, ubacujemo
       ga u levo podstablo */
    if (broj < (*koren)->broj)
        dodaj_u_stablo(&(*koren)->levo, broj);
    else
        /* Inace, ubacujemo broj u desno podstablo */
        dodaj_u_stablo(&(*koren)->desno, broj);
}

```

```
    }
54
    /* Funkcija koja oslobadja memoriju zauzetu stablom */
56 void oslobodi_stablo(Cvor ** koren)
    {
58     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
        if (*koren == NULL)
60         return;
        /* Inace ... */
62     /* Oslobadjamo memoriju zauzetu levom podstablom */
        if ((*koren)->levo)
64         oslobodi_stablo(&(*koren)->levo);
        /* Oslobadjamo memoriju zauzetu desnom podstablom */
66     if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);
68     /* Oslobadjamo memoriju zauzetu korenom */
        free(*koren);
70     /* Proglasavamo stablo praznim */
        *koren = NULL;
72 }

74 Cvor *pronadji_najmanji(Cvor * koren)
    {
76     /* ako je stablo prazno, prekidamo pretragu */
        if (koren == NULL)
78         return NULL;
        /* vrednosti koje su manje od vrednosti u korenu stabla nalaze
80         se levo od njega */
        /* ako je koren cvor koji nema levo podstablo, onda on sadrzi
82         najmanju vrednost */
        if (koren->levo == NULL)
84         return koren;
        /* inace, pretragu treba nastaviti u levom podstablu */
86     return pronadji_najmanji(koren->levo);
    }

88
89 Cvor *pronadji_najveci(Cvor * koren)
90 {
92     /* ako je stablo prazno, prekidamo pretragu */
        if (koren == NULL)
94         return NULL;
        /* vrednosti koje su vece od vrednosti u korenu stabla nalaze
96         se desno od njega */
        /* ako je koren cvor koji nema desno podstablo, onda on sadrzi
98         najveću vrednost */
        if (koren->desno == NULL)
100         return koren;
        /* inace, pretragu treba nastaviti u desnom podstablu */
        return pronadji_najveci(koren->desno);
102 }

104 /* Funkcija brise element iz stabla ciji je broj upravo jednak
```

```

106     broju n. Funkcija azurira koren stabla u pozivajucoj
        funkciji, jer u ovoj funkciji koren moze biti promenjen u
        funkciji. */
108 void obrisi_element(Cvor ** adresa_korena, int n)
    {
110     Cvor *pomocni = NULL;
        /* Izlaz iz rekurzije: ako je stablo prazno, nema sta da se
112         brise */
        if (*adresa_korena == NULL)
114             return;
        /* Ako je vrednost broja veca od vrednosti u korenu stabla,
116         tada se broj eventualno nalazi u desnom podstablu, pa treba
        rekurzivno primeniti postupak na desno podstablo. Koren
118         ovako modifikovanog stabla je nepromenjen. */
        if ((*adresa_korena)->broj < n) {
120             obrisi_element(&(*adresa_korena)->desno, n);
            return;
122         }
        /* Ako je vrednost broja manja od vrednosti korena, tada se
124         broj eventualno nalazi u levom podstablu, pa treba
        rekurzivno primeniti postupak na levo podstablo. Koren
126         ovako modifikovanog stabla je nepromenjen. */
        if ((*adresa_korena)->broj > n) {
128             obrisi_element(&(*adresa_korena)->levo, n);
            return;
130         }
        /* Slede podslucajevi vezani za slucaj kada je vrednost u
132         korenu jednaka broju koji se brise (tj. slucaj kada treba
        obrisati koren) */
134         /* Ako koren nema sinova, tada se on prosto brise, i rezultat
        je prazno stablo (vracamo NULL) */
136         if ((*adresa_korena)->levo == NULL
            && (*adresa_korena)->desno == NULL) {
138             free(*adresa_korena);
            *adresa_korena = NULL;
140             return;
        }
142         /* Ako koren ima samo levog sina, tada se brisanje vrshi tako
        sto obrisemo koren, a novi koren postaje levo sin */
144         if ((*adresa_korena)->levo != NULL
            && (*adresa_korena)->desno == NULL) {
146             pomocni = (*adresa_korena)->levo;
            free(*adresa_korena);
148             *adresa_korena = pomocni;
            return;
150         }
        /* Ako koren ima samo desnog sina, tada se brisanje vrshi tako
152         sto obrisemo koren, a novi koren postaje desno sin */
        if ((*adresa_korena)->desno != NULL
            && (*adresa_korena)->levo == NULL) {
154             pomocni = (*adresa_korena)->desno;
            free(*adresa_korena);
156             *adresa_korena = pomocni;

```

```

    *adresa_korena = pomocni;
158     return;
    }
160     /* Slučaj kada koren ima oba sina. Tada se brisanje vrši na
        sledeći način: - najpre se potraži sledbenik korena (u
162         smislu poretka) u stablu. To je upravo po vrednosti
        najmanji čvor u desnom podstablu. On se može pronaći npr.
164         funkcijom pronadji_najmanji(). - Nakon toga se u koren
        smesti vrednost tog čvora, a u taj čvor se smesti vrednost
166         korena (tj. broj koji se briše). - Onda se prosto
        rekursivno pozove funkcija za brisanje na desno podstablo.
168         S obzirom da u njemu treba obrisati najmanji element, a on
        definitivno ima najviše jednog potomka, jasno je da će
170         njegovo brisanje biti obavljeno na jedan od jednostavnijih
        načina koji su gore opisani. */
172     pomocni = pronadji_najmanji((*adresa_korena)->desno);
    (*adresa_korena)->broj = pomocni->broj;
174     pomocni->broj = n;
    obrisi_element(&(*adresa_korena)->desno, n);
176 }

178 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje
        elemente u rastućem poretku) */
180 void prikazi_stablo(Cvor * koren)
    {
182     /* izlaz iz rekurziije */
    if (koren == NULL)
184         return;
    prikazi_stablo(koren->levo);
186     printf("%d ", koren->broj);
    prikazi_stablo(koren->desno);
188 }

190 Cvor *pretrazi_stablo(Cvor * koren, int broj)
    {
192     /* ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
        */
    if (koren == NULL)
194         return NULL;
    /* ako je tražena vrednost sadržana u korenu */
    if (koren->broj == broj) {
196         /* prekidamo pretragu */
        return koren;
200     }
    /* inače, ako je broj manji od vrednosti sadržane u korenu */
    if (broj < koren->broj)
202         /* pretragu nastavljamo u levom podstablu */
        return pretrazi_stablo(koren->levo, broj);
    else
204         /* u suprotnom, pretragu nastavljamo u desnom podstablu */
        return pretrazi_stablo(koren->desno, broj);
206 }
208 }

```



```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima - pogledati uvodni
   zadatak ove glave */
5
6  #include "stabla.h"
7
8
9
10 /* Funkcija koja proverava da li su dva stabla koja sadrže cele
   brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
11
12 int identitet(Cvor * koren1, Cvor * koren2)
13 {
14     /* Ako su oba stabla prazna, jednaka su */
15     if (koren1 == NULL && koren2 == NULL)
16         return 1;
17
18     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
       jednaka */
19     if (koren1 == NULL || koren2 == NULL)
20         return 0;
21
22     /* Ako su oba stabla neprazna i u korenu se nalaze različite
       vrednosti, možemo da zaključimo da se razlikuju */
23     if (koren1->broj != koren2->broj)
24         return 0;
25
26     /* Inace, proveravamo da li vazi i jednakost u levih
       podstabala i desnih podstabala */
27     return (identitet(koren1->levo, koren2->levo)
28             && identitet(koren1->desno, koren2->desno));
29 }
30
31
32
33 int main()
34 {
35
36     int broj;
37     Cvor *koren1, *koren2;
38
39
40     koren1 = NULL;
41     /* učitavamo elemente prvog stabla */
42     printf("Prvo stablo: ");
43     scanf("%d", &broj);
44     while (broj != 0) {
45         dodaj_u_stablo(&koren1, broj);
46         scanf("%d", &broj);
47     }
48
49     koren2 = NULL;
50     /* učitavamo elemente drugog stabla */
51     printf("Drugog stablo: ");
```

```
52  scanf("%d", &broj);
    while (broj != 0) {
54      dodaj_u_stablo(&koren2, broj);
      scanf("%d", &broj);
56    }

58    /* pozivamo funkciju koja ispituje identitet stabala */
    if (identitet(koren1, koren2))
60      printf("Stabla jesu identicna.\n");
    else
62      printf("Stabla nisu identicna.\n");

64    /* oslobadjamo memoriju zauzetu stablima */
    oslobodi_stablo(&koren1);
66    oslobodi_stablo(&koren2);

68    /* završavamo sa radom programa */
    return 0;
70 }
```

Rešenje 4.20

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

4     /* Uklucujemo biblioteku za rad sa stablima */
    #include "stabla.h"

6     /* Funkcija kreira novo stablo identicno stablu koje je
       dato
8     korenom. */
10 void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
    {
12
14     /* Izlaz iz rekurzije: ako je stablo prazno nema sta da se
       kopira */
        if (koren == NULL) {
16
18         *duplikat = NULL;
19
20         return;
21     }

22     /* Dupliramo koren stabla i postavljamo ga da bude koren
       novog
24     stabla */
        *duplikat = napravi_cvor(koren->broj);
26
28     prover_i_alokaciju(*duplikat);
```

```

30      /* Rekurzivno dupliramo levo podstablo i njegovu adresu
31      cuvamo
32      u pokazivacu na levo podstablo korena
33      duplikata. */
34      kopiraj_stablo(koren->levo, &(*duplikat)->levo);
35
36
37      /* Rekurzivno dupliramo desno podstablo i njegovu
38      adresu
39      cuvamo u pokazivacu na desno podstablo korena
40      duplikata. */
41      kopiraj_stablo(koren->desno, &(*duplikat)->desno);
42
43  }
44
45
46
47      /* Funkcija izracunava uniju dva stabla - rezultuje stablo
48      se
49      dobija modifikacijom prvog stabla */
50  void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
51  {
52
53      /* Ako drugo stablo nije prazno */
54      if (koren2 != NULL) {
55
56          /* dodajemo njegov koren u prvo stablo */
57          dodaj_u_stablo(adresa_korena1, koren2->broj);
58
59          /* rekurzivno racunamo uniju levog i desnog podstabla
60          drugog
61          stabla sa prvim stablom */
62          kreiraj_uniju(adresa_korena1, koren2->levo);
63
64          kreiraj_uniju(adresa_korena1, koren2->desno);
65
66      }
67
68  }
69
70
71
72      /* Funkcija izracunava presek dva stabla - rezultuje
73      stablo se
74      dobija modifikacijom prvog stabla */
75  void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
76  {
77
78      /* Ako je prvo stablo prazno, tada je i rezultat prazno
79      stablo */
80      if (*adresa_korena1 == NULL)

```

```
82     return;
84
86     /* Kreiramo presek levog i desnog podstabla sa drugim
87        stablom, tj.
88        iz levog i desnog podstabla prvog stabla
89        brisemo sve one elemente
90        koji ne postoje u drugom
91        stablu */
92     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
94     kreiraj_presek(&(*adresa_korena1)->desno, koren2);
96
97     /* Ako se koren prvog stabla ne nalazi u drugom stablu
98        tada ga
99        uklanjamo iz prvog stabla */
100    if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) ==
        NULL)
102        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
104    }
106
108
109    /* Funkcija izracunava razliku dva stabla - rezultujuce
110       stablo se
111       dobija modifikacijom prvog stabla */
112    void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
113    {
114
115        /* Ako je prvo stablo prazno, tada je i rezultat prazno
116           stablo */
117        if (*adresa_korena1 == NULL)
118            return;
119
120        /* Kreiramo razliku levog i desnog podstabla sa drugim
121           stablom, tj.
122           iz levog i desnog podstabla prvog stabla
123           brisemo sve one elemente
124           koji postoje i u drugom
125           stablu */
126        kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
128        kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
130
131        /* Ako se koren prvog stabla nalazi i u drugom stablu
132           tada ga uklanjamo iz prvog stabla */
```

```
134         if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) !=
135             NULL)
136     obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
137 }
138
139
140
141
142 int main()
143 {
144
145     Cvor * koren1;
146
147     Cvor * koren2;
148
149     Cvor * pomocni = NULL;
150
151     int n;
152
153
154
155     /* Ucitavamo elemente prvog stabla: */
156     koren1 = NULL;
157
158     printf("Prvo stablo: ");
159
160     while (scanf("%d", &n) != EOF) {
161
162         dodaj_u_stablo(&koren1, n);
163
164     }
165
166
167     /* Ucitavamo elemente drugog stabla: */
168     koren2 = NULL;
169
170     printf("Drugo stablo: ");
171
172     while (scanf("%d", &n) != EOF) {
173
174         dodaj_u_stablo(&koren2, n);
175
176     }
177
178
179
180     /* Kreiramo uniju stabala: prvo napravimo kopiju prvog
181         stabla kako
182         bi mogli da ga iskoristimo i za preostale
183         operacije */
184     kopiraj_stablo(koren1, &pomocni);
```

```
186 kreiraj_uniju(&pomocni, koren2);
188 printf("Unija: ");
190 prikazi_stablo(pomocni);
192 putchar('\n');
194     /* Oslobadjamo stablo za rezultatom operacije */
196     oslobodi_stablo(&pomocni);
198
199     /* Kreiramo presek stabala: prvo napravimo kopiju prvog
200        stabla kako
201        bi mogli da ga iskoristimo i za preostale
202        operacije;
203        */
204     kopiraj_stablo(koren1, &pomocni);
206 kreiraj_presek(&pomocni, koren2);
208 printf("Presek: ");
210 prikazi_stablo(pomocni);
212 putchar('\n');
214     /* Oslobadjamo stablo za rezultatom operacije */
216     oslobodi_stablo(&pomocni);
218
219     /* Kreiramo razliku stabala: prvo napravimo
220        kopiju prvog
221        stabla kako
222        bi mogli da ga iskoristimo i za preostale
223        operacije;
224        */
225     kopiraj_stablo(koren1, &pomocni);
226 kreiraj_razliku(&pomocni, koren2);
228 printf("Razlika: ");
230 prikazi_stablo(pomocni);
232 putchar('\n');
234     /* Oslobadjamo stablo za rezultatom operacije */
236     oslobodi_stablo(&pomocni);
```

```

238
240     /* Oslobadjamo i polazna stabla */
    oslobodi_stablo(&koren2);
242
    oslobodi_stablo(&koren1);
244
246     /* Završavamo sa programom */
    return 0;
248
}
```

Rešenje 4.21

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
7
   #define MAX 50
9
   /* Funkcija koja obilazi stablo sa leva na desno i smesta
      vrednosti cvorova u niz. Povratna vrednost funkcije je broj
11  vrednosti koje su smestene u niz. */
13  int kreiraj_niz(Cvor * koren, int a[])
   {
15     int r, s;
17
   /* Stablo je prazno - u niz je smesteno 0 elemenata */
   if (koren == NULL)
19     return 0;
21
   /* Dodajemo u niz elemente iz levog podstabla */
   r = kreiraj_niz(koren->levo, a);
23
   /* Tekuca vrednost promenljive r je broj elemenata koji su
      upisani u niz i na osnovu nje mozemo odrediti indeks novog
25  elementa */
27
   /* Smestamo vrednost iz korena */
29  a[r] = koren->broj;
31
   /* Dodajemo elemente iz desnog podstabla */
   s = kreiraj_niz(koren->desno, a + r + 1);
33
   /* Racunamo indeks na koji treba smestiti naredni element */
35  return r + s + 1;
}
```

```
37
39 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
41    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva
    u desno.
43
44    Ovaj nacin sortiranja primer sortiranja koje nije "u mestu "
45    kao sto je to slucaj sa ostalim prethodno opisanim
46    algoritmima sortiranja, jer se sortiranje vrši u pomocnoj
47    dinamičkoj strukturi, a ne razmenom elemenata niza. */
48
49 void sortiraj(int a[], int n)
50 {
51     int i;
52     Cvor *koren;
53
54     /* Kreiramo stablo smestanjem elemenata iz niza u stablo */
55     koren = NULL;
56     for (i = 0; i < n; i++)
57         dodaj_u_stablo(&koren, a[i]);
58
59     /* Infiksni obilaskom stabla elemente iz stabla prepisujemo u
60        niz a */
61     kreiraj_niz(koren, a);
62
63     /* Vise nam stablo nije potrebno i oslobadjamo memoriju */
64     oslobodi_stablo(&koren);
65 }
66
67 int main()
68 {
69     int a[MAX];
70     int n, i;
71
72     /* Ucitavamo dimenziju i elemente niza */
73     printf("n: ");
74     scanf("%d", &n);
75     if (n < 0 || n > MAX) {
76         printf("Greska: pogresna dimenzija niza!\n");
77         return 0;
78     }
79
80     printf("a: ");
81     for (i = 0; i < n; i++)
82         scanf("%d", &a[i]);
83
84     /* Pozivamo funkciju za sortiranje */
85     sortiraj(a, n);
86
87     /* Ispisujemo rezultat */
88     for (i = 0; i < n; i++)
```



```

89     printf("%d ", a[i]);
    printf("\n");
91
    /* Prekidamo sa programom */
93     return 0;
}

```

Rešenje 4.22

```

1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
7
   /* a) Funkcija koja izracunava broj cvorova stabla */
   int broj_cvorova(Cvor * koren)
9  {
11
   /* Ako je stablo prazno, broj cvorova je nula */
   if (koren == NULL)
13     return 0;
15
   /* U suprotnom je broj cvorova stabla jednak zbiru broja
      cvorova u levom podstablu i broja cvorova u desnom
17     podstablu - 1 dodajemo zato sto treba racunati i koren */
   return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) +
19     1;
   }
21
   /* b) Funkcija koja izracunava broj listova stabla */
   int broj_listova(Cvor * koren)
23  {
25
   /* Ako je stablo prazno, broj listova je nula */
   if (koren == NULL)
27     return 0;
29
   /* Proveravamo da li je tekuci cvor list */
   if (koren->levo == NULL && koren->desno == NULL)
31     /* i ako jeste vratimo 1 - to ce kasnije zbog rekurzivnih
       poziva uvecati broj listova za 1 */
       return 1;
33
   /* U suprotnom prebrojavamo listove koje se nalaze u
       podstablima */
35     return broj_listova(koren->levo) + broj_listova(koren->desno);
37
   }
39
   /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
   void pozitivni_listovi(Cvor * koren)
41

```

```
43 {
45     /* Slučaj kada je stablo prazno */
46     if (koren == NULL)
47         return;
49     /* Ako je cvor list i sadrži pozitivnu vrednost */
50     if (koren->levo == NULL && koren->desno == NULL
51         && koren->broj > 0)
52         /* Stampamo ga */
53         printf("%d ", koren->broj);
55     /* Nastavljamo sa stampanjem pozitivnih listova u podstablama */
56     pozitivni_listovi(koren->levo);
57     pozitivni_listovi(koren->desno);
58 }
59
61 /* d) Funkcija koja izračunava zbir cvorova stabla */
62 int zbir_cvorova(Cvor * koren)
63 {
65     /* Ako je stablo prazno, zbir cvorova je 0 */
66     if (koren == NULL)
67         return 0;
69     /* Inace, zbir cvorova stabla izračunavamo kao zbir korena i
70      svih elemenata u podstablama */
71     return koren->broj + zbir_cvorova(koren->levo) +
72            zbir_cvorova(koren->desno);
73 }
75 /* e) Funkcija koja izračunava najveći element stabla. */
76 Cvor *najveci_element(Cvor * koren)
77 {
79     /* Ako je stablo prazno, obustavljamo pretragu */
80     if (koren == NULL)
81         return NULL;
83     /* Zbog prirode pretraživackog stabla, sigurni smo da su
84      vrednosti veće od korena u desnom podstablu */
85
86     /* Ako desnog podstabla nema */
87     if (koren->desno == NULL)
88         /* Najveća vrednost je koren */
89         return koren;
91     /* Inace, najveću vrednost tražimo još desno */
92     return najveci_element(koren->desno);
93 }
```

```
95 /* f) Funkcija koja izracunava dubinu stabla */
96 int dubina_stabla(Cvor * koren)
97 {
98
99     /* Dubina praznog stabla je 0 */
100    if (koren == NULL)
101        return 0;
102
103    /* Izracunavamo dubinu levog podstabla */
104    int dubina_levo = dubina_stabla(koren->levo);
105
106    /* Izracunavamo dubinu desnog podstabla */
107    int dubina_desno = dubina_stabla(koren->desno);
108
109    /* dubina stabla odgovara vecoj od dubina podstabala - 1
110       dodajemo jer racunamo i koren */
111    return dubina_levo >
112           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
113 }
114
115 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou */
116 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
117 {
118     /* ideja je da ste spustamo kroz stablo sve dok ne stignemo do
119        trazenog nivoa */
120
121     /* Ako nema vise cvorova, ne mozemo da se spustamo niz stablo */
122     if (koren == NULL)
123         return 0;
124
125     /* Ako smo stigli do trazenog nivoa, vracamo 1 - to ce kasnije
126        zbog rekurzivnih poziva uvecati broj pojavljivanja za 1 */
127     if (i == 0)
128         return 1;
129
130     /* inace, spustamo se jedan nivo nize i u levom i u desnom
131        postablu */
132     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
133        + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
134 }
135
136
137 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
138 void ispis_nivo(Cvor * koren, int i)
139 {
140     /* ideja je slicna ideji iz prethodne funkcije */
141
142     /* nema vise cvorova, ne mozemo da se spustamo kroz stablo */
143     if (koren == NULL)
144         return;
145
146     /* ako smo na trazenom nivou - ispisujemo vrednost */
```

```
147     if (i == 0) {
148         printf("%d ", koren->broj);
149         return;
150     }
151     /* inace, spustamo se jedan nivo nize i u levom i u desnom
152        podstablu */
153     ispis_nivo(koren->levo, i - 1);
154     ispis_nivo(koren->desno, i - 1);
155 }

157 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
158     nivou stabla */
159 Cvor *max_nivo(Cvor * koren, int i)
160 {
161     /* Ako je stablo prazno, obustavljamo pretragu */
162     if (koren == NULL)
163         return NULL;
164
165     /* Ako smo na trazenom nivou, takodje prekidamo pretragu */
166     if (i == 0)
167         return koren;
168
169     /* Pronalazimo maksimum sa i-tog nivoa levog podstabla */
170     Cvor *a = max_nivo(koren->levo, i - 1);
171
172     /* Pronalazimo maksimum sa i-tog nivoa desnog podstabla */
173     Cvor *b = max_nivo(koren->desno, i - 1);
174
175     /* Trazimo i vracamo maksimum izracunatih vrednosti */
176     if (a == NULL && b == NULL)
177         return NULL;
178     if (a == NULL)
179         return b;
180     if (b == NULL)
181         return a;
182     return a->broj > b->broj ? a : b;
183 }

185 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
186 int zbir_nivo(Cvor * koren, int i)
187 {
188     /* Ako je stablo prazno, zbir je nula */
189     if (koren == NULL)
190         return 0;
191
192     /* Ako smo na trazenom nivou, vracamo vrednost */
193     if (i == 0)
194         return koren->broj;
195
196     /* Inace, spustamo se jedan nivo nize i trazimo sume iz levog
```

```

199     i desnog podstabla */
200     return zbir_nivo(koren->levo, i - 1) + zbir_nivo(koren->desno,
201                                                       i - 1);
202 }
203
204 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje
205     su manje ili jednake od date vrednosti x */
206 int suma(Cvor * koren, int x)
207 {
208     /* Ako je stablo prazno, zbir je nula */
209     if (koren == NULL)
210         return 0;
211
212     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
213         prirode pretrazivackog stabla treba obici i levo i desno
214         podstablo */
215     if (koren->broj < x)
216         return koren->broj + suma(koren->levo,
217                                   x) + suma(koren->desno, x);
218
219     /* Inace, racunamo samo sumu vrednosti iz levog podstabla jer
220         medju njima jedino moze biti onih koje zadovoljavaju uslov */
221     return suma(koren->levo, x);
222 }
223
224 int main(int argc, char **argv)
225 {
226     /* Analiziramo argumente komandne linije */
227     if (argc != 3) {
228         fprintf(stderr, "Greska! Program se poziva sa: ./a.out nivo
229     broj_za_pretragu\n");
230         exit(EXIT_FAILURE);
231     }
232     int i = atoi(argv[1]);
233     int x = atoi(argv[2]);
234
235     /* Kreiramo stablo */
236     Cvor *koren = NULL;
237     int broj;
238     while (scanf("%d", &broj) != EOF)
239         dodaj_u_stablo(&koren, broj);
240
241     /* ispisujemo rezultat rada funkcija */
242     printf("broj cvorova: %d\n", br_cvorova(koren));
243     printf("broj listova: %d\n", br_listova(koren));
244     printf("pozitivni listovi: ");
245     pozitivni_listovi(koren);
246     printf("zbir cvorova: %d\n", suma_cvorova(koren));

```

```
251     if (najveci_element(koren) == NULL)
253         printf("najveci element: ne postoji\n");
255     else
256         printf("najveci element: %d\n",
257             najveci_element(koren)->broj);
259
261     printf("dubina stabla: %d\n", dubina_stabla(koren));
262     printf("\n");
263     printf("broj cvorova na %d. nivou: %d\n", i,
264         cvorovi_nivo(koren, i));
265     printf("elementi na %d. nivou: ", i);
266     ispis_nivo(koren, i);
267     printf("\n");
268     if (max_nivo(koren, i) == NULL)
269         printf("Nema elemenata na %d. nivou!\n", i);
270     else
271         printf("maksimalni na %d. nivou: %d\n", i,
272             max_nivo(koren, i)->broj);
274
275     printf("zbir na %d. nivou: %d\n", i, zbir_nivo(koren, i));
276     printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
277         suma(koren, x));
279
280     /* Oslobadjamo memoriju zauzetu stablom */
281     oslobodi_stablo(&koren);
283
284     /* Prekidamo izvršavanje programa */
285     return 0;
286 }
```

Rešenje 4.23

Rešenje 4.24

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija koja izračunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
9  {
10
11     /* Dubina praznog stabla je 0 */
12     if (koren == NULL)
13         return 0;
14
15     /* Izračunavamo dubinu levog podstabla */
```

```

17     int dubina_levo = dubina_stabla(koren->levo);

19     /* Izracunavamo dubinu desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

21     /* Dubina stabla odgovara vecoj od dubina podstabala - 1
        dodajemo jer racunamo i koren */
23     return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
25 }

27 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
29 {
    /* Ideja je slicna ideji iz prethodne funkcije */

31     /* Nema vise cvorova, ne mozemo da se spustamo kroz stablo */
33     if (koren == NULL)
        return;

35     /* Ako smo na trazenom nivou - ispisujemo vrednost */
37     if (i == 0) {
        printf("%d ", koren->broj);
39         return;
    }

41     /* Inace, spustamo se jedan nivo nize i u levom i u desnom
        podstablu */
43     ispisi_nivo(koren->levo, i - 1);
    ispisi_nivo(koren->desno, i - 1);
45 }

47 /* Funkcija koja ispisuje stablo po nivoima */
void ispisi_stablo_po_nivoima(Cvor * koren)
49 {

51     int i;

53     /* Prvo izracunavamo dubinu stabla */
    int dubina;
55     dubina = dubina_stabla(koren);

57     /* Ispisujemo nivo po nivo stabla */
    for (i = 0; i < dubina; i++) {
59         printf("%d. nivo: ", i);
        ispisi_nivo(koren, i);
61         printf("\n");
    }
63 }

65 int main(int argc, char **argv)
{
67     Cvor *koren;

```

```
    int broj;

69
    /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
71    koren = NULL;
    while (scanf("%d", &broj) != EOF) {
73        dodaj_u_stablo(&koren, broj);
    }

75
    /* Ispisujemo stablo po nivoima */
77    ispisi_stablo_po_nivoima(koren);

79
    /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);

81
    /* Prekidamo izvršavanje programa */
83    return 0;
}
```

Rešenje 4.25

Rešenje 4.26

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* Funkcija koja izračunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
9  {

11     /* Dubina praznog stabla je 0 */
    if (koren == NULL)
13         return 0;

15     /* Izračunavamo dubinu levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

17     /* Izračunavamo dubinu desnog podstabla */
19     int dubina_desno = dubina_stabla(koren->desno);

21     /* Dubina stabla odgovara vecoj od dubina podstabala - 1
       dodajemo jer racunamo i koren */
23     return dubina_levo >
           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
25 }

27 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
   AVL stablo */
```



```

29 int avl(Cvor * koren)
{
31     int dubina_levo, dubina_desno;

33     /* Ako je stablo prazno, zaustavljamo brojanje */
    if (koren == NULL) {
35         return 0;
    }

37     /* Izracunavamo dubinu levog podstabla korena */
39     dubina_levo = dubina_stabla(koren->levo);

41     /* Izracunavamo dubinu desnog podstabla korena */
    dubina_desno = dubina_stabla(koren->desno);

43     /* Ako je uslov za AVL stablo ispunjen */
45     if (abs(dubina_desno - dubina_levo) <= 1) {
        /* Racunamo broj avl cvorova u levom i desnom podstablu i
47         uvecavamo za jedan iz razloga sto koren ispunjava uslov */
        return 1 + avl(koren->levo) + avl(koren->desno);
49     } else {
        /* Inace, racunamo samo broj avl cvorova u podstablina */
51         return avl(koren->levo) + avl(koren->desno);
    }
53 }

55 int main(int argc, char **argv)
{
57     Cvor *koren;
    int broj;

59     /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
61     koren = NULL;
    while (scanf("%d", &broj) != EOF) {
63         dodaj_u_stablo(&koren, broj);
    }

65     /* Racunamo i ispisujemo broj AVL cvorova */
67     printf("%d\n", avl(koren));

69     /* Oslobadjamo memoriju zauzetu stablom */
    oslobodi_stablo(&koren);

71     /* Prekidamo izvršavanje programa */
73     return 0;
}

```

Rešenje 4.27

```

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

```

```
3  /* Uključujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* Funkcija proverava da li je zadato binarno stablo celih
   9  pozitivnih brojeva heap. Ideja koju ćemo implementirati u
  11  osnovi ima pronalazenje maksimalne vrednosti levog i
   maksimalne vrednosti desnog podstabla - ako je vrednost u
  13  korenu veća od izračunatih vrednosti uoceni fragment stabla
   zadovoljava uslov za heap. Zato će funkcija vratiti
   maksimalne vrednosti iz uocenog podstabala ili vrednost -1
   ukoliko zaključimo da stablo nije heap. */
15  int heap(Cvor * koren)
   {
17
   int max_levo, max_desno;

19
   /* Prazno sablo je heap. */
21  if (koren == NULL) {
   /* posto je 0 najmanji pozitivan broj, može nam poslužiti
   23  kao indikator */
   return 0;

25  }

27  /* Ukoliko je stablo list ... */
   if (koren->levo == NULL && koren->desno == NULL) {
29  /* ... vraćamo njegovu vrednost */
   return koren->broj;

31  }

33  /* Proveravamo svojstvo za levo podstablo. */
   max_levo = heap(koren->levo);

35

   /* Proveravamo svojstvo za desno podstablo. */
37  max_desno = heap(koren->desno);
   /* Ako levo ili desno podstablo uocenog cvora nije heap, onda
   39  nije ni celo stablo. */
   if (max_levo == -1 || max_desno == -1) {
41  return -1;
   }

43

   /* U suprotnom proveravamo da li svojstvo važi za uoceni
   45  cvor. */
   if (koren->broj > max_levo && koren->broj > max_desno) {
47  /* ako važi, vraćamo vrednost korena */
   return koren->broj;

49  }

51  /* u suprotnom zaključujemo da stablo nije heap */
   return -1;

53  }
```

```
55 int main(int argc, char **argv)
56 {
57     Cvor *koren;
58     int heap_indikator;
59
60     /* Kreiramo stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2 7
61        */
62     koren = NULL;
63     koren = napravi_cvor(100);
64     koren->levo = napravi_cvor(19);
65     koren->levo->levo = napravi_cvor(17);
66     koren->levo->levo->levo = napravi_cvor(2);
67     koren->levo->levo->desno = napravi_cvor(7);
68     koren->levo->desno = napravi_cvor(3);
69     koren->desno = napravi_cvor(36);
70     koren->desno->levo = napravi_cvor(25);
71     koren->desno->desno = napravi_cvor(1);
72
73     /* pozivamo funkciju kojom proveravamo da li je stablo heap */
74     heap_indikator = heap(koren);
75
76     /* i ispisujemo rezultat */
77     if (heap_indikator == -1) {
78         printf("Zadato tablo nije heap\n");
79     } else {
80         printf("Zadato stablo je heap!\n");
81     }
82
83     /* Oslobadjamo memoriju zauzetu stablom. */
84     oslobodi_stablo(&koren);
85
86     /* Završavamo sa programom */
87     return 0;
88 }
89 }
```


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 alokira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost -1 i prekinuti izvršavanje programa.

Test 1

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
Izlaz:
Ispit
Matematika
Programiranje
```

Test 2

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
2
maksimalano
poena
Izlaz:
```

Test 3

```
|| Poziv: ./a.out ulaz.txt
|| Problem: datoteka
|| ulaz.txt ne postoji
|| Izlaz:
|| -1
```

Test 4

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

[Rešenje 5.1]

Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na n -tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj n , a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test 1

```
|| Ulaz:
|| 2 8 10 3 6 14 13 7 4 0
|| Izlaz:
|| 20
```

Test 2

```
|| Ulaz:
|| 0 50 14 5 2 4 56 8 52 7 1 0
|| Izlaz:
|| 50
```

[Rešenje 5.2]

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A , a zatim i elementi matrice A . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

<p><i>Test 1</i></p> <pre> Ulaz: 4 5 1 2 3 4 5 -1 2 -3 4 -5 -5 -4 -3 -2 1 -1 0 0 0 0 Izlaz: 0 </pre>	<p><i>Test 2</i></p> <pre> Ulaz: 2 3 0 0 -5 1 2 -4 Izlaz: 2 </pre>	<p><i>Test 3</i></p> <pre> Ulaz: -2 Izlaz (na stderr): -1 </pre>
--	--	--

[Rešenje 5.3]

5.2 Programiranje 2, praktični deo ispita, jul 2015.

Zadatak 5.4

Napisati program koji kao prvi argument komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera. Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

<p><i>Test 1</i></p> <pre> Poziv: ./a.out ulaz.txt test ulaz.txt: Ovo je test primer. U njemu se rec test javlja vise puta. testtesttest Izlaz: 5 </pre>	<p><i>Test 2</i></p> <pre> Poziv: ./a.out Izlaz (na stderr): -1 </pre>
<p><i>Test 3</i></p> <pre> Poziv: ./a.out ulaz.txt foo ulaz.txt: (ne postoji) Izlaz (na stderr): -1 </pre>	<p><i>Test 4</i></p> <pre> Poziv: ./a.out ulaz.txt . ulaz.txt: (prazna) Izlaz: 0 </pre>

[Rešenje 5.4]

Zadatak 5.5 Jelena: Uključeno rešenje prethodnog zadatka. Dodati rešenje ovog zadatka. Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva

5 Ispitni rokovi

trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac: $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$, gde je s poluobim trougla). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

<p><i>Test 1</i></p> <pre> Datoteka: 4 0 0 0 1.2 1 0 0.3 0.3 0.5 0.5 0.9 1 -2 0 0 0 0 1 2 0 2 2 -1 -1 Izlaz: 2 0 2 2 -1 -1 -2 0 0 0 0 1 0 0 0 1.2 1 0 0.3 0.3 0.5 0.5 0.9 1</pre>	<p><i>Test 2</i></p> <pre> Datoteka: 3 1.2 3.2 1.1 4.3 Izlaz: -1</pre>
<p><i>Test 3</i></p> <pre> Datoteka: (nema datoteke) Izlaz: -1</pre>	<p><i>Test 4</i></p> <pre> Datoteka: 0 Izlaz:</pre>

[Rešenje 5.5]

Zadatak 5.6 Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju

```
int f3(Cvor *koren, int n)
```

koja u datom stablu prebrojava čvorove na n -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

<p><i>Test 1</i></p> <pre> Ulaz: 1 5 3 6 1 4 7 9 Izlaz: 1</pre>	<p><i>Test 2</i></p> <pre> Ulaz: 2 5 3 6 1 0 4 7 9 Izlaz: 2</pre>	<p><i>Test 3</i></p> <pre> Ulaz: 0 4 2 5 Izlaz: 0</pre>
<p><i>Test 4</i></p> <pre> Ulaz: 3 Izlaz: 0</pre>	<p><i>Test 5</i></p> <pre> Ulaz: -1 4 5 1 7 Izlaz: 0</pre>	

[Rešenje 5.6]

5.3 Rešenja

Rešenje 5.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #define MAX 50
5
6  void greska()
7  {
8      printf("-1\n");
9      exit(EXIT_FAILURE);
10 }
11
12 int main(int argc, char *argv[])
13 {
14
15     FILE *ulaz;
16     char **linije;
17     int i, j, n;
18
19     /* Proveravamo argumente komandne linije. */
20     if (argc != 2) {
21         greska();
22     }
23
24     /* Otvaramo datoteku cije ime je navedeno kao argument
25        komandne linije neposredno nakon imena programa koji se
26        poziva. */
27     ulaz = fopen(argv[1], "r");
28     if (ulaz == NULL) {
29         greska();
30     }
31
32     /* Ucitavamo broj linija. */
33     fscanf(ulaz, "%d", &n);
34
35     /* Alociramo memoriju na osnovu ucitanog broja linija. */
36     linije = (char **) malloc(n * sizeof(char *));
37     if (linije == NULL) {
38         greska();
39     }
40     for (i = 0; i < n; i++) {
41         linije[i] = malloc(MAX * sizeof(char));
42         if (linije[i] == NULL) {
43             for (j = 0; j < i; j++) {
44                 free(linije[j]);
45             }
46             free(linije);
47         }
48     }
49 }
```

```
47     greska();
48     }
49 }

51 /* Ucitavamo svih n linija iz datoteke. */
52 for (i = 0; i < n; i++) {
53     fscanf(ulaz, "%s", linije[i]);
54 }

55
56 /* Ispisujemo u odgovarajucem poretку ucitane linije koje
57    zadovoljavaju kriterijum. */
58 for (i = n - 1; i >= 0; i--) {
59     if (isupper(linije[i][0])) {
60         printf("%s\n", linije[i]);
61     }
62 }

63
64 /* Oslobadjamo memoriju koju smo dinamički alocirali. */
65 for (i = 0; i < n; i++) {
66     free(linije[i]);
67 }

68
69 free(linije);

71 /* Zatvaramo datoteku. */
72 fclose(ulaz);

73
74 /* Završavamo sa programom. */
75 return 0;
76
77 }
```

Rešenje 5.2

```
1 #include <stdio.h>
2 #include "stabla.h"

3
4
5 int sumirajN (Cvor * koren, int n){
6     if(koren==NULL){
7         return 0;
8     }

9
10    if(n==0){
11        return koren->broj;
12    }

13
14    return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
15 }
16
```

```

18 int main(){
    Cvor* koren=NULL;
20     int n;
    int nivo;

22     /* Citamo vrednost nivoa */
24     scanf("%d", &nivo);

26

    while(1){

28         /* Citamo broj sa standardnog ulaza */
30         scanf("%d", &n);

32         /* Ukoliko je korisnik uneo 0, prekidamo dalje citanje. */
        if(n==0){
34             break;
        }

36         /* A ako nije, dodajemo procitani broj u stablo. */
38         dodaj_u_stablo(&koren, n);

40     }

42     /* Ispisujemo rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren,nivo));

44     /* Oslobadjamo memoriju */
46     oslobodi_stablo(&koren);

48

    /* Prekidamo izvršavanje programa */
50     return 0;
}

```

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "stabla.h"

5  Cvor* napravi_cvor(int b ) {
    Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
7    if( novi == NULL)
        return NULL;

9

    /* Inicijalizacija polja novog Cvora */
11    novi->broj = b;
    novi->levo = NULL;
13    novi->desno = NULL;

15    return novi;
17 }

```

```

19 void oslobodi_stablo(Cvor** adresa_korena) {
    /* Prazno stablo i nema sta da se oslobadja */
21     if( *adresa_korena == NULL)
        return;
23
    /* Rekurzivno oslobadjamo najpre levo, a onda i desno podstablo*/
25     if( (*adresa_korena)->levo )
        oslobodi_stablo(&(*adresa_korena)->levo);
27     if( (*adresa_korena)->desno)
        oslobodi_stablo(&(*adresa_korena)->desno);
29
    free(*adresa_korena);
31     *adresa_korena =NULL;
    }
33
35 void prover_i_alokaciju( Cvor* novi) {
    if( novi == NULL) {
37         fprintf(stderr, "Malloc greska za nov cvor!\n");
        exit(EXIT_FAILURE);
39     }
    }
41
43 void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
    /* Postojece stablo je prazno*/
    if( *adresa_korena == NULL){
45         Cvor* novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
47         *adresa_korena = novi; /* Kreirani Cvor novi ce biti od
        sada koren stabla*/
        return;
49     }

    /* Brojeve smestamo u uredjeno binarno stablo, pa
    ako je broj koji ubacujemo manji od broja koji je u korenu */
51     if( broj < (*adresa_korena)->broj)
        /* Dodajemo u levo podstablo */
53         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
    /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
    dodajemo nov Cvor desno od korena */
55     else
        dodaj_u_stablo(&(*adresa_korena)->desno, broj);
57
59 }

#ifdef __STABLA_H__
2 #define __STABLA_H__ 1

4 /* Struktura kojom se predstavlja Cvor stabla */
typedef struct dcvor{
6     int broj;
    struct dcvor* levo, *desno;

```

```

8  } Cvor;

10 /* Funkcija alocira prostor za novi Cvor stabla, inicijalizuje polja
    strukture i vraća pokazivac na nov Cvor */
12 Cvor* napravi_cvor(int b );

14 /* Oslobadjamo dinamički alociran prostor za stablo
    * Nakon oslobadjanja se u pozivajućoj funkciji koren
16 * postavlja NULL, jer je stablo prazno */
void oslobodi_stablo(Cvor** adresa_korena);

18

20 /* Funkcija proverava da li je novi Cvor ispravno alociran,
    * i nakon toga prekida program */
22 void prover_i_alokaciju( Cvor* novi);

24

26 /* Funkcija dodaje nov Cvor u stablo i
    * azurira vrednost korena stabla u pozivajućoj funkciji.
    */
28 void dodaj_u_stablo(Cvor** adresa_korena, int broj);

30 #endif

```

Rešenje 5.3

```

#include <stdio.h>
2 #define MAX 50

4

int main()
6 {
    int m[MAX][MAX];
    int v, k;
    int i, j;
10    int max_broj_negativnih, max_indeks_kolone;
    int broj_negativnih;

12    /* Ucitavamo dimenzije matrice */
14    scanf("%d", &v);
    if (v < 0 || v > MAX) {
16        fprintf(stderr, "-1\n");
        return 0;
18    }

20    scanf("%d", &k);
    if (k < 0 || k > MAX) {
22        fprintf(stderr, "-1\n");
        return 0;
24    }

```

```

26  /* Ucitavamo elemente matrice */
    for (i = 0; i < v; i++) {
28      for (j = 0; j < k; j++) {
          scanf("%d", &m[i][j]);
30      }
    }

32
    /* Pronalazimo kolonu koja sadrzi najveći broj negativnih
34     elemenata */
    max_indeks_kolone = 0;

36
    max_broj_negativnih = 0;
38    for (i = 0; i < v; i++) {
        if (m[i][0] < 0) {
40            max_broj_negativnih++;
        }

42    }

44
    for (j = 0; j < k; j++) {
46        broj_negativnih = 0;
        for (i = 0; i < v; i++) {
48            if (m[i][j] < 0) {
                broj_negativnih++;
50            }
            if (broj_negativnih > max_broj_negativnih) {
52                max_indeks_kolone = j;
            }

54        }

56    }

58    /* Ispisujemo traženi rezultat */
    printf("%d\n", max_indeks_kolone);

60
    /* Završavamo program */
62    return 0;
}

```

Rešenje 5.4

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #define MAX 128

5
int main(int argc, char **argv)
7 {
    FILE *f;
9    int brojac = 0;
    char linija[MAX], *p;

```

```

11  if (argc != 3) {
13      fprintf(stderr, "-1\n");
14      exit(EXIT_FAILURE);
15  }

17  if ((f = fopen(argv[1], "r")) == NULL) {
18      fprintf(stderr, "-1\n");
19      exit(EXIT_FAILURE);
20  }

21  while (fgets(linija, MAX, f) != NULL) {
22      p = linija;
23      while (1) {
24          p = strstr(p, argv[2]);
25          if (p == NULL)
26              break;
27          brojac++;
28          p = p + strlen(argv[2]);
29      }
30  }

31  fclose(f);

32  printf("%d\n", brojac);

33  return 0;
34  }

```

Rešenje 5.5

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  typedef struct _trougao {
6      double xa, ya, xb, yb, xc, yc;
7  } trougao;
8
9  double duzina(double x1, double y1, double x2, double y2) {
10     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
11 }
12
13 double povrsina(trougao t) {
14     double a = duzina(t.xb, t.yb, t.xc, t.yc);
15     double b = duzina(t.xa, t.ya, t.xc, t.yc);
16     double c = duzina(t.xa, t.ya, t.xb, t.yb);
17     double s = (a + b + c) / 2;
18     return sqrt(s * (s - a) * (s - b) * (s - c));
19 }
20

```

```
int poredi(const void *a, const void *b) {
22   trougao x = *(trougao*)a;
   trougao y = *(trougao*)b;
24   double xp = povrsina(x);
   double yp = povrsina(y);
26   if (xp < yp)
       return 1;
28   if (xp > yp)
       return -1;
30   return 0;
}

32
int main() {
34   FILE *f;
   int n, i;
36   trougao *niz;

38   if ((f = fopen("trouglovi.txt", "r")) == NULL) {
       fprintf(stderr, "-1\n");
40       exit(EXIT_FAILURE);
   }

42
   if (fscanf(f, "%d", &n) != 1) {
44       fprintf(stderr, "-1\n");
       exit(EXIT_FAILURE);
46   }

48   if ((niz = malloc(n * sizeof(trougao))) == NULL) {
       fprintf(stderr, "-1\n");
50       exit(EXIT_FAILURE);
   }

52
   for (i = 0; i < n; i++) {
54       if (fscanf(f, "%lf%lf%lf%lf%lf%lf",
                   &niz[i].xa, &niz[i].ya,
56                   &niz[i].xb, &niz[i].yb,
                   &niz[i].xc, &niz[i].yc) != 6) {
58           fprintf(stderr, "-1\n");
           exit(EXIT_FAILURE);
60       }
   }

62
   qsort(niz, n, sizeof(trougao), &poredi);

64
   for (i = 0; i < n; i++)
66       printf("%g %g %g %g %g %g\n",
              niz[i].xa, niz[i].ya,
68              niz[i].xb, niz[i].yb,
              niz[i].xc, niz[i].yc);
70
   free(niz);
72   fclose(f);
}
```



```
74     return 0;
75 }

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"

5  Cvor *napravi_cvor(int broj)
6  {
7
8      /* Dinamicki kreiramo cvor */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

11     /* U slucaju greske ... */
12     if (novi == NULL) {
13         fprintf(stderr, "-1\n");
14         exit(1);
15     }

17     /* Inicijalizacija */
18     novi->vrednost = broj;
19     novi->levi = NULL;
20     novi->desni = NULL;

21     /* Vracamo adresu novog cvora */
22     return novi;
23 }

25 void dodaj_u_stablo(Cvor **koren, int broj)
26 {
27
28     /* Izlaz iz rekurzije: ako je stablo bilo prazno,
29        novi koren je upravo novi cvor */
30     if (*koren == NULL) {
31         *koren = napravi_cvor(broj);
32         return;
33     }

34     /* Ako je stablo neprazno, i koren sadrzi manju vrednost
35        od datog broja, broj se umece u desno podstablo,
36        rekurzivnim pozivom */
37     if ((*koren)->vrednost < broj)
38         dodaj_u_stablo(&(*koren)->desni, broj);
39     /* Ako je stablo neprazno, i koren sadrzi vecu vrednost
40        od datog broja, broj se umece u levo podstablo,
41        rekurzivnim pozivom */
42     else if ((*koren)->vrednost > broj)
43         dodaj_u_stablo(&(*koren)->levi, broj);
44
45 }
46 }
```

```

49 void prikazi_stablo(Cvor * koren)
{
51  /* Izlaz iz rekurzije */
    if (koren == NULL)
53      return;

    prikazi_stablo(koren->levi);
    printf("%d ", koren->vrednost);
57     prikazi_stablo(koren->desni);
}

59 Cvor* ucitaj_stablo() {
    Cvor *koren = NULL;
    int x;
61     while (scanf("%d", &x) == 1)
        dodaj_u_stablo(&koren, x);
63     return koren;
65 }

67 void oslobodi_stablo(Cvor **koren)
69 {
71     /* Izlaz iz rekurzije */
        if (*koren == NULL)
73         return;

        oslobodi_stablo(&(*koren)->levi);
        oslobodi_stablo(&(*koren)->desni);
75         free(*koren);
77     }

79     *koren = NULL;
}

```

```

1  #ifndef __STABLA_H__
    #define __STABLA_H__ 1
3
    /* Struktura koja predstavlja cvor stabla */
5  typedef struct cvor {
        int vrednost;    /* Vrednost koja se cuva */
        struct cvor *levi;    /* Pokazivac na levo podstablo */
        struct cvor *desni;    /* Pokazivac na desno podstablo */
9  } Cvor;

11 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira
    dinamicki, funkcijom malloc(). U slucaju greske program
13     se prekida i ispisuje se poruka o gresci. U slucaju
    uspeha inicijalizuje se vrednost datim brojem, a pokazivaci
15     na podstabla se inicijalizuju na NULL. Funkcija vraća
    adresu novokreiranog cvora */
17 Cvor *napravi_cvor(int broj);

19 /* Funkcija dodaje novi cvor u stablo sa datim korenom.

```

```

21     Ukoliko broj vec postoji u stablu, ne radi nista.
    Cvor se kreira funkcijom napravi_cvor(). */
void dodaj_u_stablo(Cvor **koren, int broj);
23
/* Funkcija prikazuje stablo s leva u desno (tj.
25     prikazuje elemente u rastucem poretku) */
void prikazi_stablo(Cvor * koren);
27
/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
    vraca
29     pokazican na njegov koren */
Cvor* ucitaj_stablo();
31
/* Funkcija oslobadja prostor koji je alocirana za
33     cvorove stabla. */
void oslobodi_stablo(Cvor **koren);
35
#endif

```

Rešenje 5.6

```

#include <stdio.h>
#include "stabla.h"
2
4 int f3(Cvor * koren, int n)
{
6     if (koren == NULL || n < 0)
        return 0;
8     if (n == 0) {
        if (koren->levi == NULL && koren->desni != NULL)
10         return 1;
        if (koren->levi != NULL && koren->desni == NULL)
12         return 1;
        return 0;
14     }
    return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
16 }

18 int main()
{
20     Cvor *koren;
    int n;
22
    scanf("%d", &n);
24     koren = ucitaj_stablo();

26     printf("%d\n", f3(koren, n));

28     oslobodi_stablo(&koren);

30     return 0;

```

5 Ispitni rokovi

}
