

## **PROGRAMIRANJE 2**



Univerzitet u Beogradu  
Matematički fakultet

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

**PROGRAMIRANJE 2  
Zbirka zadataka sa rešenjima**

**Beograd  
2016.**

Autori:

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

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

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Andelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

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

Recenzenti:

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

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

Obrada teksta, crteži i korice: *autori*

Štampa: Skripta internacional, Beograd

Tiraž: 150

CIP Каталогизација у публикацији

Народна библиотека Србије, Београд

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

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



# Sadržaj

<b>1 Uvodni zadaci</b>	<b>1</b>
1.1 Podela koda po datotekama . . . . .	1
1.2 Algoritmi za rad sa bitovima . . . . .	5
1.3 Rekurzija . . . . .	11
1.4 Rešenja . . . . .	20
<b>2 Pokazivači</b>	<b>69</b>
2.1 Pokazivačka aritmetika . . . . .	69
2.2 Višedimenzioni nizovi . . . . .	73
2.3 Dinamička alokacija memorije . . . . .	77
2.4 Pokazivači na funkcije . . . . .	84
2.5 Rešenja . . . . .	86
<b>3 Algoritmi pretrage i sortiranja</b>	<b>127</b>
3.1 Algoritmi pretrage . . . . .	127
3.2 Algoritmi sortiranja . . . . .	133
3.3 Bibliotečke funkcije pretrage i sortiranja . . . . .	143
3.4 Rešenja . . . . .	148
<b>4 Dinamičke strukture podataka</b>	<b>221</b>
4.1 Liste . . . . .	221
4.2 Stabla . . . . .	232
4.3 Rešenja . . . . .	241
<b>A Ispitni rokovi</b>	<b>337</b>
A.1 Praktični deo ispita, jun 2015. . . . .	337
A.2 Praktični deo ispita, jul 2015. . . . .	339
A.3 Praktični deo ispita, septembar 2015. . . . .	341
A.4 Praktični deo ispita, januar 2016. . . . .	343
A.5 Rešenja . . . . .	345



# 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 održanih ispita. Elektronska verzija zbirke i radni repozitorijum elektronskih verzija rešenja zadataka, dostupni su u okviru strane kursa [www.programiranje2.matf.bg.ac.rs](http://www.programiranje2.matf.bg.ac.rs), besplatno.

U prvom poglavlju zbirke obrađene su uvodne teme koje obuhvataju osnovne tehnike koje se koriste u rešavanju svih ostalih zadataka u zbirci: podela koda po datotekama i rekurzivni pristup rešavanju problema. Takođe, u okviru ovog poglavlja dati su i osnovni algoritmi za rad sa bitovima. Drugo poglavlje je posvećeno pokazivačima: pokazivačkoj aritmetici, višedimenzionim nizovima, dinamičkoj alokaciji memorije i radu sa pokazivačima na funkcije. Treće poglavlje obrađuje algoritme pretrage i sortiranja, a četvrto dinamičke strukture podataka: liste i stabla. Dodatak sadrži najvažnije ispitne rokove iz jedne akademске godine. Većina zadataka je rešena, a teži zadaci su obeleženi zvezdicom.

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. Zbog toga smatramo da je naš osnovni doprinos što smo objedinili, precizno formulisali, rešili i detaljno iskomentarisali sve najvažnije zadatke koji su potrebni za uspešno savladavanje koncepata koji se obrađuju u okviru kursa.

Zahvaljujemo recenzentima, Gordani Pavlović Lažetić i Draganu Uroševiću, na veoma pažljivom čitanju rukopisa i na brojnim korisnim sugestijama. Takođe, zahvaljujemo studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli u uobličavanju ovog materijala.

Svi komentari i sugestije na zadatke i rešenja zbirke su dobrodošli i osećajte se slobodno da ih pošaljete elektronskom poštom bilo kome od autora<sup>1</sup>.

*Autori*

---

<sup>1</sup>Adrese autora su: milena, jgraovac, aspasic, mirko, andjelkaz, nina, sa nastavkom @matf.bg.ac.rs

# 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

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

- (a) Definisati strukturu `KompleksanBroj` koja opisuje kompleksan broj zadat njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj `z` sa standardnog ulaza.
- (c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` koja ispisuje kompleksan broj `z` na standardni izlaz u odgovarajućem formatu.
- (d) Napisati funkciju `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja `z`.
- (e) Napisati funkciju `float imaginarni_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja `z`.
- (f) Napisati funkciju `float moduo(KompleksanBroj z)` koja vraća moduo kompleksnog broja `z`.
- (g) Napisati funkciju `KompleksanBroj konjugovan(KompleksanBroj z)` koja vraća konjugovano-kompleksni broj broja `z`.
- (h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća zbir dva kompleksna broja `z1` i `z2`.

## 1 Uvodni zadaci

---

- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća razliku dva kompleksna broja  $z_1$  i  $z_2$ .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća proizvod dva kompleksna broja  $z_1$  i  $z_2$ .
- (k) Napisati funkciju `float argument(KompleksanBroj z)` koja vraća argument kompleksnog broja  $z$ .

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza uneti dva kompleksna broja  $z_1$  i  $z_2$ , a zatim ispisati realni deo, imaginarni deo, moduo, konjugovano-kompleksan broj i argument broja koji se dobija kao zbir, razlika ili proizvod brojeva  $z_1$  i  $z_2$  u zavisnosti od znaka ('+', '-', '\*') koji se unosi sa standardnog ulaza.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite realni i imaginarni deo kompleksnog broja: 1 -3  
(1.00 - 3.00 i)  
Unesite realni i imaginarni deo kompleksnog broja: -1 4  
(-1.00 + 4.00 i)  
Unesite znak (+,-,*): -  
(1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)  
Realni_deo: 2  
Imaginarni_deo: -7.000000  
Moduo: 7.280110  
Konjugovano kompleksan broj: (2.00 + 7.00 i)  
Argument kompleksnog broja: - 1.292497
```

[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. Napisati program koji testira ovu biblioteku. Sa standardnog ulaza uneti kompleksan broj, a zatim na standardni izlaz ispisati njegov polarni oblik.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite realni i imaginarni deo kompleksnog broja: -5 2  
Polarni oblik kompleksnog broja je 5.39 * e^i * 2.76
```

[Rešenje 1.2]

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

- (a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20 koji je zadat nizom svojih koeficijenata tako da se na i-toj poziciji u nizu nalazi koeficijent uz i-ti stepen polinoma.
- (b) Napisati funkciju `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz, od najvišeg ka najnižem stepenu. Ispisati samo koeficijente koji su različiti od nule.
- (c) Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom najpre uneti stepen, a zatim njegove koeficijente.
- (d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` koja vraća vrednosti polinoma `p` u dатој тачки `x` користећи Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja vraća zbir dva polinoma `p` и `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja vraća proizvod dva polinoma `p` и `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja vraća izvod polinoma `p`.
- (h) Napisati funkciju `Polinom n_izvod(const Polinom * p, int n)` koja vraća `n`-ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati polinome `p` i `q`, a zatim ih ispisati na standardni izlaz u odgovarajućem formatu. Izračunati i ispisati zbir `z` i proizvod `r` unetih polinoma `p` i `q`. Sa standardnog ulaza učitati realni broj `x`, a zatim na standardni izlaz ispisati vrednost polinoma `z` u tački `x` zaokruženu na dve decimale. Na kraju, sa standardnog ulaza učitati broj `n` i na izlaz ispisati `n`-ti izvod polinoma `r`.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite polinom p (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):  
3 1.2 3.5 2.1 4.2  
Unesite polinom q (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):  
2 2.1 0 -3.9  
Zbir polinoma je polinom z:  
1.20x^3+5.60x^2+2.10x+0.30  
Prozvod polinoma je polinom r:  
2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38  
Unesite tacku u kojoj racunate vrednost polinoma z:  
0  
Vrednost polinoma z u tacki 0.00 je 0.30  
Unesite izvod polinoma koji zelite:  
3  
3. izvod polinoma r je: 151.20x^2+176.40x-1.62
```

## 1 Uvodni zadaci

---

[Rešenje 1.3]

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

- (a) Definisati strukturu `Razlomak` koja opisuje razlomak.
- (b) Napisati funkciju `Razlomak ucitaj()` za učitavanje razlomka.
- (c) Napisati funkciju `void ispisi(const Razlomak * r)` koja ispisuje razlomak `r`.
- (d) Napisati funkciju `int brojilac(const Razlomak * r)` koja vraćaja brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraćaja imenilac razlomka `r`.
- (f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.
- (g) Napisati funkciju `double reciprocna_vrednost(const Razlomak * r)` koja vraća recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skrati(const Razlomak * r)` koja vraća skraćenu vrednost datog razlomka `r`.
- (i) Napisati funkciju `Razlomak saberi(const Razlomak * r1, const Razlomak * r2)` koja vraća zbir dva razlomka `r1` i `r2`.
- (j) Napisati funkciju `Razlomak oduzmi(const Razlomak * r1, const Razlomak * r2)` koja vraća razliku dva razlomka `r1` i `r2`.
- (k) Napisati funkciju `Razlomak pomnozi(const Razlomak * r1, const Razlomak * r2)` koja vraća proizvod dva razlomka `r1` i `r2`.
- (l) Napisati funkciju `Razlomak podeli(const Razlomak * r1, const Razlomak * r2)` koja vraća količnik dva razlomka `r1` i `r2`.

Napisati program koji testira prethodne funkcije. Sa standardnog ulaza učitati dva razlomka `r1` i `r2`. Na standardni izlaz ispisati skraćene vrednosti zbira, razlike, proizvoda i količnika razlomaka `r1` i recipročne vrednosti razlomka `r2`.

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite imenilac i brojilac prvog razlomka: 1 2  
|| Unesite imenilac i brojilac drugog razlomka: 2 3  
|| 1/2 + 3/2 = 2  
|| 1/2 - 3/2 = -1  
|| 1/2 * 3/2 = 3/4  
|| 1/2 / 3/2 = 1/3
```

## 1.2 Algoritmi za rad sa bitovima

**Zadatak 1.5** Napisati biblioteku `stampanje_bitova` za rad sa bitovima. Biblioteka treba da sadrži funkcije `stampanje_bitova`, `stampanje_bitova_short` i `stampanje_bitova_char` za štampanje bitova u binarnom zapisu celog broja tipa `int`, `short` i `char`, koji se zadaje kao argument funkcije. Napisati program koji testira napisanu biblioteku. Sa standardnog ulaza učitati u heksadekadnom formatu cele brojeve tipa `int`, `short` i `char` i na standardni izlaz ispisati njihovu binarnu reprezentaciju.

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite broj tipa int: 0x4f4f4f4f  
|| Binarna reprezentacija: 0100111010011110100111101001111  
|| Unesite broj tipa short: 0x4f4f  
|| Binarna reprezentacija: 010011101001111  
|| Unesite broj tipa char: 0x4f  
|| Binarna reprezentacija: 01001111
```

[Rešenje 1.5]

**Zadatak 1.6** Napisati funkcije `prebroj_bitove_1` i `prebroj_bitove_2` koje vraćaju broj jedinica u binarnom zapisu označenog celog broja  $x$  koji se zadaje kao argument funkcije. Prebrojavanje bitova ostvariti na dva načina:

- (a) formiranjem odgovarajuće maske i njenim pomeranjem (funkcija `prebroj_bitove_1`)
- (b) formiranjem odgovarajuće maske i pomeranjem promenljive  $x$  (funkcija `prebroj_bitove_2`).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati ceo broj u heksadekasnomb formatu i redni broj funkcije koju treba primeniti ('1' ili '2'), a zatim na standardni izlaz ispisati broj jedinica u binarnom zapisu učitanog broja pozivom izabrane funkcije. Ukoliko korisnik ne unese ispravnu vrednost za

## 1 Uvodni zadaci

---

redni broj funkcije, prekinuti izvršavanje programa i ispisati odgovarajuću poruku na standardni izlaz za greške.

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite broj: 0x7F  
|| Unesite redni broj funkcije: 1  
|| Poziva se funkcija prebroj_bitove_1  
|| Broj jedinica u zapisu je 7
```

Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite broj: -0x7F  
|| Unesite redni broj funkcije: 2  
|| Poziva se funkcija prebroj_bitove_2  
|| Broj jedinica u zapisu je 26
```

Primer 3

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite broj: 0x00FF00FF  
|| Unesite redni broj funkcije: 2  
|| Poziva se funkcija prebroj_bitove_2  
|| Broj jedinica u zapisu je 16
```

Primer 4

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite broj: 0x00FF00FF  
|| Unesite redni broj funkcije: 3  
|| IZLAZ ZA GREŠKU:  
|| Neodgovarajuci redni broj funkcije!
```

[Rešenje 1.6]

**Zadatak 1.7** Napisati funkcije `unsigned najveci(unsigned x)` i `unsigned najmanji(unsigned x)` koje vraćaju najveći, odnosno najmanji neoznačen ceo broj koji se može zapisati istim binarnim ciframa kao broj `x`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati neoznačen ceo broj u heksadekadnom formatu, a zatim ispisati binarnu reprezentaciju najvećeg i najmanjeg broja koji se može zapisati istim binarnim ciframa kao učitani broj.

Test 1

```
|| ULAZ:  
|| 0x7F  
|| IZLAZ:  
|| Najveci:  
|| 11111100000000000000000000000000  
|| Najmanji:  
|| 0000000000000000000000000000111111
```

Test 2

```
|| ULAZ:  
|| 0x80  
|| IZLAZ:  
|| Najveci:  
|| 10000000000000000000000000000000  
|| Najmanji:  
|| 00000000000000000000000000000000
```

Test 3

```
|| ULAZ:  
|| 0x00FF00FF  
|| IZLAZ:  
|| Najveci:  
|| 11111111111111000000000000000000  
|| Najmanji:  
|| 00000000000000011111111111111111
```

Test 4

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

[Rešenje 1.7]

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

- (a) Napisati funkciju `unsigned postavi_0(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , postave na 0.
- (b) Napisati funkciju `unsigned postavi_1(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se  $n$  bitova datog broja  $x$ , počevši od pozicije  $p$ , postave na 1.
- (c) Napisati funkciju `unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)` koja vraća broj u kome se  $n$  bitova najmanje težine poklapa sa  $n$  bitova broja  $x$  počevši od pozicije  $p$ , dok su mu ostali bitovi postavljeni na 0.
- (d) Napisati funkciju `unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p, unsigned y)` koja vraća broj koji se dobija upisivanjem poslednjih  $n$  bitova najmanje težine broja  $y$  u broj  $x$ , počevši od pozicije  $p$ .
- (e) Napisati funkciju `unsigned invertuj(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija invertovanjem  $n$  bitova broja  $x$  počevši od pozicije  $p$ .

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve  $x, n, p, y$  koji se unose sa standardnog ulaza. Na standardni izlaz ispisati binarne reprezenatacije brojeva  $x$  i  $y$ , a zatim i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija.

*NAPOMENA: Bit najmanje težine je krajnji desni bit i njegova pozicija se označava nultom dok se pozicije ostalih bitova uvećavaju za jedan, sa desna na levo.*

## 1 Uvodni zadaci

---

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite neoznacen ceo broj x: 235  
Unesite neoznacen ceo broj n: 9  
Unesite neoznacen ceo broj p: 24  
Unesite neoznacen ceo broj y: 127  
x = 235 = 00000000000000000000000000000000011101011  
postavi_0( 235, 9, 24) = 00000000000000000000000000000000011101011  
  
x = 235 = 00000000000000000000000000000000011101011  
postavi_1( 235, 9, 24) = 0000000111111110000000011101011  
  
x = 235 = 00000000000000000000000000000000011101011  
vrati_bitove( 235, 9, 24) = 0000000000000000000000000000000000000000000000000  
  
x = 235 = 00000000000000000000000000000000011101011  
y = 127 = 000000000000000000000000000000001111111  
postavi_1_n_bitove( 235, 9, 24, 127) = 000000000111111100000000011101011  
  
x = 235 = 00000000000000000000000000000000011101011  
invertuj( 235, 9, 24) = 00000001111111100000000011101011
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite neoznacen ceo broj x: 2882398951  
Unesite neoznacen ceo broj n: 5  
Unesite neoznacen ceo broj p: 10  
Unesite neoznacen ceo broj y: 35156526  
x = 2882398951 = 10101011110011011110101011100111  
postavi_0(2882398951, 5, 10) = 10101011110011011110100000100111  
  
x = 2882398951 = 10101011110011011110101011100111  
postavi_1(2882398951, 5, 10) = 1010101111001101111011111100111  
  
x = 2882398951 = 10101011110011011110101011100111  
vrati_bitove(2882398951, 5, 10) = 000000000000000000000000000000001011  
  
x = 2882398951 = 10101011110011011110101011100111  
y = 35156526 = 00000010000110000111001000101110  
postavi_1_n_bitove(2882398951, 5, 10, 35156526) = 10101011110011011110101110100111  
  
x = 2882398951 = 10101011110011011110101011100111  
invertuj(2882398951, 5, 10) = 10101011110011011110110100100111
```

[Rešenje 1.8]

**Zadatak 1.9** Pod rotiranjem bitova uлево подразумева се помјеравање свих битова за једну позицију улево, с тим што се бит са позиције највеће тежине помјерава на позицију најмање тежине. Аналогно, ротирање битова удесно подразумева помјеравање свих битова за једну позицију удесно, с тим што се бит са позиције најмање тежине помјерава на позицију највеће тежине.

- (a) Написати функцију `unsigned rotiraj_ulevo(unsigned x, unsigned n)` која

vraća broj koji se dobija rotiranjem n puta ulevo datog celog neoznačenog broja x.

- (b) Napisati funkciju `unsigned rotiraj_udesno(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem n puta udesno datog celog neoznačenog broja x.
- (c) Napisati funkciju `int rotiraj_udesno_oznaceni(int x, unsigned n)` koja vraća broj koji se dobija rotiranjem n puta udesno datog celog broja x.

Napisati program koji sa standardnog ulaza učitava neoznačene cele brojeve x i n koji se unose u heksadekasnem formatu, tatim ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem tri prethodno napisane funkcije sa argumentima x i n, a na kraju ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem funkcije `rotiraj_udesno_oznaceni` za argumente -x i n.

#### *Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite neoznacen ceo broj x: ba11a7
Unesite neoznacen ceo broj n: 5
x = 00000000101110100001000110100111
rotiraj_ulevo(ba11a7, 5)      = 00010111010000100011010011100000
rotiraj_udesno(ba11a7, 5)     = 00111000000001011101000010001101
rotiraj_udesno_oznaceni(ba11a7, 5) = 00111000000001011101000010001101
rotiraj_udesno_oznaceni(-ba11a7, 5) = 1100011111110100010111101110010
```

[Rešenje 1.9]

**Zadatak 1.10** Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća ceo broj čiji binarni zapis predstavlja sliku u ogledalu binarnog zapisu broja x. Napisati program koji testira datu funkciju za broj koji se sa standardnog ulaza zadaje u heksadekadnom formatu. Najpre ispisati binarnu reprezentaciju unetog broja, a zatim i binarnu reprezentaciju broja dobijenog kao njegova slika u ogledalu.

#### *Test 1*

```
ULAZ:
255
IZLAZ:
0000000000000000000000000000001001010101
101010100100000000000000000000000000000000
```

#### *Test 2*

```
ULAZ:
-15
IZLAZ:
1111111111111111111111111111111010111
110101111111111111111111111111111111111111
```

[Rešenje 1.10]

**Zadatak 1.11** Napisati funkciju `int broj_01(unsigned int n)` koja za dati broj n vraća 1 ako u njegovom binarnom zapisu ima više jedinica nego nula,

## 1 Uvodni zadaci

---

a inače vraća 0. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1

ULAZ:
10
IZLAZ:
0

Test 2

ULAZ:
2147377146
IZLAZ:
1

Test 3

ULAZ:
1111111115
IZLAZ:
0

[Rešenje [1.11](#)]

**Zadatak 1.12** Napisati funkciju `int broj_parova(unsigned int x)` koja vraća broj pojava dve uzastopne jedinice u binarnom zapisu celog neoznačenog broja `x`. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Tri uzastopne jedinice sadrže dve uzastopne jedinice dva puta.*

Test 1

ULAZ:
11
IZLAZ:
1

Test 2

ULAZ:
1024
IZLAZ:
0

Test 3

ULAZ:
2147377146
IZLAZ:
22

[Rešenje [1.12](#)]

\* **Zadatak 1.13** Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama  $i$  i  $j$ . Pozicije  $i$  i  $j$  učitati kao parametre komandne linije. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore  $+$ ,  $-$ ,  $/$ ,  $*$ ,  $\%$ .

Primer 1

POKRETANJE:
./a.out 1 2
INTERAKCIJA SA PROGRAMOM:
ULAZ:
11
IZLAZ:
13

Primer 2

POKRETANJE:
./a.out 1 2
INTERAKCIJA SA PROGRAMOM:
ULAZ:
1024
IZLAZ:
1024

Primer 2

POKRETANJE:
./a.out 12 12
INTERAKCIJA SA PROGRAMOM:
ULAZ:
12345
IZLAZ:
12345

\* **Zadatak 1.14** Napisati funkciju `void prevod(unsigned int x, char s[])` koja na osnovu neoznačenog broja  $x$  formira nisku  $s$  koja sadrži heksadekadni zapis broja  $x$  koristeći algoritam za brzo prevodenje binarnog u heksade-

kadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksadekadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1	Test 2	Test 3
<b>ULAZ:</b> <code>11</code> <b>IZLAZ:</b> <code>0000000B</code>	<b>ULAZ:</b> <code>1024</code> <b>IZLAZ:</b> <code>00000400</code>	<b>ULAZ:</b> <code>12345</code> <b>IZLAZ:</b> <code>00003039</code>

[Rešenje [1.14](#)]

#### \* Zadatak 1.15

Napisati funkciju koja za dva neoznačena broja  $x$  i  $y$  invertuje one bitove u broju  $x$  koji se poklapaju sa odgovarajućim bitovima u broju  $y$ . Ostali bitovi treba da ostanu nepromjenjeni. Napisati program koji testira tu funkciju za brojeve koji se zadaju sa standardnog ulaza.

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

**Zadatak 1.16** Napisati funkciju koja vraća broj petica u oktalnom zapisu neoznačenog celog broja  $x$ . Napisati program koji testira tu funkciju za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

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

## 1.3 Rekurzija

**Zadatak 1.17** Napisati rekurzivnu funkciju koja izračunava  $x^k$ , za dati ceo broj  $x$  i prirodan broj  $k$

- (a) tako da rešenje bude linearne složenosti,

## 1 Uvodni zadaci

---

(b) tako da rešenje bude logaritamske složenosti.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1' ili '2'), ceo broj  $x$  i prirodan broj  $k$ , a zatim na standarni izlaz ispisati rezultat primene izabrane funkcije na unete brojeve. Ukoliko se na ulazu unese pogrešan redni broj funkcije, ispisati odgovarajuću poruku o grešci na standardni izlaz i prekinuti izvršavanje programa.

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite redni broj funkcije (1/2):  
|| 1  
|| Unesite broj x: 2  
|| Unesite broj k: 10  
|| 1024
```

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite redni broj funkcije (1/2):  
|| 2  
|| Unesite broj x: 9  
|| Unesite broj k: 4  
|| 6561
```

[Rešenje 1.17]

**Zadatak 1.18** Koristeći uzajamnu (posrednu) rekurziju napisati:

- funkciju `unsigned paran(unsigned n)` koja proverava da li je broj cifara broja  $x$  paran i vraća 1 ako jeste, a 0 inače;
- i funkciju `unsigned neparan(unsigned n)` koja proverava da li je broj cifara broja  $x$  neparan i vraća 1 ako jeste, a 0 inače.

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

*Test 1*

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

*Test 2*

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

[Rešenje 1.18]

**Zadatak 1.19** Napisati repno-rekurzivnu funkciju koja izračunava faktorijel broja  $n$ . Napisati program koji testira napisanu funkciju za proizvoljan broj  $n$  ( $n \leq 12$ ) unet sa standardnog ulaza. NAPOMENA: *Gornja vrednost za  $n$  je postavljena na 12 zbog ograničenja veličine broja koji može da stane u promenljivu tipa `int` i činjenice da niz faktorijela brzo raste.*

*Primer 1*

```
||| INTERAKCIJA SA PROGRAMOM:
    Unesite n (<= 12):  5
    5! = 120
```

*Primer 2*

```
||| INTERAKCIJA SA PROGRAMOM:
    Unesite n (<= 12):  0
    0! = 1
```

[Rešenje 1.19]

**Zadatak 1.20** Napisati funkciju koja vraća  $n$ -ti element u nizu Fibonačijevih brojeva. Elementi niza Fibonačijevih brojeva  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2).$$

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati prirodan broj  $n$  i na standardni izlaz ispisati rezultat primene napisane funkcije na prirodan broj  $n$ .

*Primer 1*

```
||| INTERAKCIJA SA PROGRAMOM:
    Unesite koji clan niza se racuna:  5
    F(5) = 5
```

*Primer 2*

```
||| INTERAKCIJA SA PROGRAMOM:
    Unesite koji clan niza se racuna:  8
    F(8) = 21
```

**Zadatak 1.21** Elementi niza  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

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

Napisati funkciju koja računa  $n$ -ti element u nizu  $F$

- (a) iterativno,
- (b) tako da funkcija bude rekurzivna i da koristi navedene rekurentne relacije,
- (c) tako da funkcija bude rekurzivna ali da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1','2','3'), vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ . Na standardni izlaz ispisati rezultat primene odabrane funkcije

## 1 Uvodni zadaci

---

nad učitanim podacima, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa. NAPOMENA: *Niz F definisan na ovaj način predstavlja uopštenje Fibonačijevih brojeva.*

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite redni broj funkcije koju zelite:  
|| 1 - iterativna  
|| 2 - rekurzivna  
|| 3 - rekurzivna napredna  
|| 1  
|| Unesite koeficijente: 2 3  
|| Unesite koji clan niza se racuna: 5  
|| F(5) = 61
```

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite redni broj funkcije koju zelite:  
|| 1 - iterativna  
|| 2 - rekurzivna  
|| 3 - rekurzivna napredna  
|| 3  
|| Unesite koeficijente: 4 2  
|| Unesite koji clan niza se racuna: 8  
|| F(8) = 31360
```

[Rešenje 1.21]

**Zadatak 1.22** 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 1.22]

**Zadatak 1.23** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva

- sabirajući elemente počev od početka niza ka kraju niza,
- sabirajući elemente počev od kraja niza ka početku niza.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije ('1' ili '2'), zatim dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a potom i elemente niza. Na standardni izlaz ispisati rezultat primene odabrane

funkcije nad učitanim nizom, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije (1 ili 2): 1
Unesite dimenziju niza: 5
Unesite elemente niza:
1 2 3 4 5
Suma elemenata je 15
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije (1 ili 2): 2
Unesite dimenziju niza: 4
Unesite elemente niza:
-5 2 -3 6
Suma elemenata je 0
```

[Rešenje 1.23]

**Zadatak 1.24** Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. Elementi niza se unose sve do kraja ulaza (EOF). Pretpostaviti da niz neće imati više od 256 elemenata.

#### Test 1

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

#### Test 2

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

#### Test 3

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

[Rešenje 1.24]

**Zadatak 1.25** Napisati rekurzivnu funkciju koja izračunava skalarni proizvod dva vektora celih brojeva. Napisati program koji testira ovu funkciju za nizove (vektore) koji se unose sa standardnog ulaza. Prvo treba uneti dimenziju nizova, a zatim i njihove elemente. Na standardni izlaz ispisati skalarni proizvod unetih nizova. Pretpostaviti da nizovi neće imati više od 256 elemenata.

#### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju nizova: 3
Unesite elemente prvog niza:
1 2 3
Unesite elemente drugog niza:
1 2 3
Skalarni proizvod je 14
```

#### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju nizova: 2
Unesite elemente prvog niza:
3 5
Unesite elemente drugog niza:
2 6
Skalarni proizvod je 36
```

[Rešenje 1.25]

**Zadatak 1.26** Napisati rekurzivnu funkciju koja vraća broj pojavljivanja elementa  $x$  u nizu  $a$  dužine  $n$ . Napisati program koji testira ovu funkciju za broj

## 1 Uvodni zadaci

---

$x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do kraja ulaza. Prepostaviti da nizovi neće imati više od 256 elemenata.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite ceo broj:  
4  
Unesite elemente niza:  
1 2 3 4  
Broj pojavljivanja je 1
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite ceo broj:  
11  
Unesite elemente niza:  
3 2 11 14 11 43 1  
Broj pojavljivanja je 2
```

### Primer 3

```
INTERAKCIJA SA PROGRAMOM:  
Unesite ceo broj:  
1  
Unesite elemente niza:  
3 21 5 6  
Broj pojavljivanja je 0
```

[Rešenje 1.26]

**Zadatak 1.27** Napisati rekurzivnu funkciju kojom se proverava da li su tri data cela broja uzastopni članovi datog celobrojnog niza. Sa standardnog ulaza učitati tri broja, a zatim elemente niza sve do kraja ulaza. Na standardni izlaz ispisati rezultat primene funkcije nad učitanim podacima. Prepostaviti da neće biti uneto više od 256 brojeva.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite tri cela broja:  
1 2 3  
Unesite elemente niza:  
4 1 2 3 4 5  
Uneti brojevi jesu uzastopni clanovi niza.
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite tri cela broja:  
1 2 3  
Unesite elemente niza:  
11 1 2 4 3 6  
Uneti brojevi nisu uzastopni clanovi niza.
```

[Rešenje 1.27]

**Zadatak 1.28** Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja  $x$ . 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:	0x00FF00FF
IZLAZ:	16

*Test 3*

ULAZ:	0xFFFFFFFF
IZLAZ:	32

[Rešenje 1.28]

**Zadatak 1.29** 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

*Test 2*

ULAZ:	0
IZLAZ:	00000000000000000000000000000000

**Zadatak 1.30** 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

[Rešenje 1.30]

**Zadatak 1.31** 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

## 1 Uvodni zadaci

---

[Rešenje 1.31]

**Zadatak 1.32** Napisati rekurzivnu funkciju `int palindrom(char s[], int n)` koja ispituje da li je data niska `s` palindrom. Napisati program koji testira ovu funkciju za nisku koja se zadaje sa standardnog ulaza. Prepostaviti da niska neće imati više od 31 karaktera.

Test 1	Test 2	Test 3
<code>ULAZ:</code> <code>a</code> <code>IZLAZ:</code> <code>da</code>	<code>ULAZ:</code> <code>aa</code> <code>IZLAZ:</code> <code>da</code>	<code>ULAZ:</code> <code>aba</code> <code>IZLAZ:</code> <code>da</code>
Test 4	Test 5	
<code>ULAZ:</code> <code>programiranje</code> <code>IZLAZ:</code> <code>ne</code>	<code>ULAZ:</code> <code>anavolimilovana</code> <code>IZLAZ:</code> <code>da</code>	

[Rešenje 1.32]

\* **Zadatak 1.33** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za proizvoljan prirodan broj  $n$  ( $n \leq 15$ ) unet sa standardnog ulaza.

Test 1	Test 2	Test 3
<code>ULAZ:</code> <code>2</code> <code>IZLAZ:</code> <code>1 2</code> <code>2 1</code>	<code>ULAZ:</code> <code>3</code> <code>IZLAZ:</code> <code>1 2 3</code> <code>1 3 2</code> <code>2 1 3</code> <code>2 3 1</code> <code>3 1 2</code> <code>3 2 1</code>	<code>ULAZ:</code> <code>-5</code> <code>Duzina</code> <code>permutacije</code> <code>mora biti</code> <code>broj iz</code> <code>intervala</code> <code>[0, 15]!</code>

[Rešenje 1.33]

\* **Zadatak 1.34** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbiru dve vrednosti koje su u susedna dva polja iznad. Izuzetak su jedinice na krajevima. Vrednosti brojeva Paskalovog trougla odgovaraju binomnim koeficijentima tj. vrednost polja  $(n, k)$ , gde je  $n$  redni broj hipotenuze, a  $k$  redni broj elementa u tom redu (na toj hipotenuzi) odgovara binomnom koeficijentu  $\binom{n}{k}$ , pri čemu brojanje počinje od nule. Na primer, vrednost polja  $(4, 2)$  je 6.

$$\begin{array}{ccccccc}
 & & & & 1 & & \\
 & & & & 1 & 1 & \\
 & & & & 1 & 2 & 1 \\
 & & & & 1 & 3 & 3 & 1 \\
 & & & & 1 & 4 & 6 & 4 & 1 \\
 & & & & 1 & 5 & 10 & 10 & 5 & 1
 \end{array}$$

- (a) Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta  $\binom{n}{k}$  koristeći osobine Paskalovog trougla.
- (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 redni broj hipotenuze najpre isrtava Paskalov trougao, a zatim štampa sumu elemenata hipotenuze.

Test 1

```

ULAZ:
5 3
IZLAZ:
      1
      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   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 1.34]

\* **Zadatak 1.35** 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:
2
IZLAZ:
a a
a b
b a
b b
    
```

Test 2

```

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
    
```

## 1 Uvodni zadaci

---

\* **Zadatak 1.36 Hanojske kule:** Data su tri vertikalna štapa. Na jednom od njih 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 prenesti diskove sa jednog na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg. Preostali štap koristiti kao pomoćni štap prilikom premeštanja.

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

\* **Zadatak 1.37 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 prenesti diskove na drugi štap tako da budu u istom redosledu, prenestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg. Preostala dva štapa koristiti kao pomoćne štapove prilikom premeštanja.

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

## 1.4 Rešenja

### Rešenje 1.1

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

4 /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
5   imaginarni deo kompleksnog broja */
6 typedef struct {
7     float real;
8     float imag;
9 } KompleksanBroj;

12 /* Funkcija ucitava sa standardnog ulaza realan i imaginarni deo
13   kompleksnog broja i smesta ih u strukturu cija je adresa argument
14   funkcije */
15 void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
17     /* Ucitavanje vrednosti sa standardnog ulaza */
18     printf("Unesite realni i imaginarni deo kompleksnog broja: ");
19     scanf("%f", &z->real);
20     scanf("%f", &z->imag);
21 }
22 }
```

```

24  /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
25   obliku (x + i y). Ovoj funkciji se argument prenosi po vrednosti
26   jer se u samoj funkciji ne menja njegova vrednost */
27 void ispisi_kompleksan_broj(KompleksanBroj z)
28 {
29     /* Zapocinje se sa ispisom */
30     printf("(");
31
32     /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja
33      razlicit od nule: tada se realni deo ispisuje na standardni
34      izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
35      imaginarni deo pozitivan ili negativan, a potom i apsolutna
36      vrednost imaginarnog dela kompleksnog broja 2) realni deo
37      kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
38      s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
39      decimalnih mesta */
40
41     if (z.real != 0) {
42         printf("%.2f", z.real);
43
44         if (z.imag > 0)
45             printf(" + %.2f i", z.imag);
46         else if (z.imag < 0)
47             printf(" - %.2f i", -z.imag);
48     } else {
49         if (z.imag == 0)
50             printf("0");
51         else
52             printf("%.2f i", z.imag);
53     }
54
55     /* Zavrsava se sa ispisom */
56     printf(")");
57 }
58
59 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
60 float realan_deo(KompleksanBroj z)
61 {
62     return z.real;
63 }
64
65 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
66 float imaginaran_deo(KompleksanBroj z)
67 {
68     return z.imag;
69 }
70
71 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
72 float moduo(KompleksanBroj z)
73 {
74     return sqrt(z.real * z.real + z.imag * z.imag);

```

## 1 Uvodni zadaci

---

```
76    }
77
78    /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
79     odgovara kompleksnom broju argumentu */
80    KompleksanBroj konjugovan(KompleksanBroj z)
81    {
82        /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
83         imaginarnom delu kompleksnog broja */
84
85        KompleksanBroj z1 = z;
86
87        z1.imag *= -1;
88
89        return z1;
90    }
91
92    /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
93     argumenata funkcije */
94    KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
95    {
96        /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
97         broj ciji je realan deo zbir realnih delova kompleksnih brojeva
98         z1 i z2, a imaginarni deo zbir imaginarnih delova kompleksnih
99         brojeva z1 i z2 */
100
101        KompleksanBroj z = z1;
102
103        z.real += z2.real;
104        z.imag += z2.imag;
105
106        return z;
107    }
108
109    /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
110     argumenata funkcije */
111    KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
112    {
113        /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
114         broj ciji je realan deo razlika realnih delova kompleksnih
115         brojeva z1 i z2, a imaginarni deo razlika imaginarnih delova
116         kompleksnih brojeva z1 i z2 */
117
118        KompleksanBroj z = z1;
119
120        z.real -= z2.real;
121        z.imag -= z2.imag;
122
123        return z;
124    }
125
126    /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
127     argumenata funkcije */
```

```

128 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
129 {
130     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
131        broj ciji se realan i imaginarni deo racunaju po formuli za
132        mnozenje kompleksnih brojeva z1 i z2 */
133
134     KompleksanBroj z;
135
136     z.real = z1.real * z2.real - z1.imag * z2.imag;
137     z.imag = z1.real * z2.imag + z1.imag * z2.real;
138
139     return z;
140 }
141
142 /* Funkcija vraca argument zadatog kompleksnog broja */
143 float argument(KompleksanBroj z)
144 {
145     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
146        iz biblioteke math.h */
147
148     return atan2(z.imag, z.real);
149 }
150
151 int main()
152 {
153     char c;
154
155     /* Deklaracija 3 promenljive tipa KompleksanBroj */
156     KompleksanBroj z1, z2, z;
157
158     /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
159        ispisivanje na standardni izlaz */
160     ucitaj_kompleksan_broj(&z1);
161     ispisi_kompleksan_broj(z1);
162     printf("\n");
163
164     /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
165        na standardni izlaz */
166     ucitaj_kompleksan_broj(&z2);
167     ispisi_kompleksan_broj(z2);
168     printf("\n");
169
170     /* Ucitavanje i provera znaka na osnovu koga korisnik bira
171        aritmeticku operaciju koja ce se izvrsiti nad kompleksnim
172        brojevima */
173     getchar();
174     printf("Unesite znak (+,-,*): ");
175     scanf("%c", &c);
176     if (c != '+' && c != '-' && c != '*') {
177         printf("Greska: nedozvoljena vrednost operatora!\n");
178         exit(EXIT_FAILURE);
179     }

```

## 1 Uvodni zadaci

---

```
180  /* Analizira se uneti operator */
181  if (c == '+') {
182      /* Racuna se zbir */
183      z = saberi(z1, z2);
184  } else if (c == '-') {
185      /* Racuna se razlika */
186      z = oduzmi(z1, z2);
187  } else {
188      /* Racuna se proizvod */
189      z = mnozi(z1, z2);
190  }

191  /* Ispisuje se rezultat */
192  ispisi_kompleksan_broj(z1);
193  printf(" %c ", c);
194  ispisi_kompleksan_broj(z2);
195  printf(" = ");
196  ispisi_kompleksan_broj(z);

197  /* Ispisuje se realan, imaginarni deo i moduo prvog kompleksnog
198  broja */
199  printf("\nRealni_deo: %.f\nImaginarni_deo: %f\nModuo: %f\n",
200        realan_deo(z), imaginarni_deo(z), moduo(z));

201  /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
202  kompleksnog broja */
203  printf("Konjugovano kompleksan broj: ");
204  ispisi_kompleksan_broj(konjugovan(z));
205  printf("\n");

206  /* Testira se funkcija koja racuna argument kompleksnog broja */
207  printf("Argument kompleksnog broja: %f\n", argument(z));

208  exit(EXIT_SUCCESS);
209 }
```

### Rešenje 1.2

*kompleksan\_broj.h*

```
1  /* Zaglavljekompleksan_broj.h sadrzi definiciju tipa KompleksanBroj
2   * i
3   * deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavljek
4   * nikada ne treba da sadrzi definicije funkcija. Da bi neki program
5   * mogao da koristi ove brojeve i funkcije iz ove biblioteke,
6   * neophodno je da ukljuci ovo zaglavlje. */
7
8  /* Ovim pretprocesorskim direktivama se zaključava zaglavljek
9   * onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
```

```

9  posle ključne reci ifndef je proizvoljna, ali treba da se ponovi u
10 narednoj pretrcesorskoj define direktivi. */
11 #ifndef _KOMPLEKSAN_BROJ_H
12 #define _KOMPLEKSAN_BROJ_H
13
14 /* Zaglavljva standardne biblioteke koje sadrže deklaracije funkcija
15 koje se koriste u definicijama funkcija navedenim u
16 kompleksan_broj.c */
17 #include <stdio.h>
18 #include <math.h>
19
20 /* Struktura KompleksanBroj */
21 typedef struct {
22     float real;
23     float imag;
24 } KompleksanBroj;
25
26 /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
27 definisane u kompleksan_broj.c */
28
29 /* Funkcija ucitava sa standardnog ulaza realan i imaginarni deo
30 kompleksnog broja i smesta ih u strukturu cija je adresa argument
31 funkcije */
32 void ucitaj_kompleksan_broj(KompleksanBroj * z);
33
34 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
35 obliku (x + i y) */
36 void ispisi_kompleksan_broj(KompleksanBroj z);
37
38 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
39 float realan_deo(KompleksanBroj z);
40
41 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
42 float imaginarni_deo(KompleksanBroj z);
43
44 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
45 float moduo(KompleksanBroj z);
46
47 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
48 odgovara kompleksnom broju argumentu */
49 KompleksanBroj konjugovan(KompleksanBroj z);
50
51 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
52 argumenata funkcije */
53 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
54
55 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
56 argumenata funkcije */
57 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
58
59 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
60 argumenata funkcije */

```

## 1 Uvodni zadaci

---

```
59 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);  
61 /* Funkcija vraca argument zadatog kompleksnog broja */  
63 float argument(KompleksanBroj z);  
65 /* Kraj zakljucanog dela */  
#endif
```

*kompleksan\_broj.c*

```
1 /* Uključuje se zaglavlje za rad sa kompleksnim brojevima, jer je  
2 neophodno da bude poznata definicija tipa KompleksanBroj. Takođe,  
3 time su uključena zaglavlja standardne biblioteke koja su navedena  
4 u kompleksan_broj.h */  
5 #include "kompleksan_broj.h"  
6  
8 void ucitaj_kompleksan_broj(KompleksanBroj * z)  
{  
10    /* Ucitavanje vrednosti sa standardnog ulaza */  
11    printf("Unesite realan i imaginarni deo kompleksnog broja: ");  
12    scanf("%f", &z->real);  
13    scanf("%f", &z->imag);  
14}  
16 void ispisi_kompleksan_broj(KompleksanBroj z)  
{  
18    /* Zapocinje se sa ispisom */  
19    printf("(");  
20  
22    /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja  
23     razlicit od nule: tada se realni deo ispisuje na standardni  
24     izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je  
25     imaginarni deo pozitivan ili negativan, a potom i apsolutna  
26     vrednost imaginarnog dela kompleksnog broja 2) realni deo  
27     kompleksnog broja je nula: tada se samo ispisuje imaginarni deo,  
28     s tim što se ukoliko su oba dela nula ispisuje samo 0, bez  
29     decimalnih mesta */  
30  
31    if (z.real != 0) {  
32        printf("%.2f", z.real);  
33  
34        if (z.imag > 0)  
35            printf(" + %.2f i", z.imag);  
36        else if (z.imag < 0)  
37            printf(" - %.2f i", -z.imag);  
38    } else {  
39        if (z.imag == 0)  
40            printf("0");  
41        else  
42            printf("%.2f i", z.imag);
```

```

42     }
44     /* Zavrsava se sa ispisom */
45     printf(")");
46 }
47
48 float realan_deo(KompleksanBroj z)
49 {
50     /* Vraca se vrednost realnog dela kompleksnog broja */
51     return z.real;
52 }
53
54 float imaginaran_deo(KompleksanBroj z)
55 {
56     /* Vraca se vrednost imaginarnog dela kompleksnog broja */
57     return z.imag;
58 }
59
60 float moduo(KompleksanBroj z)
61 {
62     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
63     return sqrt(z.real * z.real + z.imag * z.imag);
64 }
65
66 KompleksanBroj konjugovan(KompleksanBroj z)
67 {
68     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
69      se promeni znak imaginarnom delu kompleksnog broja */
70     KompleksanBroj z1 = z;
71     z1.imag *= -1;
72     return z1;
73 }
74
75 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76 {
77     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
78      broj ciji je realan deo zbir realnih delova kompleksnih brojeva
79      z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
80      brojeva z1 i z2 */
81     KompleksanBroj z = z1;
82
83     z.real += z2.real;
84     z.imag += z2.imag;
85
86     return z;
87 }
88
89 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
90 {
91     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
92      broj ciji je realan deo razlika realnih delova kompleksnih
93      brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
94      */
95 }
```

## 1 Uvodni zadaci

---

```
94     kompleksnih brojeva z1 i z2 */
95     KompleksanBroj z = z1;
96     z.real -= z2.real;
97     z.imag -= z2.imag;
98     return z;
99 }
100
101 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
102 {
103     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
104     broj ciji se realan i imaginarni deo racunaju po formuli za
105     mnozenje kompleksnih brojeva z1 i z2 */
106     KompleksanBroj z;
107
108     z.real = z1.real * z2.real - z1.imag * z2.imag;
109     z.imag = z1.real * z2.imag + z1.imag * z2.real;
110
111     return z;
112 }
113
114 float argument(KompleksanBroj z)
115 {
116     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
117      iz biblioteke math.h */
118     return atan2(z.imag, z.real);
119 }
```

*main.c*

```
1 ****
2 Ovaj program koristi korektno definisaniu biblioteku kompleksnih
3 brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
4     kompleksnog broja
5 i popis deklaracija podrzanih funkcija, a u kompleksan_broj.c se
6     nalaze
7 njihove definicije.
8
9 Kompilacija programa se najjednostavnije postize naredbom
10 gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c
11
12 Kompilacija se moze uraditi i na sledeci nacin:
13 gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
14 gcc -Wall -c -o main.o main.c
15 gcc -lm -o kompleksan_broj kompleksan_broj.o main.o
16
17 Napomena: Prethodne komande se koriste kada se sva tri navedena
18 dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
19     header_dir
20 prevodjenje se vrsti dodavanjem opcije -I header_dir
```

```

19 gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c main
     .c
*****
21

23 #include <stdio.h>
/* Uključuje se zaglavlje neophodno za rad sa kompleksnim brojevima
 */
25 #include "kompleksan_broj.h"

27 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
   polarni oblik */
29 int main()
{
31   KompleksanBroj z;

33   /* Ucitavamo kompleksan broj */
34   ucitaj_kompleksan_broj(&z);

35   /* Ispisujemo njegov polarni oblik */
36   printf("Polarni oblik kompleksnog broja je %.2f * e^i * %.2f\n",
          moduo(z), argument(z));
37

39   return 0;
41 }

```

### Rešenje 1.3

*polinom.h*

```

1 #ifndef _POLINOM_H
2 #define _POLINOM_H

3
4 #include <stdio.h>
5 #include <stdlib.h>

7 /* Maksimalni stepen polinoma */
8 #define MAKS_STEPEN 20

11 /* Polinomi se predstavljaju strukturom koja cuva koeficijente
12   (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
13 typedef struct {
14   double koef[MAKS_STEPEN + 1];
15   int stepen;
16 } Polinom;

17 /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
18   obliku. Polinom se prenosi po adresi da bi se usteđela memorija:
19   ne kopira se cela struktura, vec se samo prenosi adresa na kojoj

```

## 1 Uvodni zadaci

---

```
21   se nalazi polinom koji ispisujemo */
22 void ispisi(const Polinom * p);
23
24 /* Funkcija koja ucitava polinom sa tastature */
25 Polinom ucitaj();
26
27 /* Funkcija racuna i vraca vrednost polinoma p u tacki x Hornerovim
28   algoritmom */
29 double izracunaj(const Polinom * p, double x);
30
31 /* Funkcija koja sabira dva polinoma */
32 Polinom saberi(const Polinom * p, const Polinom * q);
33
34 /* Funkcija koja mnozi dva polinoma p i q */
35 Polinom pomnozi(const Polinom * p, const Polinom * q);
36
37 /* Funkcija koja racuna izvod polinoma p */
38 Polinom izvod(const Polinom * p);
39
40 /* Funkcija koja racuna n-ti izvod polinoma p */
41 Polinom n_izvod(const Polinom * p, int n);
42 #endif
```

*polinom.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "polinom.h"
4
5 void ispisi(const Polinom * p)
6 {
7     int nulaPolinom = 1;
8     int i;
9     /* Ispisivanje polinoma pocinje od najviseg stepena ka najnizem da
10       bi polinom bio isписан na prirodan nacin. Ipisuju se samo oni
11       koeficijenti koji su razliciti od nule. Ispred pozitivnih
12       koeficijenata je potrebno ispisati znak + (osim u slucaju
13       koeficijenta uz najvisi stepen). */
14     for (i = p->stepen; i >= 0; i--) {
15
16         if (p->koef[i]) {
17             /* Polinom nije nula polinom, cim je neki od koeficijenata
18               razlicit od nule */
19             nulaPolinom = 0;
20             if (p->koef[i] >= 0 && i != p->stepen)
21                 putchar('+');
22             if (i > 1)
23                 printf("%.2fx^%d", p->koef[i], i);
24             else if (i == 1)
25                 printf("%.2fx", p->koef[i]);
26             else
```

```
    printf("%.2f", p->koef[i]);
28 }
29 /* U slučaju nula polinoma indikator će imati vrednost 1 i tada se
   ispisuje nula. */
30 if(nulaPolinom)
31     printf("0");
32     putchar('\n');
33 }
34
35 Polinom ucitaj()
36 {
37     int i;
38     Polinom p;
39
40     /* Ucitava se stepena polinoma */
41     scanf("%d", &p.stepen);
42
43     /* Ponavlja se učitavanje stepena sve dok se ne unese stepen iz
       dozvoljenog opsega */
44     while (p.stepen > MAKS_STEPEŃ || p.stepen < 0) {
45         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
46         scanf("%d", &p.stepen);
47     }
48
49     /* Unose se koeficijenti polinoma */
50     for (i = p.stepen; i >= 0; i--)
51         scanf("%lf", &p.koef[i]);
52
53     /* Vraca se procitani polinom */
54     return p;
55 }
56
57 double izracunaj(const Polinom * p, double x)
58 {
59     /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
       svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
       vrednost odgovarajućeg koeficijenta */
60
61     /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
        $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1$  */
62
63     double rezultat = 0;
64     int i = p->stepen;
65     for (; i >= 0; i--)
66         rezultat = rezultat * x + p->koef[i];
67     return rezultat;
68 }
69
70 Polinom saberi(const Polinom * p, const Polinom * q)
71 {
72     Polinom rez;
```

## 1 Uvodni zadaci

---

```
int i;

/* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
   stepenom */
rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

/* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
   sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
   sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
   od stepena nekog od polaznih polinoma podrazumeva se da je
   koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
   polinoma */
for (i = 0; i <= rez.stepen; i++)
    rez.koef[i] =
        (i > p->stepen ? 0 : p->koef[i]) +
        (i > q->stepen ? 0 : q->koef[i]);

/* Vraca se dobijeni polinom */
return rez;

}

Polinom pomnozi(const Polinom * p, const Polinom * q)
{
    int i, j;
    Polinom r;

    /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
     */
    r.stepen = p->stepen + q->stepen;
    if (r.stepen > MAKS_STEPEN) {
        fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
        exit(EXIT_FAILURE);
    }

    /* Svi koeficijenti rezultujuceg polinoma se inicializuju na nulu
     */
    for (i = 0; i <= r.stepen; i++)
        r.koef[i] = 0;

    /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
       za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
    for (i = 0; i <= p->stepen; i++)
        for (j = 0; j <= q->stepen; j++)
            r.koef[i + j] += p->koef[i] * q->koef[j];

    /* Vraca se dobijeni polinom */
    return r;
}

Polinom izvod(const Polinom * p)
{
```

```

130     int i;
131     Polinom r;
132
133     /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
134        polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
135        je rezultujuci polinom nula (izvod od konstante je nula). */
136     if (p->stepen > 0) {
137         r.stepen = p->stepen - 1;
138
139         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
140            polaznog polinoma */
141         for (i = 0; i <= r.stepen; i++)
142             r.koef[i] = (i + 1) * p->koef[i + 1];
143     } else
144         r.koef[0] = r.stepen = 0;
145
146     /* Vraca se dobijeni polinom */
147     return r;
148 }
149
150 Polinom n_izvod(const Polinom * p, int n)
151 {
152     int i;
153     Polinom r;
154
155     /* Provera da li je n nenegativna vrednost */
156     if (n < 0) {
157         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
158         exit(EXIT_FAILURE);
159     }
160
161     /* Nulti izvod je bas taj polinom */
162     if (n == 0)
163         return *p;
164
165     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove funkcija
166        za racunanje prvog izvoda polinoma */
167     r = izvod(p);
168     for (i = 1; i < n; i++)
169         r = izvod(&r);
170
171     /* Vraca se dobijeni polinom */
172     return r;
173 }
```

*main.c*

```

2 #include <stdio.h>
3 #include "polinom.h"
4
5 int main(int argc, char **argv)
```

## 1 Uvodni zadaci

---

```
1 {
6   Polinom p, q, z, r;
7   double x;
8   int n;
9
10  /* Unos polinoma p */
11  printf
12    ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
13      najveceg stepena do nultog):\n");
14  p = ucitaj();
15
16  /* Ispis polinoma p */
17  ispisi(&p);
18
19  /* Unos polinoma q */
20  printf
21    ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
22      od najveceg stepena do nultog):\n");
23  q = ucitaj();
24
25  /* Polinomi se sabiraju i ispisuje se izracunati zbir */
26  z = saberi(&p, &q);
27  printf("Zbir polinoma je polinom z:\n");
28  ispisi(&z);
29
30  /* Polinomi se mnoze i ispisuje se izracunati prozivod */
31  r = pomnozi(&p, &q);
32  printf("Prozvod polinoma je polinom r:\n");
33  ispisi(&r);
34
35  /* Ispisuje se vrednost polinoma u unetoj tacki */
36  printf("Unesite tacku u kojoj racunate vrednost polinoma z:\n");
37  scanf("%lf", &x);
38  printf("Vrednost polinoma z u tacki %.2f je %.2f\n", x, izracunaj(&
39    z, x));
40
41  /* Racuna se n-ti izvod polinoma i ispisuje se dobijeni polinoma
42    */
43  printf("Unesite izvod polinoma koji zelite:\n");
44  scanf("%d", &n);
45  r = n_izvod(&r, n);
46  printf("%d. izvod polinoma r je: ", n);
47  ispisi(&r);
48
49  exit(EXIT_SUCCESS);
50 }
```

Rešenje 1.5

*stampanje\_bitova.h*

```

1 #ifndef _STAMPANJE_BITOVA_H
2 #define _STAMPANJE_BITOVA_H
3
4 #include <stdio.h>
5
6 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
7   celog broja u memoriji. Bitove koji predstavljaju binarnu
8   reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
9   najveće tezine ka bitu najmanje tezine */
10 void stampaj_bitove(unsigned x);
11
12 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
13   celog broja tipa 'short' u memoriji. */
14 void stampaj_bitove_short(short x);
15
16 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
17   karaktera u memoriji. */
18 void stampaj_bitove_char(char x);
19
20#endif

```

*stampanje\_bitova.c*

```

1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 void stampaj_bitove(unsigned x)
5 {
6     /* Broj bitova celog broja */
7     unsigned velicina = sizeof(unsigned) * 8;
8
9     /* Maska koja se koristi za "ocitavanje" bitova celog broja */
10    unsigned maska;
11
12    /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
13       na mestu bita najveće tezine sadrzi jedinicu, a na svim ostalim
14       mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
15       se jedini bit jedinica pomera udesno, kako bi se odredio naredni
16       bit broja x koji je argument funkcije. Zatim se odgovarajuca
17       cifra, ('0' ili '1'), ispisuje na standardnom izlazu. Neophodno
18       je da promenljiva maska bude deklarisana kao neoznacen ceo broj
19       kako bi se pomeranjem u desno vrsilo logicko pomeranje
20       (popunjavanje nulama), a ne aritmeticko pomeranje (popunjavanje
21       znakom broja). */
22    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
23        putchar(x & maska ? '1' : '0');
24
25    putchar('\n');

```

## 1 Uvodni zadaci

---

```
26 }
27
28 void stampaj_bitove_short(short x)
29 {
30     /* Broj bitova celog broja tipa short */
31     unsigned velicina = sizeof(short) * 8;
32
33     /* Maska koja se koristi za "ocitavanje" bitova broja tipa short */
34     unsigned short maska;
35
36     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
37         putchar(x & maska ? '1' : '0');
38
39     putchar('\n');
40 }
41
42 void stampaj_bitove_char(char x)
43 {
44     /* Broj bitova karaktera */
45     unsigned velicina = sizeof(char) * 8;
46
47     /* Maska koja se koristi za "ocitavanje" bitova jednog karaktera */
48     unsigned char maska;
49
50     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
51         putchar(x & maska ? '1' : '0');
52
53     putchar('\n');
54 }
```

*main.c*

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

int main()
{
    int broj_int;
    short broj_short;
    char broj_char;

    printf("Unesite broj tipa int: ");
    /* Ucitava se broj sa ulaza */
    scanf("%x", &broj_int);

    /* I ispisuje se njegova binarna reprezentacija */
    printf("Binarna reprezentacija: ");
    stampaj_bitove(broj_int);

    printf("Unesite broj tipa short: ");
    /* Ucitava se broj sa ulaza */
```

```

20    scanf("%hx", &broj_short);

22    /* I ispisuje se njegova binarna reprezentacija */
23    printf("Binarna reprezentacija: ");
24    stampaj_bitove_short(broj_short);

26    printf("Unesite broj tipa char: ");
27    /* Ucitava se broj sa ulaza */
28    scanf("%hhx", &broj_char);

30    /* I ispisuje se njegova binarna reprezentacija */
31    printf("Binarna reprezentacija: ");
32    stampaj_bitove_char(broj_char);

34    return 0;
}

```

### Rešenje 1.6

```

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

3
4    /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
5       kreiranjem odgovarajuce maske i njenim pomeranjem */
6    int prebroj_bitove_1(int x)
7    {
8        int br = 0;
9        unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;
10
11       /* Formiranje se maska cija binarna reprezentacija izgleda
12          100000...0000000, koja sluzi za ocitavanje bita najvece tezine.
13          U svakoj iteraciji maska se pomera u desno za 1 mesto, i
14          ocitavamo sledeci bit. Petlja se zavrsava kada vise nema
15          jedinica tj. kada maska postane nula. */
16       unsigned maska = 1 << broj_pomeranja;
17       for (; maska != 0; maska >>= 1)
18           x & maska ? br++ : 1;
19
20       return br;
21   }

23   /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
24      formiranjem odgovarajuce maske i pomeranjem promenljive x */
25   int prebroj_bitove_2(int x)
26   {
27       int br = 0;
28       unsigned broj_pomeranja = sizeof(int) * 8 - 1;
29
30       /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1 bi
31          vrsila aritmeticko pomeranje u desno, tj. popunjavanje bita
32          najvece tezine bitom znaka. U tom slucaju nikad ne bi bio

```

## 1 Uvodni zadaci

---

```
33     ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
34     petlji. Zbog toga se koristi pomeranje broja x uлево i maska
35     koja ocitava bit najvece tezine. */
36
37     unsigned maska = 1 << broj_pomeranja;
38     for ( ; x != 0; x <<= 1)
39         x & maska ? br++ : 1;
40
41     return br;
42 }
43
44 int main()
45 {
46     int x, i;
47
48     /* Ucitava se broj sa ulaza */
49     printf("Unesite broj:\n");
50     scanf("%x", &x);
51
52     /* Dozvoljava se korisniku da bira na koji nacin ce biti izracunat
53      broj jedinica u zapisu broja */
54     printf("Unesite redni broj funkcije:\n");
55     scanf("%d", &i);
56
57     /* Ispisuje se rezultat */
58     if (i == 1){
59         printf("Poziva se funkcija prebroj_bitove_1\n");
60         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
61     }else if (i == 2){
62         printf("Poziva se funkcija prebroj_bitove_2\n");
63         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
64     }else {
65         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
66         exit(EXIT_FAILURE);
67     }
68
69     exit(EXIT_SUCCESS);
70 }
```

**Rešenje 1.7** NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```
1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija vraca najveci neoznacen broj sastavljen od istih bitova
5  koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
6 unsigned najveci(unsigned x)
7 {
8     unsigned velicina = sizeof(unsigned) * 8;
```

```

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

13 /* Rezultat se inicializuje vrednoscu 0 */
14 unsigned rezultat = 0;

15 /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj
16 binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
17 od nule). */
18 for (; x != 0; x <= 1) {
19     /* Za svaku jedinicu koja se koriscenjem maske detektuje na
20        poziciji najvece tezine u binarnoj reprezentaciji promenjive
21        x, potiskuje se jedna nova jedinicu sa leva u rezultat */
22     if (x & maska) {
23         rezultat >= 1;
24         rezultat |= maska;
25     }
26 }

29 /* Vraca se dobijena vrednost */
30 return rezultat;
31 }

33 /* Funkcija vraca najmanji neoznacen broj sastavljen od istih bitova
34   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
35 unsigned najmanji(unsigned x)
36 {
37     /* Rezultat se inicializuje vrednoscu 0 */
38     unsigned rezultat = 0;

39     /* Promenljiva x se pomera u desno sve dok postoje jedinice u
40       njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
41       razlicita od nule). */
42     for (; x != 0; x >= 1) {
43         /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
44            detektuje na poziciji najmanje tezine u binarnoj
45            reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
46            sa desna u rezultat */
47         if (x & 1) {
48             rezultat <= 1;
49             rezultat |= 1;
50         }
51     }

53     /* Vraca se dobijena vrednost */
54     return rezultat;
55 }

57 int main()
58 {
59     int broj;

```

## 1 Uvodni zadaci

---

```
61  /* Ucitava se broj sa ulaza */
63  scanf("%x", &broj);

65  /* Ispisuju se, redom, najveci i najmanji broj formirani od bitova
   unetog broja */
67  printf("Najveci:\n");
68  stampaj_bitove(najveci(broj));

69  printf("Najmanji:\n");
70  stampaj_bitove(najmanji(broj));

72  return 0;
73 }
```

**Rešenje 1.8**      NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```
1 #include <stdio.h>
2 #include "stampanje_bitova.h"

4 /* Funckija postavlja na nulu n bitova pocev od pozicije p. */
5 unsigned postavi_0(unsigned x, unsigned n, unsigned p)
6 {
7  *****
8  Formira se maska cija binarna reprezentacija ima n bitova
9  postavljenih na 0 pocev od pozicije p, dok su svi ostali
10 postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
11 1111 1111 1111 1111 1111 1000 0011 1111
12 To se postize na sledeci nacin:
13          1111 1111 1111 1111 1111 1111 1111 1111
14 (~0 << n)          1111 1111 1111 1111 1111 1111 1110 0000
15 ~(~0 << n)          0000 0000 0000 0000 0000 0000 0001 1111
16 (~(~0 << n) << (p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000
17 ~(~(~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
18 *****
19 unsigned maska = ~(~(~0 << n) << (p - n + 1));
20
21 return x & maska;
22 }

24 /* Funckija postavlja na jedinicu n bitova pocev od pozicije p. */
25 unsigned postavi_1(unsigned x, unsigned n, unsigned p)
26 {
27 *****
28  Formira se maska kod koje je samo n bitova pocev od pocev od
29  pozicije p jednako 1, a ostali su 0.
30  Na primer, za n=5 i p=10 formira se maska oblika
31  0000 0000 0000 0000 0000 0111 1100 0000
32 }
```

```

34    ****
35    unsigned maska = ~(~0 << n) << (p - n + 1);
36
37    return x | maska;
38 }
39
40 /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
41 predstavlja n bitova pocev od pozicije p u binarnoj
42 reprezentaciji broja x. */
43 unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
44 {
45
46     ****
47     Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
48     Na primer, za n=5
49     0000 0000 0000 0000 0000 0001 1111
50     ****
51     unsigned maska = ~(~0 << n);
52
53     /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
54     polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
55     zeljenih n i funkcija vraca tako dobijenu vrednost */
56     return maska & (x >> (p - n + 1));
57 }
58
59 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
60 postavljeni na vrednosti n bitova najmanje tezine binarne
61 reprezentacije broja y */
62 unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
63     unsigned y)
64 {
65     /* Kreira se maska kod kod koje su poslednjih n bitova 1, a
66     ostali su 0. */
67     unsigned poslednjih_n_1 = ~(~0 << n);
68
69     /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima n
70     bitova postavljenih na 0 pocevsi od pozicije p, dok su
71     ostali bitovi 1. */
72     unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));
73
74     /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se u
75     binarnoj reprezentaciji vrednosti promenljive x postavi na 0 n
76     bitova na pozicijama pocev od p */
77     unsigned x_postavi_0 = x & srednjih_n_0;
78
79     /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
80     binarne reprezentacije vrednosti promenljive y cijih je n bitova
81     najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
82     bitovi su nule. Sa (y & poslednjih_n_1) postave na 0 svi bitovi
83     osim najnizih n */
84     unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);

```

## 1 Uvodni zadaci

---

```
84     return x_postavi_0 ^ y_pomeri_srednje;
85 }
86
87 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
88    njih n */
89 unsigned invertuj(unsigned x, unsigned n, unsigned p)
90 {
91     /* Formira se maska sa n jedinica pocev od pozicije p. */
92     unsigned maska = ~(~0 << n) << (p - n + 1);
93
94     /* Operator ekskluzivno ili invertuje sve bitove gde je
95        odgovarajuci bit maske 1. Ostali bitovi ostaju nepromjenjeni. */
96     return maska ^ x;
97 }
98
99 int main()
100 {
101     unsigned x, p, n, y;
102
103     /* Ucitavaju se vrednosti sa standardnog ulaza */
104     printf("Unesite neoznacen ceo broj x:\n");
105     scanf("%u", &x);
106     printf("Unesite neoznacen ceo broj n:\n");
107     scanf("%u", &n);
108     printf("Unesite neoznacen ceo broj p:\n");
109     scanf("%u", &p);
110     printf("Unesite neoznacen ceo broj y:\n");
111     scanf("%u", &y);
112
113     /* Ispisuju se binarne reprezentacije broja x i broja koji se
114        dobije kada se primeni funkcija postavi_0 za x, n i p*/
115     printf("x = %10u %36s = ", x, "");
116     stampaj_bitove(x);
117     printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
118     stampaj_bitove( postavi_0(x, n, p));
119     printf("\n");
120
121     /* Ispisuju se binarne reprezentacije broja x i broja koji se
122        dobije kada se primeni funkcija postavi_1 za x, n i p*/
123     printf("x = %10u %36s = ", x, "");
124     stampaj_bitove(x);
125     printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
126     stampaj_bitove( postavi_1(x, n, p));
127     printf("\n");
128
129     /* Ispisuju se binarne reprezentacije broja x i broja koji se
130        dobije kada se primeni funkcija vrati_bitove za x, n i p*/
131     printf("x = %10u %36s = ", x, "");
132     stampaj_bitove(x);
133     printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
134     stampaj_bitove( vrati_bitove(x, n, p));
135     printf("\n");
```

```

136  /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
137   dobije kada se primeni funkcija postavi_1_n_bitova za x, n i p*/
138   printf("x = %10u %36s = ", x, "");
139   stampaj_bitove(x);
140   printf("y = %10u %36s = ", y, "");
141   stampaj_bitove(y);
142   printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
143   stampaj_bitove( postavi_1_n_bitova(x, n, p, y));
144   printf("\n");
145
146  /* Ispisuju se binarne reprezentacije broja x i broja koji se
147   dobije kada se primeni funkcija invertuj za x, n i p*/
148  printf("x = %10u %36s = ", x, "");
149  stampaj_bitove(x);
150  printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
151  stampaj_bitove( invertuj(x, n, p));
152
153  return 0;
154 }
```

**Rešenje 1.9** NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```

1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija ceo broj x rotira u levo za n mesta. */
5 unsigned rotiraj_ulevo(int x, unsigned n)
6 {
7     unsigned bit_najvece_tezine;
8
9     /* Maska koja ima samo bit na poziciji najvece tezine postavljen na
10      1 je neophodna da bi pre pomeranja u levo za 1 bit na poziciji
11      najvece tezine bio sacuvan */
12     unsigned bit_najvece_tezine_maska = 1 << (sizeof(unsigned) * 8 - 1)
13     ;
14     int i;
15
16     /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
17      se odredi bit na poziciji najvece tezine, a potom se pomera
18      binarna reprezentacija trenutne vrednosti promenljive x u levo
19      za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
20      na vrednost koju je imao bit na poziciji najvece tezine koji je
21      istisnut pomeranjem */
22     for (i = 0; i < n; i++) {
23         bit_najvece_tezine = x & bit_najvece_tezine_maska;
24         x = x << 1 | (bit_najvece_tezine ? 1 : 0);
25     }
26 }
```

## 1 Uvodni zadaci

---

```
26     /* Vraca se dobijena vrednost */
27     return x;
28 }

29 /* Funkcija neoznacen broj x rotira u desno za n mesta. */
30 unsigned rotiraj_udesno(unsigned x, unsigned n)
31 {
32     unsigned bit_najmanje_tezine;
33     int i;

34     /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
35      iteraciji se određuje bit na poziciji najmanje tezine broja x,
36      zatim tako određeni bit se pomera u levo tako da bit na
37      poziciji najmanje tezine dodje do pozicije najveće tezine.
38      Zatim, nakon pomeranja binarne reprezentacije trenutne vrednosti
39      promenljive x za 1 u desno, bit na poziciji najveće tezine se
40      postavlja na vrednost vec zapamcenog bita koji je bio na
41      poziciji najmanje tezine. */
42     for (i = 0; i < n; i++) {
43         bit_najmanje_tezine = x & 1;
44         x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
45     }

46     /* Vraca se dobijena vrednost */
47     return x;
48 }

49 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
50    argument funkcije x označeni ceo broj */
51 int rotiraj_udesno_oznaceni(int x, unsigned n)
52 {
53     unsigned bit_najmanje_tezine;
54     int i;

55     /* U svakoj iteraciji se određuje bit na poziciji najmanje tezine
56      i smesta u promenljivu bit_najmanje_tezine. Kako je x označen
57      ceo broj, tada se prilikom pomeranja u desno vrši aritmeticko
58      pomeranje i cuva se znak broja. Dakle, razlikuju se dva slučaja
59      u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
60      izvrši pre petlje, s obzirom da rotiranjem u desno na poziciju
61      nejveće tezine može doći i 0 i 1, nezavisno od početnog znaka
62      broja smestenog u promenljivu x. */
63     for (i = 0; i < n; i++) {
64         bit_najmanje_tezine = x & 1;

65         if (x < 0)
66             *****
67             Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
68             na poziciji najveće tezine. Na primer ako je x
69             1010 1011 1100 1101 1110 0001 0010 001b
70             (sa b je označen ili 1 ili 0 na poziciji najmanje tezine)
71             Onda je sadrzaj promenljive bit_najmanje_tezine:
```

```

78      0000 0000 0000 0000 0000 0000 0000 000b
Nakon siftovanja sadrzaja promenljive x za 1 u desno
80      1101 0101 1110 0110 1111 0000 1001 0001
82      Kako bi umesto 1 na poziciji najvece tezine u trenutnoj binarnoj
     reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
     poziciju najvece tezine jer bi se time dobile 0, a u ovom slučaju
     su potrebne jedinice zbog bitovskog & zato se prvo vrši
     komplementiranje, a zatim pomeranje
86      ~bit_najmanje_tezine << (sizeof(int)*8 -1)
B000 0000 0000 0000 0000 0000 0000 0000
88      gde B označava ~b.
Potom se ponovo vrši komplementiranje kako bi se b nalazilo na
90      poziciji najveće tezine i sve jedinice na ostalim pozicijama
~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
92      b111 1111 1111 1111 1111 1111 1111 1111
*****x = (x >> 1) & ~(~bit_najmanje_tezine << (sizeof(int) * 8 - 1))
94      ;
95      else
96          x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
}
98
/* Vraca se dobijena vrednost */
100    return x;
}
102
int main()
{
    unsigned x, n;

106
/* Ucitavaju se vrednosti sa standardnog ulaza */
108    printf("Unesite neoznacen ceo broj x:");
scanf("%x", &x);
110    printf("Unesite neoznacen ceo broj n:");
scanf("%x", &n);

112
/* Ispisuje se binarna reprezentacija broja x */
114    printf("x\t\t\t\t= ");
stampaj_bitove(x);

116
/* Testira se rad napisanih funkcija */
118    printf("rotiraj_ulevo(%x,%u)\t\t= ", x, n);
stampaj_bitove(rotiraj_ulevo(x, n));

120
printf("rotiraj_udesno(%x,%u)\t\t= ", x, n);
stampaj_bitove(rotiraj_udesno(x, n));

124
printf("rotiraj_udesno_oznaceni(%x,%u)\t\t= ", x, n);
stampaj_bitove(rotiraj_udesno_oznaceni(x, n));

126
printf("rotiraj_udesno_oznaceni(-%x,%u)\t\t= ", x, n);
stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));
}

```

## 1 Uvodni zadaci

---

```
130     return 0;
}
```

**Rešenje 1.10** NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```
1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija vraca vrednost cija je binarna reprezentacija slika u
5    ogledalu binarne reprezentacije broja x. */
6 unsigned ogledalo(unsigned x)
7 {
8     unsigned najnizi_bit;
9     unsigned rezultat = 0;
10
11    int i;
12    /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuce
13       vrednosti broja x se odredjuje i pamti u promenljivoj
14       najnizi_bit, nakon cega se na promenljivu x primeni pomeranje u
15       desno */
16    for (i = 0; i < sizeof(x) * 8; i++) {
17        najnizi_bit = x & 1;
18        x >>= 1;
19        /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
20           postavljeni bitovi dobijaju vecu poziciju. Novi bit se
21           postavlja na najnizu poziciju */
22        rezultat <=> 1;
23        rezultat |= najnizi_bit;
24    }
25
26    /* Vraca se dobijena vrednost */
27    return rezultat;
28}
29
30 int main()
31 {
32     int broj;
33
34     /* Ucitava se broj sa ulaza */
35     scanf("%x", &broj);
36
37     /* Ispisuje se njegova binarna reprezentacija */
38     stampaj_bitove(broj);
39
40     /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
41       funkcije ogledalo */
42     stampaj_bitove(ogledalo(broj));
43 }
```

```
45     return 0;
}
```

### Rešenje 1.11

```
#include <stdio.h>
1
2 /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
   3   jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
4 int broj_01(unsigned int n)
5 {
6     int broj_nula, broj_jedinica;
7     unsigned int maska;
8
9     broj_nula = 0;
10    broj_jedinica = 0;
11
12    /* Masku je inicializovana tako da moze da analizira bit najvece
13       tezine */
14    maska = 1 << (sizeof(unsigned int) * 8 - 1);
15
16    /* Cilj je proći kroz sve bitove broja x, zato se maska u svakoj
17       iteraciji pomera u desno pa će jedini bit koji je postavljen na
18       1 biti na svim pozicijama u binarnoj reprezentaciji maske */
19    while (maska != 0) {
20
21        /* Provera da li se na poziciji koju određuje maska nalazi 0 ili
22           1 i uveća se odgovarajući brojac */
23        if (n & maska) {
24            broj_jedinica++;
25        } else {
26            broj_nula++;
27        }
28
29        /* Pomera se maska u desnu stranu */
30        maska = maska >> 1;
31    }
32
33    /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
34       suprotnom vraca 0 */
35    return (broj_jedinica > broj_nula) ? 1 : 0;
36}
37
38int main()
39{
40    unsigned int n;
41
42    /* Ucitava se broj sa ulaza */
43    scanf("%u", &n);
44
45    /* Ispisuje se rezultat */
46}
```

## 1 Uvodni zadaci

---

```
    printf("%d\n", broj_01(n));
48
49     return 0;
50 }
```

### Rešenje 1.12

```
#include <stdio.h>
2
3 /* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju u
4  binarnom zapisu celog čneoznaenog broja x */
5 int broj_parova(unsigned int x)
6 {
7     int broj_parova;
8     unsigned int maska;
9
10    /* Vrednost promenljive koja predstavlja broj parova se
11       inicijalizuje na 0 */
12    broj_parova = 0;
13
14    /* Postavlja se maska tako da moze da procitamo da li su dva
15       najmanja bita u zapisu broja x 11 */
16    /* Binarna reprezentacija broja 3 je 000....00011 */
17    maska = 3;
18
19    while (x != 0) {
20        /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
21           */
22        if ((x & maska) == maska) {
23            broj_parova++;
24        }
25
26        /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
27           proveravao sledeći par bitova. Pomeranjem u desno bit najveće
28           tezine se popunjava nulom jer je x neoznacen broj */
29        x = x >> 1;
30    }
31
32    /* Vraca se dobijena vrednost */
33    return broj_parova;
34}
35
36 int main()
37 {
38     unsigned int x;
39
40     /* Ucitava se broj sa ulaza */
41     scanf("%u", &x);
42
43     /* Ispisuje se rezultat */
44     printf("%d\n", broj_parova(x));
```

```

44     return 0;
45 }

```

### Rešenje 1.14

```

1 #include <stdio.h>
2
3 /* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
4    su za svaku heksadekadnu cifru potrebne 4 binarne cifre i jedna
5    dodatna pozicija za terminirajucu nulu.
6    Prethodni izraz se moze zapisati kao sizeof(unsigned int)*2+1. */
7 #define MAKS_DUZINA sizeof(unsigned int)*2 +1
8
9 /* Funkcija za neoznacen broj x formira nisku s koja sadrzi njegov
10   heksadekadni zapis */
11 void prevod(unsigned int x, char s[])
12 {
13     int i;
14     unsigned int maska;
15     int vrednost;
16
17     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
18       maska za citanje 4 uzastopne cifre */
19     maska = 15;
20
21     /***** Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
22       tezine. Na primer za broj cija je binarna reprezentacija
23       00000000001101000100001111010101
24       u prvom koraku se citaju bitovi izdvojeni sa <...>:
25       0000000000110100010000111101<0101>
26       u drugom koraku:
27       000000000011010001000011<1101>0101
28       u trećem koraku:
29       00000000001101000100<0011>11010101 i tako redom...
30
31     Indeks i označava poziciju na koju se smesta vrednost.
32     *****/
33     for (i = MAKS_DUZINA - 2; i >= 0; i--) {
34         /* Vrednost izdvojene cifre */
35         vrednost = x & maska;
36
37         /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
38            dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
39            10 do 15 odgovarajuci karakter se dobija tako sto se prvo
40            oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na takoj
41            dobijenu vrednost doda ASCII kod 'A' (time se dobija
42            odgovarajuce slovo 'A', 'B', ... 'F') */
43         if (vrednost < 10) {
44             s[i] = vrednost + '0';

```

## 1 Uvodni zadaci

---

```
46     } else {
47         s[i] = vrednost - 10 + 'A';
48     }
49
50     /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u
51      narednoj iteraciji posmatraju sledeca 4 bita */
52     x = x >> 4;
53 }
54
55 /* Upisuje se terminirajuca nula */
56 s[MAKS_DUZINA - 1] = '\0';
57 }
58
59 int main()
60 {
61     unsigned int x;
62     char s[MAKS_DUZINA];
63
64     /* Ucitava se broj sa ulaza */
65     scanf("%u", &x);
66
67     /* Poziva se funkcija za prevodjenje */
68     prevod(x, s);
69
70     /* I stampa se dobijena niska */
71     printf("%s\n", s);
72
73     return 0;
74 }
```

### Rešenje 1.17

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

4 /***** Resenje linearne slozenosti: *****
5   x^0 = 1
6   x^k = x * x^(k-1)
7 *****/
8 int stepen(int x, int k)
9 {
10     if (k == 0)
11         return 1;
12
13     return x * stepen(x, k - 1);
14     /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
15 }
16
17 /***** Resenje logaritamske slozenosti: *****
18
```

```

20     x^0 =1;
21     x^k = x * (x^2 )^(k/2) , za neparno k
22     x^k = (x^2)^{(k/2)} , za parno k
23     Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
24     se doslo do rezultata, i stoga je efikasnije.
25 ****
26 int stepen_2(int x, int k)
27 {
28     if (k == 0)
29         return 1;
30
31     /* Ako je stepen paran */
32     if ((k % 2) == 0)
33         return stepen_2(x * x, k / 2);
34
35     /* Inace (ukoliko je stepen neparan) */
36     return x * stepen_2(x * x, k / 2);
37 }
38
39 int main()
40 {
41     int x, k, ind;
42
43     /* Ucitava se redni broj funkcije koja ce se primeniti */
44     printf("Unesite redni broj funkcije (1/2):\n");
45     scanf("%d", &ind);
46
47     /* Ucitavaju se vrednosti za x i k */
48     printf("Unesite broj x:\n");
49     scanf("%d%d", &x);
50     printf("Unesite broj k:\n");
51     scanf("%d%d", &k);
52
53     /* Ispisuje se vrednost koju vraca odgovarajuci funkcija */
54     if (x == 1)
55         printf("%d\n", stepen(x, k));
56     else if (x == 2)
57         printf("%d\n", stepen_2(x, k));
58     else {
59         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
60         exit(EXIT_FAILURE);
61     }
62
63     exit(EXIT_SUCCESS);
64 }
```

**Rešenje 1.18**

```

1 #include <stdio.h>
2
3 /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
```

## 1 Uvodni zadaci

---

```
4     (posrednu) rekurziju */

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

12 /* Funkcija vraca 1 ako broj n ima paran broj cifara, inace vraca 0
13 */
14 unsigned paran(unsigned n)
15 {
16     if (n <= 9)
17         return 0;
18     else
19         return neparan(n / 10);
20 }

22 /* Funkcija vraca 1 ako broj n ima neparan broj cifara, inace vraca
23 0 */
24 unsigned neparan(unsigned n)
25 {
26     if (n <= 9)
27         return 1;
28     else
29         return paran(n / 10);
30 }

32 int main()
33 {
34     int n;

36     /* Ucitava se ceo broj */
37     scanf("%d", &n);

39     /* Ispisuje se rezultat */
40     printf("Uneti broj ima %sparan broj cifara.\n",
41           (paran(n) == 1 ? "" : "ne"));

43     return 0;
44 }
```

### Rešenje 1.19

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
5  rekurziju. Result je argument u kome se akumulira do tada
6  izracunatu vrednost faktorijela. Kada dodje do izlaza iz rekurzije
7  iz rekurzije potrebno je da vratimo result. */
```

```

1 int faktorijel_repna(int n, int result)
2 {
3     if (n == 0)
4         return result;
5
6     return faktorijel_repna(n - 1, n * result);
7 }
8
9 /* U sledecim funkcijama je prikazan postupak oslobođanja od
10    repne rekurzije koja postoji u funkciji faktorijel_repna. */
11
12 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
13    naredbama kojima se vrednost argumenta funkcije postavlja na
14    vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
15    goto naredbe za vracanje na pocetak tela funkcije. */
16
17 int faktorijel_repna_v1(int n, int result)
18 {
19     pocetak:
20         if (n == 0)
21             return result;
22
23         result = n * result;
24         n = n - 1;
25         goto pocetak;
26 }
27
28 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
29    praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
30    iterativno resenje bez bezuslovnih skokova */
31
32 int faktorijel_repna_v2(int n, int result)
33 {
34     while (n != 0) {
35         result = n * result;
36         n = n - 1;
37     }
38
39     return result;
40 }
41
42
43 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
44    broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
45    ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
46    udobnosti korisnika, jer je sasvim prirodno da za faktorijel
47    zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
48    sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
49    faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
50    funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
51    poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
52    pozivom funkcije faktorijel_repna_v2. */
53
54 int faktorijel(int n)
55 {
56     return faktorijel_repna(n, 1);
57 }
```

## 1 Uvodni zadaci

---

```
61    }
62
63 int main()
64 {
65     int n;
66
67     /* Ucitava se ceo broj */
68     printf("Unesite n (<= 12): ");
69     scanf("%d", &n);
70     if (n > 12) {
71         fprintf(stderr, "Greska: nedozvoljena vrednost za n!\n");
72         exit(EXIT_FAILURE);
73     }
74
75     /* Ispisuje se rezultat */
76     printf("%d! = %d\n", n, faktorijel(n));
77
78     exit(EXIT_SUCCESS);
79 }
```

### Rešenje 1.21

```
1 #include <stdio.h>
2
3 /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
4 int f_iterativna(int n, int a, int b)
5 {
6     int i;
7     int f_0 = 0;
8     int f_1 = 1;
9     int tmp;
10
11    if (n == 0)
12        return 0;
13
14    for (i = 2; i <= n; i++) {
15        tmp = a * f_1 + b * f_0;
16        f_0 = f_1;
17        f_1 = tmp;
18    }
19
20    return f_1;
21}
22
23 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
24 int f_rekursivna(int n, int a, int b)
25 {
26     /* Izlaz iz rekurzije */
27     if (n == 0 || n == 1)
28         return n;
29 }
```

```

/* Rekurzivni pozivi */
31 return a * f_rekurzivna(n - 1, a, b) +
      b * f_rekurzivna(n - 2, a, b);
33 }

35 /* NAPOMENA: U slučaju da se rekurzijom problem svodi na vise manjih
36 podproblema koji se mogu preklapati, postoji opasnost da se
37 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
38 Npr.  $F(20) = a \cdot F(19) + b \cdot F(18)$ , a  $F(19) = a \cdot F(18) + b \cdot F(17)$ , tj.
39 problem  $F(18)$  se resava dva puta! Problemi manjih
40 dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
41 problem je kombinacija rekurzije sa dinamickim programiranjem.
42 Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
43 memoriji (obicno u nizovima ili matricama), odakle se koriste ako
44 tokom resavanja ponovo budu potrebni.

45 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
46 statickom nizu celih brojeva, jer se staticki niz ne smesta
47 na stek, kao sto je to slučaj sa lokalnim promenljivama, vec na
48 segment podataka, odakle je dostupan svim pozivima
49 rekurzivne funkcije. */

51 /* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
52 verzija */
53 int f_napredna(int n, int a, int b)
54 {
55     /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
56     staticke promenljive na podrazumevane vrednosti. Stoga, elemente
57     celobrojnog niza inicijalizuje na 0 */
58     static int f[20];

59     /* Ako je podproblem vec ranije resen, koristi se resenje koje je
60     vec izracunato i */
61     if (f[n] != 0)
62         return f[n];

63     /* Izlaz iz rekurzije */
64     if (n == 0 || n == 1)
65         return f[n] = n;

66     /* Rekurzivni pozivi */
67     return f[n] =
68         a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
69 }

70 int main()
71 {
72     int n, a, b, ind;

73     /* Unosi se redni broj funkcije koja ce se primeniti */
74     printf("Unesite redni broj funkcije koju zelite:\n");
75     printf("1 - iterativna\n");

```

## 1 Uvodni zadaci

---

```
83     printf("2 - rekurzivna\n");
84     printf("3 - rekurzivna napredna\n");
85     scanf("%d", &ind);
86
87     /* Ucitavaju se koeficijenti a i b */
88     printf("Unesite koeficijente:\n");
89     scanf("%d%d", &a, &b);
90
91     /* Ucitava se broj n */
92     printf("Unesite koji clan niza se racuna:\n");
93     scanf("%d", &n);
94
95     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
96      funkcije f_iterativna, f_rekurzivna ili f_napredna */
97     if (ind == 0)
98         printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
99     else if (ind == 1)
100        printf("F(%d) = %d\n", n, f_rekurzivna(n, a, b));
101    else
102        printf("F(%d) = %d\n", n, f_napredna(n, a, b));
103
104    return 0;
105 }
```

### Rešenje 1.22

```
1 #include <stdio.h>
2
3     /* Funkcija određuje zbir cifara zadatog broja x */
4     int zbir_cifara(unsigned int x)
5     {
6         /* Izlazak iz rekurzije: ako je broj jednociрен */
7         if (x < 10)
8             return x;
9
10        /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
11           poslednje cifre + poslednja cifra tog broja */
12        return zbir_cifara(x / 10) + x % 10;
13    }
14
15    int main()
16    {
17        unsigned int x;
18
19        /* Ucitava se ceo broj */
20        scanf("%u", &x);
21
22        /* Ispisuje se zbir cifara ucitanog broja */
23        printf("%d\n", zbir_cifara(x));
24
25        return 0;
26    }
```

26 }

## Rešenje 1.23

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAKS_DIM 100
4
5 /* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
6   suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
7   element niza. */
8 int suma_niza_1(int *a, int n)
9 {
10    if (n <= 0)
11       return 0;
12
13    return suma_niza_1(a, n - 1) + a[n - 1];
14 }
15
16 /* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
17   jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
18   elementa niza i sume preostalih n-1 elementa. */
19 int suma_niza_2(int *a, int n)
20 {
21    if (n <= 0)
22       return 0;
23
24    return a[0] + suma_niza_2(a + 1, n - 1);
25 }
26
27 int main()
28 {
29    int a[MAKS_DIM];
30    int n, i = 0, ind;
31
32    /* Ucitava se redni broj funkcije */
33    printf("Unesite redni broj funkcije (1 ili 2):\n");
34    scanf("%d", &ind);
35
36    /* Ucitava se broj elemenata niza */
37    printf("Unesite dimenziju niza:");
38    scanf("%d", &n);
39
40    /* Ucitava se n elemenata niza. */
41    printf("Unesite elemente niza:");
42    for (i = 0; i < n; i++)
43       scanf("%d", &a[i]);
44
45    /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
46      funkcije suma_niza_1, ondosno suma_niza_2 */
47    if (ind == 1)
48

```

## 1 Uvodni zadaci

---

```
48     printf("Suma elemenata je %d\n", suma_niza_1(a, n));
49 else if (ind == 2)
50     printf("Suma elemenata je %d\n", suma_niza_2(a, n));
51 else{
52     fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
53     exit(EXIT_FAILURE);
54 }
55
56 exit(EXIT_SUCCESS);
57 }
```

### Rešenje 1.24

```
#include <stdio.h>
#define MAKS_DIM 256

/* Rekursivna funkcija koja određuje 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];

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

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

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

    /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
       Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
       ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
       i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
    int i = 0;
    while (scanf("%d", &brojevi[i]) != EOF) {
        i++;
        if (i == MAKS_DIM)
            break;
    }
    n = i;

    /* Stampa se maksimum unetog niza brojeva */
}
```

```

40     printf("%d\n", maksimum_niza(brojevi, n));
41
42 }
```

### Rešenje 1.25

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Funkcija koja izracunava skalarni proizvod dva data vektora */
5 int skalarno(int a[], int b[], int n)
6 {
7     /* Izlazak iz rekurzije: vektori su duzine 0 */
8     if (n == 0)
9         return 0;
10
11    /* Na osnovu resenja problema dimenzije n-1, resava se problem
12       dimenzije n primenom definicije skalarnog proizvoda
13       a*b = a[0]*b[0] + a[1]*b[1] +...+ a[n-2]*a[n-2] + a[n-1]*a[n-1]
14       Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
15       na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
16       polazna dva vektora otklanjanjem poslednjih elemenata, doda
17       proizvod poslednja dva elementa polaznih vektora. */
18    else
19        return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
20 }
21
22 int main()
23 {
24     int i, a[MAKS_DIM], b[MAKS_DIM], n;
25
26     /* Unosi se dimenzija nizova */
27     printf("Unesite dimenziju nizova:");
28     scanf("%d", &n);
29
30     /* Provera da li je dimenzija niza odgovarajuca */
31     if (n < 0 || n > MAKS_DIM) {
32         printf("Dimenzija mora biti prirodan broj <= %d!\n", MAKS_DIM);
33         return 0;
34     }
35
36     /* A zatim i elementi nizova */
37     printf("Unesite elemente prvog niza:");
38     for (i = 0; i < n; i++)
39         scanf("%d", &a[i]);
40
41     printf("Unesite elemente drugog niza:");
42     for (i = 0; i < n; i++)
43         scanf("%d", &b[i]);
```

## 1 Uvodni zadaci

---

```
45  /* Ispisuje se rezultat skalarnog proizvoda dva ucitana niza */
46  printf("Skalarni proizvod je %d\n", skalarno(a, b, n));
47
48  return 0;
49 }
```

### Rešenje 1.26

```
#include <stdio.h>
#define MAKS_DIM 256

/* Funkcija koja racuna broj pojavljivanja elementa x u nizu a duzine n */
int br_pojave(int x, int a[], int n)
{
    /* Izlazak iz rekurzije: za niz duzine jedan broj pojava broja x u nizu je 1 ukoliko je jedini element a[0] bas x ili 0 inace */
    if (n == 1)
        return a[0] == x ? 1 : 0;

    /* U promenljivu bp se smesta broj pojave broja x u prvih n-1 elemenata niza a. Ukupan broj pojavljivanja broja x u celom nizu a je jednak bp uvecanom za jedan ukoliko je se na poziciji n-1 u nizu a nalazi broj x */
    int bp = br_pojave(x, a, n - 1);
    return a[n - 1] == x ? 1 + bp : bp;
}

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

    /* Ucitava se ceo broj */
    printf("Unesite ceo broj:");
    scanf("%d", &x);

    /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz. Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
    printf("Unesite elemente niza:");
    i = 0;
    while (scanf("%d", &a[i]) != EOF) {
        i++;
        if (i == MAKS_DIM)
            break;
    }
    n = i;

    /* Ispisuje se broj pojavljivanja */
```

```

44     printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
45
46     return 0;
}

```

### Rešenje 1.27

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Funkcija koja proverava da li su tri zadata broja uzastopni
   clanovi niza */
5 int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
6 {
7     /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i vraca
       se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi niza */
8     if (n < 3)
9         return 0;
10
11    /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
      dovoljno je da su oni poslednja tri clana niza ili da se oni
      rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
12    return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
13        || tri_uzastopna_clana(x, y, z, a, n - 1);
14
15 }
16
17 int main()
18 {
19     int x, y, z, a[MAKS_DIM];
20     int n;
21
22     /* Ucitavaju se tri cela broja za koje se ispituje da li su
      uzastopni clanovi niza */
23     printf("Unesite tri cela broja:");
24     scanf("%d%d%d", &x, &y, &z);
25
26     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
       Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
       ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
       i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
27     printf("Unesite elemente niza:");
28     int i = 0;
29     while (scanf("%d", &a[i]) != EOF) {
30         i++;
31         if (i == MAKS_DIM)
32             break;
33     }
34     n = i;
35
36     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
       se odgovarajuca poruka */
37
38 }
39
40
41
42
43
44

```

## 1 Uvodni zadaci

---

```
46     if (tri_uzastopna_clana(x, y, z, a, n))
47         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
48     else
49         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
50
51     return 0;
52 }
```

### Rešenje 1.28

```
#include <stdio.h>

2
3 /* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
5 {
6     /* Izlaz iz rekurzije */
7     if (x == 0)
8         return 0;
9
10    /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
11       postavljen na 1. Koriscenjem odgovarajuce maske proverava se
12       vrednost bita na poziciji najvece tezine i na osnovu toga se
13       razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
14       broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
15       x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
16       na poziciji najvece tezine jedinica, rezultat dobijen
17       rekurzivnim pozivom funkcije za x<<1 treba uvecati za jedan.
18       Za rekurzivni poziv se salje vrednost koja se dobija kada se x
19       pomeri u levo. Napomena: argument funkcije x je ozначен ceo
20       broj, usled cega se ne koristi pomeranje udesno, jer funkciji
21       moze biti prosledjen i negativan broj. Iz tog razloga,
22       odlucujemo se da proveramo najvisi, umesto najnizeg bita */
23    if (x & (1 << (sizeof(x) * 8 - 1)))
24        return 1 + prebroj(x << 1);
25    else
26        return prebroj(x << 1);
27    /**************************************************************************
28     Krace zapisano
29     return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
30    **************************************************************************/
31 }
32
33 int main()
34 {
35     int x;
36
37     /* Ucitava se ceo broj */
38     scanf("%x", &x);
39
40     /* Ispisuje se rezultat */
41     printf("%d\n", prebroj(x));
```

```

42     return 0;
43 }

```

### Rešenje 1.30

```

1 #include <stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
4    */
5 int maks_oktalna_cifra(unsigned x)
6 {
7     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8        vrednost najveće oktalne cifre u broju 0 */
9     if (x == 0)
10        return 0;
11
12     /* Odredjivanje poslednje oktalne cifre u broju */
13     int poslednja_cifra = x & 7;
14
15     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
16        izbriše poslednja oktalna cifra */
17     int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);
18
19     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
20           : maks_bez_poslednje_cifre;
21 }
22
23 int main()
24 {
25     unsigned x;
26
27     /* Ucitava se neoznacen ceo broj */
28     scanf("%u", &x);
29
30     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
31     printf("%d\n", maks_oktalna_cifra(x));
32
33     return 0;
34 }

```

### Rešenje 1.31

```

1 #include <stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
4    broju */
5 int maks_heksadekadna_cifra(unsigned x)
6 {

```

## 1 Uvodni zadaci

---

```
8  /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
   vrednost najvece heksadekadne cifre u broju 0 */
10 if (x == 0)
11     return 0;
12
13 /* Odredjivanje poslednje heksadekadne cifre u broju */
14 int poslednja_cifra = x & 15;
15
16 /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
   njega izbriše poslednja heksadekadna cifra */
17 int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);
18
19 return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
20      : maks_bez_poslednje_cifre;
21 }
22
23 int main()
24 {
25     unsigned x;
26
27     /* Ucitava se neoznacen ceo broj */
28     scanf("%u", &x);
29
30     /* Ispisuje se vrednost najvece heksadekadne cifre unetog broja */
31     printf("%d\n", maks_heksadekadna_cifra(x));
32
33     return 0;
34 }
```

### Rešenje 1.32

```
1 #include <stdio.h>
2 #include <string.h>
3
4 /* Niska moze imati najvise 31 karaktera + 1 za terminalnu nulu */
5 #define MAKS_DIM 32
6
7 /* Funkcija ispituje da li je zadata niska duzine n palindrom */
8 int palindrom(char s[], int n)
9 {
10    /* Izlaz iz rekurzije - trivijalno, niska duzine 0 ili 1 je
   palindrom */
11    if ((n == 1) || (n == 0))
12        return 1;
13
14    /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
   poslednji karakter i da je palindrom niska koja nastaje kada se
   polaznoj nisci otklone prvi i poslednji karakter */
15    return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
16
17 }
```

```

1 int main()
2 {
3     char s[MAKS_DIM];
4     int n;
5
6     /* Ucitavanje niske sa standardnog ulaza */
7     scanf("%s", s);
8
9     /* Odredjuje se duzina niske */
10    n = strlen(s);
11
12    /* Ispisuje se poruka da li je niska palindrom ili nije */
13    if (palindrom(s, n))
14        printf("da\n");
15    else
16        printf("ne\n");
17
18    return 0;
19}

```

### Rešenje 1.33

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAKS_DUZINA_PERMUTACIJE 15
4
5 /* Funkcija koja ispisuje elemente niza a duzine n */
6 void ispisi_niz(int a[], int n)
7 {
8     int i;
9
10    for (i = 1; i <= n; i++)
11        printf("%d ", a[i]);
12    printf("\n");
13}
14
15 /* Funkcija koja proverava da li se broj x nalazi u nizu a duzine n
16 */
17 int koriscen(int a[], int n, int x)
18 {
19     int i;
20
21     /* Obilaze se svi elementi niza */
22     for (i = 1; i <= n; i++)
23         /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
24         if (a[i] == x)
25             return 1;
26
27     /* Zaključuje se da broj nije pronadjen */
28     return 0;
29}

```

## 1 Uvodni zadaci

---

```
30 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija
31   kao argument niz a[] u koji se smesta permutacija, broj m označava
32   da se u okviru tog poziva funkcije na m-tu poziciju u permutaciji
33   smesta jedan od preostalih celih brojeva, n je veličina skupa koji
34   se permutuje. Funkciju se inicijalno poziva sa argumentom m = 1,
35   jer formiranje permutacije pocinje od pozicije broj 1. Stoga, a[0]
36   se ne koristi. */
37 void permutacija(int a[], int m, int n)
38 {
39     int i;
40
41     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
42        premašila veličinu skupa, onda se svi brojevi vec nalaze u
43        permutaciji i ispisuje se permutacija. */
44     if (m > n) {
45         ispisi_niz(a, n);
46         return;
47     }
48
49     /* Ideja: pronalazi se prvi broj koji može da se postavi na m-to
50       mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
51       Zatim, rekursivno se pronalaze one permutacije koje odgovaraju
52       ovako postavljenom pocetku permutacije. Kada se to završi, vrši
53       se provjera da li postoji još neki broj koji može da se stavi na
54       m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
55       funkcija zavrsava sa radom. Ukoliko takav broj postoji, onda se
56       ponovo poziva rekursivno pronalaženje odgovarajućih permutacija,
57       ali sada sa drugacije postavljenim prefiksom. */
58     for (i = 1; i <= n; i++) {
59         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
60           pozicije, onda se on postavlja na poziciju m i poziva se
61           ponovo funkcija da dopuni ostatak permutacije posle upisivanja
62           i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
63           se nije pojavio do sada u permutaciji. */
64         if (!koriscen(a, m - 1, i)) {
65             a[m] = i;
66             permutacija(a, m + 1, n);
67         }
68     }
69
70     int main(void)
71 {
72     int n;
73     int a[MAKS_DUZINA_PERMUTACIJE+1];
74
75     /* Ucitava se broja n i provjerava se da li je u odgovarajućem opsegu
76      */
77     scanf("%d", &n);
78     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
79         fprintf(stderr,
```

```

80         "Duzina permutacije mora biti broj iz intervala [0, %d]!\n"
81         n",
82         MAKS_DUZINA_PERMUTACIJE);
83     exit(EXIT_FAILURE);
84 }
85
86 /* Ispisuju se permutacije duzine n */
87 permutacija(a, 1, n);
88
89 exit(EXIT_SUCCESS);
}

```

### Rešenje 1.34

```

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

4 /* Rekurzivna funkcija za racunanje binomnog koeficijenta */
5 int binomni_koeficijent(int n, int k)
6 {
7     /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0.
8     Ukoliko je k strogog izmedju 0 i n, onda se koristi formula
9     bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
10    koja se moze izvesti iz definicije binomnog koeficijenta */
11    return (0 < k && k < n) ?
12        binomni_koeficijent(n - 1, k - 1) + binomni_koeficijent(n - 1,
13                                                               k) : 1;
14 }

16 **** Iterativno izracunavanje datog binomnog koeficijenta
17
18 int binomni_koeficijent (int n, int k) {
19     int i, j, b;
20
21     for (b=i=1, j=n; i<=k; b =b * j-- / i++);
22
23     return b;
24 }
25
26 Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
27 resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
28 ****
30
31 /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
32 zbir 2 elemenata iz n-1 hipotenuze. Uključujući i pomenute dve
33 ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
34 veca od sume elemenata prethodne hipotenuze. */
35 int suma_elemenata_hipotenuze(int n)
36 {
37     return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
}

```

## 1 Uvodni zadaci

---

```
38 }
39
40 int main()
41 {
42     int n, k, i, d, r;
43
44     /* Ucitavaju se brojevi d i r */
45     scanf("%d %d", &d, &r);
46
47     /* Ispisuje se Paskalov trougao */
48     putchar('\n');
49     for (n = 0; n <= d; n++) {
50         for (i = 0; i < d - n; i++)
51             printf(" ");
52         for (k = 0; k <= n; k++)
53             printf("%4d", binomni_koeficijent(n, k));
54         putchar('\n');
55     }
56
57     /* Provera da li je r nenegativan */
58     if (r < 0) {
59         fprintf(stderr,
60                 "Redni broj hipotenuze mora biti veci ili jednak od 0!\n");
61         exit(EXIT_FAILURE);
62     }
63
64     /* Ispisuje se suma elemenata hipotenuze */
65     printf("%d\n", suma_elemenata_hipotenuze(r));
66
67     exit(EXIT_SUCCESS);
68 }
```

# 2

## Pokazivači

### 2.1 Pokazivačka aritmetika

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

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

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

*Primer 1*

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

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite dimenziju niza: 0  
Greska: neodgovarajuća dimenzija niza.
```

[Rešenje 2.1]

**Zadatak 2.2** Dat je niz realnih brojeva dimenzije  $n$ . Korišćenjem pokazivačke sintakse, napisati:

## 2 Pokazivači

---

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

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

*Primer 1*

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

*Primer 2*

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

[Rešenje 2.2]

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

*Primer 1*

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

*Primer 2*

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

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite dimenziju niza: 0  
Greska: neodgovarajuća dimenzija niza.
```

*Primer 4*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite dimenziju niza: 101  
Greska: neodgovarajuća dimenzija niza.
```

[Rešenje 2.3]

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

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

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

### Primer 1

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

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

### Primer 2

```
POKRETANJE: ./a.out

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

[Rešenje 2.4]

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

### Primer 1

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

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

### Primer 2

```
POKRETANJE: ./a.out

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

[Rešenje 2.5]

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

## 2 Pokazivači

---

### Primer 1

```
POKRETANJE: ./a.out ulaz.txt 1  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima  
reci koje imaju 1 karakter  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci ciji je broj karaktera 1 je 3.
```

### Primer 2

```
POKRETANJE: ./a.out ulaz.txt  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima  
reci koje imaju 1 karakter  
  
INTERAKCIJA SA PROGRAMOM:  
Greska: Nedovoljan broj argumenata  
komandne linije.  
Program se poziva sa  
.a.out ime_dat br_karaktera.
```

### Primer 3

```
POKRETANJE: ./a.out ulaz.txt 2  
  
DATOTEKA ULAZ.TXT NE POSTOJI  
  
INTERAKCIJA SA PROGRAMOM:  
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

[Rešenje 2.6]

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

### Primer 1

```
POKRETANJE: ./a.out ulaz.txt ke -s  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima reci  
koje se zavrsavaju na ke  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci koje se zavrsavaju na ke je 2.
```

### Primer 2

```
POKRETANJE: ./a.out ulaz.txt sa -p  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima reci  
koje pocinju sa sa  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci koje pocinju na sa je 3.
```

*Primer 3*

```
POKRETANJE: ./a.out ulaz.txt sa -p
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje
datoteke ulaz.txt.
```

*Primer 4*

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
Ovo je sadrzaj ulaza.
INTERAKCIJA SA PROGRAMOM:
Greska: Nedovoljan broj argumenata
komandne linije.
Program se poziva sa
./a.out ime_dat suf/pref -s/-p.
```

[Rešenje 2.7]

## 2.2 Višedimenzioni nizovi

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

- 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 broj vrsta (ili kolona) kvadratne matrice  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrice. Na standardni izlaz ispisati učitanu matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 -2 3
4 -5 6
7 -8 9
Trag matrice je 5.
Euklidska norma matrice je 16.88.
Vandijagonalna norma matrice je 11.
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 0
Greska: neodgovarajuca dimenzija matrice.
```

[Rešenje 2.8]

## 2 Pokazivači

---

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

- 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 broj vrsta kvadratnih matrica  $n$  ( $0 < n \leq 100$ ), a zatim i elemente matrica. Na standardni izlaz ispisati da li su matrice jednake, a zatim ispisati zbir i proizvod učitanih matrica.

### Primer 1

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

[Rešenje 2.9]

**Zadatak 2.10** Relacija se može predstaviti kvadratnom matricom nula i jedinica na sledeći način: element  $i$  je u relaciji sa elementom  $j$  ukoliko se u preseku  $i$ -te vrste i  $j$ -te kolone matrice nalazi broj 1, a nije u relaciji ukoliko se tu nalazi broj 0.

- Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.

- (d) Napisati funkciju koja određuje refleksivno zatvorene relacije (najmanju refleksivnu relaciju koja je nadskup date).
- (e) Napisati funkciju koja određuje simetrično zatvorene relacije (najmanju simetričnu relaciju koja je nadskup date).
- (f) Napisati funkciju koja određuje refleksivno-tranzitivno zatvorene 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 broj vrsta kvadratne matrice  $n$  ( $0 < n \leq 64$ ), a potom i sami elementi matrice. Na standardni izlaz ispisati rezultat testiranja napisanih funkcija.

*Primer 1*

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

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

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

[Rešenje [2.10](#)]

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

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

## 2 Pokazivači

---

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

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

*Primer 1*

```
POKRETANJE: ./a.out 3
INTERAKCIJA SA PROGRAMOM:
Unesite elemente matrice dimenzije 3x3:
1 2 3
-4 -5 -6
7 8 9
Najveci element sporedne dijagonale je 7.
Indeks kolone sa najmanjim elementom je 2.
Indeks vrste sa najvecim elementom je 2.
Broj negativnih elemenata matrice je 3.
```

*Primer 2*

```
POKRETANJE: ./a.out 4
INTERAKCIJA SA PROGRAMOM:
Unesite elemente matrice dimenzije 4x4:
-1 -2 -3 -4
-5 -6 -7 -8
-9 -10 -11 -12
-13 -14 -15 -16
Najveci element sporedne dijagonale je -4.
Indeks kolone sa najmanjim elementom je 3.
Indeks vrste sa najvecim elementom je 0.
Broj negativnih elemenata matrice je 16.
```

[Rešenje 2.11]

**Zadatak 2.12** Napisati funkciju kojom se proverava da li je zadata kvadratna matrica dimenzije  $n \times n$  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 broj vrsta celobrojne kvadratne matrice  $n$  ( $0 < n \leq 32$ ), a zatim i njene elemente. Na standardni izlaz ispisati rezultat primene napisane funkcije na učitanu matricu.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrice: 4
Unesite elemente matrice, vrstu po vrstu:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Matrica je ortonormirana.
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 2 3
5 6 7
1 4 2
Matrica nije ortonormirana.
```

[Rešenje 2.12]

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

- Napisati funkciju koja učitava elemente matrice sa standardnog ulaza
- Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice, u smeru kretanja kazaljke na satu.

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

### Primer 1

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

### Primer 2

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

[Rešenje 2.13]

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

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj vrsta kvadratne matrice: 3  
Unesite elemente matrice, vrstu po vrstu:  
1 2 3  
4 5 6  
7 8 9  
Unesite stepen koji se racuna: 8  
8. stepen matrice je:  
510008400 626654232 74330064  
1154967822 1419124617 1683281412  
1799927244 2211595002 2623262760
```

## 2.3 Dinamička alokacija memorije

## 2 Pokazivači

---

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

*Primer 1*

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

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite dimenziju niza: -1  
|| malloc(): neuspela alokacija memorije.
```

[Rešenje 2.15]

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

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

Od korisnika sa ulaza tražiti da izabere način realokacije memorije.

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite zeljeni nacin realokacije (M ili R): M  
|| Unesite brojeve, nulu za kraj:  
|| 1 -2 3 -4 0  
|| Niz u obrnutom poretku je: -4 3 -2 1
```

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite zeljeni nacin realokacije (M ili R): R  
|| Unesite brojeve, nulu za kraj:  
|| 6 -1 5 -2 4 -3 0  
|| Niz u obrnutom poretku je: -3 4 -2 5 -1 6
```

[Rešenje 2.16]

**Zadatak 2.17** Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera (prepostaviti da niske nisu duže od 50 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

*Primer 1*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite dve niske karaktera:  
|| Jedan Dva  
|| Nadovezane niske: JedanDva
```

*Primer 2*

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite dve niske karaktera:  
|| Ana Marija  
|| Nadovezane niske: AnaMarija
```

[Rešenje 2.17]

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

*Primer 1*

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

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj vrsta i broj kolona matrice:  
2 2  
Unesite elemente matrice, vrstu po vrstu:  
-0.1 -0.2  
-0.3 -0.4  
Trag unete matrice je -0.50.
```

[Rešenje 2.18]

**Zadatak 2.19** Napisati biblioteku za rad sa celobrojnim matricama.

- Napisati funkciju `int **alociraj_matricu(int n, int m)` koja dinamički alocira memoriju potrebnu za matricu dimenzija  $n \times m$ .
- Napisati funkciju `int **alociraj_kvadratnu_matricu(int n)` koja alocira memoriju za kvadratnu matricu dimenzije  $n$ .
- Napisati funkciju `int **dealociraj_matricu(int **A, int n)` koja dealocira memoriju matrice sa  $n$  vrsta. Povratna vrednost ove funkcije treba da bude “prazna” matrica.
- Napisati funkciju `void ucitaj_matricu(int **A, int n, int m)` koja učitava već alociranu matricu dimenzija  $n \times m$  sa standardnog ulaza.
- Napisati funkciju `void ucitaj_kvadratnu_matricu(int **A, int n)` koja učitava već alociranu kvadratnu matricu dimenzije  $n \times n$  sa standardnog ulaza.
- Napisati funkciju `void ispisi_matricu(int **A, int n, int m)` koja ispisuje matricu dimenzija  $n \times m$  na standardnom izlazu.
- Napisati funkciju `void ispisi_kvadratnu_matricu(int **A, int n)` koja ispisuje kvadratnu matricu dimenzije  $n \times n$  na standardnom izlazu.
- Napisati funkciju `int ucitaj_matricu_iz_datoteke(int **A, int n, int m, FILE * f)` koja učitava već alociranu matricu dimenzija  $n \times m$  iz već otvorene datoteke  $f$ . U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.

## 2 Pokazivači

---

- (i) Napisati funkciju `int ucitaj_kvadratnu_matricu_iz_datoteke(int **A, int n, FILE * f)` koja učitava već alociranu kvadratnu matricu dimenzije  $n \times n$  iz već otvorene datoteke  $f$ . U slučaju neuspjelog učitavanja vratiti vrednost različitu od 0.
- (j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzija  $n \times m$  u već otvorenu datoteku  $f$ . U slučaju neuspjelog upisivanja vratiti vrednost različitu od 0.
- (k) Napisati funkciju `int upisi_kvadratnu_matricu_u_datoteku(int **A, int n, FILE * f)` koja upisuje kvadratnu matricu dimenzije  $n \times n$  u već otvorenu datoteku  $f$ . U slučaju neuspjelog upisivanja vratiti vrednost različitu od 0.

Napisati programe koji testiraju napisanu biblioteku.

- (1) Program učitava dimenziju nekvadratne matrice sa standardnog ulaza, a zatim i samu matricu. Potom, matricu upisati u datoteku *matrica.txt*.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
Unesi broj vrsta matrice: 3  
Unesi broj kolona matrice: 4  
Unesi elemente matrice po vrstama:  
1 2 3 4  
5 6 7 8  
9 10 11 12  
  
MATRICA.TXT  
1 2 3 4  
5 6 7 8  
9 10 11 12
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
Unesi broj vrsta matrice: 5  
Unesi broj kolona matrice: 0  
Neodgovarajce dimenzije matrice
```

- (2) Program prima kao prvi argument komandne linije putanju do datoteke u kojoj se redom nalazi dimenzija kvadratne matrice i sama matrica, koju treba ispisati na standardnom izlazu.

*Test 1*

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

IZLAZ:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

*Test 2*

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
dimenzija: 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

IZLAZ:
Neispravan pocetak fajla
```

*Test 3*

```
POKRETANJE: ./a.out
IZLAZ:
Korisenje programa:
./a.out datoteka
```

[Rešenje [2.19](#)]

**Zadatak 2.20** Data je celobrojna matrica dimenzije  $n \times m$ . Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu). Napisati program koji testira napisanu funkciju. 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. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka [2.19](#).*

*Primer 1*

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

[Rešenje [2.20](#)]

**Zadatak 2.21** Za zadatu matricu dimenzije  $n \times m$  napisati funkciju koja izračunava redni broj kolone matrice čiji je zbir maksimalan. Napisati program koji testira ovu funkciju. Sa standardnog ulaza učitati dimenzije matrice  $n$  i  $m$  (ne praviti nikakve pretpostavke o njihovoj veličini), a zatim elemente matrice. Na standardni izlaz ispisati redni broj kolone matrice sa maksimalnim zbirom. Ukoliko ima više takvih, ispisati prvu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka [2.19](#).*

## 2 Pokazivači

---

### Primer 1

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

### Primer 2

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

**Zadatak 2.22** Data je realna kvadratna matrica dimenzije  $n \times n$ .

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

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

### Primer 1

```
POKRETANJE: ./a.out matrica.txt  
MATRICA.TXT  
3  
1.1 -2.2 3.3  
-4.4 5.5 -6.6  
7.7 -8.8 9.9
```

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

[Rešenje 2.22]

**Zadatak 2.23** Napisati program koji na osnovu dve realne matrice dimenzija  $m \times n$  formira matricu dimenzije  $2 \cdot m \times n$  tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci `matrice.txt`. U prvom redu se nalaze dimenzije matrica  $m$  i  $n$ , u narednih  $m$  redova se nalaze vrste prve matrice, a u narednih  $m$  redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz.

*Primer 1*

```
POKRETANJE: ./a.out matrice.txt
MATRICE.TXT
3
1.1 -2.2 3.3
-4.4 5.5 -6.6
7.7 -8.8 9.9
-1.1 2.2 -3.3
4.4 -5.5 6.6
-7.7 8.8 -9.9
```

INTERAKCIJA SA PROGRAMOM:		
1.1	-2.2	3.3
-1.1	2.2	-3.3
-4.4	5.5	-6.6
4.4	-5.5	6.6
7.7	-8.8	9.9
-7.7	8.8	-9.9

**Zadatak 2.24** 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 uлево. Napisati program koji testira ovu funkciju. Rezultujuću matricu ispisati na standardni izlaz. NAPOMENA: Koristiti biblioteku sa celobrojnim matricama iz zadatka [2.19](#).

*Primer 1*

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

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite elemente niza, nulu za kraj:
-5 -2 -4 -1 0
Trazena matrica je:
-5 -2 -4 -1
-2 -4 -1 -5
-4 -1 -5 -2
-1 -5 -2 -4
```

**Zadatak 2.25** 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*  
Pomožite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronađe ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. NAPOMENA: Za realokaciju memorije koristiti *realloc()* funkciju.

*Primer 1*

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

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

## 2 Pokazivači

---

**\*\* Zadatak 2.26** U datoteci `temena.txt` se nalaze tačke koje predstavljaju temena nekog  $n$ -tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom  $n$ -touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan.

### Primer 1

TEMENA.TXT	INTERAKCIJA SA PROGRAMOM: U datoteci su zadata temena cetvorougla. Obim je 8. Povrsina je 4.
-1 -1 1 -1 1 1 -1 1	

### Primer 2

TEMENA.TXT	INTERAKCIJA SA PROGRAMOM: U datoteci su zadata temena petougla. Obim je 18.80. Povrsina je 22.59.
-1.75 -1.5 3 1.5 2.2 3.1 -2 4 -4.1 1	

## 2.4 Pokazivači na funkcije

**Zadatak 2.27** Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije u  $n$  ekvidistantnih tačaka na intervalu  $[a, b]$ . 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`, `sqrt`, `floor`, `ceil`, `sqr`).

### Primer 1

POKRETANJE: ./a.out sin	
INTERAKCIJA SA PROGRAMOM: Unesite krajeve intervala:	
-0.5 1	
Koliko tacaka ima na ekvidistantnoj	
mrezi (uključujući krajeve intervala)?	
4	
x sin(x)	
-----	
-0.50000   -0.47943	
0.00000   0.00000	
0.50000   0.47943	
1.00000   0.84147	
-----	

### Primer 2

POKRETANJE: ./a.out cos	
INTERAKCIJA SA PROGRAMOM: Unesite krajeve intervala:	
0 2	
Koliko tacaka ima na ekvidistantnoj	
mrezi (uključujući krajeve intervala)?	
4	
x cos(x)	
-----	
0.00000   1.00000	
0.66667   0.78589	
1.33333   0.23524	
2.00000   -0.41615	
-----	

[Rešenje 2.27]

**Zadatak 2.28** 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):

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite ime funkcije, n i a:  
tan 10000 1.570795  
Limes funkcije tan je -10134.46.
```

Primer 2

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

**Zadatak 2.29** Napisati funkciju koja određuje integral funkcije  $f(x)$  na intervalu  $[a, b]$ . Adresa funkcije  $f$  se prenosi kao parametar. Integral se računa prema formuli:

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

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

Primer 1

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

Primer 1

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

**Zadatak 2.30** Napisati funkciju koja približno izračunava integral funkcije  $f(x)$  na intervalu  $[a, b]$ . Funkcija  $f$  se prosleđuje kao parametar, a integral se procenjuje po Simpsonovoj formuli:

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

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

### Primer 1

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

### Primer 2

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

## 2.5 Rešenja

### Rešenje 2.1

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

```

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

41 ****
43 Drugi nacin za obrtanje niza

45 for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
        prvi++, poslednji--) {
47     int t = *prvi;
48     *prvi = *poslednji;
49     *poslednji = t;
50 }
51 ****
52 */

53 int main()
54 {
55     /* Deklarise se niz od najvise MAX elemenata */
56     int a[MAX];
57
58     /* Broj elemenata niza a */
59     int n;
60
61     /* Pokazivac na elemente niza */
62     int *p;
63
64     printf("Unesite dimenziju niza: ");
65     scanf("%d", &n);
66
67     /* Proverava se da li je doslo do prekoracenja ogranicenja
68      dimenzije */
69     if (n <= 0 || n > MAX) {
70         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
71         exit(EXIT_FAILURE);
72     }
73
74
75     printf("Unesite elemente niza:\n");
76     for (p = a; p - a < n; p++)
77         scanf("%d", p);
78
79     obrni_niz_v1(a, n);
80
81     printf("Nakon obrtanja elemenata, niz je:\n");
82
83     for (p = a; p - a < n; p++)
84         printf("%d ", *p);
85     printf("\n");
86
87     obrni_niz_v2(a, n);
}

```

## 2 Pokazivači

---

```
89     printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
91
92     for (p = a; p - a < n; p++)
93         printf("%d ", *p);
94     printf("\n");
95
96     exit(EXIT_SUCCESS);
97 }
```

### Rešenje 2.2

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

```

40     return min;
42 }

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

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

54     return max;
55 }

56 int main()
57 {
58     double a[MAX];
59     int n, i;

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

62     /* Proverava se da li je doslo do prekoracenja ogranicenja
63      dimenzije */
64     if (n <= 0 || n > MAX) {
65         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
66         exit(EXIT_FAILURE);
67     }

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

71     /* Vrsi se testiranje definisanih funkcija */
72     printf("Zbir elemenata niza je %5.3f.\n", zbir(a, n));
73     printf("Proizvod elemenata niza je %5.3f.\n", proizvod(a, n));
74     printf("Minimalni element niza je %5.3f.\n", min(a, n));
75     printf("Maksimalni element niza je %5.3f.\n", max(a, n));

76     exit(EXIT_SUCCESS);
77 }

```

**Rešenje 2.3**

## 2 Pokazivači

---

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX 100
4
5 /* Funkcija povecava za jedan sve elemente u prvoj polovini niza a
6    smanjuje za jedan sve elemente u drugoj polovini niza. Ukoliko niz
7    ima neparan broj elemenata, srednji element ostaje nepromenjen */
8 void povecaj_smanji(int *a, int n)
9 {
10     int *prvi = a;
11     int *poslednji = a + n - 1;
12
13     while (prvi < poslednji) {
14
15         /* Povecava se vrednost elementa na koji pokazuje pokazivac prvi
16         */
17         (*prvi)++;
18
19         /* Pokazivac prvi se pomera na sledeci element */
20         prvi++;
21
22         /* Smanjuje se vrednost elementa na koji pokazuje pokazivac
23            poslednji */
24         (*poslednji)--;
25
26         /* Pokazivac poslednji se pomera na prethodni element */
27         poslednji--;
28     }
29
30     *****
31     Drugi nacin:
32     while (prvi < poslednji) {
33         (*prvi)++;
34         (*poslednji)--;
35     }
36     *****
37
38     int main()
39 {
40     int a[MAX];
41     int n;
42     int *p;
43
44     printf("Unesite dimenziju niza: ");
45     scanf("%d", &n);
46
47     /* Proverava se da li je doslo do prekoracenja ogranicenja
48        dimenzije */
49     if (n <= 0 || n > MAX) {
50         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
51         exit(EXIT_FAILURE);
52 }
```

```

53    }
54
55    printf("Unesite elemente niza:\n");
56    for (p = a; p - a < n; p++)
57        scanf("%d", p);
58
59    povecaj_smanji(a, n);
60
61    printf("Transformisan niz je:\n");
62    for (p = a; p - a < n; p++)
63        printf("%d ", *p);
64    printf("\n");
65
66    exit(EXIT_SUCCESS);
67}

```

### Rešenje 2.4

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     int i;
6     char tip_ispisa;
7
8     printf("Broj argumenata komandne linije je %d.\n", argc);
9
10    printf("Kako zelite da ispisete argumente, koriscenjem"
11          " indeksne ili pokazivacke sintakse (I ili P)? ");
12    scanf("%c", &tip_ispisa);
13
14    printf("Argumenti komandne linije su:\n");
15    if (tip_ispisa == 'I') {
16        /* Ispisuju se argumenti komandne linije koriscenjem indeksne
17           sintakse */
18        for (i = 0; i < argc; i++)
19            printf("%d %s\n", i, argv[i]);
20    } else if (tip_ispisa == 'P') {
21        /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
22           sintakse */
23        i = argc;
24        for (; argc > 0; argc--)
25            printf("%d %s\n", i - argc, *argv++);
26
27        /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje na
28           polje u memoriji koje se nalazi iza poslednjeg argumenta
29           komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
30           broja argumenta komandne linije to sada moze ponovo da se
31           postavi "argv" da pokazuje na nulti argument komandne linije
32        */
33    argv = argv - i;
34}

```

## 2 Pokazivači

---

```
33     argc = i;
34 }
35
36 printf("Pocetna slova argumenata komandne linije:\n");
37 if (tip_ispisa == 'I') {
38     /* koristeci indeksnu sintaksu */
39     for (i = 0; i < argc; i++)
40         printf("%c ", argv[i][0]);
41     printf("\n");
42 } else if (tip_ispisa == 'P') {
43     /* koristeci pokazivacku sintaksu */
44     for (i = 0; i < argc; i++)
45         printf("%c ", **argv++);
46     printf("\n");
47 }
48
49 return 0;
50 }
```

### Rešenje 2.5

```
1 #include <stdio.h>
2 #include <string.h>
3 #define MAX 100
4
5 /* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
6    cita spreda i odpozadi */
7 int palindrom(char *niska)
8 {
9     int i, j;
10    for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
11        if (*(niska + i) != *(niska + j))
12            return 0;
13    return 1;
14 }
15
16 int main(int argc, char **argv)
17 {
18     int i, n = 0;
19
20     /* Nulti argument komandne linije je ime izvrsnog programa */
21     for (i = 1; i < argc; i++)
22         if (palindrom(*(argv + i)))
23             n++;
24
25     printf
26         ("Broj argumenata komandne linije koji su palindromi je %d.\n",
27          n);
28     return 0;
29 }
```

## Rešenje 2.6

```

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

```

## 2 Pokazivači

---

```
50     exit(EXIT_SUCCESS);
51 }
52 }
```

### Rešenje 2.7

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

#define MAX_KARAKTERA 100

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

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

/* Implementacija funkcije strlen() iz standardne biblioteke */
int duzina_niske(char *s)
{
    int i;
    for (i = 0; *(s + i); i++);
    return i;
}

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

/* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
```

```

46     prefiks niske zadate prvi argumentom funkcije */
47 int prefiks_niske(char *niska, char *prefiks)
48 {
49     int i;
50     int duzina_prefiksa = duzina_niske(prefiks);
51     int duzina_niske_pom = duzina_niske(niska);
52     if (duzina_prefiksa <= duzina_niske_pom) {
53         for (i = 0; i < duzina_prefiksa; i++)
54             if (*(prefiks + i) != *(niska + i))
55                 return 0;
56     return 1;
57 } else
58     return 0;
59 }
60
int main(int argc, char **argv)
61 {
62     /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
63      greska */
64     if (argc < 4) {
65         printf("Greska: ");
66         printf("Nedovoljan broj argumenata komandne linije.\n");
67         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
68               argv[0]);
69         exit(EXIT_FAILURE);
70     }
71
FILE *in;
72     int br = 0;
char rec[MAX_KARAKTERA+1];
73
74     in = fopen(*(argv + 1), "r");
75     if (in == NULL) {
76         fprintf(stderr, "Greska: ");
77         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
78         exit(EXIT_FAILURE);
79     }
80
/* Provera se opcija kojom je pozvan program a zatim se ucitavaju
   reci iz datoteke i broji se koliko njih zadovoljava trazeni
   uslov */
81     if (!(poredjenje_niski(*(argv + 3), "-s"))) {
82         while (fscanf(in, "%s", rec) != EOF)
83             br += sufiks_niske(rec, *(argv + 2));
84         printf("Broj reci koje se zavrsavaju na %s je %d.\n", *(argv + 2),
85               br);
86     } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
87         while (fscanf(in, "%s", rec) != EOF)
88             br += prefiks_niske(rec, *(argv + 2));
89         printf("Broj reci koje pocinju na %s je %d.\n", *(argv + 2), br);
90     }
91 }
92
93
94
95

```

## 2 Pokazivači

---

```
98     fclose(in);  
100    exit(EXIT_SUCCESS);  
}
```

### Rešenje 2.8

```
1 #include <stdio.h>  
2 #include <math.h>  
3 #include <stdlib.h>  
4  
5 #define MAX 100  
6  
7 /* Funkcija izracunava trag matrice */  
8 int trag(int M[][][MAX], int n)  
9 {  
10     int trag = 0, i;  
11     for (i = 0; i < n; i++)  
12         trag += M[i][i];  
13     return trag;  
14 }  
15  
16 /* Funkcija izracunava euklidsku normu matrice */  
17 double euklidska_norma(int M[][][MAX], int n)  
18 {  
19     double norma = 0.0;  
20     int i, j;  
21  
22     for (i = 0; i < n; i++)  
23         for (j = 0; j < n; j++)  
24             norma += M[i][j] * M[i][j];  
25  
26     return sqrt(norma);  
27 }  
28  
29 /* Funkcija izracunava gornju vandijagonalnu normu matrice */  
30 int gornja_vandijagonalna_norma(int M[][][MAX], int n)  
31 {  
32     int norma = 0;  
33     int i, j;  
34  
35     for (i = 0; i < n; i++) {  
36         for (j = i + 1; j < n; j++)  
37             norma += abs(M[i][j]);  
38     }  
39  
40     return norma;  
41 }  
42  
43 int main()
```

```

45   {
46     int A[MAX][MAX];
47     int i, j, n;
48
49     printf("Unesite broj vrsta matrice: ");
50     scanf("%d", &n);
51
52     /* Provera prekoracenja dimenzije matrice */
53     if (n > MAX || n <= 0) {
54       fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
55       exit(EXIT_FAILURE);
56     }
57
58     printf("Unesite elemente matrice, vrstu po vrstu:\n ");
59     for (i = 0; i < n; i++)
60       for (j = 0; j < n; j++)
61         scanf("%d", &A[i][j]);
62
63     /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
64     for (i = 0; i < n; i++) {
65       /* Ispis elemenata i-te vrste */
66       for (j = 0; j < n; j++)
67         printf("%d ", A[i][j]);
68       printf("\n");
69     }
70
71     /***** Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
72     Kod ovako definisane matrice, elementi su uzastopno smesteni u
73     memoriju, kao na traci. To znaci da su svi elementi prve vrste
74     redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
75     prve vrste smesten je prvi element druge vrste za kojim sledi
76     svi elementi te vrste i tako dalje redom.
77
78     for( i = 0; i < n ; i++ ) {
79       for ( j=0 ; j<n ; j++)
80         printf("%d ", *(*(A+i)+j));
81       printf("\n");
82     }
83     *****/
84
85     /* Ispisuje se rezultat na standardni izlaz */
86     int tr = trag(A, n);
87     printf("Trag matrice je %d.\n", tr);
88
89     printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
90     ;
91     printf("Vandijagonalna norma matrice je = %d.\n",
92           gornja_vandijagonalna_norma(A, n));
93
94     exit(EXIT_SUCCESS);
95   }

```

### Rešenje 2.9

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
7 standardnog ulaza */
8 void ucitaj_matricu(int m[][][MAX], int n)
9 {
10     int i, j;
11
12     for (i = 0; i < n; i++)
13         for (j = 0; j < n; j++)
14             scanf("%d", &m[i][j]);
15 }
16
17 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
18 standardni izlaz */
19 void ispisi_matricu(int m[][][MAX], int n)
20 {
21     int i, j;
22
23     for (i = 0; i < n; i++) {
24         for (j = 0; j < n; j++)
25             printf("%d ", m[i][j]);
26         printf("\n");
27     }
28 }
29
30 /* Funkcija proverava da li su zadate kvadratne matrice a i b
31 dimenzije n jednake */
32 int jednake_matrice(int a[][][MAX], int b[][][MAX], int n)
33 {
34     int i, j;
35
36     for (i = 0; i < n; i++)
37         for (j = 0; j < n; j++)
38             if (a[i][j] != b[i][j])
39                 return 0;
40
41     /* Prosla je provera jednakosti za sve parove elemenata koji su na
42     istim pozicijama. To znaci da su matrice jednake */
43     return 1;
44 }
45
46 /* Funkcija izracunava zbir dve kvadratne matice */
47 void saberi(int a[][][MAX], int b[][][MAX], int c[][][MAX], int n)
48 {
49     int i, j;
```

```

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

```

## 2 Pokazivači

---

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

### Rešenje 2.10

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

#define MAX 64

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

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

    return 1;
}

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

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

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

```

38 /* Funkcija proverava da li je relacija simetricna. Relacija je
40 simetricna ako za svaki par elemenata vazi: ako je element "i" u
42 relaciji sa elementom "j", onda je i element "j" u relaciji sa
elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
44 dijagonalu */
45 int simetricnost(int m[][][MAX], int n)
46 {
47     int i, j;
48
49     /* Obilaze se elementi ispod glavne dijagonale matrice i uporedjuju
50     se sa njima simetricnim elementima */
51     for (i = 0; i < n; i++)
52         for (j = 0; j < i; j++)
53             if (m[i][j] != m[j][i])
54                 return 0;
55
56     return 1;
57 }
58
59 /* Funkcija odredjuje simetricno zatvorene zadate relacije. Ono je
60 odredjeno matricom koja sadrzi sve elemente polazne matrice
61 dopunjene tako da matrica postane simetricna u odnosu na glavnu
62 dijagonalu */
63 void sim_zatvorenje(int m[][][MAX], int n, int zatvorenje[][][MAX])
64 {
65     int i, j;
66
67     for (i = 0; i < n; i++)
68         for (j = 0; j < n; j++)
69             zatvorenje[i][j] = m[i][j];
70
71     for (i = 0; i < n; i++)
72         for (j = 0; j < n; j++)
73             if (zatvorenje[i][j] == 1)
74                 zatvorenje[j][i] = 1;
75 }
76
77 /* Funkcija proverava da li je relacija tranzitivna. Relacija je
78 tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
79 relaciji sa elementom "j" i element "j" u relaciji sa elementom
80 "k", onda je i element "i" u relaciji sa elementom "k" */
81 int tranzitivnost(int m[][][MAX], int n)
82 {
83     int i, j, k;
84
85     for (i = 0; i < n; i++)
86         for (j = 0; j < n; j++)
87             /* Ispituje se da li postoji element koji narusava *
88             tranzitivnost */
89             for (k = 0; k < n; k++)
90                 if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)

```

## 2 Pokazivači

---

```
90         return 0;
92
93     return 1;
94 }
95
96 /* Funkcija određuje refleksivno-tranzitivno zatvorene zadate
97    relacije koriscenjem Varsalovog algoritma */
98 void ref_tran_zatvorene(int m[][][MAX], int n, int zatvorene[][][MAX])
99 {
100     int i, j, k;
101
102     /* Prepisuju se vrednosti elemenata pocetne matrice */
103     for (i = 0; i < n; i++)
104         for (j = 0; j < n; j++)
105             zatvorene[i][j] = m[i][j];
106
107     /* Određuje se reflektivno zatvorene matrice */
108     for (i = 0; i < n; i++)
109         zatvorene[i][i] = 1;
110
111     /* Primenom Varsalovog algoritma određuje se tranzitivno
112        zatvorene matrice */
113     for (k = 0; k < n; k++)
114         for (i = 0; i < n; i++)
115             for (j = 0; j < n; j++)
116                 if ((zatvorene[i][k] == 1) && (zatvorene[k][j] == 1)
117                     && (zatvorene[i][j] == 0))
118                     zatvorene[i][j] = 1;
119
120     /* Funkcija ispisuje elemente matrice */
121 void pisi_matricu(int m[][][MAX], int n)
122 {
123     int i, j;
124
125     for (i = 0; i < n; i++) {
126         for (j = 0; j < n; j++)
127             printf("%d ", m[i][j]);
128         printf("\n");
129     }
130 }
131
132 int main(int argc, char *argv[])
133 {
134     FILE *ulaz;
135     int m[MAX][MAX];
136     int pomocna[MAX][MAX];
137     int n, i, j;
138
139     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
140        */
141 }
```

```

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

```

## 2 Pokazivači

---

```
194     /* Zatvara se datoteka */
195     fclose(ulaz);
196
197     exit(EXIT_SUCCESS);
198 }
```

### Rešenje 2.11

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

#define MAX 32

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

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

    return max_na_sporednoj_dijagonali;
}

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

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

    return indeks_kolone;
}

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

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

```

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

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

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

61     return broj_negativnih;
62 }

63 int main(int argc, char *argv[])
64 {
65     int m[MAX][MAX];
66     int n;
67     int i, j;

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

75     /* Ucitava se broj vrsta matrice */
76     n = atoi(argv[1]);

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

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

87     printf("Najveci element sporedne dijagonale je %d.\n",
88

```

## 2 Pokazivači

---

```
96         max_sporedna_dijagonalna(m, n));  
98     printf("Indeks kolone sa najmanjim elementom je %d.\n",  
99            indeks_min(m, n));  
100    printf("Indeks vrste sa najvecim elementom je %d.\n",  
101       indeks_max(m, n));  
104    printf("Broj negativnih elemenata matrice je %d.\n",  
105       broj_negativnih(m, n));  
106    exit(EXIT_SUCCESS);  
108 }
```

### Rešenje 2.12

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 #define MAX 32  
5  
6 /* Funkcija ucitava elemente kvadratne matrice sa standardnog ulaza  
7  * */  
8 void ucitaj_matricu(int m[][][MAX], int n)  
9 {  
10    int i, j;  
11  
12    for (i = 0; i < n; i++)  
13        for (j = 0; j < n; j++)  
14            scanf("%d", &m[i][j]);  
15  
16    /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz  
17     * */  
18    void ispisi_matricu(int m[][][MAX], int n)  
19    {  
20        int i, j;  
21  
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, odnosno,  
29     * da li je normirana i ortogonalna. Matrica je normirana ako je  
30     * proizvod svake vrste matrice sa samom sobom jednak jedinici.  
31     * Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite  
32     * vrste matrice jednak nuli */  
33    int ortonormirana(int m[][][MAX], int n)
```

```

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

```

## 2 Pokazivači

---

```
87     exit(EXIT_SUCCESS);
}
```

### Rešenje 2.13

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
5
6 /* Funkcija ucitava elemente kvadratne matrice sa standardnog ulaza
7    */
8 void ucitaj_matricu(int m[][][MAX], int n)
{
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 na standardni izlaz
17    */
18 void ispisi_matricu(int m[][][MAX], int n)
{
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 /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
29 da li je normirana i ortogonalna. Matrica je normirana ako je
30 proizvod svake vrste matrice sa samom sobom jednak jedinici.
31 Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
32 vrste matrice jednak nuli */
33 int ortonormirana(int m[][][MAX], int n)
{
34     int i, j, k;
35     int proizvod;
36
37     /* Ispituje se uslov normiranosti */
38     for (i = 0; i < n; i++) {
39         proizvod = 0;
40
41         for (j = 0; j < n; j++)
42             proizvod += m[i][j] * m[i][j];
43     }
44
45     for (i = 0; i < n; i++) {
46         for (j = 0; j < n; j++)
47             if (m[i][j] != 0) {
48                 for (k = 0; k < n; k++)
49                     if (m[i][k] * m[j][k] != 0)
50                         return 0;
51             }
52     }
53
54     return 1;
55}
```

```

45     if (proizvod != 1)
46         return 0;
47     }

49     /* Ispituje se uslov ortogonalnosti */
50     for (i = 0; i < n - 1; i++) {
51         for (j = i + 1; j < n; j++) {

53             proizvod = 0;

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

57             if (proizvod != 0)
58                 return 0;
59         }
60     }

63     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
64     return 1;
65 }

67 int main()
68 {
69     int A[MAX][MAX];
70     int n;
71
72     printf("Unesite broj vrsta matrice: ");
73     scanf("%d", &n);

74     if (n > MAX || n <= 0) {
75         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
76         fprintf(stderr, "matrice.\n");
77         exit(EXIT_FAILURE);
78     }

79     printf("Unesite elemente matrice, vrstu po vrstu:\n");
80     ucitaj_matricu(A, n);

81     printf("Matrica %s ortonormirana.\n",
82           ortonormirana(A, n) ? "je" : "nije");
83
84     exit(EXIT_SUCCESS);
85 }
```

**Rešenje 2.15**

```

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

## 2 Pokazivači

---

```
1 int main()
2 {
3     int *p = NULL;
4     int i, n;
5
6     printf("Unesite dimenziju niza: ");
7     scanf("%d", &n);
8
9     /* Alocira se prostor za n celih brojeva */
10    if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
11        fprintf(stderr, "malloc(): ");
12        fprintf(stderr, "greska pri alokaciji memorije.\n");
13        exit(EXIT_FAILURE);
14    }
15
16    printf("Unesite elemente niza: ");
17    for (i = 0; i < n; i++)
18        scanf("%d", &p[i]);
19
20    printf("Niz u obrnutom poretku je: ");
21    for (i = n - 1; i >= 0; i--)
22        printf("%d ", p[i]);
23    printf("\n");
24
25    /* Oslobadja se prostor rezervisan funkcijom malloc() */
26    free(p);
27
28    exit(EXIT_SUCCESS);
29}
```

### Rešenje 2.16

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define KORAK 10
4
5 int main()
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 ucitava sa ulaza */
17    int x;
18    int i;
19    int *b = NULL;
```

```

20  char realokacija;
21
22  /* Inicijalizacija */
23  alocirano = n = 0;
24
25  printf("Unesite zeljeni nacin realokacije (M ili R):\n");
26  scanf("%c", &realokacija);
27
28  printf("Unesite brojeve, nulu za kraj:\n");
29  scanf("%d", &x);
30
31  while (x != 0) {
32      if (n == alocirano) {
33          alocirano = alocirano + KORAK;
34
35          if (realokacija == 'M') {
36              /* Vrsi se realokacija memorije sa novom velicinom */
37              b = (int *) malloc(alocirano * sizeof(int));
38
39              if (b == NULL) {
40                  fprintf(stderr, "malloc(): ");
41                  fprintf(stderr, "greska pri alokaciji memorije.\n");
42                  free(a);
43                  exit(EXIT_FAILURE);
44              }
45
46              /* Svih n elemenata koji pocinju na adresi a prepisujemo na
47                 novu adresu b */
48              for (i = 0; i < n; i++)
49                  b[i] = a[i];
50
51              free(a);
52
53              /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
54                 bloka koji je prilikom alokacije zapamcen u promenljivoj b
55                 */
56              a = b;
57          } else if (realokacija == 'R') {
58
59              /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
60                 "a" bude inicializovano na NULL */
61
62              a = (int *) realloc(a, alocirano * sizeof(int));
63              if (a == NULL) {
64                  fprintf(stderr, "realloc(): ");
65                  fprintf(stderr, "greska pri alokaciji memorije.\n");
66                  exit(EXIT_FAILURE);
67              }
68          }
69      }
70
71      a[n++] = x;

```

## 2 Pokazivači

---

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

### Rešenje 2.17

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 1000
6
7 /* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8    nadovezivanja niski. Adresa niza se vraca kao povratna vrednost.
9    */
10 char *nadovezi(char *s, char *t)
11 {
12     char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
13                               * sizeof(char));
14
15     /* Proverava se da li je memorija uspesno alocirana */
16     if (p == NULL) {
17         fprintf(stderr, "malloc(): ");
18         fprintf(stderr, "greska pri alokaciji memorije.\n");
19         exit(EXIT_FAILURE);
20     }
21
22     /* Kopiraju se i nadovezuju niske karaktera */
23     strcpy(p, s);
24     strcat(p, t);
25
26     return p;
27 }
28
29 int main()
30 {
31     char *s = NULL;
32     char s1[MAX], s2[MAX];
33
34     printf("Unesite dve niske karaktera:\n");
```

```

34     scanf("%s", s1);
35     scanf("%s", s2);
36
37     /* Poziva se funkcija koja nadovezuje niske */
38     s = nadovezi(s1, s2);
39
40     /* Prikazuje se rezultat */
41     printf("Nadovezane niske: %s\n", s);
42
43     /* Oslobadja se memorija alocirana u funkciji nadovezi() */
44     free(s);
45
46     exit(EXIT_SUCCESS);
47 }
```

**Rešenje 2.18**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int main()
6 {
7     int i, j;
8
9     /* Pokazivac na niz vrsta matrice realnih brojeva */
10    double **A = NULL;
11
12    /* Broj vrsta i broj kolona */
13    int n = 0, m = 0;
14
15    /* Trag matice */
16    double trag = 0;
17
18    printf("Unesite broj vrsta i broj kolona matrice:\n ");
19    scanf("%d%d", &n, &m);
20
21    /* Činjenica se alocira prostor za niz vrsta matrice */
22    A = (double **)malloc(sizeof(double *) * n);
23
24    /* Provera se da li je doslo do greske pri alokaciji */
25    if (A == NULL) {
26        fprintf(stderr, "malloc(): ");
27        fprintf(stderr, "greska pri alokaciji memorije.\n");
28        exit(EXIT_FAILURE);
29    }
30
31    /* Dinamicki se alocira prostor za elemente u vrstama */
32    for (i = 0; i < n; i++) {
33        A[i] = (double **)malloc(sizeof(double) * m);
```

## 2 Pokazivači

---

```
35     /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
36      potrebno je osloboditi svih i-1 prethodno alociranih vrsta, i
37      alociran niz pokazivaca */
38     if (A[i] == NULL) {
39         for (j = 0; j < i; j++)
40             free(A[j]);
41         free(A);
42         exit(EXIT_FAILURE);
43     }
44
45     printf("Unesite elemente matrice, vrstu po vrstu:\n");
46     for (i = 0; i < n; i++)
47         for (j = 0; j < m; j++)
48             scanf("%lf", &A[i][j]);
49
50     /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
51      dijagonali */
52     trag = 0.0;
53
54     for (i = 0; i < n; i++)
55         trag += A[i][i];
56
57     printf("Trag unete matrice je %.2f.\n", trag);
58
59     /* Oslobadja se prostor rezervisan za svaku vrstu */
60     for (j = 0; j < n; j++)
61         free(A[j]);
62
63     /* Oslobadjaju se memorija za niz pokazivaca na vrste */
64     free(A);
65
66     exit(EXIT_SUCCESS);
67 }
```

### Rešenje 2.19

*matrica.h*

```
1 #ifndef _MATRICA_H_
2 #define _MATRICA_H_ 1
3
4 /* Funkcija dinamicki alocira memoriju za matricu dimenzija n x m */
5 int **alociraj_matricu(int n, int m);
6
7 /* Funkcija dinamicki alocira memoriju za kvadratnu matricu dimenzije
8   n */
9 int **alociraj_kvadratnu_matricu(int n);
10
11 /* Funkcija dealocira memoriju za matricu sa n vrsta */
```

```

13 int **dealociraj_matricu(int **matrica, int n);
14 /* Funkcija ucitava vec alociranu matricu dimenzija n x m sa
15    standardnog ulaza */
16 void ucitaj_matricu(int **matrica, int n, int m);
17 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
18    standardnog ulaza */
19 void ucitaj_kvadratnu_matricu(int **matrica, int n);
20 /* Funkcija ispisuje matricu dimenzija n x m na standardnom izlazu */
21 void ispisi_matricu(int **matrica, int n, int m);
22 /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
23    izlazu */
24 void ispisi_kvadratnu_matricu(int **matrica, int n);
25 /* Funkcija ucitava vec alociranu matricu dimenzija n x m iz datoteke
26    f */
27 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
28 ;
29 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n iz
30    datoteke f */
31 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
32                                            FILE * f);
33 /* Funkcija upisuje matricu dimenzija n x m u datoteku f */
34 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);
35 /* Funkcija upisuje kvadratnu matricu dimenzije n u datoteku f */
36 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
37                                         FILE * f);
38
39 #endif

```

*matrica.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 int **alociraj_matricu(int n, int m)
6 {
7     int **matrica = NULL;
8     int i, j;
9
10    /* Alocira se prostor za niz vrsta matrice */
11    matrica = (int **) malloc(n * sizeof(int *));
12    /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
13       biti NULL, sto mora biti provereno u main funkciji */

```

## 2 Pokazivači

---

```
15     if (matrica == NULL)
16         return NULL;
17
18     /* Alocira se prostor za svaku vrstu matrice */
19     for (i = 0; i < n; i++) {
20         matrica[i] = (int *) malloc(m * sizeof(int));
21         /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
22          prethodno alocirani resursi, i povratna vrednost je NULL */
23         if (matrica[i] == NULL) {
24             for (j = 0; j < i; j++)
25                 free(matrica[j]);
26             free(matrica);
27             return NULL;
28         }
29     }
30
31     return matrica;
32 }
33
34 int **alociraj_kvadratnu_matricu(int n)
35 {
36     /* Alociranje matrice dimenzije n x n */
37     return alociraj_matricu(n, n);
38 }
39
40 int **dealociraj_matricu(int **matrica, int n)
41 {
42     int i;
43     /* Oslobadja se prostor rezervisan za svaku vrstu */
44     for (i = 0; i < n; i++)
45         free(matrica[i]);
46     /* Oslobadja se memorija za niz pokazivaca na vrste */
47     free(matrica);
48
49     /* Matrica postaje prazna, tj. nealocirana */
50     return NULL;
51 }
52
53 void ucitaj_matricu(int **matrica, int n, int m)
54 {
55     int i, j;
56     /* Elementi matrice se ucitacaju po vrstama */
57     for (i = 0; i < n; i++)
58         for (j = 0; j < m; j++)
59             scanf("%d", &matrica[i][j]);
60
61     void ucitaj_kvadratnu_matricu(int **matrica, int n)
62     {
63         /* Ucitavanje matrice n x n */
64         ucitaj_matricu(matrica, n, n);
65     }
```

```

67 void ispisi_matricu(int **matrica, int n, int m)
68 {
69     int i, j;
70     /* Ispis po vrstama */
71     for (i = 0; i < n; i++) {
72         for (j = 0; j < m; j++)
73             printf("%d ", matrica[i][j]);
74         printf("\n");
75     }
76 }
77 void ispisi_kvadratnu_matricu(int **matrica, int n)
78 {
79     /* Ispis matrice n x n */
80     ispisi_matricu(matrica, n, n);
81 }
82
83 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
84 {
85     int i, j;
86     /* Elementi matrice se ucitacaju po vrstama */
87     for (i = 0; i < n; i++)
88         for (j = 0; j < m; j++)
89             /* Ako je nemoguce ucitati sledeci element, povratna vrednost
89              funkcije je 1, kao indikator neuspesnog ucitavanja */
90             if (fscanf(f, "%d", &matrica[i][j]) != 1)
91                 return 1;
92
93     /* Uspesno ucitana matrica */
94     return 0;
95 }
96
97 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
98                                             FILE * f)
99 {
100    /* Ucitavanje matrice n x n iz datoteke */
101    return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
102 }
103
104 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
105 {
106     int i, j;
107     /* Ispis po vrstama */
108     for (i = 0; i < n; i++) {
109         for (j = 0; j < m; j++)
110             /* Ako je nemoguce ispisati sledeci element, povratna vrednost
110              funkcije je 1, kao indikator neuspesnog ispisa */
111             if (fprintf(f, "%d ", matrica[i][j]) <= 0)
112                 return 1;
113             fprintf(f, "\n");
114     }
115 }
116
117

```

## 2 Pokazivači

---

```
119     /* Uspesno upisana matrica */
120     return 0;
121 }
122 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n, FILE * f
123 )
124 {
125     /* Ispis matrice n x n u datoteku */
126     return upisi_matricu_u_datoteku(matrica, n, n, f);
127 }
```

*main\_a.c*

```
#include <stdio.h>
2 #include <stdlib.h>
# include "matrica.h"
4
int main()
6 {
    int **matrica = NULL;
8    int n, m;
    FILE *f;
10
/* Ucitavanje dimenzije matrice */
12    printf("Unesi broj vrsta matrice: ");
    scanf("%d", &n);
14    printf("Unesi broj kolona matrice: ");
    scanf("%d", &m);
16
/* Provera dimenzija matrice */
18    if (n <= 0 || m <= 0) {
        fprintf(stderr, "Neodgovarajce dimenzije matrice\n");
        exit(EXIT_FAILURE);
    }
22
/* Alokacija matrice i provera alokacije */
24    matrica = alociraj_matricu(n, m);
    if (matrica == NULL) {
26        fprintf(stderr, "Neuspesna alokacija matrice\n");
        exit(EXIT_FAILURE);
    }
28
/* Ucitavanje matrice sa standardnog ulaza */
30    printf("Unesi elemente matrice po vrstama:\n");
32    ucitaj_matricu(matrica, n, m);
34
/* Otvaranje fajla za upis matrice */
36    if ((f = fopen("matrica.txt", "w")) == NULL) {
        fprintf(stderr, "fopen() error\n");
        matrica = dealociraj_matricu(matrica, n);
        exit(EXIT_FAILURE);
    }
38
```

```

40    }
41
42    /* Upis matrice u fajl */
43    if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
44        fprintf(stderr, "Neuspesno upisivanje matrice u datoteku\n");
45        matrica = dealociraj_matricu(matrica, n);
46        exit(EXIT_FAILURE);
47    }
48
49    /* Zatvaranje fajla */
50    fclose(f);
51
52    /* Dealokacija matrice */
53    matrica = dealociraj_matricu(matrica, n);
54
55    exit(EXIT_SUCCESS);
56}

```

*main\_b.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 int main(int argc, char **argv)
6 {
7     int **matrica = NULL;
8     int n;
9     FILE *f;
10
11    /* Provera argumenata komandne linije */
12    if (argc != 2) {
13        fprintf(stderr, "Koriscenje programa: %s datoteka\n", argv[0]);
14        exit(EXIT_FAILURE);
15    }
16
17    /* Otvaranje fajla za citanje */
18    if ((f = fopen(argv[1], "r")) == NULL) {
19        fprintf(stderr, "fopen() error\n");
20        exit(EXIT_FAILURE);
21    }
22
23    /* Ucitavanje dimenzije matrice */
24    if (fscanf(f, "%d", &n) != 1) {
25        fprintf(stderr, "Neispravan pocetak fajla\n");
26        exit(EXIT_FAILURE);
27    }
28
29    /* Provera dimenzije matrice */
30    if (n <= 0) {
31        fprintf(stderr, "Neodgovarajca dimenzija matrice\n");
32    }

```

## 2 Pokazivači

---

```
    exit(EXIT_FAILURE);
33 }

35 /* Alokacija matrice i provera alokacije */
36 matrica = alociraj_kvadratnu_matricu(n);
37 if (matrica == NULL) {
38     fprintf(stderr, "Neuspesna alokacija matrice\n");
39     exit(EXIT_FAILURE);
40 }

41 /* Ucitavanje matrice iz datoteke */
42 if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
43     fprintf(stderr, "Neuspesno ucitavanje matrice iz datoteke\n");
44     matrica = dealociraj_matricu(matrica, n);
45     exit(EXIT_FAILURE);
46 }

47 /* Zatvaranje fajla */
48 fclose(f);

49 /* Ispis matrice na standardnom izlazu */
50 ispis_i_kvadratnu_matricu(matrica, n);

51 /* Dealokacija matrice */
52 matrica = dealociraj_matricu(matrica, n);

53 exit(EXIT_SUCCESS);
54 }
```

### Rešenje 2.20

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

5 /* Funkcija ispisuje elemente matrice ispod glavne dijagonale */
6 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
7 {
8     int i, j;

9     for (i = 0; i < n; i++) {
10         for (j = 0; j <= i; j++)
11             printf("%d ", M[i][j]);
12         printf("\n");
13     }
14 }

15 int main()
16 {
17     int m, n;
```

```

21 int **matrica = NULL;
22
23 printf("Unesite broj vrsta i broj kolona matrice:\n ");
24 scanf("%d %d", &n, &m);
25
26 /* Alocira se matrica */
27 matrica = alociraj_matricu(n, m);
28 /* Provera alokacije */
29 if (matrica == NULL) {
30     fprintf(stderr, "Neuspesna alokacija matrice\n");
31     exit(EXIT_FAILURE);
32 }
33
34 printf("Unesite elemente matrice, vrstu po vrstu:\n");
35 ucitaj_matricu(matrica, n, m);
36
37 printf("Elementi ispod glavne dijagonale matrice:\n");
38 ispis_i_elemente_ispod_dijagonale(matrica, n, m);
39
40 /* Oslobođanje memorije */
41 matrica = dealociraj_matricu(matrica, n);
42
43 exit(EXIT_SUCCESS);
}

```

### Rešenje 2.22

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 /* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni(float **a, int n)
7 {
8     int i, j;
9
10    for (i = 0; i < n; i++)
11        for (j = 0; j < n; j++)
12            if (i < j)
13                a[i][j] /= 2;
14            else if (i > j)
15                a[i][j] *= 2;
16 }
17
18 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
19 sporedne dijagonale. Element se nalazi ispod sporedne dijagonale
20 ukoliko je zbir indeksa vrste i indeksa kolone elementa veci od
21 n-1 */
22 float zbir_ispod_sporedne_dijagonale(float **m, int n)
23 {
24     int i, j;

```

## 2 Pokazivači

---

```
1     float zbir = 0;
2
3     for (i = 0; i < n; i++)
4         for (j = n-i; j < n; j++)
5             if (i + j > n - 1)
6                 zbir += fabs(m[i][j]);
7
8     return zbir;
9 }
10
11 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
12    datoteke */
13 void ucitaj_matricu(FILE * ulaz, float **m, int n)
14 {
15     int i, j;
16
17     for (i = 0; i < n; i++)
18         for (j = 0; j < n; j++)
19             fscanf(ulaz, "%f", &m[i][j]);
20 }
21
22 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
23    standardni izlaz */
24 void ispisi_matricu(float **m, int n)
25 {
26     int i, j;
27
28     for (i = 0; i < n; i++) {
29         for (j = 0; j < n; j++)
30             printf("%.2f ", m[i][j]);
31         printf("\n");
32     }
33
34     /* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
35 float **alociraj_memoriju(int n)
36 {
37     int i, j;
38     float **m;
39
40     m = (float **) malloc(n * sizeof(float *));
41     if (m == NULL) {
42         fprintf(stderr, "malloc(): Neuspela alokacija\n");
43         exit(EXIT_FAILURE);
44     }
45
46     for (i = 0; i < n; i++) {
47         m[i] = (float *) malloc(n * sizeof(float));
48
49         if (m[i] == NULL) {
50             printf("malloc(): neuspela alokacija memorije!\n");
51             for (j = 0; j < i; j++)
52                 free(m[j]);
53             free(m);
54             exit(EXIT_FAILURE);
55         }
56     }
57 }
```

```

    free(m[i]);
    free(m);
    exit(EXIT_FAILURE);
}
}
return m;
}

/* Funckija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
   n */
void oslobodi_memoriju(float **m, int n)
{
    int i;
    for (i = 0; i < n; i++)
        free(m[i]);
    free(m);
}

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

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

    /* Otvara se datoteka za citanje */
    ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
        fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
    }

    /* Cita se dimenzija matrice */
    fscanf(ulaz, "%d", &n);

    /* Alocira se memorija */
    a = alociraj_memoriju(n);

    /* Ucitavaju se elementi matrice */
    ucitaj_matricu(ulaz, a, n);

    float zbir = zbir_ispod_sporedne_dijagonale(a, n);
}

```

## 2 Pokazivači

---

```
128  /* Poziva se funkcija za transformaciju matrice */
129  izmeni(a, n);
130
132  /* Ispisuje se rezultat */
133  printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
134  printf("je %.2f.\n", zbir);
135
136  printf("Transformisana matrica je:\n");
137  ispisi_matricu(a, n);
138
139  /* Oslobadja se memorija */
140  osloboidi_memoriju(a, n);
141
142  /* Zatvara se datoteka */
143  fclose(ulaz);
144
145  exit(EXIT_SUCCESS);
146 }
```

### Rešenje 2.27

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <string.h>
5
6 /* Funkcija tabela() prihvata granice intervala a i b, broj
7  ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
8  funkciju koja prihvata double argument, i vraca double vrednost.
9  Za tako datu funkciju ispisuju se njene vrednosti u intervalu
10 [a,b] u n ekvidistantnih tacaka intervala */
11 void tabela(double a, double b, int n, double (*fp) (double))
12 {
13     int i;
14     double x;
15
16     printf("-----\n");
17     for (i = 0; i < n; i++) {
18         x = a + i * (b - a) / (n - 1);
19         printf(" | %8.5f | %8.5f |\n", x, (*fp)(x));
20     }
21     printf("-----\n");
22 }
23
24 double sqr(double a)
25 {
26     return a * a;
27 }
```

```

1 int main(int argc, char *argv[])
31 {
32     double a, b;
33     int n;
34
35     char ime_funkcije[6];
36
37     /* Pokazivac na funkciju koja ima jedan argument tipa double i
38      povratnu vrednost istog tipa */
39     double (*fp) (double);
40
41     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
42      */
43     if (argc < 2) {
44         printf("Greska: ");
45         printf("Nedovoljan broj argumenata komandne linije.\n");
46         printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
47                argv[0]);
48         exit(EXIT_FAILURE);
49     }
50
51     /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
52      u
53      komandnoj liniji */
54     strcpy(ime_funkcije, argv[1]);
55
56     /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
57      */
58     if (strcmp(ime_funkcije, "sin") == 0)
59         fp = &sin;
60     else if (strcmp(ime_funkcije, "cos") == 0)
61         fp = &cos;
62     else if (strcmp(ime_funkcije, "tan") == 0)
63         fp = &tan;
64     else if (strcmp(ime_funkcije, "atan") == 0)
65         fp = &atan;
66     else if (strcmp(ime_funkcije, "acos") == 0)
67         fp = &acos;
68     else if (strcmp(ime_funkcije, "asin") == 0)
69         fp = &asin;
70     else if (strcmp(ime_funkcije, "exp") == 0)
71         fp = &exp;
72     else if (strcmp(ime_funkcije, "log") == 0)
73         fp = &log;
74     else if (strcmp(ime_funkcije, "log10") == 0)
75         fp = &log10;
76     else if (strcmp(ime_funkcije, "sqrt") == 0)
77         fp = &sqrt;
78     else if (strcmp(ime_funkcije, "floor") == 0)
79         fp = &floor;
80     else if (strcmp(ime_funkcije, "ceil") == 0)
81         fp = &ceil;

```

## 2 Pokazivači

---

```
81     else if (strcmp(ime_funkcije, "sqr") == 0)
82         fp = &sqr;
83     else {
84         printf("Program jos uvek ne podrzava trazenu funkciju!\n");
85         exit(EXIT_SUCCESS);
86     }
87
88     printf("Unesite krajeve intervala:\n");
89     scanf("%lf %lf", &a, &b);
90
91     printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
92     printf("(uključujući krajeve intervala)?\n");
93     scanf("%d", &n);
94
95     /* Mreza mora da uključuje bar krajeve intervala, tako da se mora
96      uneti broj veci od 2 */
97     if (n < 2) {
98         fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
99         exit(EXIT_FAILURE);
100    }
101
102    /* Ispisuje se ime funkcije */
103    printf("      x %10s(x)\n", ime_funkcije);
104
105    /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
106       komandne linije */
107    tabela(a, b, n, fp);
108
109    exit(EXIT_SUCCESS);
110 }
```

# 3

## Algoritmi pretrage i sortiranja

### 3.1 Algoritmi pretrage

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

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

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

### 3 Algoritmi pretrage i sortiranja

---

Test 1

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

Test 2

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

[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. Linearu pretragu implementirati na dva načina, svođenjem pretrage na prefiks i na sufiks niza. Prvo se unosi broj koji se traži, a zatim sortirani elementi niza sve do kraja ulaza. Pretpostaviti da niz brojeva koji se unosi neće biti duži od 1024 elemenata.

Primer 1

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

Primer 2

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

[Rešenje 3.2]

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenata po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti **-indeks** ili **-prezime**. U slučaju neuspjehnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

#### Primer 1

```
POKRETANJE: ./a.out datoteka.txt -indeks  
  
DATOTEKA.TXT  
20140003 Marina Petrovic  
20140012 Stefan Mitrovic  
20140032 Dejan Popovic  
20140049 Mirko Brankovic  
20140076 Sonja Stevanovic  
20140104 Ivan Popovic  
20140187 Vlada Stankovic  
20140234 Darko Brankovic  
  
INTERAKCIJA SA PROGRAMOM:  
Unesite indeks studenta  
cije informacije zelite:  
20140076  
Indeks: 20140076,  
Ime i prezime: Sonja Stevanovic
```

#### Primer 2

```
POKRETANJE: ./a.out datoteka.txt -prezime  
  
DATOTEKA.TXT  
20140003 Marina Petrovic  
20140012 Stefan Mitrovic  
20140032 Dejan Popovic  
20140049 Mirko Brankovic  
20140076 Sonja Stevanovic  
20140104 Ivan Popovic  
20140187 Vlada Stankovic  
20140234 Darko Brankovic  
  
INTERAKCIJA SA PROGRAMOM:  
Unesite prezime studenta  
cije informacije zelite:  
Popovic  
Indeks: 20140032,  
Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

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

[Rešenje 3.4]

**Zadatak 3.5** U datoteci koja se zadaje kao prvi argument komandne linije, nalaze se koordinate tačaka. U zavisnosti od prisustva opcija komandne linije (**-x** ili **-y**), pronaći onu koja je najbliža  $x$ , ili  $y$  osi, ili koordinatnom početku, ako

### 3 Algoritmi pretrage i sortiranja

---

nije prisutna nijedna opcija. Pretpostaviti da je broj tačaka u datateci veći od 0 i ne veći od 1024.

Test 1	Test 2	Test 3
POKRETANJE: ./a.out dat.txt -	POKRETANJE: ./a.out dat.txt	POKRETANJE: ./a.out dat.txt -y
DAT.TXT 12 53 2.342 34.1 -0.3 23 -1 23.1 123.5 756.12	DAT.TXT 12 53 2.342 34.1 -0.3 23 -1 2.1 123.5 756.12	DAT.TXT 12 53 2.342 34.1 -0.3 0.23 -1 2.1 123.5 756.12

[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, metod polovljenja intervala. NAPOMENA: Ovaj metod se može primeniti na funkciju  $\cos(x)$  na intervalu  $[0, 2]$  zato što je ona na ovom intervalu neprekidna, i vrednosti funkcije na krajevima intervala su različitog znaka.

Test 1
IZLAZ: 1.57031

[Rešenje 3.6]

**Zadatak 3.7** Napisati funkciju koja metodom polovljenja intervala određuje nulu izabrane funkcije na proizvolnjom intervalu sa tačnošću  $epsilon$ . Ime funkcije se zadaje kao prvi argument komandne linije, a interval i tačnost se unose sa standardnog ulaza. Pretpostaviti da je izabrana funkcija na tom intervalu neprekidna. UPUTSTVO: U okviru algoritma pretrage koristiti pokazivač na odgovarajuću funkciju (na primer, kao u zadatku 2.27).

*Primer 1*

```
POKRETANJE: ./a.out cos
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 0 2
Unesite preciznost: 0.001
1.57031
```

*Primer 2*

```
POKRETANJE: ./a.out sin
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 1 5
Unesite preciznost: 0.00001
3.1416
```

*Primer 3*

```
POKRETANJE: ./a.out tan
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: -1.1 1
Unesite preciznost: 0.00001
3.8147e-06
```

*Primer 4*

```
POKRETANJE: ./a.out sin
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 1 3
Funkcija sin na intervalu [1, 3]
ne zadovoljava uslove
```

[Rešenje 3.7]

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

*Test 1*

```
ULAZ:
-151 -44 5 12 13 15
IZLAZ:
2
```

*Test 2*

```
ULAZ:
-100 -15 -11 -8 -7 -5
IZLAZ:
-1
```

*Test 3*

```
ULAZ:
-100 -15 0 13 55 124
258 315 516 7000
IZLAZ:
3
```

[Rešenje 3.8]

**Zadatak 3.9** Napisati funkciju koja u opadajuće sortiranom nizu celih brojeva binarnom pretragom pronalazi indeks prvog elementa manjeg od nule. Ukoliko nema elemenata manjih od nule, funkcija kao rezultat vraća  $-1$ . Napisati program koji testira ovu funkciju za opadajući niz celih brojeva koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

### 3 Algoritmi pretrage i sortiranja

---

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

[Rešenje 3.9]

**Zadatak 3.10** Napisati funkciju koja određuje ceo deo logaritma za osnovu 2 datog neoznačenog celog broja koristeći samo bitske i relacione operatore.

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

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

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

[Rešenje 3.10]

\* **Zadatak 3.11** U prvom kvadrantu dato je  $1 \leq N \leq 10000$  duži svojim koordinatama (duži mogu da se sekut, preklapaju, itd.). Napisati program koji pronalazi najmanji ugao  $0 \leq \alpha \leq 90^\circ$ , na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom  $\alpha$  jednak (neke duži bivaju presečene, a neke ne). Program prvo učitava broj  $N$ , a zatim i same koordinate temena duži. UPUTSTVO: *Vršiti binarnu pretragu intervala  $[0, 90^\circ]$ .*

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesi broj tacaka: 2  
|| Unesi koordinate tacaka:  
|| 2 0 2 1  
|| 1 2 2 2  
|| 26.57
```

Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesi broj tacaka: 2  
|| Unesi koordinate tacaka:  
|| 1 0 1 1  
|| 0 1 1 1  
|| 45
```

Primer 3

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesi broj tacaka: 3  
|| Unesi koordinate tacaka:  
|| 1 0 1 1  
|| 2 0 2 1  
|| 1 2 2 2  
|| 26.57
```

## 3.2 Algoritmi sortiranja

**Zadatak 3.12** Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. *selection sort*), sortiranja spajanjem (engl. *merge sort*), brzog sortiranja (engl. *quick sort*), mehurastog sortiranja (engl. *bubble sort*), sortiranja direktnim umetanjem (engl. *insertion sort*) i sortiranja umetanjem sa inkrementom (engl. *shell sort*). Upotrebiti biblioteku kako bi se napravilo po-ređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za sortiranje spajanjem, **-q** za brzo sortiranje, **-b** za mehurasto, **-i** za sortiranje direktnim umetanjem ili **-s** za sortiranje umetanjem sa inkrementom. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati algoritmom sortiranja izborom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija **-r**, nerastuće ako je prisutna opcija **-o** ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom **time**. Analizirati porast vremena sa porastom dimenzije *n*.

Test 1

```
|| POKRETANJE: time ./a.out  
|| 200000  
|| IZLAZ:  
|| real 0m42.168s  
|| user 0m42.100s  
|| sys 0m0.000s
```

Test 2

```
|| POKRETANJE: time ./a.out  
|| 400000  
|| IZLAZ:  
|| real 2m48.395s  
|| user 2m48.128s  
|| sys 0m0.000s
```

Test 3

```
|| POKRETANJE: time ./a.out  
|| 800000  
|| IZLAZ:  
|| real 11m13.703s  
|| user 11m12.636s  
|| sys 0m0.000s
```

### 3 Algoritmi pretrage i sortiranja

---

Test 4

```
|| POKRETANJE: time ./a.out
  800000 -r
  || IZLAZ:
    real 11m21.533s
    user 11m20.436s
    sys 0m0.020s
```

Test 5

```
|| POKRETANJE: time ./a.out
  800000 -q
  || IZLAZ:
    real 0m0.159s
    user 0m0.156s
    sys 0m0.000s
```

Test 6

```
|| POKRETANJE: time ./a.out
  800000 -m
  || IZLAZ:
    real 0m0.137s
    user 0m0.136s
    sys 0m0.000s
```

[Rešenje 3.12]

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

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
  Unesite prvu nisku anagram
  Unesite drugu nisku ramgana
  jesu
```

Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
  Unesite prvu nisku anagram
  Unesite drugu nisku anagr
  nisu
```

Primer 3

```
|| INTERAKCIJA SA PROGRAMOM:
  Unesite prvu nisku test
  Unesite drugu nisku tset
  jesu
```

[Rešenje 3.13]

**Zadatak 3.14** U datom nizu brojeva treba pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, ali neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati razliku pronađena dva broja. UPUTSTVO: *Prvo sortirati niz.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.*

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.14]

**Zadatak 3.15** 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*

### 3.2 Algoritmi sortiranja

niz, a zatim naći najdužu sekvencu jednakačih elemenata. NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.

Test 1

ULAZ:	4 23 5 2 4 6 7 34 6 4 5
IZLAZ:	4

Test 2

ULAZ:	2 4 6 2 6 7 99 1
IZLAZ:	2

Test 3

ULAZ:	123
IZLAZ:	123

[Rešenje 3.15]

**Zadatak 3.16** Napisati funkciju koja proverava da li u datom nizu postoje dva elementa čiji zbir je jednak zadatom celom broju. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz. Elementi niza se unose sve do kraja ulaza. Pretpostaviti da u niz neće biti uneto više od 256 brojeva. UPUTSTVO: Prvo sortirati niz. NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.

Primer 1

INTERAKCIJA SA PROGRAMOM:	Unesite trazeni zbir: 34
	Unesite elemente niza:
	134 4 1 6 30 23
	da

Primer 2

INTERAKCIJA SA PROGRAMOM:	Unesite trazeni zbir: 12
	Unesite elemente niza:
	53 1 43 3 56 13
	ne

Primer 3

INTERAKCIJA SA PROGRAMOM:	Unesite trazeni zbir: 52
	Unesite elemente niza:
	52
	ne

[Rešenje 3.16]

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

Primer 1

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

Primer 2

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

### 3 Algoritmi pretrage i sortiranja

---

[Rešenje 3.17]

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

*Test 1*

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

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

[Rešenje 3.18]

**Zadatak 3.19** Napisati funkcije koje sortiraju niz struktura tačaka na osnovu sledećih kriterijuma: (i) njihovog rastojanja od koordinatnog početka, (ii) x koordinata tačaka, (iii) y koordinata tačaka. Napisati program koji učitava niz tačaka iz datoteke čije se ime zadaje kao drugi argument komandne linije, i u zavisnosti od prisutnih opcija (prvi argument) u komandnoj liniji (`-o`, `-x` ili `-y`) sortira tačke po jednom od prethodna tri kriterijuma i rezultat upisuje u datoteku čije se ime zadaje kao treći argument komandne linije. U ulaznoj datoteci nije zadato više od 128 tačaka.

### 3.2 Algoritmi sortiranja

Test 1

```
POKRETANJE: ./a.out -x in.txt out.txt  
  
IN.TXT  
3 4  
11 6  
7 3  
2 82  
-1 6  
  
OUT.TXT  
-1 6  
2 82  
3 4  
7 3  
11 6
```

Test 2

```
POKRETANJE: ./a.out -o in.txt out.txt  
  
IN.TXT  
3 4  
11 6  
7 3  
2 82  
-1 6  
  
OUT.TXT  
3 4  
-1 6  
7 3  
11 6  
2 82
```

[Rešenje 3.19]

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

Test 1

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

Test 2

```
BIRACKI-SPISAK.TXT  
Milan Milicevic  
  
IZLAZ:  
1
```

Test 3

```
DATOTEKA BIRACKI-SPISAK.TXT  
NE POSTOJI  
  
IZLAZ ZA GREŠKU:  
Neuspesno otvaranje  
datoteke  
biracki-spisak.txt.
```

[Rešenje 3.20]

**Zadatak 3.21** Definisati strukturu koja čuva imena, prezimena i godišta dece. Prepostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument

### 3 Algoritmi pretrage i sortiranja

---

komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Prepostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

*Test 1*

```
||| POKRETANJE: ./a.out in.txt out.txt
||| IN.OUT
||| Petar Petrovic 2007
||| Milica Antonic 2008
||| Ana Petrovic 2007
||| Ivana Ivanovic 2009
||| Dragana Markovic 2010
||| Marija Antic 2007
||| OUT.TXT
||| Marija Antic 2007
||| Ana Petrovic 2007
||| Petar Petrovic 2007
||| Milica Antonic 2008
||| Ivana Ivanovic 2009
||| Dragana Markovic 2010
```

*Test 2*

```
||| POKRETANJE: ./a.out in.txt out.txt
||| IN.OUT
||| Milijana Maric 2009
||| OUT.TXT
||| Milijana Maric 2009
```

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

*Test 1*

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

#### Primer 1

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

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

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

UKUPNO: 270.00 dinara.

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

232
GRESKA: Ne postoji proizvod sa traenim 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

### 3 Algoritmi pretrage i sortiranja

---

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

#### Test 1

AKTIVNOSTI.TXT	DAT2.TXT
Aleksandra Cvetic 4 6	Studenti sortirani po broju zadataka
Bojan Golubovic 4 3	opadajuće, pa po duzini imena rastuce:
Dragan Markovic 3 5	Aleksandra Cvetic 4 6
Ivana Stankovic 3 1	Uros Milic 2 5
Marija Stankovic 1 3	Dragan Markovic 3 5
Ognjen Peric 1 2	Andrija Petrovic 2 5
Uros Milic 2 5	Nenad Brankovic 2 4
Andrija Petrovic 2 5	Lazar Micic 1 3
Anja Ilic 3 1	Bojan Golubovic 4 3
Lazar Micic 1 3	Marija Stankovic 1 3
Nenad Brankovic 2 4	Ognjen Peric 1 2
	Anja Ilic 3 1
	Ivana Stankovic 3 1
DAT1.TXT	DAT3.TXT
Studenti sortirani po imenu	Studenti sortirani po prisustvu
leksikografski rastuce:	opadajuće, pa po broju zadataka,
Aleksandra Cvetic 4 6	pa po prezimenima leksikografski
Andrija Petrovic 2 5	opadajuće:
Anja Ilic 3 1	Aleksandra Cvetic 4 6
Bojan Golubovic 4 3	Bojan Golubovic 4 3
Dragan Markovic 3 5	Dragan Markovic 3 5
Ivana Stankovic 3 1	Ivana Stankovic 3 1
Lazar Micic 1 3	Anja Ilic 3 1
Marija Stankovic 1 3	Andrija Petrovic 2 5
Nenad Brankovic 2 4	Uros Milic 2 5
Ognjen Peric 1 2	Nenad Brankovic 2 4
Uros Milic 2 5	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 prepostavki o maksimalnoj dužini imena izvođača i naslova pesme.

*Test 1*

```
POKRETANJE: ./a.out
PESME.TXT
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*

```
POKRETANJE: ./a.out -i
PESME.TXT
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*

```
POKRETANJE: ./a.out -n
PESME.TXT
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]

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

*Primer 1*

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

\* **Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze

### 3 Algoritmi pretrage i sortiranja

---

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
1	1	4	4
2	2	— —	5

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

*Test 1*

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

**Zadatak 3.28** Za zadatu 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. NAPOMENA: Koristiti biblioteku sa celobrojnim matricama iz zadatka [2.19](#).

*Test 1*

INTERAKCIJA SA PROGRAMOM:	Unesite dimenzije matrice: 3 2
	Unesite elemente matrice po vrstama:
	6 -5
	-4 3
	2 1
	Sortirana matrica je:
	-4 3
	6 -5
	2 1

*Test 2*

INTERAKCIJA SA PROGRAMOM:	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 [3.28](#)]

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.29** Za zadatu 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. NAPOMENA: Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.

Primer 1

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

Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
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
```

### 3.3 Bibliotečke funkcije pretrage i sortiranja

**Zadatak 3.30** Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime, a zatim se učitava jedan karakter i pronalazi i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati  $-1$  na standardnom izlazu za greške. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretazivanje niza vršiti bibliotečkom funkcijom bsearch. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

Test 1

```
ULAZ:  
R  
  
IZLAZ:  
Dusko Radovic
```

Test 2

```
ULAZ:  
E  
  
IZLAZ:  
Dobrica Eric
```

Test 3

```
ULAZ:  
X  
  
IZLAZ:  
-1
```

### 3 Algoritmi pretrage i sortiranja

---

**Zadatak 3.31** Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva, ne veća od 100, a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu. Na standardnom izlazu ispisati odgovarajuću poruku.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 11  
Uneti elemente niza:  
5 3 1 6 8 90 34 5 3 432 34  
Sortirani niz u rastucem poretku:  
1 3 3 5 6 8 34 34 90 432  
Uneti element koji se trazi u nizu: 34  
Binarna pretraga:  
Element je nadjen na poziciji 8  
Linearna pretraga (lfind):  
Element je nadjen na poziciji 7
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
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 3.31]

**Zadatak 3.32** Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva (ne veću od 100), a potom i same elemente niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku prema broju delilaca i tako dobijeni niz odštampati na standardnom izlazu.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:  
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
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 1  
Uneti elemente niza:  
234  
Sortirani niz u rastucem  
poretku prema broju delilaca  
234
```

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 0  
Uneti elemente niza:  
Sortirani niz u rastucem  
poretku prema broju  
delilaca:
```

[Rešenje 3.32]

**Zadatak 3.33** Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

#### Primer 1

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

INTERAKCIJA SA PROGRAMOM:
Leksikografski sortirane niske:
aleksandar ana andjela ljubica matej milica milos nikola petar
Uneti trazenu nisku: matej
Niska "matej"je pronadjena u nizu na poziciji 4
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
Niska "matej"je pronadjena u nizu na poziciji 1
```

[Rešenje 3.33]

**Zadatak 3.34** Uraditi prethodni zadatak 3.33 sa dinamički alociranim niskama i sortiranjem niza pokazivača, umesto niza niski.

[Rešenje 3.34]

**Zadatak 3.35** 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. Prepostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

### 3 Algoritmi pretrage i sortiranja

---

*Primer 1*

```
POKRETANJE: ./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 SA PROGRAMOM:
Studenti sortirani po broju poena
opadajuce, pa po prezimenu rastuce:
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 3.35]

**Zadatak 3.36** Uraditi zadatak 3.13, ali korišćenjem bibliotečke qsort funkcije.

[Rešenje 3.36]

**Zadatak 3.37** Napisati program koji sa standardnog ulaza učitava prvo ceo broj  $n$  ( $n \leq 10$ ), a zatim niz  $S$  od  $n$  niski. Maksimalna dužina svake niske je 31 karakter. Sortirati niz  $S$  bibliotečkom funkcijom qsort i proveriti da li u njemu ima identičnih niski.

*Primer 1*

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

*Primer 2*

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

*Primer 3*

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

[Rešenje 3.37]

**Zadatak 3.38** Datoteka studenti.txt sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. mr15125, mm14001), ime, prezime i broj poena. Ni ime, ni prezime neće biti duže od 20 karaktera. Napisati

### 3.3 Bibliotečke funkcije pretrage i sortiranja

program koji korišćenjem funkcije `qsort` sortira studente po broju poena opadaće, 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

```
POKRETANJE: ./a.out -n mm13321
STUDENTI.TXT
mr14123 Marko Antic 20
mm13321 Marija Radic 12
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mv14003 Jovan Jovanovic 17

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

IZLAZ:
mm13321 Marija Radic 12
```

#### Test 2

```
POKRETANJE: /a.out -p
STUDENTI.TXT
mr14123 Marko Antic 20
mm13321 Marija Radic 12
ml13011 Ivana Mitrovic 19
ml13066 Pera Simic 15
mv14003 Jovan Jovanovic 17

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

[Rešenje 3.38]

**Zadatak 3.39** Definisati strukturu `Datum`. Napisati funkciju koja poredi dva datuma hronološki. Potom, napisati i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma), sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza postoje među prethodno unetim datumima. Datumi se učitavaju sve do kraja ulaza.

#### Primer 1

```
POKRETANJE: ./a.out datoteka.txt
DATOTEKA.TXT
1.1.2013.
13.12.2016.
11.11.2011.
3.5.2015.
5.2.2009.

INTERAKCIJA SA PROGRAMOM:
Unesi sledeći datum: 13.12.2016.
postoji
Unesi sledeći datum: 10.5.2015.
ne postoji
Unesi sledeći datum: 5.2.2009.
postoji
```

## 3.4 Rešenja

### Rešenje 3.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
7 -lrt zbog funkcije clock_gettime() */
8
9 /* Naredne tri funkcije koje vrse pretragu, ukoliko se traženi
10 element pronadje u nizu, vracaju indeks pozicije na kojoj je
11 element pronadjen. Ovaj indeks je uvek nenegativan. Ako element
12 nije pronadjen u nizu, funkcije vracaju negativnu vrednost -1, kao
13 indikator neuspesne pretrage. */
14
15 /* Linearna pretraga: Funkcija pretrazuje niz a[] celih brojeva
16 duzine n, trazeci u njemu prvo pojavljivanje elementa x. Pretraga
17 se vrsti prostom iteracijom kroz niz. */
18 int linearna_pretraga(int a[], int n, int x)
19 {
20     int i;
21     for (i = 0; i < n; i++)
22         if (a[i] == x)
23             return i;
24     return -1;
25 }
26
27 /* Binarna pretraga: Funkcija trazi u sortiranom nizu a[] duzine n
28 broj x. Pretraga koristi osobinu sortiranosti niza i u svakoj
29 iteraciji polovi interval pretrage. */
30 int binarna_pretraga(int a[], int n, int x)
31 {
32     int levi = 0;
33     int desni = n - 1;
34     int srednji;
35     /* Dokle god je indeks levi levo od indeksa desni */
36     while (levi <= desni) {
37         /* Srednji indeks je njihova aritmeticka sredina */
38         srednji = (levi + desni) / 2;
39         /* Ako je element sa sredisnjim indeksom veci od x, tada se x
40         mora nalaziti u levom delu niza */
41         if (x < a[srednji])
42             desni = srednji - 1;
43         /* Ako je element sa sredisnjim indeksom manji od x, tada se x
44         mora nalaziti u desnom delu niza */
45         else if (x > a[srednji])
46             levi = srednji + 1;
47 }
```

```

47     else
48         /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
49            x pronadjen na poziciji srednji */
50         return srednji;
51     }
52     /* Ako element x nije pronadjen, vraca se -1 */
53     return -1;
54 }
55
56 /* Interpolaciona pretraga: Funkcija trazi u sortiranom nizu a[]
57    duzine n broj x. Pretraga koristi osobinu sortiranosti niza i
58    zasniva se na linearnej interpolaciji vrednosti koja se trazi
59    vrednostima na krajevima prostora pretrage. */
60 int interpolaciona_pretraga(int a[], int n, int x)
61 {
62     int levi = 0;
63     int desni = n - 1;
64     int srednji;
65     /* Dokle god je indeks levi levo od indeksa desni... */
66     while (levi <= desni) {
67         /* Ako je trazen element manji od pocetnog ili veci od
68            poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
69            nije u tom delu niza. Ova provera je neophodna, da se ne bi
70            dogodilo da se prilikom izracunavanja indeksa srednji izadje
71            izvan opsega indeksa [levi,desni] */
72         if (x < a[levi] || x > a[desni])
73             return -1;
74         /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
75            a[levi] i a[desni] jednaki, tada je jasno da je trazen broj x
76            jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
77            desni). Ova provera je neophodna, jer bi se u suprotnom
78            prilikom izracunavanja indeksa srednji pojavilo deljenje
79            nulom. */
80         else if (a[levi] == a[desni])
81             return levi;
82         /* Racunanje srednjeg indeksa */
83         srednji =
84             levi +
85             ((int) ((double) (x - a[levi]) / (a[desni] - a[levi]) *
86                     (desni - levi)));
87         /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
88            verovatno biti blize trazenoj vrednosti nego da je prosto uvek
89            uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
90            porediti sa pretragom recnika: ako neko trazi rec na slovo
91            'B', sigurno nece da otvorи recnik na polovini, vec verovatno
92            negde blize pocetku. */
93         /* Ako je element sa indeksom srednji veci od trazenog, tada se
94            trazen element mora nalaziti u levoj polovini niza */
95         if (x < a[srednji])
96             desni = srednji - 1;
97         /* Ako je element sa indeksom srednji manji od trazenog, tada se
98            trazen element mora nalaziti u desnoj polovini niza */

```

### 3 Algoritmi pretrage i sortiranja

---

```
99     else if (x > a[srednji])
100         levi = srednji + 1;
101     else
102         /* Ako je element sa indeksom srednji jednak trazenom, onda se
103            pretraga zavrsava na poziciji srednji */
104         return srednji;
105     }
106     /* U slucaju neuspesne pretrage vraca se -1 */
107     return -1;
108 }
109
110 int main(int argc, char **argv)
111 {
112     int a[MAX];
113     int n, i, x;
114     struct timespec vreme1, vreme2, vreme3, vreme4, vreme5, vreme6;
115     FILE *f;
116     /* Provera argumenata komandne linije */
117     if (argc != 3) {
118         fprintf(stderr,
119                 "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
120         ;
121         exit(EXIT_FAILURE);
122     }
123
124     /* Dimenzija niza */
125     n = atoi(argv[1]);
126     if (n > MAX || n <= 0) {
127         fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
128         exit(EXIT_FAILURE);
129     }
130
131     /* Broj koji se trazi */
132     x = atoi(argv[2]);
133     /* Elementi niza se generisu slucajno, tako da je svaki sledeci
134        veci od prethodnog. Funkcija srand() inicijalizuje pocetnu
135        vrednost sa kojom se kreće u izracunavanje sekvene
136        pseudo-slucajnih brojeva. Kako generisani niz ne bi uvek bio
137        isti, ova vrednost se postavlja na tekuce vreme u sekundama od
138        Nove godine 1970, tako da je za svako sledece pokretanje
139        programa (u vremenskim intervalima vecim od jedne sekunde) ove
140        vrednost drugacija. random()%100 vraca brojeve izmedju 0 i 99 */
141     srand(time(NULL));
142     for (i = 0; i < n; i++)
143         a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
144     /* Linearna pretraga */
145     printf("Linearna pretraga:\n");
146     /* Vreme proteklo od Nove godine 1970 */
147     clock_gettime(CLOCK_REALTIME, &vreme1);
148     i = linearna_pretraga(a, n, x);
149     /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
        linearu pretragu */
```

```

151     clock_gettime(CLOCK_REALTIME, &vreme2);
152     if (i == -1)
153         printf("Element nije u nizu\n");
154     else
155         printf("Element je u nizu na poziciji %d\n", i);
156     /* Binarna pretraga */
157     printf("Binarna pretraga:\n");
158     clock_gettime(CLOCK_REALTIME, &vreme3);
159     i = binarna_pretraga(a, n, x);
160     clock_gettime(CLOCK_REALTIME, &vreme4);
161     if (i == -1)
162         printf("Element nije u nizu\n");
163     else
164         printf("Element je u nizu na poziciji %d\n", i);
165     /* Interpolaciona pretraga */
166     printf("Interpolaciona pretraga:\n");
167     clock_gettime(CLOCK_REALTIME, &vreme5);
168     i = interpolaciona_pretraga(a, n, x);
169     clock_gettime(CLOCK_REALTIME, &vreme6);
170     if (i == -1)
171         printf("Element nije u nizu\n");
172     else
173         printf("Element je u nizu na poziciji %d\n", i);
174     /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
175     if ((f = fopen("vremena.txt", "a")) == NULL) {
176         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
177         exit(EXIT_FAILURE);
178     }
179
180     fprintf(f, "Dimenzija niza: %d\n", n);
181     fprintf(f, "\tLinearna: %10ld ns\n",
182             (vreme2.tv_sec - vreme1.tv_sec) * 1000000000 +
183             vreme2.tv_nsec - vreme1.tv_nsec);
184     fprintf(f, "\tBinarna: %19ld ns\n",
185             (vreme4.tv_sec - vreme3.tv_sec) * 1000000000 +
186             vreme4.tv_nsec - vreme3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
188             (vreme6.tv_sec - vreme5.tv_sec) * 1000000000 +
189             vreme6.tv_nsec - vreme5.tv_nsec);
190     /* Zatvaranje datoteke */
191     fclose(f);
192
193     exit(EXIT_SUCCESS);
}

```

## Rešenje 3.2

```

2 #include <stdio.h>
3 #include <stdlib.h>
4 #define MAX 1024

```

### 3 Algoritmi pretrage i sortiranja

---

```
6 int linearna_pretraga_r1(int a[], int n, int x)
7 {
8     int tmp;
9     /* Izlaz iz rekurzije */
10    if (n <= 0)
11        return -1;
12    /* Ako je prvi element trazeni */
13    if (a[0] == x)
14        return 0;
15    /* Pretraga ostatka niza */
16    tmp = linearna_pretraga_r1(a + 1, n - 1, x);
17    return tmp < 0 ? tmp : tmp + 1;
18 }

20 int linearna_pretraga_r2(int a[], int n, int x)
21 {
22     /* Izlaz iz rekurzije */
23     if (n <= 0)
24         return -1;
25     /* Ako je poslednji element trazeni */
26     if (a[n - 1] == x)
27         return n - 1;
28     /* Pretraga ostatka niza */
29     return linearna_pretraga_r2(a, n - 1, x);
30 }

32 int binarna_pretraga_r(int a[], int l, int d, int x)
33 {
34     int srednji;
35     if (l > d)
36         return -1;
37     /* Sredisnja pozicija na kojoj se trazi vrednost x */
38     srednji = (l + d) / 2;
39     /* Ako je element na sredisnoj poziciji trazeni */
40     if (a[srednji] == x)
41         return srednji;
42     /* Ako je trazeni broj veci od broja na sredisnoj poziciji,
43      pretrazuje se desna polovina niza */
44     if (a[srednji] < x)
45         return binarna_pretraga_r(a, srednji + 1, d, x);
46     /* Ako je trazeni broj manji od broja na sredisnoj poziciji,
47      pretrazuje se leva polovina niza */
48     else
49         return binarna_pretraga_r(a, l, srednji - 1, x);
50 }

52 int interpolaciona_pretraga_r(int a[], int l, int d, int x)
53 {
54     int p;
55     if (x < a[l] || x > a[d])
```

```

    return -1;
58  if (a[d] == a[1])
      return 1;
59  /* Pozicija na kojoj se trazi vrednost x */
60  p = 1 + (d - 1) * (x - a[1]) / (a[d] - a[1]);
61  if (a[p] == x)
62    return p;
63  if (a[p] < x)
64    return interpolaciona_pretraga_r(a, p + 1, d, x);
65  else
66    return interpolaciona_pretraga_r(a, 1, p - 1, x);
67 }

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

73   /* Ucitavanje traženog broja */
74   printf("Unesite traženi broj: ");
75   scanf("%d", &x);

76   /* Ucitavanje elemenata niza sve do kraja ulaza - očekuje se da
77    korisnik pritisne CTRL+D za naznaku kraja */
78   i = 0;
79   printf("Unesite sortiran niz elemenata: ");
80   while (i < MAX && scanf("%d", &a[i]) == 1) {
81     if (i > 0 && a[i] < a[i - 1]) {
82       fprintf(stderr,
83             "Elementi moraju biti uneseni u neopadajućem poretku\n");
84       exit(EXIT_FAILURE);
85     }
86     i++;
87   }

88   /* Linearna pretraga */
89   printf("Linearna pretraga\n");
90   indeks = linearna_pretraga_r1(a, i, x);
91   if (indeks == -1)
92     printf("Element se ne nalazi u nizu.\n");
93   else
94     printf("Pozicija elementa je %d.\n", indeks);

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
108     /* Interpolaciona pretraga */
109     printf("Interpolaciona pretraga\n");
110     indeks = interpolaciona_pretraga_r(a, 0, i - 1, x);
111     if (indeks == -1)
112         printf("Element se ne nalazi u nizu.\n");
113     else
114         printf("Pozicija elementa je %d.\n", indeks);
115
116     return 0;
117 }
```

#### Rešenje 3.3

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

/* O svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
typedef struct {
    /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
       int, npr. 20140123 */
    long indeks;
    char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
} Student;

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

/* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
   sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
   ako element nije pronadjen. */
int binarna_pretraga(Student a[], int n, long x)
{
    int levi = 0;
    int desni = n - 1;
    int srednji;
    /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
        /* Racuna se srednja pozicija */
        srednji = (levi + desni) / 2;
        /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
           se trazeni indeks mora nalaziti u levoj polovini niza */
        if (x < a[srednji].indeks)
```

```

38     desni = srednji - 1;
39     /* Ako je pak manji od traženog, tada se on mora nalaziti u
40      desnoj polovini niza */
41     else if (x > a[srednji].indeks)
42         levi = srednji + 1;
43     else
44         /* Ako je jednak traženom indeksu x, tada je pronađen student
45          sa traženom indeksom na poziciji srednji */
46         return srednji;
47     }
48     /* Ako nije pronađen, vraca se -1 */
49     return -1;
50 }

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

62 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
63   prosledjuju kao argumenti komandne linije */
int main(int argc, char *argv[])
{
    Student dosije[MAX_STUDENATA];
    FILE *fin = NULL;
    int i;
    int br_studenata = 0;
    long tražen_indeks = 0;
    char traženo_preszime[MAX_DUZINA];
    int bin_pretraga;

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

    /* Provera prosledjene opcije */
    if (strcmp(argv[2], "-indeks") == 0)
        bin_pretraga = 1;
    else if (strcmp(argv[2], "-prezime") == 0)
        bin_pretraga = 0;
    else {

```

### 3 Algoritmi pretrage i sortiranja

---

```
90     fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
91     exit(EXIT_FAILURE);
92 }

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

102 /* Citanje se vrsti sve dok postoji red sa informacijama o studentu
103 */
104 i = 0;
105 while (1) {
106     if (i == MAX_STUDENATA)
107         break;
108     if (fscanf(
109         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
110         dosije[i].prezime) != 3)
111         break;
112     i++;
113 }
114 br_studenata = i;

116 /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
117 fclose(fin);

118 /* Pretraga po indeksu */
119 if (bin_pretraga) {
120     /* Unos indeksa koji se binarno trazi u nizu */
121     printf("Unesite indeks studenta cije informacije zelite: ");
122     scanf("%ld", &trazen_indeks);
123     i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
124     /* Rezultat binarne pretrage */
125     if (i == -1)
126         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
127     else
128         printf("Indeks: %ld, Ime i prezime: %s %s\n",
129                dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
130 }
131 /* Pretraga po prezimenu */
132 else {
133     /* Unos prezimena koje se linearno trazi u nizu */
134     printf("Unesite prezime studenta cije informacije zelite: ");
135     scanf("%s", trazeno_prezime);
136     i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
137     /* Rezultat linearne pretrage */
138     if (i == -1)
139         printf("Ne postoji student sa prezimenom %s\n",
140                trazeno_prezime);
```

```

142     else
143         printf("Indeks: %ld, Ime i prezime: %s %s\n",
144             dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
145     }
146     exit(EXIT_SUCCESS);
}

```

### Rešenje 3.4

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
7
8 typedef struct {
9     long indeks;
10    char ime[MAX_DUZINA];
11    char prezime[MAX_DUZINA];
12 } Student;
13
14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                     long x)
16 {
17     /* Ako je pozicija elementa na levom kraju veca od pozicije
18      elementa na desnom kraju dela niza koji se pretrazuje, onda se
19      zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20      trazenog elementa pa se vraca -1 */
21     if (levi > desni)
22         return -1;
23     /* Racunanje pozicije srednjeg elementa */
24     int srednji = (levi + desni) / 2;
25     /* Da li je srednji bas onaj trazeni */
26     if (a[srednji].indeks == x) {
27         return srednji;
28     }
29     /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30      poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
31      je poznato da je niz sortiran po indeksu u rastucem poretku. */
32     if (x < a[srednji].indeks)
33         return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34     /* Inace ga treba traziti u desnoj polovini */
35     else
36         return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }
38
39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
41     /* Ako je niz prazan, vraca se -1 */
42 }

```

### 3 Algoritmi pretrage i sortiranja

---

```
if (n == 0)
    return -1;
/* Kako se trazi prvi student sa traenim prezimenom, pocinje se sa
   prvim studentom u nizu. */
if (strcmp(a[0].prezime, x) == 0)
    return 0;
int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
return i >= 0 ? 1 + i : -1;
}

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

/* Main funkcija mora imati argumente jer se ime datoteke i opcija
   prosledjuju kao argumenti komandne linije */
int main(int argc, char *argv[])
{
    Student dosije[MAX_STUDENATA];
    FILE *fin = NULL;
    int i;
    int br_studenata = 0;
    long trazen_indeks = 0;
    char trazeno_prezime[MAX_DUZINA];
    int bin_pretraga;

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

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

```

95  /* Otvaranje datoteke */
96  fin = fopen(argv[1], "r");
97  if (fin == NULL) {
98      fprintf(stderr,
99          "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
100     exit(EXIT_FAILURE);
101 }

103 /* Citanje se vrši sve dok postoji red sa informacijama o studentu
104 */
105 i = 0;
106 while (1) {
107     if (i == MAX_STUDENATA)
108         break;
109     if (fscanf(
110         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
111         dosije[i].prezime) != 3)
112         break;
113     i++;
114 }
115 br_studenata = i;

116 /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
117 fclose(fin);

118 /* Pretraga po indeksu */
119 if (bin_pretraga) {
120     /* Unos indeksa koji se binarno trazi u nizu */
121     printf("Unesite indeks studenta cije informacije zelite: ");
122     scanf("%ld", &trazen_indeks);
123     i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
124                                         trazen_indeks);
125
126     /* Rezultat binarne pretrage */
127     if (i == -1)
128         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
129     else
130         printf("Indeks: %ld, Ime i prezime: %s %s\n",
131               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132 }
133 /* Pretraga po prezimenu */
134 else {
135     /* Unos prezimena koje se linearno trazi u nizu */
136     printf("Unesite prezime studenta cije informacije zelite: ");
137     scanf("%s", trazeno_prezime);
138     i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
139                                           trazeno_prezime);
140
141     /* Rezultat linearne pretrage */
142     if (i == -1)
143         printf("Ne postoji student sa prezimenom %s\n",
144               trazeno_prezime);
145 }

```

### 3 Algoritmi pretrage i sortiranja

---

```
145     printf("Indeks: %ld, Ime i prezime: %s %s\n",
146            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
147 }
148 exit(EXIT_SUCCESS);
149 }
```

#### 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 tache 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 nizu
20    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 tacaka
35    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++) {
```

```

43     if (fabs(t[i].x) < fabs(najbliza.x)) {
44         najbliza = t[i];
45     }
46 }
47 return najbliza;
}
49 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
50  t dimenzije n */
51 Tacka najbliza_y_osi(Tacka t[], int n)
52 {
53     Tacka najbliza;
54     int i;
55     najbliza = t[0];
56     for (i = 1; i < n; i++) {
57         if (fabs(t[i].y) < fabs(najbliza.y)) {
58             najbliza = t[i];
59         }
60     }
61     return najbliza;
}
63 }

65 #define MAX 1024

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

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

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

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
95     }
96     n = i;
97
98     /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99        dodatnih argumenata */
100    if (argc == 2)
101        /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
102        najbliza = najbliza_koordinatnom(tacke, n);
103    /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
104       pitanju opcija -x */
105    else if (strcmp(argv[2], "-x") == 0)
106        /* Racuna se rastojanje u odnosu na x osu */
107        najbliza = najbliza_x_osi(tacke, n);
108    /* Ako je u pitanju opcija -y */
109    else if (strcmp(argv[2], "-y") == 0)
110        /* Racuna se rastojanje u odnosu na y osu */
111        najbliza = najbliza_y_osi(tacke, n);
112    else {
113        /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
114           korisnika i prekida se izvrsavanje programa */
115        fprintf(stderr, "Pogresna opcija\n");
116        exit(EXIT_FAILURE);
117    }
118
119    /* Stampanje koordinata trazene tacke */
120    printf("%g %g\n", najbliza.x, najbliza.y);
121
122    /* Zatvaranje datoteke */
123    fclose(ulaz);
124
125    exit(EXIT_SUCCESS);
126}
```

#### Rešenje 3.6

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

/* Tacnost */
#define EPS 0.001

int main()
{
    double l, d, s;

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

    /* Sve dok se ne pronadje trazena vrednost argumenta */
}
```

```

16 while (1) {
17     /* Polovi se interval */
18     s = (l + d) / 2;
19     /* Ako je absolutna vrednost kosinusa u ovoj tacki manja od
20      zadate tacnosti, prekida se pretraga */
21     if (fabs(cos(s)) < EPS) {
22         break;
23     }
24     /* Ako je nula u levom delu intervala, nastavlja se pretraga na
25      [l, s] */
26     if (cos(l) * cos(s) < 0)
27         d = s;
28     else
29         /* Inace, na intervalu [s, d] */
30         l = s;
31     }
32
33     /* Stampanje vrednosti trazene tacke */
34     printf("%g\n", s);
35
36     return 0;
37 }
```

**Rešenje 3.7**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 int main(int argc, char **argv)
7 {
8     double l, d, s, epsilon;
9
10    char ime_funkcije[6];
11
12    /* Pokazivac na funkciju koja ima jedan argument tipa double i
13       povratnu vrednost istog tipa */
14    double (*fp) (double);
15
16    /* Ako korisnik nije uneo argument, prijavljuje se greska */
17    if (argc != 2) {
18        fprintf(stderr, "Greska: ");
19        fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
20        fprintf(stderr,
21                "Program se poziva sa %s ime_funkcije iz math.h.\n",
22                argv[0]);
23        exit(EXIT_FAILURE);
24    }
25
26    /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
```

### 3 Algoritmi pretrage i sortiranja

---

```
    u komandnoj liniji */
28 strcpy(ime_funkcije, argv[1]);

30 /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
31 if (strcmp(ime_funkcije, "sin") == 0)
32     fp = &sin;
33 else if (strcmp(ime_funkcije, "cos") == 0)
34     fp = &cos;
35 else if (strcmp(ime_funkcije, "tan") == 0)
36     fp = &tan;
37 else if (strcmp(ime_funkcije, "atan") == 0)
38     fp = &atan;
39 else if (strcmp(ime_funkcije, "asin") == 0)
40     fp = &asin;
41 else if (strcmp(ime_funkcije, "log") == 0)
42     fp = &log;
43 else if (strcmp(ime_funkcije, "log10") == 0)
44     fp = &log10;
45 else {
46     fprintf(stderr, "Program ne podrzava trazenu funkciju!\n");
47     exit(EXIT_SUCCESS);
48 }

49 printf("Unesite krajeve intervala: ");
50 scanf("%lf %lf", &l, &d);

51 if ((*fp) (l) * (*fp) (d) >= 0) {
52     fprintf(stderr,
53             "Funkcija %s na intervalu [%g, %g] ne zadovoljava uslove\n",
54             ime_funkcije, l, d);
55     exit(EXIT_FAILURE);
56 }

57 printf("Unesite preciznost: ");
58 scanf("%lf", &epsilon);

59 /* Sve dok se ne pronadje trazena vrednost argumenta */
60 while (1) {
61     /* Polovi se interval */
62     s = (l + d) / 2;
63     /* Ako je apsolutna vrednost trazene funkcije u ovoj tacki manja
64      od zadate tacnosti, prekida se pretraga */
65     if (fabs((*fp) (s)) < epsilon) {
66         break;
67     }
68     /* Ako je nula u levom delu intervala, nastavlja se pretraga na
69      [l, s] */
70     if ((*fp) (l) * (*fp) (s) < 0)
71         d = s;
72     else
73         /* Inace, na intervalu [s, d] */
```

```

78     l = s;
79 }
80 /* Stampanje vrednosti trazene tacke */
81 printf("%g\n", s);
82
83 return 0;
84 }
```

### Rešenje 3.8

```

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

### 3 Algoritmi pretrage i sortiranja

---

```
42     return 0;
}
```

#### Rešenje 3.9

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

## Rešenje 3.10

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 unsigned int logaritam_a(unsigned int x)
5 {
6     /* Izlaz iz rekurzije */
7     if (x == 1)
8         return 0;
9     /* Rekursivni korak */
10    return 1 + logaritam_a(x >> 1);
11}
12
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
16      najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
17      do 31 */
18     int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
20     /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
22         /* Racuna se sredisnja pozicija */
23         s = (l + d) / 2;
24         /* Proverava se da li je na toj poziciji trazena jedinica */
25         if ((1 << s) <= x && (1 << (s + 1)) > x)
26             return s;
27         /* Pretraga desne polovine binarnog zapisa */
28         if ((1 << s) > x)
29             l = s - 1;
30         /* Pretraga leve polovine binarnog zapisa */
31         else
32             d = s + 1;
33     }
34     return s;
35 }
36
37 int main()
38 {
39     unsigned int x;
40
41     /* Unos podatka */
42     scanf("%u", &x);
43
44     /* Provera da li je uneti broj pozitivan */
45     if (x == 0) {
46         fprintf(stderr, "Logaritam od nule nije definisan\n");
47         exit(EXIT_FAILURE);
48     }
49
50     /* Ispis povratnih vrednosti funkcija */

```

### 3 Algoritmi pretrage i sortiranja

---

```
51     printf("%u %u\n", logaritam_a(x), logaritam_b(x));  
53     exit(EXIT_SUCCESS);  
}
```

#### Rešenje 3.12

*sort.h*

```
#ifndef _SORT_H_  
#define _SORT_H_ 1  
  
4 /* Selection sort: Funkcija sortira niz celih brojeva metodom  
5  sortiranja izborom. Ideja algoritma je sledeća: U svakoj  
6  iteraciji pronađi se najmanji element i premestiti ga na pocetak  
7  niza. Dakle, u prvoj iteraciji, pronađi se najmanji element, i  
8  dovodi ga na nulto mesto u nizu. U i-toj iteraciji najmanjih i-1  
9  elemenata su već na svojim pozicijama, pa se od elemenata sa  
10 indeksima od i do n-1 trazi najmanji, koji se dovodi na i-tu  
11 poziciju. */  
12 void selection_sort(int a[], int n);  
  
14 /* Insertion sort: Funkcija sortira niz celih brojeva metodom  
15 sortiranja umetanjem. Ideja algoritma je sledeća: neka je na  
16 pocetku i-te iteracije niz prvih i elemenata  
17 (a[0],a[1],...,a[i-1]) sortirano. U i-toj iteraciji treba element  
18 a[i] umetnuti na pravu poziciju među prvih i elemenata tako da se  
19 dobije niz duzine i+1 koji je sortiran. Ovo se radi tako što se  
20 i-ti element najpre uporedi sa njegovim prvim levim susedom  
21 (a[i-1]). Ako je a[i] veći, tada je on vec na pravom mestu, i niz  
22 a[0],a[1],...,a[i] je sortiran, pa se može preci na sledecu  
23 iteraciju. Ako je a[i-1] veći, tada se zamjenjuju a[i] i a[i-1], a  
24 zatim se proverava da li je potrebno dalje potiskivanje elemenata u  
25 levo, poredeci ga sa njegovim novim levim susedom. Ovim uzastopnim  
26 premestanjem se a[i] umeće na pravo mesto u nizu. */  
27 void insertion_sort(int a[], int n);  
  
28 /* Bubble sort: Funkcija sortira niz celih brojeva metodom mehurica.  
29 Ideja algoritma je sledeća: prolazi se kroz niz redom poredeći  
30 susedne elemente, i pri tom ih zamjenjujući ako su u pogresnom  
31 poretku. Ovim se najveći element poput mehurica istiskuje na  
32 "površinu", tj. na krajnju desnu poziciju. Nakon toga je potrebno  
33 ovaj postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih  
34 n-1 elemenata niza bez poslednjeg koji je postavljen na pravu  
35 poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i  
36 kracim prefiksima niza, cime se jedan po jedan istiskuju  
37 elementi na svoje prave pozicije. */  
38 void bubble_sort(int a[], int n);  
  
40 /* Selsort: Ovaj algoritam je jednostavno prosirenje sortiranja
```

```

42 umetanjem koje dopusta direktnu razmenu udaljenih elemenata.
43 Prosirenje se sastoji u tome da se kroz algoritam umetanja prolazi
44 vise puta; u prvom prolazu, umesto koraka i uzima se neki korak h
45 koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
46 tako se dobija h-sortiran niz, tj. niz u kome su elementi na
47 rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
48 drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
49 za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
50 kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
51 h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
52 algoritma, ali bilo koja vrednost ce rezultovati ispravnim
53 sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
54 vrednost 1. */
55 void shell_sort(int a[], int n);
56
57 /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
58 Sortiranje se vrsti od elementa na poziciji l do onog na poziciji
59 d. Na pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a
60 d je jednako poslednjem validnom indeksu u nizu. Funkcija niz
61 podeli na dve polovine, levu i desnu, koje zatim rekursivno
62 sortira. Od ova dva sortirana podniza, sortiran niz se dobija
63 ucesljavanjem, tj. istovremenim prolaskom kroz oba niza i izborom
64 trenutnog manjeg elementa koji se smesta u pomocni niz. Na kraju
65 algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u
66 originalni niz. */
67 void merge_sort(int a[], int l, int d);
68
69 /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
70 i d. Njena ideja sortiranja je izbor jednog elementa niza, koji se
71 naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
72 svi elementi levo od njega bice manji, a svi desno bice veci od
73 njega. Kako je pivot doveden na svoje mesto, da bi niz bio
74 kompletno sortiran, potrebno je sortirati elemente levo (manje) od
75 njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
76 manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
77 deo moze se uraditi rekursivno. */
78 void quick_sort(int a[], int l, int d);
79
80#endif

```

*sort.c*

```

1 #include "sort.h"
2
3 #define MAX 1000000
4
5 void selection_sort(int a[], int n)
6 {
7     int i, j;
8     int min;
9     int pom;

```

### 3 Algoritmi pretrage i sortiranja

---

```
10  /* U svakoj iteraciji ove petlje pronađazi se najmanji element
11   medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
12   poziciju i, dok se element na poziciji i premesta na poziciju
13   min, na kojoj se nalazio najmanji od gore navedenih elemenata.
14   */
15   for (i = 0; i < n - 1; i++) {
16     /* Unutrašnja petlja pronađazi poziciju min, na kojoj se nalazi
17      najmanji od elemenata a[i],...,a[n-1]. */
18     min = i;
19     for (j = i + 1; j < n; j++)
20       if (a[j] < a[min])
21         min = j;
22
23     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
24      su (i) i min razliciti, inace je nepotrebno.*/
25     if (min != i) {
26       pom = a[i];
27       a[i] = a[min];
28       a[min] = pom;
29     }
30   }
31
32 void insertion_sort(int a[], int n)
33 {
34   int i, j;
35
36   /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
37    sortiran */
38   for (i = 1; i < n; i++) {
39
40     /* U ovoj petlji se redom potiskuje element a[i] uлево koliko je
41      potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
42      bude sortiran. Indeks j je trenutna pozicija na kojoj se
43      element koji se umeće nalazi. Petlja se zavrsava ili kada
44      element dodje do levog kraja (j==0) ili kada se naidje na
45      element a[j-1] koji je manji od a[j]. */
46     int temp = a[i];
47     for (j = i; j > 0 && temp < a[j - 1]; j--)
48       a[j] = a[j - 1];
49     a[j] = temp;
50   }
51 }
52
53 void bubble_sort(int a[], int n)
54 {
55   int i, j;
56   int ind;
57
58   for (i = n, ind = 1; i > 1 && ind; i--)
59
60 }
```

```

62  /* Poput "mehurica" potiskuje se najveci element medju elementima
63   od a[0] do a[i-1] na poziciju i-1 uporedjujuci susedne
64   elemente niza i potiskujuci veci u desno */
65   for (j = 0, ind = 0; j < i - 1; j++) {
66     if (a[j] > a[j + 1]) {
67       int temp = a[j];
68       a[j] = a[j + 1];
69       a[j + 1] = temp;
70
71     /* Promenljiva ind registruje da je bilo prenestanja. Samo u
72      tom slucaju ima smisla ici na sledecu iteraciju, jer ako
73      nije bilo prenestanja, znaci da su svi elementi vec u
74      dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
75      niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
76      ispravno, ali manje efikasano, jer bi se cesto nepotrebno
77      vrsila mnoga uporedjivanja, kada je vec jasno da je
78      sortiranje zavrzeno. */
79     ind = 1;
80   }
81
82 void shell_sort(int a[], int n)
83 {
84   int h = n / 2, i, j;
85   while (h > 0) {
86     /* Insertion sort sa korakom h */
87     for (i = h; i < n; i++) {
88       int temp = a[i];
89       j = i;
90       while (j >= h && a[j - h] > temp) {
91         a[j] = a[j - h];
92         j -= h;
93       }
94       a[j] = temp;
95     }
96     h = h / 2;
97   }
98
99 void merge_sort(int a[], int l, int d)
100 {
101   int s;
102   static int b[MAX];           /* Pomocni niz */
103   int i, j, k;
104
105   /* Izlaz iz rekurzije */
106   if (l >= d)
107     return;
108
109   /* Odredjivanje sredisnjeg indeksa */
110   s = (l + d) / 2;
111
112

```

### 3 Algoritmi pretrage i sortiranja

---

```
114  /* Rekurzivni pozivi */
115  merge_sort(a, l, s);
116  merge_sort(a, s + 1, d);

117  /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
118  niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
119  prolazi kroz pomocni niz b[] */
120  i = l;
121  j = s + 1;
122  k = 0;

123  /* "Ucesljanje" koriscenjem pomocnog niza b[] */
124  while (i <= s && j <= d) {
125      if (a[i] < a[j])
126          b[k++] = a[i++];
127      else
128          b[k++] = a[j++];
129  }

130  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive j
131  iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
132  polovine niza */
133  while (i <= s)
134      b[k++] = a[i++];

135  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive i
136  iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
137  polovine niza */
138  while (j <= d)
139      b[k++] = a[j++];

140  /* Prepisuje se "ucesljeni" niz u originalni niz */
141  for (k = 0, i = l; i <= d; i++, k++)
142      a[i] = b[k];
143  }

144  /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
145  void swap(int a[], int i, int j)
146  {
147      int tmp = a[i];
148      a[i] = a[j];
149      a[j] = tmp;
150  }

151  void quick_sort(int a[], int l, int d)
152  {
153      int i, pivot_pozicija;

154      /* Izlaz iz rekurzije -- prazan niz */
155      if (l >= d)
156          return;
157  }
```

```

166  /* Particionisanje niza. Svi elementi na pozicijama levo od
168    pivot_pozicija (izuzev same pozicije 1) su strogo manji od
170    pivota. Kada se pronadje neki element manji od pivota, uvecava
172    se promenljiva pivot_pozicija i na tu poziciju se premesta
174    nadjeni element. Na kraju ce pivot_pozicija zaista biti pozicija
176    na koju treba smestiti pivot, jer ce svi elementi levo od te
178    pozicije biti manji a desno biti veci ili jednaki od pivota. */
179    pivot_pozicija = 1;
180    for (i = 1 + 1; i <= d; i++)
181      if (a[i] < a[1])
182        swap(a, ++pivot_pozicija, i);
183
184    /* Postavljanje pivota na svoje mesto */
185    swap(a, 1, pivot_pozicija);
186
187    /* Rekursivno sortiranje elemenata manjih od pivota */
188    quick_sort(a, 1, pivot_pozicija - 1);
189    /* Rekursivno sortiranje elemenata vecih od pivota */
190    quick_sort(a, pivot_pozicija + 1, d);
191  }

```

main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "sort.h"
5
6 /* Maksimalna duzina niza */
7 #define MAX 1000000
8
9 int main(int argc, char *argv[])
10 {
11   //*****
12   tip_sortiranja == 0 => selectionsort, (podrazumevano)
13   tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
14   tip_sortiranja == 2 => bubblesort, -b opcija komandne linije
15   tip_sortiranja == 3 => shellsort, -s opcija komandne linije
16   tip_sortiranja == 4 => mergesort, -m opcija komandne linije
17   tip_sortiranja == 5 => quicksort, -q opcija komandne linije
18   ****
19   int tip_sortiranja = 0;
20   //*****
21   tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
22   tip_niza == 1 => rastuce sortirani nizovi, -r opcija
23   tip_niza == 2 => opadajuce sortirani nizovi, -o opcija
24   ****
25   int tip_niza = 0;
26
27   /* Dimenzija niza koji se sortira */

```

### 3 Algoritmi pretrage i sortiranja

---

```
int dimenzija;
int i;
int niz[MAX];

/* Provera argumenata komandne linije */
if (argc < 2) {
    fprintf(stderr,
            "Program zahteva bar 2 argumenta komandne linije!\n");
    exit(EXIT_FAILURE);
}

/* Ocitavanje opcija i argumenata prilikom poziva programa */
for (i = 1; i < argc; i++) {
    /* Ako je u pitanju opcija... */
    if (argv[i][0] == '-') {
        switch (argv[i][1]) {
        case 'i':
            tip_sortiranja = 1;
            break;
        case 'b':
            tip_sortiranja = 2;
            break;
        case 's':
            tip_sortiranja = 3;
            break;
        case 'm':
            tip_sortiranja = 4;
            break;
        case 'q':
            tip_sortiranja = 5;
            break;
        case 'r':
            tip_niza = 1;
            break;
        case 'o':
            tip_niza = 2;
            break;
        default:
            printf("Pogresna opcija -%c\n", argv[i][1]);
            return 1;
            break;
        }
    }
    /* Ako je u pitanju argument, onda je to duzina niza koji treba
     da se sortira */
    else {
        dimenzija = atoi(argv[i]);
        if (dimenzija <= 0 || dimenzija > MAX) {
            fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
            exit(EXIT_FAILURE);
        }
    }
}
```

```

    }

81  /* Elementi niza se odredjuju slučajno, ali vodeći računa o tipu
83   niza dobijenom iz komandne linije. srand() funkcija obezbeđuje
85   novi seed za pozivanje rand funkcije, i kako generisani niz ne
86   bi uvek bio isti seed je postavljen na tekuce vreme u sekundama
87   od Nove godine 1970. rand()%100 daje brojeve između 0 i 99 */
88   srand(time(NULL));
89   if (tip_niza == 0)
90     for (i = 0; i < dimenzija; i++)
91       niz[i] = rand();
92   else if (tip_niza == 1)
93     for (i = 0; i < dimenzija; i++)
94       niz[i] = i == 0 ? rand() % 100 : niz[i - 1] + rand() % 100;
95   else
96     for (i = 0; i < dimenzija; i++)
97       niz[i] = i == 0 ? rand() % 100 : niz[i - 1] - rand() % 100;

98  /* Ispisivanje elemenata niza */
99  ****
100  Ovaj deo je komentarisani jer sledeći ispis ne treba da se nadje
101  na standardnom izlazu. Njegova svrha je samo bila provjera da li
102  je niz generisan u skladu sa opcijama komandne linije.

103  printf("Niz koji sortiramo je:\n");
104  for (i = 0; i < dimenzija; i++)
105    printf("%d\n", niz[i]);
106  ****

107  /* Sortiranje niza na odgovarajući način */
108  if (tip_sortiranja == 0)
109    selection_sort(niz, dimenzija);
110  else if (tip_sortiranja == 1)
111    insertion_sort(niz, dimenzija);
112  else if (tip_sortiranja == 2)
113    bubble_sort(niz, dimenzija);
114  else if (tip_sortiranja == 3)
115    shell_sort(niz, dimenzija);
116  else if (tip_sortiranja == 4)
117    merge_sort(niz, 0, dimenzija - 1);
118  else
119    quick_sort(niz, 0, dimenzija - 1);

120  /* Ispis elemenata niza */
121  ****
122  Ovaj deo je komentarisani jer vreme potrebno za njegovo
123  izvršavanje ne bi trebalo da bude uključeno u vreme izmereno
124  programom time. Takođe, kako je svrha ovog programa da prikaže
125  vremena razlicitih algoritama sortiranja, dimenzije nizova će
126  biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
127  od toliko elemenata. Ovaj deo je koriscen u razvoju programa
128
129
130
131

```

### 3 Algoritmi pretrage i sortiranja

---

```
133     zarad testiranja korektnosti.  
134  
135     printf("Sortiran niz je:\n");  
136     for (i = 0; i < dimenzija; i++)  
137         printf("%d\n", niz[i]);  
138     ****  
139     exit(EXIT_SUCCESS);  
140 }
```

#### Rešenje 3.13

```
1 #include <stdio.h>  
2 #include <string.h>  
3  
4 #define MAX_DIM 128  
5  
6 /* Funkcija za sortiranje niza karaktera */  
7 void selectionSort(char s[])  
8 {  
9     int i, j, min;  
10    char pom;  
11    for (i = 0; s[i] != '\0'; i++) {  
12        min = i;  
13        for (j = i + 1; s[j] != '\0'; j++)  
14            if (s[j] < s[min])  
15                min = j;  
16        if (min != i) {  
17            pom = s[i];  
18            s[i] = s[min];  
19            s[min] = pom;  
20        }  
21    }  
22 }  
23  
24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */  
25 int anagrami(char s[], char t[])  
26 {  
27     int i;  
28  
29     /* Ako dve niske imaju razlicit broj karaktera onda one nisu  
30     anagrami */  
31     if (strlen(s) != strlen(t))  
32         return 0;  
33  
34     /* Sortiramo niske */  
35     selectionSort(s);  
36     selectionSort(t);  
37  
38     /* Dve sortirane niske su anagrami ako i samo ako su jednake */  
39     for (i = 0; s[i] != '\0'; i++)
```

```

40     if (s[i] != t[i])
41         return 0;
42     return 1;
43 }
44
45 int main()
46 {
47     char s[MAX_DIM], t[MAX_DIM];
48
49     /* Ucitavanje niski sa ulaza */
50     printf("Unesite prvu nisku: ");
51     scanf("%s", s);
52     printf("Unesite drugu nisku: ");
53     scanf("%s", t);
54
55     /* Poziv funkcije */
56     if (anagrami(s, t))
57         printf("jesu\n");
58     else
59         printf("nisu\n");
60
61     return 0;
62 }
```

**Rešenje 3.14**

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX 256
4
5 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
6    sortiranom nizu celih brojeva */
7 int najmanje_rastojanje(int a[], int n)
8 {
9     int i, min;
10    min = a[1] - a[0];
11    for (i = 2; i < n; i++)
12        if (a[i] - a[i - 1] < min)
13            min = a[i] - a[i - 1];
14    return min;
15 }
16
17 int main()
18 {
19     int i, a[MAX];
20
21     /* Ucitavaju se elementi niza sve do kraja ulaza */
22     i = 0;
23     while (scanf("%d", &a[i]) != EOF)
24         i++;
25 }
```

### 3 Algoritmi pretrage i sortiranja

---

```
27  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
28   sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
29   se selection sort. */
30   selection_sort(a, i);
31
32   /* Ispis rezultata */
33   printf("%d\n", najmanje_rastojanje(a, i));
34
35   return 0;
36 }
```

#### Rešenje 3.15

```
1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX_DIM 256
4
5 /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
6   najvise puta pojavio u tom nizu */
7 int najvise puta(int a[], int n)
8 {
9     int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
10    /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
11    for (i = 0; i < n; i = j) {
12        br_pojava = 1;
13        for (j = i + 1; j < n && a[i] == a[j]; j++)
14            br_pojava++;
15        /* Ispitivanje da li se do tog trenutka i-ti element pojavio
16           najvise puta u nizu */
17        if (br_pojava > max_br_pojava) {
18            max_br_pojava = br_pojava;
19            i_max_pojava = i;
20        }
21    }
22    /* Vraca se element koji se najvise puta pojavio u nizu */
23    return a[i_max_pojava];
24}
25
26 int main()
27 {
28     int a[MAX_DIM], i;
29
30     /* Ucitavanje elemenata niza sve do kraja ulaza */
31     i = 0;
32     while (scanf("%d", &a[i]) != EOF)
33         i++;
34
35     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
36      sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
37      se merge sort. */
38     merge_sort(a, 0, i - 1);
39 }
```

```

39  /* Određuje se broj koji se najvise puta pojavio u nizu */
40  printf("%d\n", najvise_puta(a, i));
41
42  return 0;
43 }

```

### Rešenje 3.16

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX_DIM 256
4
5 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
6   u nizu, a 0 inace. Prepostavlja se da je niz sortiran u rastucem
7   poretku */
8 int binarna_pretraga(int a[], int n, int x)
9 {
10    int levi = 0, desni = n - 1, srednji;
11
12    while (levi <= desni) {
13        srednji = (levi + desni) / 2;
14        if (a[srednji] == x)
15            return 1;
16        else if (a[srednji] > x)
17            desni = srednji - 1;
18        else if (a[srednji] < x)
19            levi = srednji + 1;
20    }
21    return 0;
22}
23
24 int main()
25 {
26    int a[MAX_DIM], n = 0, zbir, i;
27
28    /* Ucitava se traženi zbir */
29    printf("Unesite traženi zbir: ");
30    scanf("%d", &zbir);
31
32    /* Ucitavaju se elementi niza sve do kraja ulaza */
33    i = 0;
34    printf("Unesite elemente niza: ");
35    while (scanf("%d", &a[i]) != EOF)
36        i++;
37    n = i;
38
39    /* Za sortiranje niza može se koristiti bilo koja od funkcija
40     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
41     se quick sort. */
42    quick_sort(a, 0, n - 1);
43 }

```

### 3 Algoritmi pretrage i sortiranja

---

```
43     for (i = 0; i < n; i++)  
44         /* Za i-ti element niza binarno se pretrazuje da li se u ostatku  
        niza nalazi element koji sabran sa njim ima ucitanu vrednost  
        zbiru */  
45         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {  
46             printf("da\n");  
47             return 0;  
48         }  
49         printf("ne\n");  
50     }  
51     return 0;  
52 }
```

#### Rešenje 3.17

```
#include <stdio.h>  
#define MAX_DIM 256  
  
/* Funkcija objedinjuje nizove niz1 i niz2 dimenzija dim1 i dim2, a  
   rezultat cuva u nizu dim3 za koji je rezervisano dim3 elemenata */  
int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,  
          int dim3)  
{  
    int i = 0, j = 0, k = 0;  
    /* U slučaju da je dimenzija treceg niza manja od neophodne,  
       funkcija vraca -1 */  
    if (dim3 < dim1 + dim2)  
        return -1;  
  
    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog  
       od njih */  
    while (i < dim1 && j < dim2) {  
        if (niz1[i] < niz2[j])  
            niz3[k++] = niz1[i++];  
        else  
            niz3[k++] = niz2[j++];  
    }  
    /* Ostatak prvog niza prepisujemo u treci */  
    while (i < dim1)  
        niz3[k++] = niz1[i++];  
  
    /* Ostatak drugog niza prepisujemo u treci */  
    while (j < dim2)  
        niz3[k++] = niz2[j++];  
    return dim1 + dim2;  
}  
  
int main()  
{  
    int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
```

```

36 int i = 0, j = 0, k, dim3;
37
38 /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
39    Prepostavka je da na ulazu nece biti vise od MAX_DIM elemenata
40 */
41 printf("Unesite elemente prvog niza: ");
42 while (1) {
43     scanf("%d", &niz1[i]);
44     if (niz1[i] == 0)
45         break;
46     i++;
47 }
48 printf("Unesite elemente drugog niza: ");
49 while (1) {
50     scanf("%d", &niz2[j]);
51     if (niz2[j] == 0)
52         break;
53     j++;
54 }
55
56 /* Poziv trazene funkcije */
57 dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);
58
59 /* Ispis niza */
60 for (k = 0; k < dim3; k++)
61     printf("%d ", niz3[k]);
62     printf("\n");
63
64 return 0;
}

```

### Rešenje 3.18

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     FILE *fin1 = NULL, *fin2 = NULL;
8     FILE *fout = NULL;
9     char ime1[11], ime2[11];
10    char prezime1[16], prezime2[16];
11    int kraj1 = 0, kraj2 = 0;
12
13    /* Ako nema dovoljno argumenata komandne linije */
14    if (argc < 3) {
15        fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
16        ;
17        exit(EXIT_FAILURE);
18    }

```

### 3 Algoritmi pretrage i sortiranja

---

```
18  /* Otvaranje datoteke zadate prvim argumentom komandne linije */
19  fin1 = fopen(argv[1], "r");
20  if (fin1 == NULL) {
21      fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
22      exit(EXIT_FAILURE);
23  }

24  /* Otvaranje datoteke zadate drugim argumentom komandne linije */
25  fin2 = fopen(argv[2], "r");
26  if (fin2 == NULL) {
27      fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
28      exit(EXIT_FAILURE);
29  }

30  /* Otvaranje datoteke za upis rezultata */
31  fout = fopen("ceo-tok.txt", "w");
32  if (fout == NULL) {
33      fprintf(stderr,
34              "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
35      exit(EXIT_FAILURE);
36  }

37  /* Citanje narednog studenta iz prve datoteke */
38  if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
39      kraj1 = 1;

40  /* Citanje narednog studenta iz druge datoteke */
41  if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
42      kraj2 = 1;

43  /* Sve dok nije dostignut kraj neke datoteke */
44  while (!kraj1 && !kraj2) {
45      int tmp = strcmp(ime1, ime2);
46      if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
47          /* Ime i prezime iz prve datoteke je leksikografski ranije, i
48             biva upisano u izlaznu datoteku */
49          fprintf(fout, "%s %s\n", ime1, prezime1);
50          /* Citanje narednog studenta iz prve datoteke */
51          if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
52              kraj1 = 1;
53      } else {
54          /* Ime i prezime iz druge datoteke je leksikografski ranije, i
55             biva upisano u izlaznu datoteku */
56          fprintf(fout, "%s %s\n", ime2, prezime2);
57          /* Citanje narednog studenta iz druge datoteke */
58          if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
59              kraj2 = 1;
60      }
61  }

62  /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
```

```

70     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
71     treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72 */
73 while (!kraj1) {
74     fprintf(fout, "%s %s\n", ime1, prezime1);
75     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
76         kraj1 = 1;
77 }
78
79 /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
80    datoteke, onda ima jos studenata u drugoj datoteci, koje treba
81    prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
82 while (!kraj2) {
83     fprintf(fout, "%s %s\n", ime2, prezime2);
84     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
85         kraj2 = 1;
86 }
87
88 /* Zatvaranje datoteka */
89 fclose(fin1);
90 fclose(fin2);
91 fclose(fout);
92
93 exit(EXIT_SUCCESS);
94 }
```

### Rešenje 3.19

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 #define MAX_BR_TACAKA 128
7
8 /* Struktura koja reprezentuje koordinate tacke */
9 typedef struct Tacka {
10     int x;
11     int y;
12 } Tacka;
13
14 /* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka
15   (0,0) */
16 float rastojanje(Tacka A)
17 {
18     return sqrt(A.x * A.x + A.y * A.y);
19 }
20
21 /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
22   pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)
```

### 3 Algoritmi pretrage i sortiranja

---

```
25 {  
26     int min, i, j;  
27     Tacka tmp;  
28  
29     for (i = 0; i < n - 1; i++) {  
30         min = i;  
31         for (j = i + 1; j < n; j++) {  
32             if (rastojanje(t[j]) < rastojanje(t[min])) {  
33                 min = j;  
34             }  
35         }  
36         if (min != i) {  
37             tmp = t[i];  
38             t[i] = t[min];  
39             t[min] = tmp;  
40         }  
41     }  
42  
43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */  
44 void sortiraj_po_x(Tacka t[], int n)  
45 {  
46     int min, i, j;  
47     Tacka tmp;  
48  
49     for (i = 0; i < n - 1; i++) {  
50         min = i;  
51         for (j = i + 1; j < n; j++) {  
52             if (abs(t[j].x) < abs(t[min].x)) {  
53                 min = j;  
54             }  
55         }  
56         if (min != i) {  
57             tmp = t[i];  
58             t[i] = t[min];  
59             t[min] = tmp;  
60         }  
61     }  
62  
63 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */  
64 void sortiraj_po_y(Tacka t[], int n)  
65 {  
66     int min, i, j;  
67     Tacka tmp;  
68  
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         }
```

```

77     }
78     if (min != i) {
79         tmp = t[i];
80         t[i] = t[min];
81         t[min] = tmp;
82     }
83 }

85 int main(int argc, char *argv[])
{
86     FILE *ulaz;
87     FILE *izlaz;
88     Tacka tacke[MAX_BR_TACAKA];
89     int i, n;
90
91     /* Proveravanje broja argumenata komandne linije: očekuje se ime
92      izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
93      datoteke, tj. 4 argumenta */
94     if (argc != 4) {
95         fprintf(stderr,
96             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
97         return 0;
98     }
99
100    /* Otvaranje datoteke u kojoj su zadate tacke */
101    ulaz = fopen(argv[2], "r");
102    if (ulaz == NULL) {
103        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
104            argv[2]);
105        return 0;
106    }
107
108    /* Otvaranje datoteke u koju treba upisati rezultat */
109    izlaz = fopen(argv[3], "w");
110    if (izlaz == NULL) {
111        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
112            argv[3]);
113        return 0;
114    }
115
116    /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
117       koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
118       brojacem i. */
119    i = 0;
120    while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
121        i++;
122    }
123
124    /* Ukupan broj procitanih tacaka */
125    n = i;
126
127

```

### 3 Algoritmi pretrage i sortiranja

---

```
129  /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
130   "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
131   (karakter -), a karakter argv[1][1] odreduje kriterijum
132   sortiranja */
133   switch (argv[1][1]) {
134     case 'x':
135       /* Sortiranje po vrednosti x koordinate */
136       sortiraj_po_x(tacke, n);
137       break;
138     case 'y':
139       /* Sortiranje po vrednosti y koordinate */
140       sortiraj_po_y(tacke, n);
141       break;
142     case 'o':
143       /* Sortiranje po udaljenosti od koordinatnog pocetka */
144       sortiraj_po_rastojanju(tacke, n);
145       break;
146   }
147
148   /* Upisivanje dobijenog niza u izlaznu datoteku */
149   for (i = 0; i < n; i++) {
150     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
151   }
152
153   /* Zatvaranje otvorenih datoteka */
154   fclose(ulaz);
155   fclose(izlaz);
156
157   return 0;
}
```

### Rešenje 3.20

```
1 #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 pronađe poziciju min, na kojoj se nalazi
23      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)
27             min = j;
28     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
29      su (i) i min razliciti, inace je nepotrebno. */
30     if (min != i) {
31         pom = a[i];
32         a[i] = a[min];
33         a[min] = pom;
34     }
35 }
36
37 /* Funkcija sortira niz gradjana rastuce po prezimenima */
38 void sort_prezime(Gradjanin a[], int n)
39 {
40     int i, j;
41     int min;
42     Gradjanin pom;
43
44     for (i = 0; i < n - 1; i++) {
45         /* Unutrasnja petlja pronađe poziciju min, na kojoj se nalazi
46          najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
47         min = i;
48         for (j = i + 1; j < n; j++)
49             if (strcmp(a[j].prezime, a[min].prezime) < 0)
50                 min = j;
51         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
52          su (i) i min razliciti, inace je nepotrebno. */
53         if (min != i) {
54             pom = a[i];
55             a[i] = a[min];
56             a[min] = pom;
57         }
58     }
59 }
60
61 /* Pretraga niza Gradjana */
62 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
63 {
64     int i;
65     for (i = 0; i < n; i++)
66         if (strcmp(a[i].ime, x->ime) == 0
67             && strcmp(a[i].prezime, x->prezime) == 0)
68             return i;
69     return -1;

```

### 3 Algoritmi pretrage i sortiranja

---

```
71 }
73
74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;
80
81     /* Otvaranje datoteke */
82     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
83         fprintf(stderr,
84             "Neunesno otvaranje datoteke biracki-spisak.txt.\n");
85         exit(EXIT_FAILURE);
86     }
87
88     /* Citanje sadrzaja */
89     for (i = 0;
90         fscanf(fp, "%s %s", spisak1[i].ime,
91                 spisak1[i].prezime) != EOF; i++)
92         spisak2[i] = spisak1[i];
93     n = i;
94
95     /* Zatvaranje datoteke */
96     fclose(fp);
97
98     sort_ime(spisak1, n);
99
100    *****
101    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102        sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104    printf("Biracki spisak [uredjen prema imenima]:\n");
105    for(i=0; i<n; i++)
106        printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107    *****
108
109    sort_prezime(spisak2, n);
110
111    *****
112    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
113        sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
115    printf("Biracki spisak [uredjen prema prezimenima]:\n");
116    for(i=0; i<n; i++)
117        printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118    *****
119
120    /* Linearno pretrazivanje nizova */
121    for (i = 0; i < n; i++)
122        if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
```

```

123     isti_rbr++;

125     /* Alternativno (efikasnije) resenje */
126     //*****
127     for(i=0; i<n ;i++)
128         if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
129             strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130             isti_rbr++;
131     //***** */

133     /* Ispis rezultata */
134     printf("%d\n", isti_rbr);
135
136     exit(EXIT_SUCCESS);
137 }
```

### Rešenje 3.22

```

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

5 #define MAX_BR_REC 128
6 #define MAX_DUZINA_REC 32
7
8 /* Funkcija koja izracunava broj suglasnika u reci */
9 int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Prolaz karakter po karakter kroz zadatu nisku */
15     for (i = 0; s[i]; i++) {
16         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17            pokriven slucaj i malih i velikih suglasnika. */
18         if (isalpha(s[i])) {
19             c = toupper(s[i]);
20             /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
21            */
22             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
23                 suglasnici++;
24         }
25     }
26     /* Vraca se izracunata vrednost */
27     return suglasnici;
28 }
29
30 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
31    duzini reci se mora proslediti zbog pravilnog upravljanja
32    memorijom */
33 void sortiraj_recu(char reci[][MAX_DUZINA_REC], int n)
```

### 3 Algoritmi pretrage i sortiranja

---

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

```

85  /* Sve dok se moze pročitati sledeća rec */
86  while (fscanf(ulaz, "%s", reci[i]) != EOF) {
87      /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
88         prekida se ucitavanje */
89      if (i == MAX_BR_REC)
90          break;
91      /* Priprema brojaca za narednu iteraciju */
92      i++;
93  }

94  /* n je duzina niza reci i predstavlja poslednju vrednost
95   koriscenog brojaca */
96  n = i;
97  /* Poziv funkcije za sortiranje reci */
98  sortiraj_reci(reci, n);

100  /* Ispis sortiranog niza reci */
101  for (i = 0; i < n; i++) {
102      printf("%s ", reci[i]);
103  }
104  printf("\n");

105  /* Zatvaranje datoteke */
106  fclose(ulaz);

107  return 0;
108 }

```

### Rešenje 3.23

```

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

4
5 #define MAX_ARTIKALA 100000

6
7 /* Struktura koja predstavlja jedan artikal */
8 typedef struct art {
9     long kod;
10    char naziv[20];
11    char proizvodjac[20];
12    float cena;
13 } Artikal;
14
15 /* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
16   traženim bar kodom */
17 int binarna_pretraga(Artikal a[], int n, long x)
18 {
19     int levi = 0;
20     int desni = n - 1;

```

### 3 Algoritmi pretrage i sortiranja

---

```
22  /* Dokle god je indeks levi levo od indeksa desni */
23  while (levi <= desni) {
24      /* Racuna se sredisnji indeks */
25      int srednji = (levi + desni) / 2;
26      /* Ako je sredisnji element veci od trazenog, tada se trazeni
27          mora nalaziti u levoj polovini niza */
28      if (x < a[srednji].kod)
29          desni = srednji - 1;
30      /* Ako je sredisnji element manji od trazenog, tada se trazeni
31          mora nalaziti u desnoj polovini niza */
32      else if (x > a[srednji].kod)
33          levi = srednji + 1;
34      else
35          /* Ako je sredisnji element jednak trazenom, tada je artikal sa
36              bar kodom x pronadjen na poziciji srednji */
37          return srednji;
38      }
39      /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
40      return -1;
41  }
42
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  }
62
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, "Neuspesno otvaranje datoteke artikli.txt.\n");
75     exit(EXIT_FAILURE);
76 }

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

87 n = i;

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

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

112 kod = 0;
113 while (1) {
114     printf("-----\n");
115     printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
116     printf("- Za nov racun unesite kod artikla!\n\n");
117     /* Unos bar koda provog artikla sledeceg kupca */
118     if (scanf("%ld", &kod) == EOF)
119         break;
120     /* Trenutni racun novog kupca */
121     racun = 0;
122     /* Za sve artikle trenutnog kupca */
123 }

```

### 3 Algoritmi pretrage i sortiranja

---

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

#### Rešenje 3.24

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 500
6
7 /* Struktura sa svim informacijama o pojedinacnom studentu */
8 typedef struct {
9     char ime[21];
10    char prezime[26];
11    int prisustvo;
12    int zadaci;
13 } Student;
14
15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
16    rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
18 {
19     int i, j;
20     int min;
21     Student pom;
```

```

23   for (i = 0; i < n - 1; i++) {
24     min = i;
25     for (j = i + 1; j < n; j++)
26       if (strcmp(niz[j].ime, niz[min].ime) < 0)
27         min = j;
28
29     if (min != i) {
30       pom = niz[min];
31       niz[min] = niz[i];
32       niz[i] = pom;
33     }
34   }
35 }
36
37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38  zadataka opadajuce, a ukoliko neki studenti imaju isti broj
39  uradjenih zadataka sortiraju se po duzini imena rastuce. */
40 void sort_zadatke_pa_imena(Student niz[], int n)
41 {
42   int i, j;
43   int max;
44   Student pom;
45   for (i = 0; i < n - 1; i++) {
46     max = i;
47     for (j = i + 1; j < n; j++)
48       if (niz[j].zadaci > niz[max].zadaci)
49         max = j;
50     else if (niz[j].zadaci == niz[max].zadaci
51               && strlen(niz[j].ime) < strlen(niz[max].ime))
52       max = j;
53     if (max != i) {
54       pom = niz[max];
55       niz[max] = niz[i];
56       niz[i] = pom;
57     }
58   }
59 }
60
61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
62  su bili opadajuce. Ukoliko neki studenti imaju isti broj casova,
63  sortiraju se opadajuce po broju uradjenih zadataka, a ukoliko se
64  i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65  prezimenu opadajuce. */
66 void sort_prisustvo_pa_zadatke_pa_prezimena(Student niz[], int n)
67 {
68   int i, j;
69   int max;
70   Student pom;
71   for (i = 0; i < n - 1; i++) {
72     max = i;
73     for (j = i + 1; j < n; j++)
74       if (niz[j].prisustvo > niz[max].prisustvo)

```

### 3 Algoritmi pretrage i sortiranja

---

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

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

```

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

129
130  /* Kreiranje drugog spiska studenata po drugom kriterijumu */
131  sort_zadatke_pa_imena(praktikum, br_studenata);
132  /* Otvaranje datoteke za pisanje */
133  if ((fp = fopen("dat2.txt", "w")) == NULL) {
134      fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
135      exit(EXIT_FAILURE);
136  }
137  /* Upis niza u datoteku */
138  fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
139  fprintf(fp, "pa po duzini imena rastuce:\n");
140  for (i = 0; i < br_studenata; i++) {
141      fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
142              praktikum[i].prezime, praktikum[i].prisustvo,
143              praktikum[i].zadaci);
144  /* Zatvaranje datoteke */
145  fclose(fp);

146  /* Kreiranje treceg spiska studenata po trećem kriterijumu */
147  sort_prisustvo_pa_zadatke_pa_prezimena(praktikum, br_studenata);
148  /* Otvaranje datoteke za pisanje */
149  if ((fp = fopen("dat3.txt", "w")) == NULL) {
150      fprintf(stderr, "Neupesno otvaranje datoteke dat3.txt.\n");
151      exit(EXIT_FAILURE);
152  }
153  /* Upis niza u datoteku */
154  fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
155  fprintf(fp, "pa po broju zadataka,\n");
156  fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
157  for (i = 0; i < br_studenata; i++) {
158      fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
159              praktikum[i].prezime, praktikum[i].prisustvo,
160              praktikum[i].zadaci);
161  /* Zatvaranje datoteke */
162  fclose(fp);

163  exit(EXIT_SUCCESS);
164 }

```

**Rešenje 3.25**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define KORAK 10
6
7 /* Struktura koja opisuje jednu pesmu */
8 typedef struct {

```

### 3 Algoritmi pretrage i sortiranja

---

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

14 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
15    rad qsort funkcije) */
16 int uporedi_gledanost(const void *pp1, const void *pp2)
17 {
18     Pesma *p1 = (Pesma *) pp1;
19     Pesma *p2 = (Pesma *) pp2;
20
21     return p2->broj_gledanja - p1->broj_gledanja;
22 }

24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
25    funkcije) */
26 int uporedi_naslove(const void *pp1, const void *pp2)
27 {
28     Pesma *p1 = (Pesma *) pp1;
29     Pesma *p2 = (Pesma *) pp2;
30
31     return strcmp(p1->naslov, p2->naslov);
32 }

34 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
35    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
37 {
38     Pesma *p1 = (Pesma *) pp1;
39     Pesma *p2 = (Pesma *) pp2;
40
41     return strcmp(p1->izvodjac, p2->izvodjac);
42 }

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

```

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

```

### 3 Algoritmi pretrage i sortiranja

---

```
112     pesme[i].izvodjac = NULL;      /* Memorija za smestanje procitanih
113                                karaktera */
114
115     /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
116        izvodjaca) citaju se karakteri iz datoteke */
117     while ((c = fgetc(ulaz)) != ' ') {
118         /* Proverav da li postoji dovoljno memorije za smestanje
119            procitanog karaktera */
120         if (j == alocirano) {
121
122             /* Ako ne, ako je potrosena sva alocirana memorija, alocira
123                se novih KORAK mesta */
124             alocirano += KORAK;
125             pesme[i].izvodjac =
126                 (char *) realloc(pesme[i].izvodjac,
127                                 alocirano * sizeof(char));
128
129             /* Provera da li je nova alokacija uspesna */
130             if (pesme[i].izvodjac == NULL) {
131                 /* Ako nije oslobadja se sva memorija zauzeta do ovog
132                   koraka */
133                 for (k = 0; k < i; k++) {
134                     free(pesme[k].izvodjac);
135                     free(pesme[k].naslov);
136                 }
137                 free(pesme);
138                 /* I prekida sa izvrsavanjem programa */
139                 return 0;
140             }
141
142         }
143         /* Ako postoji dovoljno memorije, smestamo procitani karakter
144           */
145         pesme[i].izvodjac[j] = c;
146         j++;
147         /* I nastavlja se sa citanjem */
148     }
149
150     /* Upis terminirajuce nule na kraj reci */
151     pesme[i].izvodjac[j] = '\0';
152
153     /* Preskace se karakter - */
154     fgetc(ulaz);
155
156     /* Preskace se razmak */
157     fgetc(ulaz);
158
159     /* Cita se naslov pesme */
160     j = 0;                      /* Pozicija na koju treba smestiti
161                                procitani karakter */
162     alocirano = 0;               /* Broj alociranih mesta */
163     pesme[i].naslov = NULL;     /* Memorija za smestanje procitanih
```

```

164                                     karaktera */
165
166     /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
167     karakteri iz datoteke */
168     while ((c = fgetc(ulaz)) != ',') {
169         /* Provera da li postoji dovoljno memorije za smestanje
170         procitanog karaktera */
171         if (j == alocirano) {
172             /* Ako ne, ako je potrosena sva alocirana memorija, alocira
173             se novih KORAK mesta */
174             alocirano += KORAK;
175             pesme[i].naslov =
176                 (char *) realloc(pesme[i].naslov,
177                                 alocirano * sizeof(char));
178
179             /* Provera da li je nova alokacija uspesna */
180             if (pesme[i].naslov == NULL) {
181                 /* Ako nije, oslobadja se sva memorija zauzeta do ovog
182                 koraka */
183                 for (k = 0; k < i; k++) {
184                     free(pesme[k].izvodjac);
185                     free(pesme[k].naslov);
186                 }
187                 free(pesme[i].izvodjac);
188                 free(pesme);
189
190                 /* I prekida izvrsavanje programa */
191                 return 0;
192             }
193             /* Ako postoji dovoljno memorije, smesta se procitani karakter
194             */
195             pesme[i].naslov[j] = c;
196             j++;
197             /* I nastavlja dalje sa citanjem */
198         }
199         /* Upisuje se terminirajuca nula na kraj reci */
200         pesme[i].naslov[j] = '\0';
201
202         /* Preskace se razmak */
203         fgetc(ulaz);
204
205         /* Cita se broj gledanja */
206         broj_gledanja = 0;
207
208         /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
209         datoteke */
210         while ((c = fgetc(ulaz)) != '\n') {
211             broj_gledanja = broj_gledanja * 10 + (c - '0');
212         }
213         pesme[i].broj_gledanja = broj_gledanja;

```

### 3 Algoritmi pretrage i sortiranja

---

```
214     /* Prelazi se na citanje sledece pesme */
215     i++;
216 }
217
218 /* Informacija o broju pročitanih pesama */
219 n = i;
220 /* Zatvaranje nepotrebne datoteke */
221 fclose(ulaz);
222
223 /* Analiza argumenta komandne linije */
224 if (argc == 1) {
225     /* Nema dodatnih opcija => sortiranje po broju gledanja */
226     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
227 } else {
228     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
229         /* Sortiranje po naslovu */
230         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
231     } else {
232         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
233             /* Sortiranje po izvodjacu */
234             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
235         } else {
236             fprintf(stderr, "Nedozvoljeni argumenti!\n");
237             free(pesme);
238             return 0;
239         }
240     }
241
242     /* Ispis rezultata */
243     for (i = 0; i < n; i++) {
244         printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
245                pesme[i].broj_gledanja);
246     }
247
248     /* Oslobođjanje memorije */
249     for (i = 0; i < n; i++) {
250         free(pesme[i].izvodjac);
251         free(pesme[i].naslov);
252     }
253     free(pesme);
254
255     return 0;
256 }
```

#### Rešenje 3.28

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

```

6  /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
7   * kolona */
8  int zbir_vrste(int **a, int v, int m)
9  {
10    int i, zbir = 0;
11
12    for (i = 0; i < m; i++) {
13      zbir += a[v][i];
14    }
15    return zbir;
16}
17
18 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
19   * osnovu zbira koriscenjem selection sort algoritma */
20 void sortiraj_vrste(int **a, int n, int m)
21 {
22   int i, j, min;
23
24   for (i = 0; i < n - 1; i++) {
25     min = i;
26     for (j = i + 1; j < n; j++) {
27       if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
28         min = j;
29       }
30     }
31     if (min != i) {
32       int *tmp;
33       tmp = a[i];
34       a[i] = a[min];
35       a[min] = tmp;
36     }
37   }
38
39   int main(int argc, char *argv[])
40 {
41   int **a;
42   int n, m;
43
44   /* Unos dimenzija matrice */
45   printf("Unesite dimenzije matrice: ");
46   scanf("%d %d", &n, &m);
47
48   /* Alokacija memorije */
49   a = alociraj_matricu(n, m);
50   if (a == NULL) {
51     fprintf(stderr, "Neuspesna alokacija matrice\n");
52     exit(EXIT_FAILURE);
53   }
54
55   /* Ucitavanje elementa matrice */
56   printf("Unesite elemente matrice po vrstama:\n");

```

### 3 Algoritmi pretrage i sortiranja

---

```
58    ucitaj_matricu(a, n, m);
59
60    /* Poziv funkcije koja sortira vrste matrice prema zbiru */
61    sortiraj_vrste(a, n, m);
62
63    /* Ispis rezultujuce matrice */
64    printf("Sortirana matrica je:\n");
65    ispisi_matricu(a, n, m);
66
67    /* Oslobadjanje memorije */
68    a = dealociraj_matricu(a, n);
69
70    return 0;
71}
```

#### Rešenje 3.31

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <search.h>
5
6 #define MAX 100
7
8 /* Funkcija poredjenja dva cela broja (neopadajuci poredak) */
9 int poredi_int(const void *a, const void *b)
10 {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
12      se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13     int b1 = *((int *) a);
14     int b2 = *((int *) b);
15
16     /* Zbog moguceg prekoracenja opsega celih brojeva, oduzimanje b1-b2
17      treba izbegavati */
18     if (b1 > b2)
19         return 1;
20     else if (b1 < b2)
21         return -1;
22     else
23         return 0;
24 }
25
26 /* Funkcija poredjenja dva cela broja (nerastuci poredak) */
27 int poredi_int_nerastuce(const void *a, const void *b)
28 {
29     /* Za obrnuti poredak treba samo promeniti znak vrednosti koju koju
30      vraca prethodna funkcija */
31     return -poredi_int(a, b);
32 }
33
34 int main()
```

```

35 {
36     size_t n;
37     int i, x;
38     int a[MAX], *p = NULL;
39
40     /* Unos dimenzije */
41     printf("Uneti dimenziju niza: ");
42     scanf("%ld", &n);
43     if (n > MAX)
44         n = MAX;
45
46     /* Unos elementa niza */
47     printf("Uneti elemente niza:\n");
48     for (i = 0; i < n; i++)
49         scanf("%d", &a[i]);
50
51     /* Sortiranje niza celih brojeva */
52     qsort(a, n, sizeof(int), &poredi_int);
53
54     /* Prikaz sortiranog niz */
55     printf("Sortirani niz u rastucem poretku:\n");
56     for (i = 0; i < n; i++)
57         printf("%d ", a[i]);
58     putchar('\n');
59
60     /* Pretrazivanje niza */
61     /* Vrednost koja ce biti trazena u nizu */
62     printf("Uneti element koji se trazi u nizu: ");
63     scanf("%d", &x);
64
65     /* Binarna pretraga */
66     printf("Binarna pretraga: \n");
67     p = bsearch(&x, a, n, sizeof(int), &poredi_int);
68     if (p == NULL)
69         printf("Elementa nema u nizu!\n");
70     else
71         printf("Element je nadjen na poziciji %ld\n", p - a);
72
73     /* Linearna pretraga */
74     printf("Linearna pretraga (lfind): \n");
75     p = lfind(&x, a, &n, sizeof(int), &poredi_int);
76     if (p == NULL)
77         printf("Elementa nema u nizu!\n");
78     else
79         printf("Element je nadjen na poziciji %ld\n", p - a);
80
81     return 0;
82 }

```

## Rešenje 3.32

### 3 Algoritmi pretrage i sortiranja

---

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <search.h>
5
6 #define MAX 100
7
8 /* Funkcija racuna broj delilaca broja x */
9 int broj_deliaca(int x)
10 {
11     int i;
12     int br;
13
14     /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15     if (x < 0)
16         x = -x;
17     if (x == 0)
18         return 0;
19     if (x == 1)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     for (i = 2; i < sqrt(x); i++)
24         if (x % i == 0)
25             /* Ako i deli x onda su delioci: i, x/i */
26             br += 2;
27     /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
28     promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29     jos jednog delioca */
30     if (i * i == x)
31         br++;
32
33     return br;
34 }
35
36 /* Funkcija poredjenja dva cela broja po broju delilaca */
37 int poredi_po_broju_deliaca(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = broj_deliaca(ak);
42     int n_d_b = broj_deliaca(bk);
43
44     return n_d_a - n_d_b;
45 }
46
47 int main()
48 {
49     size_t n;
50     int i;
51     int a[MAX];
```

```

53  /* Unos dimenzije */
54  printf("Uneti dimenziju niza: ");
55  scanf("%d", &n);
56  if (n > MAX)
57      n = MAX;
58
59  /* Unos elementa niza */
60  printf("Uneti elemente niza:\n");
61  for (i = 0; i < n; i++)
62      scanf("%d", &a[i]);
63
64  /* Sortiranje niza celih brojeva prema broju delilaca */
65  qsort(a, n, sizeof(int), &poredi_po_broju_delilaca);
66
67  /* Prikaz sortiranog niza */
68  printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
69  for (i = 0; i < n; i++)
70      printf("%d ", a[i]);
71  putchar('\n');
72
73  return 0;
}

```

### Rešenje 3.33

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <search.h>
5
6 #define MAX_NISKI 1000
7 #define MAX_DUZINA 31
8
9 ****
10 Niz nizova karaktera ovog potpisa
11 char niske[3][4];
12 se moze graficki predstaviti ovako:
13 -----
14 | a | b | c | \0 || d | e | \0|   || f | g | h | \0 ||
15 -----
16 Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
17 svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
18 nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
19 adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
20 i ona je tipa char*.
21
22 Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23 koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
24 reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25 slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp

```

### 3 Algoritmi pretrage i sortiranja

---

```
    nad njima.  
27 ****  
28 int poredi_leksikografski(const void *a, const void *b)  
29 {  
30     return strcmp((char *) a, (char *) b);  
31 }  
32  
33 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje  
   leksikografski, vec po duzini */  
34 int poredi_duzine(const void *a, const void *b)  
35 {  
36     return strlen((char *) a) - strlen((char *) b);  
37 }  
38  
39 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, "Neupesno 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     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort  
   prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2  
   niske po duzini */  
62     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);  
63  
64     printf("Leksikografski sortirane niske:\n");  
65     for (i = 0; i < n; i++)  
66         printf("%s ", niske[i]);  
67     printf("\n");  
68  
69     /* Unos trazene niske */  
70     printf("Uneti trazenu nisku: ");  
71     scanf("%s", x);  
72  
73     /* Binarna pretraga */  
74     /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
```

```

    je niz vec sortiran leksikografski. */
79 p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
              &poredi_leksikografski);

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

87 /* Sortiranje po duzini */
88 qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

89 printf("Niske sortirane po duzini:\n");
90 for (i = 0; i < n; i++)
91     printf("%s ", niske[i]);
92     printf("\n");

93 /* Linearna pretraga */
94 p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
            &poredi_leksikografski);

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

100 exit(EXIT_SUCCESS);
101 }
```

### Rešenje 3.34

```

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

5
6 #define MAX_NISKI 1000
7 #define MAX_DUZINA 31

8 *****
9 Niz pokazivaca na karaktere ovog potpisa
10 char *niske[3];
11 posle alokacije u main-u se moze graficki predstaviti ovako:
12 -----
13 | X | -----> | a | b | c | \0|
14 -----
15 | Y | -----> | d | e | \0|
16 -----
17 | Z | -----> | f | g | h | \0|
```

### 3 Algoritmi pretrage i sortiranja

---

```
19      -----  
20      -----  
21      Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti  
22      njegov element pokazivac koji pokazuje na alocirane karaktere i-te  
23      reci. Njegov tip je char*.  
24  
25      Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata  
26      koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i  
27      kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako  
28      moraju i kastovati. Da bi se leksikografski uporedili elementi X i  
29      Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se  
30      u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,  
31      kastovani na odgovarajuci tip.  
32  ****  
33  int poredi_leksikografski(const void *a, const void *b)  
34  {  
35      return strcmp(*(char **) a, *(char **) b);  
36  }  
37  /* Funkcija sлична 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  /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na  
44      element u nizu sa kojim se poredi, pa njega treba kastovati na  
45      char** i dereferencirati, (videti obrazlozenje za prvu funkciju u  
46      ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U  
47      main funkciji je to x, koji je tipa char*, tako da pokazivac a  
48      ovde samo treba kastovati i ne dereferencirati. */  
49  int poredi_leksikografski_b(const void *a, const void *b)  
50  {  
51      return strcmp((char *) a, *(char **) b);  
52  }  
53  
54  int main()  
55  {  
56      int i;  
57      size_t n;  
58      FILE *fp = NULL;  
59      char *niske[MAX_NISKI];  
60      char **p = NULL;  
61      char x[MAX_DUZINA];  
62  
63      /* Otvaranje datoteke */  
64      if ((fp = fopen("niske.txt", "r")) == NULL) {  
65          fprintf(stderr, "Neunesno otvaranje datoteke niske.txt.\n");  
66          exit(EXIT_FAILURE);  
67      }  
68  
69      /* Citanje sadrzaja datoteke */
```

```

71     i = 0;
72     while (fscanf(fp, "%s", x) != EOF) {
73         /* Alociranje dovoljne memorije za i-tu nisku */
74         if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
75             {
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     }
83
84     /* Zatvaranje datoteke */
85     fclose(fp);
86     n = i;
87
88     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
89      prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
90      niske po duzini */
91     qsort(niske, n, sizeof(char *), &poredi_leksikografski);
92
93     printf("Leksikografski sortirane niske:\n");
94     for (i = 0; i < n; i++)
95         printf("%s ", niske[i]);
96     printf("\n");
97
98     /* Unos trazene niske */
99     printf("Uneti trazenu nisku: ");
100    scanf("%s", x);
101
102    /* Binarna pretraga */
103    p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
104    if (p != NULL)
105        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
106               *p, p - niske);
107    else
108        printf("Niska nije pronadjena u nizu\n");
109
110    /* Linearna pretraga */
111    p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
112    if (p != NULL)
113        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
114               *p, p - niske);
115    else
116        printf("Niska nije pronadjena u nizu\n");
117
118    /* Sortiramo po duzini */
119    qsort(niske, n, sizeof(char *), &poredi_duzine);
120
121    printf("Niske sortirane po duzini:\n");
122    for (i = 0; i < n; i++)

```

### 3 Algoritmi pretrage i sortiranja

---

```
123     printf("%s ", niske[i]);
124     printf("\n");
125
126     /* Oslobođanje zauzete memorije */
127     for (i = 0; i < n; i++)
128         free(niske[i]);
129
130     exit(EXIT_SUCCESS);
131 }
```

#### Rešenje 3.35

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <search.h>
5
6 #define MAX 500
7
8 /* Struktura sa svim informacijama o pojedinacnom studentu */
9 typedef struct {
10     char ime[21];
11     char prezime[21];
12     int bodovi;
13 } Student;
14
15 /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16 istim brojem bodova se dodatno sortiraju leksikografski po
17 prezimenu */
18 int poređi1(const void *a, const void *b)
19 {
20     Student *prvi = (Student *) a;
21     Student *drugi = (Student *) b;
22
23     if (prvi->bodovi > drugi->bodovi)
24         return -1;
25     else if (prvi->bodovi < drugi->bodovi)
26         return 1;
27     else
28         /* Ako su jednaki po broju bodova, treba ih uporediti po
29             prezimenu */
30         return strcmp(prvi->prezime, drugi->prezime);
31 }
32
33 /* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
34    Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
35    parametar je element niza ciji se bodovi porede. */
36 int poređi2(const void *a, const void *b)
37 {
38     int bodovi = *(int *) a;
39     Student *s = (Student *) b;
```

```

40     return s->bodovi - bodovi;
}
42
43 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
45    parametar je element niza cije se prezime poredi. */
46 int poredi3(const void *a, const void *b)
{
47     char *prezime = (char *) a;
48     Student *s = (Student *) b;
49     return strcmp(prezime, s->prezime);
}
50
51
52 int main(int argc, char *argv[])
{
53     Student kolokvijum[MAX];
54     int i;
55     size_t br_studenata = 0;
56     Student *nadjen = NULL;
57     FILE *fp = NULL;
58     int bodovi;
59     char prezime[21];
60
61     /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
62        informacija korisniku kako se program koristi i prekida se
63        izvrsavanje. */
64     if (argc < 2) {
65         fprintf(stderr,
66                 "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
67                 argv[0]);
68         exit(EXIT_FAILURE);
69     }
70
71     /* Otvaranje datoteke */
72     if ((fp = fopen(argv[1], "r")) == NULL) {
73         fprintf(stderr, "Neuspelo otvaranje datoteke %s\n", argv[1]);
74         exit(EXIT_FAILURE);
75     }
76
77     /* Ucitavanje sadrzaja */
78     for (i = 0;
79          fscanf(fp, "%s%s%d", kolokvijum[i].ime,
80                  kolokvijum[i].prezime,
81                  &kolokvijum[i].bodovi) != EOF; i++);
82
83     /* Zatvaranje datoteke */
84     fclose(fp);
85     br_studenata = i;
86
87     /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
88        studenata sa istim brojem bodova sortiranje vrsti po prezimenu */
89     qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);

```

### 3 Algoritmi pretrage i sortiranja

---

```
92     printf("Studenti sortirani po broju poena opadajuce, ");
94     printf("pa po prezimenu rastuce:\n");
95     for (i = 0; i < br_studenata; i++)
96         printf("%s %s %d\n", kolokvijum[i].ime,
97                kolokvijum[i].prezime, kolokvijum[i].bodovi);
98
99     /* Pretrazivanje studenata po broju bodova se vrsti binarnom
100      pretragom jer je niz sortiran po broju bodova. */
101     printf("Unesite broj bodova: ");
102     scanf("%d", &bodovi);
103
104     nadjen =
105         bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
106                 &poredi2);
107
108     if (nadjen != NULL)
109         printf
110             ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
111              nadjen->ime, nadjen->prezime, nadjen->bodovi);
112     else
113         printf("Nema studenta sa unetim brojem bodova\n");
114
115     /* Pretraga po prezimenu se mora vrstiti linearno jer je niz
116      sortiran po bodovima. */
117     printf("Unesite prezime: ");
118     scanf("%s", prezime);
119
120     nadjen =
121         lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122               &poredi3);
123
124     if (nadjen != NULL)
125         printf
126             ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
127              nadjen->ime, nadjen->prezime, nadjen->bodovi);
128     else
129         printf("Nema studenta sa unetim prezimenom\n");
130
131     exit(EXIT_SUCCESS);
132 }
```

#### Rešenje 3.36

```
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 #define MAX 128
7
8 /* Funkcija poredi dva karaktera */
```

```

8 int uporedi_char(const void *pa, const void *pb)
9 {
10    return *(char *) pa - *(char *) pb;
11 }
12
13 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
15 {
16    /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
17    if (strlen(s) != strlen(t))
18        return 0;
19
20    /* Sortiranje niski */
21    qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22    qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);
23
24    /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
25       suprotnom, nisu */
26    return !strcmp(s, t);
27 }
28
29 int main()
30 {
31    char s[MAX], t[MAX];
32
33    /* Unos niski */
34    printf("Unesite prvu nisku: ");
35    scanf("%s", s);
36    printf("Unesite drugu nisku: ");
37    scanf("%s", t);
38
39    /* Ispituje se da li su niske anagrami */
40    if (anagrami(s, t))
41        printf("jesu\n");
42    else
43        printf("nisu\n");
44
45    return 0;
46 }
```

**Rešenje 3.37**

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 10
6 #define MAX_DUZINA 32
7
8 /* Funkcija porenenja */
9 int uporedi_niske(const void *pa, const void *pb)
```

### 3 Algoritmi pretrage i sortiranja

---

```
11 {  
12     return strcmp((char *) pa, (char *) pb);  
13 }  
14  
15 int main()  
16 {  
17     int i, n;  
18     char S[MAX][MAX_DUZINA];  
19  
20     /* Unos broja niski */  
21     printf("Unesite broj niski:");  
22     scanf("%d", &n);  
23  
24     /* Unos niza niski */  
25     printf("Unesite niske:\n");  
26     for (i = 0; i < n; i++)  
27         scanf("%s", S[i]);  
28  
29     /* Sortiranje niza niski */  
30     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);  
31  
32     //*****  
33     // Ovaj deo je komentarisan jer se u zadatku ne trazi ispis  
34     // sortiranih niski. Koriscen je samo u fazi testiranja programa.  
35  
36     printf("Sortirane niske su:\n");  
37     for(i = 0; i < n; i++)  
38         printf("%s ", S[i]);  
39     //*****  
40  
41     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja  
42     // niza biti jedna do druge */  
43     for (i = 0; i < n - 1; i++)  
44         if (strcmp(S[i], S[i + 1]) == 0) {  
45             printf("ima\n");  
46             return 0;  
47         }  
48     printf("nema\n");  
49  
50     return 0;  
51 }
```

#### Rešenje 3.38

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 #include <string.h>  
4  
5 #define MAX 21  
6  
7 /* Struktura koja predstavlja jednog studenta */
```

```

9  typedef struct student {
10    char nalog[8];
11    char ime[MAX];
12    char prezime[MAX];
13    int poeni;
14 } Student;
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 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i na
25 kraju prema indeksu */
26 int uporedi_nalog(const void *a, const void *b)
27 {
28     Student s = *(Student *) a;
29     Student t = *(Student *) b;
30     /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
31     broj indeksa */
32     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
33     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
34     char smer1 = s.nalog[1];
35     char smer2 = t.nalog[1];
36     int indeks1 =
37         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
38         s.nalog[6] - '0';
39     int indeks2 =
40         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
41         t.nalog[6] - '0';
42     if (godina1 != godina2)
43         return godina1 - godina2;
44     else if (smer1 != smer2)
45         return smer1 - smer2;
46     else
47         return indeks1 - indeks2;
48 }
49
50 /* Funkcija poredjenja po nalogu za upotrebu u biblioteckoj funkciji
51   bsearch */
52 int uporedi_bsearch(const void *a, const void *b)
53 {
54     /* Nalog studenta koji se trazi */
55     char *nalog = (char *) a;
56     /* Kljuc pretrage */
57     Student s = *(Student *) b;
58
59     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
60     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';

```

### 3 Algoritmi pretrage i sortiranja

---

```
61     char smer1 = nalog[1];
62     char smer2 = s.nalog[1];
63     int indeks1 =
64         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
65     ;
66     int indeks2 =
67         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
68         s.nalog[6] - '0';
69     if (godina1 != godina2)
70         return godina1 - godina2;
71     else if (smer1 != smer2)
72         return smer1 - smer2;
73     else
74         return indeks1 - indeks2;
75 }
76
77 int main(int argc, char **argv)
78 {
79     Student *nadjen = NULL;
80     char nalog_trazeni[8];
81     Student niz_studenata[100];
82     int i = 0, br_studenata = 0;
83     FILE *in = NULL, *out = NULL;
84
85     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
86      korisnik nije ispravno pozvao program i prijavljuje se greska.
87      */
88     if (argc != 2 && argc != 3) {
89         fprintf(stderr,
90                 "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
91         );
92         exit(EXIT_FAILURE);
93     }
94
95     /* Otvaranje datoteke za citanje */
96     in = fopen("studenti.txt", "r");
97     if (in == NULL) {
98         fprintf(stderr,
99                 "Greska prilikom otvaranja datoteke studenti.txt!\n");
100        exit(EXIT_FAILURE);
101    }
102
103    /* Otvaranje datoteke za pisanje */
104    out = fopen("izlaz.txt", "w");
105    if (out == NULL) {
106        fprintf(stderr,
107                 "Greska prilikom otvaranja datoteke izlaz.txt!\n");
108        exit(EXIT_FAILURE);
109    }
110
111    /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
112    while (fscanf
```

```

109         (in, "%s %s %s %d", niz_studenata[i].nalog,
110          niz_studenata[i].ime, niz_studenata[i].prezime,
111          &niz_studenata[i].poeni) != EOF)
112     i++;
113
114     br_studenata = i;
115
116     /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
117     if (strcmp(argv[1], "-p") == 0)
118         qsort(niz_studenata, br_studenata, sizeof(Student),
119               &uporedi_poeni);
120     /* Ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
121     else if (strcmp(argv[1], "-n") == 0)
122         qsort(niz_studenata, br_studenata, sizeof(Student),
123               &uporedi_nalog);
124
125     /* Sortirani studenti se ispisuju u izlaznu datoteku */
126     for (i = 0; i < br_studenata; i++)
127         fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
128                 niz_studenata[i].ime, niz_studenata[i].prezime,
129                 niz_studenata[i].poeni);
130
131     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
132        studenta... */
133     if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
134         strcpy(nalog_trazen, argv[2]);
135
136         /* ... pronalazi se student sa tim nalogom... */
137         nadjen =
138             (Student *) bsearch(nalog_trazen, niz_studenata,
139                                 br_studenata, sizeof(Student),
140                                 &uporedi_bsearch);
141
142         if (nadjen == NULL)
143             printf("Nije nadjen!\n");
144         else
145             printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
146                   nadjen->prezime, nadjen->poeni);
147     }
148
149     /* Zatvaranje datoteka */
150     fclose(in);
151     fclose(out);
152
153     exit(EXIT_SUCCESS);
154 }
```



# 4

## Dinamičke strukture podataka

### 4.1 Liste

**Zadatak 4.1** Napisati biblioteku za rad sa jednostrukom povezanim listom čiji čvorovi sadrže cele brojeve.

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

## 4 Dinamičke strukture podataka

---

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

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

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

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

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unosite brojeve: (za kraj CTRL+D)  
2  
Lista: [2]  
3  
Lista: [3, 2]  
14  
Lista: [14, 3, 2]  
5  
Lista: [5, 14, 3, 2]  
3  
Lista: [3, 5, 14, 3, 2]  
17  
Lista: [17, 3, 5, 14, 3, 2]  
  
Unesite broj koji se trazi: 5  
Trazeni broj 5 je u listi!
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unosite brojeve: (za kraj CTRL+D)  
23  
Lista: [23]  
14  
Lista: [14, 23]  
35  
Lista: [35, 14, 23]  
  
Unesite broj koji se trazi: 8  
Broj 8 se ne nalazi u listi!
```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unosite brojeve: (za kraj CTRL+D)  
2  
Lista: [2]  
3  
Lista: [2, 3]  
14  
Lista: [2, 3, 14]  
3  
Lista: [2, 3, 14, 3]  
3  
Lista: [2, 3, 14, 3, 3]  
17  
Lista: [2, 3, 14, 3, 3, 17]  
3  
Lista: [2, 3, 14, 3, 3, 17, 3]  
  
Unesite broj koji se brise: 3  
Lista nakon brisanja: [2, 14, 17]
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unosite brojeve: (za kraj CTRL+D)  
23  
Lista: [23]  
14  
Lista: [23, 14]  
35  
Lista: [23, 14, 35]  
  
Unesite broj koji se brise: 3  
Lista nakon brisanja: [23, 14, 35]
```

- (3) U programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite brojeve: (za kraj CTRL+D)  
2  
Lista: [2]  
3  
Lista: [2, 3]  
14  
Lista: [2, 3, 14]  
3  
Lista: [2, 3, 3]  
3  
Lista: [2, 3, 3, 3]  
5  
Lista: [2, 3, 3, 3, 5]  
  
Unesite broj koji se trazi: 14  
Trazeni broj 14 je u listi!  
  
Unesite broj koji se brise: 3  
Lista nakon brisanja: [2, 5, 14]
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite brojeve: (za kraj CTRL+D)  
23  
Lista: [23]  
14  
Lista: [14, 23]  
35  
Lista: [14, 23, 35]  
  
Unesite broj koji se trazi: 8  
Broj 8 se ne nalazi u listi!  
  
Unesite broj koji se brise: 3  
Lista nakon brisanja: [14, 23]
```

[Rešenje 4.1]

**Zadatak 4.2** Napisati biblioteku za rad sa jednostrukim povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1.*

[Rešenje 4.2]

**Zadatak 4.3** Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

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

*Test 1*

```
POKRETANJE: ./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*

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA DATOTEKA.HTML NE POSTOJI.
IZLAZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke datoteka.html.
```

[Rešenje 4.3]

**Zadatak 4.4** U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku *da* ili *ne*, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Prepostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

## 4 Dinamičke strukture podataka

---

Primer 1

```
POKRETANJE: ./a.out studenti.txt
STUDENTI.TXT
123/2014 Marko Lukic
3/2014 Ana Sokic
43/2013 Jelena Ilic
41/2009 Marija Zaric
13/2010 Milovan Lazic

INTERAKCIJA SA PROGRAMOM:
3/2014 da: Ana Sokic
235/2008 ne
41/2009 da: Marija Zaric
```

Primer 2

```
POKRETANJE: ./a.out studenti.txt
DATOTEKA STUDENTI.TXT JE PRAZNA

INTERAKCIJA SA PROGRAMOM:
3/2014 ne
235/2008 ne
41/2009 ne
```

[Rešenje 4.4]

\* **Zadatak 4.5** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

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

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

Test 1

```
BROJEVI.TXT
43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
12 15 16
```

Test 2

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke brojevi.txt.
```

Test 3

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
Rezultat.txt ce biti prazna.
```

\* **Zadatak 4.6** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspođeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. **NA-POMENA:** Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.

### Test 1

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out dat1.txt dat2.txt
DAT1.TXT
2 4 6 10 15

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ ZA GREŠKU:
Greska prilikom otvaranja datoteke
dat2.txt.
```

### Test 3

```
POKRETANJE: ./a.out dat1.txt dat2.txt
DATOTEKA DAT1.TXT JE PRAZNA

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
[5, 6, 11, 12, 14, 16]
```

### Test 4

```
POKRETANJE: ./a.out dat1.txt
IZLAZ ZA GREŠKU:
Program se poziva sa:
./a.out dat1.txt dat2.txt!
```

[Rešenje 4.6]

**\* Zadatak 4.7** Date su dve jednostruko povezane liste  $L_1$  i  $L_2$ . Napisati funkciju koja od ovih listi formira novu listu  $L$  koja sadrži naizmenično raspoređene čvorove listi  $L_1$  i  $L_2$ : prvi čvor iz  $L_1$ , prvi čvor iz  $L_2$ , drugi čvor  $L_1$ , drugi čvor  $L_2$ , itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: Iskoristiti testove 2 - 6 za zadatak 4.6.

### Test 1

```
POKRETANJE: ./a.out dat1.txt dat2.txt
DAT1.TXT
2 4 6 10 15

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
2 5 4 6 6 11 10 12 15 14 16
```

**Zadatak 4.8** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {}, [, ]. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem

## 4 Dinamičke strukture podataka

---

steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

```
IZRAZ.TXT  
{[23 + 5344] * (24 - 234)} - 23
```

IZLAZ:

Zagrade su ispravno uparene.

Test 2

```
IZRAZ.TXT  
{[23 + 5] * (9 * 2)} - {23}
```

IZLAZ:

Zagrade su ispravno uparene.

Test 3

```
IZRAZ.TXT  
{[2 + 54] / (24 * 87)} + (234 + 23)
```

IZLAZ:

Zagrade nisu ispravno uparene.

Test 4

```
IZRAZ.TXT  
{(2 - 14) / (23 + 11)} * (2 + 13)
```

IZLAZ:

Zagrade nisu ispravno uparene.

Test 5

```
DATOTEKA IZRAZ.TXT JE PRAZNA
```

IZLAZ:

Zagrade su ispravno uparene.

Test 6

```
DATOTEKA IZRAZ.TXT NE POSTOJI.
```

IZLAZ ZA GREŠKU:  
Greska prilikom otvaranja  
datoteke izraz.txt!

[Rešenje 4.8]

**Zadatak 4.9** Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.

Test 1

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML  
<html>  
<head>  
<title>Primer</title>  
</head>  
<body>  
</body>
```

IZLAZ:

Etikete nisu pravilno uparene  
(etiketa <html> nije zatvorena)

Test 2

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML  
<head>  
<title>Primer</title>  
</head>  
<body>  
</body>  
</html>
```

IZLAZ:

Etikete nisu pravilno uparene  
(nadjena je etiketa </html>  
koja nije otvorena)

*Test 3*

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
<h1>Naslov</h1>
Danas je lep i suncan dan. <br>
Sutra ce biti jos lepsi.
<a href='http://www.math.rs'>Link</a>
</body>
</html>
```

**IZLAZ:**  
Etikete su pravilno uparene!

*Test 4*

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA.HTML
<html>
<head>
<title>Primer</title>
</head>
<body>
</html>
```

**IZLAZ:**  
Etikete nisu pravilno uparene  
(nadjena je etiketa </html>, a poslednja  
otvorena je <body>)

*Test 5*

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA DATOTEKA.HTML NE POSTOJI.
```

**IZLAZ ZA GREŠKU:**  
Greska prilikom otvaranja  
datoteke datoteka.html.

*Test 6*

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA.HTML JE PRAZNA
```

**IZLAZ:**  
Etikete su pravilno uparene!

[Rešenje 4.9]

**Zadatak 4.10** Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve korisničke *JMBG* brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (*EOF*). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Program mu postavlja pitanje *Da li korisnika vracate na kraj reda?* i ukoliko on da odgovor *Da*, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije *Da*, službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke *izvestaj.txt*. Ova datoteka, za svaki obrađen zahtev, sadrži *JMBG* i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. **UPUTSTVO:** Za čuvanje korisničkih zahteva koristiti red implementiranjem listi.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Sluzbenik evidentira korisnicke zahteve:  
Novi zahtev [CTRL+D za kraj]  
JMBG: 1234567890123  
Opis problema: Otvaranje racuna  
  
Novi zahtev [CTRL+D za kraj]  
JMBG: 2345678901234  
Opis problema: Podizanje novca  
  
Novi zahtev [CTRL+D za kraj]  
JMBG: 3456789012345  
Opis problema: Reklamacija  
  
Novi zahtev [CTRL+D za kraj]  
JMBG:  
  
Sledeci je korisnik sa JMBG: 1234567890123  
i zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Da  
  
Sledeci je korisnik sa JMBG: 2345678901234  
i zahtevom: Podizanje novca  
Da li ga vracate na kraj reda? [Da/Ne] Ne  
  
Sledeci je korisnik sa JMBG: 3456789012345  
i zahtevom: Reklamacija  
Da li ga vracate na kraj reda? [Da/Ne] Da  
  
Sledeci je korisnik sa JMBG: 1234567890123  
i zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Da  
  
Sledeci je korisnik sa JMBG: 3456789012345  
i zahtevom: Reklamacija  
Da li ga vracate na kraj reda? [Da/Ne] Ne  
  
Da li je kraj smene? [Da/Ne] Ne  
  
Sledeci je korisnik sa JMBG: 1234567890123  
i zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Ne  
  
IZVESTAJ.TXT  
JMBG: 2345678901234 Zahtev: Podizanje novca  
JMBG: 3456789012345 Zahtev: Reklamacija  
JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

[Rešenje 4.10]

**Zadatak 4.11** Napisati biblioteku za rad sa dvostrukom povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- (a) Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor **`

`adresa_kraja, Cvor * tekuci)` koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave` i poslednji čvor liste na adresi `adresa_kraja`.

- (b) Napisati funkciju `void ispisi_listu_unazad(Cvor * kraj)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Zbog efikasnog izvršavanja operacija dodavanja na kraj liste i ispisivanja liste unazad treba, pored pokazivača na glavu liste, čuvati i pokazivač na poslednji čvor liste. NAPOMENA: *Funkcije testirati koristeći test primere iz zadatka 4.1*

[Rešenje 4.11]

**\* Zadatak 4.12** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ . Plesači najpre formiraju krug tako da brojevi sa njihovih kostima rastu u smeru kazaljke na satu. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na svoj kuk i tako redom. Plesač sa brojem  $n$  svoju levu ruku spušta na rame plesača sa brojem 1, a desnu na svoj kuk i tako zatvara krug. Svoju plesnu tačku izvode tako što iz formiranog kruga najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug tako što  $k - 1$ -vi stavlja ruku na rame  $k + 1$ -og i zatvara krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

Test 1	Test 2	Test 3
ULAZ: 5 3  IZLAZ: 3 1 5 2 4	ULAZ: 8 4  IZLAZ: 4 8 5 2 1 3 7 6	ULAZ: 3 8  IZLAZ ZA GREŠKU: n mora biti uvek vece od k, a 3 < 8!

**\* Zadatak 4.13** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ . Plesači najpre formiraju krug tako da brojevi sa njihovih kostima rastu u smeru kazaljke na satu. Svaki plesač levu ruku stavlja na rame plesača sa sledećim većim brojem, a desnu na rame plesača sa prvim manjim brojem. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na rame plesača sa brojem  $n$ . Plesač sa brojem  $n$  svoju desnu ruku spušta na rame plesača sa brojem  $n - 1$ ,

a levu na rame plesača sa brojem 1 i tako zatvara krug. Plesači izvode svoju plesnu tačku tako što iz formiranog kruga najpre izlazi  $k$ -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi  $k$ -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, 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 povezану кружну листу.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 5 3  IZLAZ: 3 5 4 2 1	ULAZ: 8 4  IZLAZ: 4 8 5 7 6 3 2 1	ULAZ: 5 8  IZLAZ ZA GREŠKU: n mora biti uvek vece od k, a 5 < 8!

## 4.2 Stabla

**Zadatak 4.14** Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu **Cvor** kojom se opisuje čvor stabla, a koja sadrži ceo broj **broj** i pokazivače **levo** i **desno** redom na levo i desno podstablo.
- Napisati funkciju **Cvor \*napravi\_cvor(int broj)** koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem **broj**.
- Napisati funkciju **int dodaj\_u\_stablo(Cvor \*\*adresa\_korena, int broj)** koja u stablu na koje pokazuje argument **adresa\_korena** dodaje ceo broj **broj**. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.
- Napisati funkciju **Cvor \*pretrazi\_stablo(Cvor \* koren, int broj)** koja proverava da li se ceo broj **broj** nalazi u stablu sa korenom **koren**. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.
- Napisati funkciju **Cvor \*pronadji\_najmanji(Cvor \* koren)** koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom **koren**.

- (f) Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronađe čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.
- (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 osloboodi_stablo(Cvor ** adresa_korena)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `adresa_korena`.

Korišćenjem kreirane biblioteke, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

**Primer 1**

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće stablo: 1 2 9 18 32
```

**Primer 2**

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuće stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

**Zadatak 4.15** Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova.

## 4 Dinamičke strukture podataka

---

Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku Nedostaje ime ulazne datoteke!. Može se prepostaviti da dužina reči neće biti veća od 50 karaktera.

Test 1

```
POKRETANJE: ./a.out test.txt
TEST.TXT
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUka

IZLAZ:
jabuka: 2
programiranje: 2
racunar: 1
suncce: 3
utorak: 1

Najcesca rec: sunce (pojavljuje se 3 puta)
```

Test 2

```
POKRETANJE: ./a.out suma.txt
SUMA.TXT
lipa zova hrast ZOVA breza LIPA

IZLAZ:
breza: 1
hrast: 1
lipa: 2
zova: 2

Najcesca rec: lipa
(pojavljuje se 2 puta)
```

Test 3

```
POKRETANJE: ./a.out ulaz.txt
DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

Test 4

```
POKRETANJE: ./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 prepostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Primer 1*

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858
```

## INTERAKCIJA SA PROGRAMOM:

```
Unesite ime datoteke: imenik.txt
Unesite ime i prezime: Pera Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ
```

*Primer 2*

```
DATOTEKA IMENIK1.TXT NE POSTOJI
```

## INTERAKCIJA SA PROGRAMOM:

```
Unesite ime datoteke: imenik1.txt
Greska: Neuspesno otvaranje datoteke
imenik1.txt.
```

[Rešenje 4.16]

**Zadatak 4.17** U datoteci prijemni.txt nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niska od 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 dve grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

*Test 1*

```
PRIJEMNI.TXT
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6
```

## IZLAZ:

```
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9
```

*Test 2*

```
DATOTEKA PRIJEMNI.TXT NE POSTOJI
```

## IZLAZ:

```
Greska: Neuspesno otvaranje datoteke
prijemni.txt.
```

[Rešenje 4.17]

\* **Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata **Ime Prezime DD.MM.** i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu **DD.MM.**. Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

### Primer 1

```
POKRETANJE: ./a.out rodjendani.txt
```

```
RODJENDANI.TXT
```

```
Marko Markovic 12.12.  
Milan Jevremovic 04.06.  
Maja Agic 23.04.  
Nadica Zec 01.01.  
Jovana Milic 05.05.
```

```
INTERAKCIJA SA PROGRAMOM:
```

```
Unesite datum: 23.04.  
Slavljenik: Maja Agic  
Unesite datum: 01.01.  
Slavljenik: Nadica Zec  
Unesite datum: 01.05.  
Slavljenik: Jovana Milic 05.05.  
Unesite datum: 20.12.  
Slavljenik: Nadica Zec 01.01.  
Unesite datum:
```

### Primer 2

```
POKRETANJE: ./a.out rodjendani.txt
```

```
DATOTEKA RODJENDANI.TXT NE POSTOJI
```

```
INTERAKCIJA SA PROGRAMOM:  
Greska: Neuspesno otvaranje datoteke  
rodjendani.txt.
```

[Rešenje 4.18]

**Zadatak 4.19** Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. 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: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo:
10 5 15 3 2 4 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla jesu identicna.
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo:
10 5 15 4 3 2 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla nisu identicna.
```

[Rešenje 4.19]

**\* Zadatak 4.20** Napisati program za rad sa skupovima u kojem se skupovi predstavljaju pomoću binarnih pretraživačkih stabala. Program za dva skupa čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku skupova. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Prvi skup: 1 7 8 9 2 2
Drugi skup: 3 9 6 11 1
Unija: 1 1 2 2 3 6 7 8 9 9 11
Presek: 1 9
Razlika: 2 2 7 8
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Prvi skup: 11 2 7 5
Drugi skup: 4 3 3 7
Unija: 2 3 3 4 5 7 7 11
Presek: 7
Razlika: 2 5 11
```

[Rešenje 4.20]

**Zadatak 4.21** Napisati funkciju void sortiraj(int a[], int n) koja sortira niz celih brojeva a dimenzije n korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj n manji od 50 i niz a celih brojeva dužine n, poziva funkciju sortiraj i rezultat ispisuje na standardni izlaz. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
n: 7
a: 1 11 8 6 37 25 30
1 6 8 11 25 30 37
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
n: 55
Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

**Zadatak 4.22** Dato je binarno pretraživačko stablo celih brojeva.

## 4 Dinamičke strukture podataka

---

- (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. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka [4.14](#).

Test 1

```
POKRETANJE: ./a.out 2 15
ULAZ:
10 5 15 3 2 4 30 12 14 13
IZLAZ:
Broj cvorova: 10
Broj listova: 4
Pozitivni listovi: 2 4 13 30
Zbir cvorova: 108
Najveci element: 30
Dubina stabla: 5
Broj cvorova na 2. nivou: 3
Elementi na 2. nivou: 3 12 30
Maksimalni element na 2. nivou: 30
Zbir elemenata na 2. nivou: 45
Zbir elemenata manjih ili jednakih od 15: 78
```

Test 2

```
POKRETANJE: ./a.out 3 31
ULAZ:
24 53 61 9 7 55 20 16
IZLAZ:
Broj cvorova: 8
Broj listova: 3
Pozitivni listovi: 7 16 55
Zbir cvorova: 245
Najveci element: 61
Dubina stabla: 4
Broj cvorova na 3. nivou: 2
Elementi na 3. nivou: 16 55
Maksimalni element na 3. nivou: 55
Zbir elemenata na 3. nivou: 71
Zbir elemenata manjih ili jednakih od 31: 76
```

[Rešenje 4.22]

**Zadatak 4.23** Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Test 1	Test 2	Test 3
<pre> ULAZ: 10 5 15 3 2 4 30 12 14 13 IZLAZ: 0.nivo: 10 1.nivo: 5 15 2.nivo: 3 12 30 3.nivo: 2 4 14 4.nivo: 13 </pre>	<pre> ULAZ: 6 11 8 3 -2 IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8 </pre>	<pre> ULAZ: 24 53 61 9 7 55 20 16 IZLAZ: 0.nivo: 24 1.nivo: 9 53 2.nivo: 7 20 61 3.nivo: 16 55 </pre>

[Rešenje 4.23]

\* **Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1	Primer 2
<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 14 30 1 0 Stabla su slična kao u ogledalu. </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 20 15 0 Stabla nisu slična kao u ogledalu. </pre>

**Zadatak 4.25** AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor * koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Test 1

ULAZ:  
10 5 15 2 11 16 1 13

IZLAZ:  
7

Test 2

ULAZ:  
16 30 40 24 10 18 45 22

IZLAZ:  
6

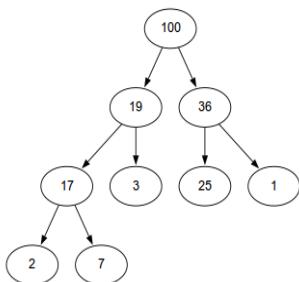
[Rešenje 4.25]

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *heap* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor * koren)` koja proverava da li je dato binarno stablo celih brojeva heap. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `heap` i ispisuje rezultat na standardni izlaz. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

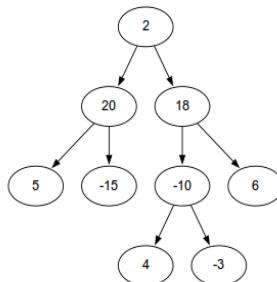
Test 1

IZLAZ:  
Zadato stablo je heap!

[Rešenje 4.26]



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronađe čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

#### Test 1

```
IZLAZ:  
Vrednost u cvoru sa maksimalnim desnim zbirom: 18  
Vrednost u cvoru sa minimalnim levim zbirom: 18  
2 18 -10 4  
2 20 -15
```

## 4.3 Rešenja

### Rešenje 4.1

*lista.h*

```
#ifndef _LISTA_H_
#define _LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste */
typedef struct cvor {
    int vrednost;
    struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void osloboodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
```

## 4 Dinamičke strukture podataka

---

```
    greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   NULL ukoliko je lista prazna. */
26 Cvor *pronadji_poslednji(Cvor * glava);

28 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
30 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov cvor sa vrednoscu broj. */
34 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

36 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
38 int dodaj_iza(Cvor * tekuci, int broj);

40 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   inace vraca 0. */
42 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

44 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadran trazeni broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
46 Cvor *pretrazi_listu(Cvor * glava, int broj);

50 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
   neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
   sadran trazeni broj ili NULL u slucaju da takav cvor ne postoji.
   */
52 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

56 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrije stara glava. */
58 void obrisi_cvor(Cvor ** adresa_glave, int broj);

60 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajući se na cinjenicu da je prosledjena lista sortirana
   neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrije stara glava liste. */
62 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

66 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
68 void ispisi_listu(Cvor * glava);

70 #endif
```

*lista.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"

5 Cvor *napravi_cvor(int broj)
{
7     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8      * alokacije */
9     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10    if (novi == NULL)
11        return NULL;

13    /* Inicijalizacija polja strukture */
14    novi->vrednost = broj;
15    novi->sledeci = NULL;

17    /* Vraca se adresa novog cvora */
18    return novi;
19}

21 void osloboodi_listu(Cvor ** adresa_glage)
{
23    Cvor *pomocni = NULL;

25    /* Ako lista nije prazna, onda treba oslobooditi memoriju */
26    while (*adresa_glage != NULL) {
27        /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
28         * oslobooditi cvor koji predstavlja glavu liste */
29        pomocni = (*adresa_glage)->sledeci;
30        free(*adresa_glage);

31        /* Sledeci cvor je nova glava liste */
32        *adresa_glage = pomocni;
33    }
35}

37 int dodaj_na_pocetak_liste(Cvor ** adresa_glage, int broj)
{
38    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39    Cvor *novi = napravi_cvor(broj);
40    if (novi == NULL)
41        return 1;

43    /* Novi cvor se uvezuje na pocetak i postaje nova glava liste */
44    novi->sledeci = *adresa_glage;
45    *adresa_glage = novi;

47    /* Vraca se indikator uspesnog dodavanja */
48    return 0;
49}

```

## 4 Dinamičke strukture podataka

---

```
51    }
51 Cvor *pronadji_poslednji(Cvor * glava)
53 {
54     /* U praznoj listi nema cvorova pa se vraca NULL */
55     if (glava == NULL)
56         return NULL;
57
58     /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
59      glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
60      ce pokazivati na cvor liste koji nema sledbenika, tj. na
61      poslednji cvor liste pa se vraca vrednost pokazivaca glava.
62
63     Pokazivac glava je argument funkcije i njegove promene nece se
64     odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
65     while (glava->sledeci != NULL)
66         glava = glava->sledeci;
67
68     return glava;
69 }
70
71 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
72 {
73     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
74     Cvor *novi = napravi_cvor(broj);
75     if (novi == NULL)
76         return 1;
77
78     /* Ako je lista prazna */
79     if (*adresa_glave == NULL) {
80         /* Glava nove liste je upravo novi cvor */
81         *adresa_glave = novi;
82     } else {
83         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
84          se dodaje na kraj liste kao sledbenik poslednjeg */
85         Cvor *poslednji = pronadji_poslednji(*adresa_glave);
86         poslednji->sledeci = novi;
87     }
88
89     /* Vraca se indikator uspesnog dodavanja */
90     return 0;
91 }
92
93 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
94 {
95     /* U praznoj listi nema takvog mesta i vraca se NULL */
96     if (glava == NULL)
97         return NULL;
98
99     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
100       pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
101       vecu ili jednaku vrednosti novog cvora.
```

```

103     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
105     provera da li se doslo do poslednjeg cvora liste pre nego sto se
106     proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
107     sledeci ne postoji, pa ni njegova vrednost. */
108     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
109         glava = glava->sledeci;
110
111     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
112      poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
113      vrednost vecu od broj. */
114     return glava;
115 }
116
117 int dodaj_iza(Cvor * tekuci, int broj)
118 {
119     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
120     Cvor *novi = napravi_cvor(broj);
121     if (novi == NULL)
122         return 1;
123
124     /* Novi cvor se dodaje iza tekuceg cvora. */
125     novi->sledeci = tekuci->sledeci;
126     tekuci->sledeci = novi;
127
128     /* Vraca se indikator uspesnog dodavanja */
129     return 0;
130 }
131
132 int dodaj_sortirano(Cvor ** adresa_glove, int broj)
133 {
134     /* Ako je lista prazna */
135     if (*adresa_glove == NULL) {
136         /* Glava nove liste je novi cvor */
137
138         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
139         Cvor *novi = napravi_cvor(broj);
140         if (novi == NULL)
141             return 1;
142
143         *adresa_glove = novi;
144
145         /* Vraca se indikator uspesnog dodavanja */
146         return 0;
147     }
148
149     /* Inace... */
150
151     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
152      treba dodati na pocetak liste */
153     if ((*adresa_glove)->vrednost >= broj) {
154         return dodaj_na_pocetak_liste(adresa_glove, broj);

```

## 4 Dinamičke strukture podataka

---

```
    }

155 /* U slučaju da je glava liste cvor sa vrednoscu manjom od broj,
   tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
157 Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
159 return dodaj_iza(pomocni, broj);
}

161 Cvor *pretrazi_listu(Cvor * glava, int broj)
{
    /* Obilaze se cvorovi liste */
    for (; glava != NULL; glava = glava->sledeci)
        /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
           se obustavlja */
        if (glava->vrednost == broj)
            return glava;

    /* Nema traženog broja u listi i vraca se NULL */
    return NULL;
}

175 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
    /* Obilaze se cvorovi liste */
    /* U uslovu ostanka u petlji, bitan je redosled provera u
       konjunkciji */
    while (glava != NULL && glava->vrednost < broj)
        glava = glava->sledeci;

    /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
       glava->vrednost veca od traženog broja i tada treba vratiti
       NULL, jer tražen broj nije nadjen medju manjim brojevima pri
       pocetku sortirane liste, pa se moze zaključiti da ga nema u
       listi. Drugi nacini, tako sto se doslo do kraja liste i glava je
       NULL ili tako sto je glava->vrednost == broj. U ova poslednja
       nacina treba vratiti pokazivac glava bilo da je NULL ili
       pokazivac na konkretan cvor. */
    if (glava->vrednost > broj)
        return NULL;
    else
        return glava;
}

197 void obrisi_cvor(Cvor ** adresu_glave, int broj)
{
    Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;

201 /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
       broju i azurira se pokazivac na glavu liste */
203 while (*adresu_glave != NULL && (*adresu_glave)->vrednost == broj)
    {
```

```

205     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
206      adresi adresa_glave */
207     pomocni = (*adresa_glave)->sledeci;
208     free(*adresa_glave);
209     *adresa_glave = pomocni;
210 }
211
212 /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
213  funkcije */
214 if (*adresa_glave == NULL)
215     return;
216
217 /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
218  pokazuje na cvor cija je vrednost razlicita od trazenog broja.
219  Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
220  sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
221  ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
222  Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
223 */
224 tekuci = *adresa_glave;
225 while (tekuci->sledeci != NULL)
226     if (tekuci->sledeci->vrednost == broj) {
227         /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
228            prvo cuva u promenljivoj pomocni. */
229         pomocni = tekuci->sledeci;
230         /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
231            njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
232            sledeci od cvora koji se brise. */
233         tekuci->sledeci = pomocni->sledeci;
234         /* Sada treba osloboziti cvor sa vrednoscu broj. */
235         free(pomocni);
236     } else {
237         /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
238            pomera na sledeci. */
239         tekuci = tekuci->sledeci;
240     }
241 }
242
243 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
244 {
245     Cvor *tekuci = NULL;
246     Cvor *pomocni = NULL;
247
248     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
249       broju i azurira se pokazivac na glavu liste. */
250     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
251     {
252         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
253           adresi adresa_glave. */
254         pomocni = (*adresa_glave)->sledeci;
255         free(*adresa_glave);

```

## 4 Dinamičke strukture podataka

---

```
255     *adresa_glave = pomocni;
256 }
257
258 /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
259    prekida. Isto se radi i ukoliko glava liste sadrzi vrednost koja
260    je veca od broja, jer kako je lista sortirana rastuce nema
261    potrebe broj traziti u repu liste. */
262 if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
263     return;
264
265 /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
266    na cvor cija vrednost je manja od trazenog broja, kao i svim
267    cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
268    je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
269    ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
270    cija vrednost je veca od trazenog broja. */
271 tekuci = *adresa_glave;
272 while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
273     )
274     if (tekuci->sledeci->vrednost == broj) {
275         pomocni = tekuci->sledeci;
276         tekuci->sledeci = tekuci->sledeci->sledeci;
277         free(pomocni);
278     } else {
279         /* Ne treba brisati sledeceg od tekuceg jer je manji od
280            trazenog i tekuci se pomera na sledeci cvor. */
281         tekuci = tekuci->sledeci;
282     }
283     return;
284 }
285 void ispisi_listu(Cvor * glava)
286 {
287     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
288        jer se lista nece menjati */
289     putchar('[');
290     /* Unutar zagrade ispisuju se vrednosti u cvorovima liste od
291        pocetka prema kraju liste. */
292     for (; glava != NULL; glava = glava->sledeci) {
293         printf("%d", glava->vrednost);
294         if (glava->sledeci != NULL)
295             printf(", ");
296     }
297     printf("]\n");
298 }
```

*main\_a.c*

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

```

4  /* 1) Glavni program */
6  int main()
{
8    /* Lista je prazna na pocetku */
10   Cvor *glava = NULL;
11   Cvor *trazeni = NULL;
12   int broj;
13
14   /* Testiranje dodavanja novog broja na pocetak liste */
15   printf("Unesite brojeve: (za kraj CTRL+D)\n");
16   while (scanf("%d", &broj) > 0) {
17     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
18     memorije za nov cvor. Memoriju alocirano za cvorove liste
19     treba osloboditi pre napustanja programa. */
20     if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
21       fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
22       oslobodi_listu(&glava);
23       exit(EXIT_FAILURE);
24     }
25     printf("\tLista: ");
26     ispisi_listu(glava);
27   }
28
29   /* Testiranje funkcije za pretragu liste */
30   printf("\nUnesite broj koji se trazi: ");
31   scanf("%d", &broj);
32
33   trazeni = pretrazi_listu(glava, broj);
34   if (trazeni == NULL)
35     printf("Broj %d se ne nalazi u listi!\n", broj);
36   else
37     printf("Trazenii broj %d je u listi!\n", trazeni->vrednost);
38
39   /* Oslobadja se memorija zauzeta listom */
40   oslobodi_listu(&glava);
41
42   exit(EXIT_SUCCESS);
}

```

*main\_b.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* 2) Glavni program */
6 int main()
{
7   /* Lista je prazna na pocetku */
8   Cvor *glava = NULL;

```

## 4 Dinamičke strukture podataka

---

```
10 int broj;
12 /* Testira se funkcija za dodavanja novog broja na kraj liste */
13 printf("Unesite brojeve: (za kraj CTRL+D)\n");
14 while (scanf("%d", &broj) > 0) {
15     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16     memorije za nov cvor. Memoriju alociranu za cvorove liste
17     treba oslobođiti pre napustanja programa. */
18     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
19         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
20         oslobodi_listu(&glava);
21         exit(EXIT_FAILURE);
22     }
23     printf("\tLista: ");
24     ispisi_listu(glava);
25 }
26
27 /* Testira se funkcije kojom se brise cvor liste */
28 printf("\nUnesite broj koji se brise: ");
29 scanf("%d", &broj);
30
31 /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
32   procitanom sa ulaza */
33 obrisi_cvor(&glava, broj);
34
35 printf("Lista nakon brisanja: ");
36 ispisi_listu(glava);
37
38 /* Oslobadja se memorija zauzeta listom */
39 oslobodi_listu(&glava);
40
41 exit(EXIT_SUCCESS);
42 }
```

main\_c.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* 3) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na početku */
9     Cvor *glava = NULL;
10    Cvor *trazenii = NULL;
11    int broj;
12
13    /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
14       uređena neopadajuće */
15    printf("Unosite brojeve (za kraj CTRL+D)\n");
```

```

16 while (scanf("%d", &broj) > 0) {
17     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
18     memorije za nov cvor. Memoriju alociranu za cvorove liste
19     treba osloboditi pre napustanja programa. */
20     if (dodaj_sortirano(&glava, broj) == 1) {
21         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
22         oslobodi_listu(&glava);
23         exit(EXIT_FAILURE);
24     }
25     printf("\tLista: ");
26     ispisi_listu(glava);
27 }
28
29 /* Testira se funkcija za pretragu liste */
30 printf("\nUnesite broj koji se trazi: ");
31 scanf("%d", &broj);
32
33 trazeni = pretrazi_listu(glava, broj);
34 if (trazeni == NULL)
35     printf("Broj %d se ne nalazi u listi!\n", broj);
36 else
37     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
38
39 /* Testira se funkcija kojom se brise cvor liste */
40 printf("\nUnesite broj koji se brise: ");
41 scanf("%d", &broj);
42
43 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
44 procitanom sa ulaza */
45 obrisi_cvor_sortirane_liste(&glava, broj);
46
47 printf("Lista nakon brisanja: ");
48 ispisi_listu(glava);
49
50 /* Oslobadja se memorija zauzeta listom */
51 oslobodi_listu(&glava);
52
53 exit(EXIT_SUCCESS);
54 }

```

## Rešenje 4.2

*lista.h*

```

1 #ifndef _LISTA_H_
2 #define _LISTA_H_
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5  podatak vrednost i pokazivac na sledeci cvor liste. */
6 typedef struct cvor {

```

## 4 Dinamičke strukture podataka

---

```
8     int vrednost;
9     struct cvor *sledeci;
10    } Cvor;
11
12   /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
13      dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14      na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
15   Cvor *napravi_cvor(int broj);
16
17   /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18      ciji se pokazivac glava nalazi na adresi adresa_glave. */
19   void osloboodi_listu(Cvor ** adresa_glave);
20
21   /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
22      bilo greske pri alokaciji memorije, inace vraca 0. */
23   int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25   /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
26      pri alokaciji memorije, inace vraca 0. */
27   int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
28
29   /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
30      ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
31      memorije, inace vraca 0. */
32   int dodaj_sortirano(Cvor ** adresa_glave, int broj);
33
34   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
35      Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
36      NULL u slucaju da takav cvor ne postoji u listi. */
37   Cvor *pretrazi_listu(Cvor * glava, int broj);
38
39   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
40      U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
41      neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
42      sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
43      */
44   Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
45
46   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
47      pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
48      obrije stara glava liste. */
49   void obrisi_cvor(Cvor ** adresa_glave, int broj);
50
51   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
52      oslanjajuci se na cinjenicu da je prosledjena lista sortirana
53      neopadajuće. Azurira pokazivac na glavu liste, koji moze biti
54      promenjen ukoliko se obrije stara glava liste. */
55   void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
56
57   /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
58      zapetama. */
59   void ispisi_vrednosti(Cvor * glava);
```

```

58 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
60   liste, razdvojene zapetama i uokvirene zagradama. */
61 void ispisi_listu(Cvor * glava);
62 #endif

```

*lista.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"

5 Cvor *napravi_cvor(int broj)
{
7   /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8     alokacije */
9   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10  if (novi == NULL)
11    return NULL;

13  /* Inicijalizacija polja strukture */
14  novi->vrednost = broj;
15  novi->sledeci = NULL;

17  /* Vraca se adresa novog cvora */
18  return novi;
19 }

21 void osloboodi_listu(Cvor ** adres_a_glave)
{
23  /* Ako je lista vec prazna */
24  if (*adres_a_glave == NULL)
25    return;

27  /* Ako lista nije prazna, treba oslobooditi memoriju. Pre
28    oslobadjanja memorije za glavu liste, treba oslobooditi rep liste
29  */
30  osloboodi_listu(&(*adres_a_glave)->sledeci);
31  /* Nakon osloboodenog repa, oslobadja se glava liste i azurira se
32    glava u pozivajuoj funkciji tako da odgovara praznoj listi */
33  free(*adres_a_glave);
34  *adres_a_glave = NULL;
35 }

37 int dodaj_na_pocetak_liste(Cvor ** adres_a_glave, int broj)
{
38  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39  Cvor *novi = napravi_cvor(broj);
40  if (novi == NULL)
41    return 1;

```

## 4 Dinamičke strukture podataka

---

```
43  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
44  novi->sledeci = *adresa_glave;
45  *adresa_glave = novi;
46
47  /* Vraca se indikator uspesnog dodavanja */
48  return 0;
49 }
50
51 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
52 {
53  /* Ako je lista prazna */
54  if (*adresa_glave == NULL) {
55
56      /* Novi cvor postaje glava liste */
57      Cvor *novi = napravi_cvor(broj);
58      /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
59      */
60      if (novi == NULL)
61          return 1;
62
63      /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
64      azurira i pokazivacka promenljiva u pozivajucoj funkciji */
65      *adresa_glave = novi;
66
67      /* Vraca se indikator uspesnog dodavanja */
68      return 0;
69 }
70
71 /* Ako lista nije prazna, broj se dodaje u rep liste. */
72 /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
73 novi broj se dodaje u rep liste u rekurzivnom pozivu.
74 Informaciju o uspesnosti alokacije u rekurzivnom pozivu funkcija
75 prosledjuje visem rekurzivnom pozivu koji tu informaciju vraca u
76 rekurzivni poziv iznad, sve dok se ne vrati u main. Tek je iz
77 main funkcije moguce pristupiti pravom pocetku liste i
78 osloboediti je celu, ako ima potrebe. Ako je funkcija vratila 0,
79 onda nije bilo greske. */
80     return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
81 }
82
83 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
84 {
85  /* Ako je lista prazna */
86  if (*adresa_glave == NULL) {
87
88      /* Novi cvor postaje glava liste */
89      Cvor *novi = napravi_cvor(broj);
90
91      /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
92      */
93      if (novi == NULL)
```

```

93     return 1;

95     /* Azurira se glava liste */
96     *adresa_glave = novi;

97     /* Vraca se indikator uspesnog dodavanja */
98     return 0;
99 }

101    /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
102       glavi liste, onda se dodaje na pocetak liste */
103    if ((*adresa_glave)->vrednost >= broj)
104        return dodaj_na_pocetak_liste(adresa_glave, broj);

105    /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
106       bude sortirana lista. */
107    return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
108 }

111 Cvor *pretrazi_listu(Cvor * glava, int broj)
112 {
113     /* U praznoj listi nema vrednosti */
114     if (glava == NULL)
115         return NULL;

116     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
117     if (glava->vrednost == broj)
118         return glava;

119     /* Inace, pretraga se nastavlja u repu liste */
120     return pretrazi_listu(glava->sledeci, broj);
121 }

125 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
126 {
127     /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
128       vrednosti u glavi liste, jer je lista neopadajuće sortirana */
129     if (glava == NULL || glava->vrednost > broj)
130         return NULL;

131     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
132     if (glava->vrednost == broj)
133         return glava;

134     /* Inace, pretraga se nastavlja u repu liste */
135     return pretrazi_listu(glava->sledeci, broj);
136 }

141 void obrisi_cvor(Cvor ** adresa_glave, int broj)
142 {
143     /* U praznoj listi nema cvorova za brisanje. */
144     if (*adresa_glave == NULL)

```

## 4 Dinamičke strukture podataka

---

```
145     return;

147     /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
148     obrisi_cvor(&(*adresa_glave)->sledeci, broj);

149     /* Preostaje provera da li glavu liste treba obrisati */
150     if ((*adresa_glave)->vrednost == broj) {
151         /* Pomocni pokazuje na cvor koji treba da se obrije */
152         Cvor *pomocni = *adresa_glave;
153         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
154          brise se cvor koji je bio glava liste. */
155         *adresa_glave = (*adresa_glave)->sledeci;
156         free(pomocni);
157     }
158 }
159 }

161 void obrisi_cvor_sortirane_liste(Cvor ** adresga_lave, int broj)
162 {
163     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
164      od broja, kako je lista sortirana rastuce nema potrebe broj
165      traziti u repu liste i zato se funkcija prekida */
166     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
167         return;
168
169     /* Brisu se cvorovi iz repa koji imaju vrednost broj */
170     obrisi_cvor(&(*adresa_glave)->sledeci, broj);

171     /* Preostaje provera da li glavu liste treba obrisati */
172     if ((*adresa_glave)->vrednost == broj) {
173         /* Pomocni pokazuje na cvor koji treba da se obrije */
174         Cvor *pomocni = *adresa_glave;
175         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
176          brise se cvor koji je bio glava liste */
177         *adresa_glave = (*adresa_glave)->sledeci;
178         free(pomocni);
179     }
180 }
181 }

183 void ispisi_vrednosti(Cvor * glava)
184 {
185     /* Prazna lista */
186     if (glava == NULL)
187         return;
188
189     /* Ispisuje se vrednost u glavi liste */
190     printf("%d", glava->vrednost);
191
192     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
193      se poziva ista funkcija za ispis ostalih. */
194     if (glava->sledeci != NULL) {
195         printf(", ");



256
```

```

197     ispisi_vrednosti(glava->sledeci);
198 }
199
200 void ispisi_listu(Cvor * glava)
201 {
202     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
203      jer nece menjati listu, pa nema ni potrebe da azurira pokazivac
204      na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
205      zgrade, a rekurzivno ispisivanje vrednosti u listi prepusta
206      rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
207      elemente razdvojene zapetom i razmakom. */
208     putchar('[');
209     ispisi_vrednosti(glava);
210     printf("]\n");
211 }

```

### Rešenje 4.3

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX_DUZINA 20
6
7 /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
8   broj pojavljanja etikete i pokazivac na sledeci cvor liste */
9 typedef struct _Cvor {
10     char etiketa[20];
11     unsigned broj_pojavljanja;
12     struct _Cvor *sledeci;
13 } Cvor;
14
15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
16   ili NULL ako alokacija nije uspesno izvrsena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
18 {
19     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
20       alokacije */
21     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
22     if (novi == NULL)
23         return NULL;
24
25     /* Inicijalizacija polja strukture */
26     novi->broj_pojavljanja = br;
27     strcpy(novi->etiketa, etiketa);
28     novi->sledeci = NULL;
29
30     /* Vraca se adresa novog cvora */
31     return novi;
32 }

```

## 4 Dinamičke strukture podataka

---

```
33  /* Funkcija oslobođa dinamicku memoriju zauzetu cvorovima liste */
34  void osloboди_listu(Cvor ** adresa_glove)
35  {
36      Cvor *pomocni = NULL;
37
38      /* Sve dok lista ni bude prazna, briše se tekuća glava liste i
39       azurira se vrednost glave liste */
40      while (*adresa_glove != NULL) {
41          pomocni = (*adresa_glove)->sledeći;
42          free(*adresa_glove);
43          *adresa_glove = pomocni;
44      }
45      /* Nakon izlaska iz petlje pokazivac glava u main funkciji koji se
46       nalazi na adresi adresa_glove bice postavljen na NULL vrednost.
47       */
48  }
49
50  /* Funkcija dodaje novi cvor na početak liste. Povratna vrednost je 1
51   ako je doslo do greske pri alokaciji memorije za nov cvor, odnosno
52   0 */
53  int dodaj_na_pocetak_liste(Cvor ** adresa_glove, unsigned br,
54                               char *etiketa)
55  {
56      /* Kreira se novi cvor i proverava se uspesnost alokacije */
57      Cvor *novi = napravi_cvor(br, etiketa);
58      if (novi == NULL)
59          return 1;
60
61      /* Dodaje se novi cvor na početak liste */
62      novi->sledeći = *adresa_glove;
63      *adresa_glove = novi;
64
65      /* Vraca se indikator uspesnog dodavanja */
66      return 0;
67  }
68
69  /* Funkcija vraca cvor koji kao vrednost sadrzi traženu etiketu ili
70   NULL ako takav cvor ne postoji u listi */
71  Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
72  {
73      Cvor *tekuci;
74
75      /* Obilazi se cvor po cvor liste */
76      for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeći)
77          /* Ako tekuci cvor sadrzi traženu etiketu, vracamo njegovu
78           vrednost */
79          if (strcmp(tekuci->etiketa, etiketa) == 0)
80              return tekuci;
81
82      /* Cvor nije pronadjen */
83      return NULL;
84  }
```

```

}
85 /* Funkcija ispisuje sadrzaj liste */
87 void ispisi_listu(Cvor * glava)
{
89 /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
    broj njihovog pojavljivanja u HTML datoteci. */
91 for (; glava != NULL; glava = glava->sledeci)
    printf("%s - %u\n", glava->etiketa, glava->broj_pojavljenja);
93 }

/* Glavni program */
95 int main(int argc, char **argv)
{
97 /* Provera se da li je program pozvan sa ispravnim brojem
   argumenata. */
99 if (argc != 2) {
101     fprintf(stderr,
102             "Greska! Program se poziva sa: ./a.out datoteka.html!\n")
103     ;
104     exit(EXIT_FAILURE);
105 }

/* Priprema datoteke za citanje */
107 FILE *in = NULL;
108 in = fopen(argv[1], "r");
109 if (in == NULL) {
110     fprintf(stderr,
111             "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
112     exit(EXIT_FAILURE);
113 }

115 char c;
116 int i = 0;
117 char procitana[MAX_DUZINA];
118 Cvor *glava = NULL;
119 Cvor *trazeni = NULL;

121 /* Cita se karakter po karakter datoteke sve dok se ne procita cela
   datoteka */
122 while ((c = fgetc(in)) != EOF) {

125 /* Proverava se da li se pocinje sa citanjem nove etikete */
126 if (c == '<') {
127     /* Proverava se da li se cita zatvarajuca etiketa */
128     if (((c = fgetc(in)) == '/') {
129         i = 0;
130         while ((c = fgetc(in)) != '>')
131             procitana[i++] = c;
132     }
133     /* Cita se zatvarajuca etiketa */
134     else {

```

## 4 Dinamičke strukture podataka

---

```
135     i = 0;
136     procitana[i++] = c;
137     while ((c = fgetc(in)) != ' ' && c != '>')
138         procitana[i++] = c;
139     }
140     procitana[i] = '\0';
141
142     /* Trazi se procitana etiketa medju postojecim cvorovima liste.
143      Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
144      sa
145      brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
146      etikete */
147     trazeni = pretrazi_listu(glava, procitana);
148     if (trazeni == NULL) {
149         if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
150             fprintf(stderr, "Neuspela alokacija za nov cvor\n");
151             osloboodi_listu(&glava);
152             exit(EXIT_FAILURE);
153         }
154     } else
155         trazeni->broj_pojavljivanja++;
156     }
157
158     /* Zatvaranje datoteke */
159     fclose(in);
160
161     /* Ispisuje se sadrzaj cvorova liste */
162     ispisi_listu(glava);
163
164     /* Oslobadja se memorija zauzeta listom */
165     osloboodi_listu(&glava);
166
167     exit(EXIT_SUCCESS);
168 }
```

### Rešenje 4.4

```
#include <stdio.h>
2 #include <stdlib.h>
# include <string.h>
4
#define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21
8
/* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
   studentu */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
11    char ime[MAX_IME_PREZIME];
12    char prezime[MAX_IME_PREZIME];
```

```

14     struct _Cvor *sledeci;
15 } Cvor;
16
17 /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
18    novi cvor ili NULL ukoliko je doslo do greske */
19 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
21     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
22        alokacije */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizacija polja strukture */
28     strcpy(novi->broj_indeksa, broj_indeksa);
29     strcpy(novi->ime, ime);
30     strcpy(novi->prezime, prezime);
31     novi->sledeci = NULL;
32
33     /* Vraca se adresa novog cvora */
34     return novi;
35 }
36
37 /* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void osloboodi_listu(Cvor ** adresa_glave)
39 {
40     /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
41     if (*adresa_glave == NULL)
42         return;
43
44     /* Rekurzivnim pozivom se oslobadja rep liste */
45     osloboodi_listu(&(*adresa_glave)->sledeci);
46
47     /* Potom se oslobadja i glava liste */
48     free(*adresa_glave);
49
50     /* Proglasava se lista praznom */
51     *adresa_glave = NULL;
52 }
53
54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
55    do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
57                             char *ime, char *prezime)
58 {
59     /* Kreira se novi cvor i proverava se uspesnost alokacije */
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
61     if (novi == NULL)
62         return 1;
63
64     /* Dodaje se novi cvor na pocetak liste */
65     novi->sledeci = *adresa_glave;

```

## 4 Dinamičke strukture podataka

---

```
66     *adresa_glave = novi;
68
69     /* Vraca se indikator uspesnog dodavanja */
70     return 0;
71 }
72
73 /* Funkcija ispisuje sadrzaj cvorova liste. */
74 void ispis_i_listu(Cvor * glava)
75 {
76     /* Pocevsi od glave liste */
77     for ( ; glava != NULL; glava = glava->sledeci)
78         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
79                glava->prezime);
80 }
81
82 /* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj indeksa.
83  U suprotnom funkcija vraca NULL */
84 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
85 {
86     /* Ako je lista prazna, ne postoji trazeni cvor */
87     if (glava == NULL)
88         return NULL;
89
90     /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste
91      */
92     if (!strcmp(glava->broj_indeksa, broj_indeksa))
93         return glava;
94
95     /* Ukoliko u glavi liste nije trazeni indeks, pretraga se nastavlja
96      u repu liste */
97     return pretrazi_listu(glava->sledeci, broj_indeksa);
98 }
99
100 /* Glavni program */
101 int main(int argc, char **argv)
102 {
103     /* Argumenti komandne linije su neophodni jer se iz komandne linije
104      dobija ime datoteke sa informacijama o studentima */
105     if (argc != 2) {
106         fprintf(stderr,
107                 "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
108         exit(EXIT_FAILURE);
109     }
110
111     /* Priprema datoteke za citanje */
112     FILE *in = NULL;
113     in = fopen(argv[1], "r");
114     if (in == NULL) {
115         fprintf(stderr,
116                 "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
117         exit(EXIT_FAILURE);
118     }
119 }
```

```

118  /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
119   listu */
120  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
121  char broj_indeksa[MAX_INDEKS];
122  Cvor *glava = NULL;
123  Cvor *trazeni = NULL;
124
125  /* Ucitavanje vrednosti u listu */
126  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
127    if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
128      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
129      osloboodi_listu(&glava);
130      exit(EXIT_FAILURE);
131    }
132
133  /* Datoteka vise nije potrebna i zatvara se. */
134  fclose(in);
135
136  /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
137  while (scanf("%s", broj_indeksa) != EOF) {
138    trazeni = pretrazi_listu(glava, broj_indeksa);
139    if (trazeni == NULL)
140      printf("ne\n");
141    else
142      printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
143  }
144
145  /* Oslobadja se memorija zauzeta za cvorove liste. */
146  osloboodi_listu(&glava);
147
148  exit(EXIT_SUCCESS);
}

```

### Rešenje 4.6

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
6   na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7   pokazivaca postojećih cvorova listi */
8 Cvor *objedini(Cvor **adresa_glave_1, Cvor **adresa_glave_2)
9 {
10   /* Pokazivaci na pocetne cvorove liste koje se prevezuju */
11   Cvor *lista1 = *adresa_glave_1;
12   Cvor *lista2 = *adresa_glave_2;
13
14   /* Pokazivac na pocetni cvor rezultujuće liste */
15   Cvor *rezultujuca = NULL;

```

## 4 Dinamičke strukture podataka

---

```
17     Cvor *tekuci = NULL;
18
19     /* Ako su obe liste prazne i rezultat je prazna lista */
20     if (lista1 == NULL && lista2 == NULL)
21         return NULL;
22
23     /* Ako je prva lista prazna, rezultat je druga lista */
24     if (lista1 == NULL)
25         return lista2;
26
27     /* Ako je druga lista prazna, rezultat je prva lista */
28     if (lista2 == NULL)
29         return lista1;
30
31     /* Odredjuje se prvi cvor rezultujuće liste - to je ili prvi cvor
32      liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
33      koji sadrži manju vrednost */
34     if (lista1->vrednost < lista2->vrednost) {
35         rezultujuca = lista1;
36         lista1 = lista1->sledeci;
37     } else {
38         rezultujuca = lista2;
39         lista2 = lista2->sledeci;
40     }
41     /* Kako promenljiva rezultujuća pokazuje na početak nove liste, ne
42      sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
43      koji sadrži adresu trenutnog cvora rezultujuće liste */
44     tekuci = rezultujuca;
45
46     /* U svakoj iteraciji petlje rezultujućoj listi se dodaje novi cvor
47      tako da bude uredjena neopadajuće. Pokazivac na listu iz koje se
48      uzima cvor se azurira tako da pokazuje na sledeći cvor */
49     while (lista1 != NULL && lista2 != NULL) {
50         if (lista1->vrednost < lista2->vrednost) {
51             tekuci->sledeci = lista1;
52             lista1 = lista1->sledeci;
53         } else {
54             tekuci->sledeci = lista2;
55             lista2 = lista2->sledeci;
56         }
57         tekuci = tekuci->sledeci;
58     }
59
60     /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
61      rezultujuću listu treba nadovezati ostatak druge liste */
62     if (lista1 == NULL)
63         tekuci->sledeci = lista2;
64     else
65         /* U suprotnom treba nadovezati ostatak prve liste */
66         tekuci->sledeci = lista1;
67
68     /* Preko adresa glava polaznih listi vrednosti pokazivaca u
```

```

69     pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
71     prethodnih listi nalaze negde unutar rezultujuće liste. Do njih
    se može doći prateći pokazivace iz glave rezultujuće liste, tako
    da stare pokazivace treba postaviti na NULL. */
73     *adresa_glave_1 = NULL;
74     *adresa_glave_2 = NULL;

75     return rezultujuca;
76 }

77 int main(int argc, char **argv)
78 {
    /* Argumenti komandne linije su neophodni */
79     if (argc != 3) {
        fprintf(stderr,
    "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
        exit(EXIT_FAILURE);
    }

87     /* Otvaramo datoteke u kojima se nalaze elementi listi */
88     FILE *in1 = NULL;
89     in1 = fopen(argv[1], "r");
90     if (in1 == NULL) {
        fprintf(stderr,
    "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
    }

95     FILE *in2 = NULL;
96     in2 = fopen(argv[2], "r");
97     if (in2 == NULL) {
        fprintf(stderr,
    "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
        exit(EXIT_FAILURE);
    }

103    /* Liste su na početku prazne */
104    int broj;
105    Cvor *lista1 = NULL;
106    Cvor *lista2 = NULL;

109    /* Ucitavanje listi */
110    while (fscanf(in1, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista1, broj);

113    while (fscanf(in2, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista2, broj);

115    /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
116    fclose(in1);
117    fclose(in2);
118
119

```

## 4 Dinamičke strukture podataka

---

```
121 /* Pokazivac rezultat ce pokazivati na glavu liste dobijene  
122 objedinjavanjem listi */  
123 Cvor *rezultat = objedini(&lista1, &lista2);  
124  
125 /* Ispis rezultujuce liste. */  
126 ispis_i_listu(rezultat);  
127  
128 /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.  
129 Njenim oslobadjanjem bice oslobođena sva zauzeta memorija. */  
130 osloboodi_listu(&rezultat);  
131  
132 exit(EXIT_SUCCESS);  
133 }
```

### Rešenje 4.8

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji  
5 predstavlja zagradu koja se koristi i pokazivac na sledeci cvor  
6 liste */  
7 typedef struct cvor {  
8     char zagrada;  
9     struct cvor *sledeci;  
10 } Cvor;  
11  
12 /* Funkcija koja oslobadja memoriju zauzetu stekom */  
13 void osloboodi_stek(Cvor ** stek)  
14 {  
15     Cvor *tekuci;  
16     Cvor *pomocni;  
17  
18     /* Oslobadja se cvor po cvor steka */  
19     tekuci = *stek;  
20     while (tekuci != NULL) {  
21         pomocni = tekuci->sledeci;  
22         free(tekuci);  
23         tekuci = pomocni;  
24     }  
25  
26     /* Stek se proglašava praznim */  
27     *stek = NULL;  
28 }  
29  
30 /* Glavni program */  
31 int main()  
32 {  
33     /* Stek je na pocetku prazan */  
34     Cvor *stek = NULL;  
35     FILE *ulaz = NULL;
```

```

1   char c;
2   Cvor *pomocni = NULL;
3
4   /* Otvaranje datoteke za citanje izraza */
5   ulaz = fopen("izraz.txt", "r");
6   if (ulaz == NULL) {
7       fprintf(stderr,
8           "Greska prilikom otvaranja datoteke izraz.txt!\n");
9       exit(EXIT_FAILURE);
10  }
11
12  /* Cita se karakter po karakter iz datoteke dok se ne dodje do
13     kraja */
14  while ((c = fgetc(ulaz)) != EOF) {
15      /* Ako je ucitana otvorena zagrada, stavlja se na stek */
16      if (c == '(' || c == '{' || c == '[') {
17          /* Alocira se memorija za novi cvor liste i proverava se
18             uspesnost alokacije */
19          pomocni = (Cvor *) malloc(sizeof(Cvor));
20          if (pomocni == NULL) {
21              fprintf(stderr, "Greska prilikom alokacije memorije!\n");
22              /* Oslobadja se memorija zauzeta stekom */
23              oslobodi_stek(&stek);
24              /* I prekida se sa izvrsavanjem programa */
25              exit(EXIT_FAILURE);
26          }
27
28          /* Inicijalizacija polja strukture */
29          pomocni->zagrada = c;
30
31          /* Promena vrha steka */
32          pomocni->sledeci = stek;
33          stek = pomocni;
34      }
35
36      /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
37         prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
38         otvorena zagrada */
39      else {
40          if (c == ')' || c == '}' || c == ']') {
41              if (stek != NULL && ((stek->zagrada == '(' && c == ')')
42                  || (stek->zagrada == '{' && c == '}')
43                  || (stek->zagrada == '[' && c == ']')))
44              {
45                  /* Sa vrha steka se uklanja otvorena zagrada */
46                  pomocni = stek->sledeci;
47                  free(stek);
48                  stek = pomocni;
49              } else {
50                  /* Inace, zaključujemo da zagrade u izrazu nisu ispravno
51                     uparene */
52                  break;
53              }
54          }
55      }
56  }
57
58  /* Islobadjanje memorije */
59  oslobodi_stek(&stek);
60
61  /* Ispis rezultata */
62  printf("Rezultat izraza je: %s\n", rezultat);
63
64  /* Islobadjanje memorije */
65  oslobodi_stek(&stek);
66
67  /* Islobadjanje memorije */
68  free(pomocni);
69
70  /* Islobadjanje memorije */
71  free(stek);
72
73  /* Islobadjanje memorije */
74  free(rezultat);
75
76  /* Islobadjanje memorije */
77  free(izraz);
78
79  /* Islobadjanje memorije */
80  free(zagrade);
81
82  /* Islobadjanje memorije */
83  free(izracun);
84
85  /* Islobadjanje memorije */
86  free(izraz);

```

## 4 Dinamičke strukture podataka

---

```
87         }
88     }
89 }
90
91 /* Procitana je cela datoteka i treba je zatvoriti */
92 fclose(ulaz);
93
94 /* Ako je stek prazan i procitana je cela datoteka, zgrade su
95    ispravno uparene */
96 if (stek == NULL && c == EOF)
97     printf("Zgrade su ispravno uparene.\n");
98 else {
99     /* U suprotnom se zaključuje da zgrade nisu ispravno uparene */
100    printf("Zgrade nisu ispravno uparene.\n");
101    /* Oslobođa se memorija koja je ostala zauzeta stekom */
102    osloboди_stek(&stek);
103 }
104
105 exit(EXIT_SUCCESS);
106 }
```

### Rešenje 4.9

*stek.h*

```
1 #ifndef _STEK_H_
2 #define _STEK_H_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <ctype.h>
8
9 #define MAX 100
10
11 #define OTVORENA 1
12 #define ZATVORENA 2
13
14 #define VAN_ETIKETE 0
15 #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2
17
18 /* Struktura kojim se predstavlja cvor liste sadrži ime etikete i
19    pokazivac na sledeći cvor */
20 typedef struct cvor {
21     char etiketa[MAX];
22     struct cvor *sledeci;
23 } Cvor;
24
25 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
```

```

26     adresu ili NULL ako alokacija nije bila uspesna */
27 Cvor *napravi_cvor(char *etiketa);
28
29 /* Funkcija oslobođa memoriju zauzetu stekom */
30 void osloboodi_stek(Cvor ** adresa_vrha);
31
32 /* Funkcija postavlja na vrh steka novu etiketu. U slučaju greske pri
33  alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);
35
36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
37  pokazivac razlicit od NULL, tada u niz karaktera na koji on
38  pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
39  suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
40  samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
41  suprotnom */
42 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);
43
44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
45  steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);
47
48 /* Funkcija prikazuje stek od vrha prema dnu */
49 void prikazi_stek(Cvor * vrh);
50
51 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledeću
52  etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
53  Vraca EOF u slučaju da se dodje do kraja datoteke pre nego što se
54  procita etiketa. Vraca OTVORENA, ako je procitana otvorena
55  etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
56 int uzmi_etiketu(FILE * f, char *etiketa);
57
58 #endif

```

*stek.c*

```

1 #include "stek.h"
2
3 Cvor *napravi_cvor(char *etiketa)
4 {
5     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
6      alokacije */
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8     if (novi == NULL)
9         return NULL;
10
11    /* Inicijalizacija polja u novom cvoru */
12    if (strlen(etiketa) >= MAX) {
13        fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
14        etiketa[MAX - 1] = '\0';
15    }

```

## 4 Dinamičke strukture podataka

---

```
16    strcpy(novi->etiketa, etiketa);
17    novi->sledeci = NULL;
18
19    /* Vraca se adresa novog cvora */
20    return novi;
21}
22
23 void osloboodi_stek(Cvor ** adresса_vrha)
24 {
25    Cvor *pomocni;
26
27    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
28    while (*adresса_vrha != NULL) {
29        /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
30         oslobooditi cvor koji predstavlja vrh steka */
31        pomocni = *adresса_vrha;
32        /* Sledeci cvor je novi vrh steka */
33        *adresса_vrha = (*adresса_vrha)->sledeci;
34        free(pomocni);
35    }
36
37    /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
38     adresса_vrha ce pokazivati na NULL. */
39}
40
41 int potisni_na_stek(Cvor ** adresса_vrha, char *etiketa)
42 {
43    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
44    Cvor *novi = napravi_cvor(etiketa);
45    if (novi == NULL)
46        return 1;
47
48    /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
49    novi->sledeci = *adresса_vrha;
50    *adresса_vrha = novi;
51    return 0;
52}
53
54 int skinji_sa_steka(Cvor ** adresса_vrha, char *etiketa)
55 {
56    Cvor *pomocni;
57
58    /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
59     vraca se 0 */
60    if (*adresса_vrha == NULL)
61        return 0;
62
63    /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
64     adresu kopira etiketa sa vrha steka */
65    if (etiketa != NULL)
66        strcpy(etiketa, (*adresса_vrha)->etiketa);
```

```

68  /* Element sa vrha steka se uklanja */
69  pomocni = *adresa_vrha;
70  *adresa_vrha = (*adresa_vrha)->sledeci;
71  free(pomocni);
72
73  /* Vraca se indikator uspesno izvrsene radnje */
74  return 1;
75 }
76
77 char *vrh_steka(Cvor * vrh)
78 {
79  /* Prazan stek nema cvor koji je vrh i vraca se NULL */
80  if (vrh == NULL)
81      return NULL;
82
83  /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
84   koji je na vrhu steka. */
85  return vrh->etiketa;
86 }
87
88 void prikazi_stek(Cvor * vrh)
89 {
90  /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
91  for ( ; vrh != NULL; vrh = vrh->sledeci)
92      printf("<%s>\n", vrh->etiketa);
93 }
94
95 int uzmi_etiketu(FILE * f, char *etiketa)
96 {
97  int c;
98  int i = 0;
99
100  /* Stanje predstavlja informaciju dokle se stalo sa citanjem
101   etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
102   nije zapoceto citanje. */
103  /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
104   OTVORENA ili ZATVORENA. */
105  int stanje = VAN_ETIKETE;
106  int tip;
107
108  /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
109   to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
110   samo malim slovima. */
111  while ((c = fgetc(f)) != EOF) {
112      switch (stanje) {
113      case VAN_ETIKETE:
114          if (c == '<')
115              stanje = PROCITANO_MANJE;
116          break;
117      case PROCITANO_MANJE:
118          if (c == '/') {
119              /* Cita se zatvorena etiketa */
120              tip = ZATVORENA;
121          }
122      }
123  }
124 }
```

## 4 Dinamičke strukture podataka

---

```
120     } else {
121         if (isalpha(c)) {
122             /* Cita se otvorena etiketa */
123             tip = OTVORENA;
124             etiketa[i++] = tolower(c);
125         }
126     }
127     /* Od sada se cita etiketa i zato se menja stanje */
128     stanje = U_ETIKETI;
129     break;
130 case U_ETIKETI:
131     if (isalpha(c) && i < MAX - 1) {
132         /* Ako je procitani karakter slovo i nije prekoracena
133            dozvoljena duzina etikete, procitani karakter se smanjuje
134            i smesta u etiketu */
135         etiketa[i++] = tolower(c);
136     } else {
137         /* Inace, staje se sa citanjem etikete. Korektno se zavrsava
138            niska koja sadrzi procitanu etiketu i vraca se njen tip */
139         etiketa[i] = '\0';
140         return tip;
141     }
142     break;
143 }
144 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
145    etiketa i vraca se EOF. */
146 return EOF;
147 }
```

*main.c*

```
1 #include "stek.h"
2
3 /* Glavni program */
4 int main(int argc, char **argv)
5 {
6     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
7        nije procitana. */
8     Cvor *vrh = NULL;
9     char etiketa[MAX];
10    int tip;
11    int uparene = 1;
12    FILE *f = NULL;
13
14    /* Ime datoteke se preuzima iz komandne linije. */
15    if (argc < 2) {
16        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
17        exit(EXIT_FAILURE);
18    }
```

```

20  /* Datoteka se otvara za citanje */
21  if ((f = fopen(argv[1], "r")) == NULL) {
22      fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
23              argv[1]);
24      exit(EXIT_FAILURE);
25  }
26
27  /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
28  while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
29      /* Ako je otvorena etiketa, stavља se na stek. Izuzetak su
30      etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
31      potrebno zatvoriti. U HTML-u postoje još neke etikete koje
32      koje nemaju sadrzaj (npr link). Zbog jednostavnosti
33      prepostavlja se da njih nema u HTML dokumentu. */
34      if (tip == OTVORENA) {
35          if (strcmp(etiketa, "br") != 0
36              && strcmp(etiketa, "hr") != 0
37              && strcmp(etiketa, "meta") != 0)
38              if (potisni_na_stek(&vrh, etiketa) == 1) {
39                  fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40                  oslobodi_stek(&vrh);
41                  exit(EXIT_FAILURE);
42              }
43      }
44      /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
45      u pitanju zatvaranje etikete koja je poslednja otvorena, a još
46      uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
47      je taj uslov ispunjen, skida se sa steka, jer je upravo
48      zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
49      nisu pravilno uparene. */
50      else if (tip == ZATVORENA) {
51          if (vrh_steka(vrh) != NULL
52              && strcmp(vrh_steka(vrh), etiketa) == 0)
53              skinji_sa_steka(&vrh, NULL);
54          else {
55              printf("Etikete nisu pravilno uparene\n");
56              printf("(nadjena je etiketa </%s>, etiketa);
57              if (vrh_steka(vrh) != NULL)
58                  printf(", a poslednja otvorena je <%s>\n", vrh_steka(vrh))
59              ;
60              else
61                  printf(" koja nije otvorena)\n");
62              uparene = 0;
63              break;
64          }
65      }
66      /* Zavrseno je citanje i datoteka se zatvara */
67      fclose(f);
68
69      /* Ako do sada nije pronadljeno pogresno uparivanje, stek bi trebal
70      da bude prazan. Ukoliko nije, tada postoje etikete koje su

```

## 4 Dinamičke strukture podataka

---

```
    ostale otvorene */
72 if (uparene) {
    if (vrh_steka(vrh) == NULL)
        printf("Etikete su pravilno uparene!\n");
74 else {
    printf("Etikete nisu pravilno uparene\n");
    printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
78 /* Oslobadja se memorija zauzeta stekom */
    oslobodi_stek(&vrh);
80 }
82 exit(EXIT_SUCCESS);
84 }
```

### Rešenje 4.10

*red.h*

```
1 #ifndef _RED_H_
#define _RED_H_
3
# include <stdio.h>
5 # include <stdlib.h>
7
#define MAX 1000
#define JMBG_DUZINA 14
9
/* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11 opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
/* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
17     korisnika i pokazivac na sledeci cvor liste. */
18 typedef struct cvor {
19     Zahtev nalog;
20     struct cvor *sledeci;
21 } Cvor;
22
/* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
23     poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
24     greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
25     treba smestiti u novi cvor zbog smestanja manjeg podatka na
26     sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
27     bajtovima(B) u odnosu na strukturu Zahtev. */
28 Cvor *napravi_cvor(Zahtev * zahtev);
29
30
```

```

1  /* Funkcija prazni red oslobadjajući memoriju koji je red zauzimao */
33 void osloboodi_red(Cvor ** pocetak, Cvor ** kraj);

35 /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
   greske pri alokaciji memorije za novi cvor, inace vraca 0. */
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

39 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
41 pokazivac razlicit od NULL, tada se u strukturu na koju on
43 pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
45 suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
47 suprotnom. */
49 int skinji_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                      Zahtev * zahtev);

51 /* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
53 na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
55 Zahtev *pocetak_reda(Cvor * pocetak);

57 /* Funkcija prikazuje sadrzaj reda. */
59 void prikazi_red(Cvor * pocetak);

55 #endif

```

*red.c*

```

1 #include "red.h"

3 Cvor *napravi_cvor(Zahtev * zahtev)
{
5   /* Alocira se memorija za novi cvor liste i proverava uspesnost
     alokacije */
7   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9   if (novi == NULL)
     return NULL;

11  /* Inicijalizacija polja strukture */
12  novi->nalog = *zahtev;
13  novi->sledeci = NULL;

15  /* Vraca se adresa novog cvora */
16  return novi;
17}

19 void osloboodi_red(Cvor ** pocetak, Cvor ** kraj)
{
21   Cvor *pomocni = NULL;

23   /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
24   while (*pocetak != NULL) {

```

## 4 Dinamičke strukture podataka

---

```
25     /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
26      osloboditi cvor sa pocetka reda */
27     pomocni = *pocetak;
28     *pocetak = (*pocetak)->sledeci;
29     free(pomocni);
30 }
31 /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
32  treba postaviti na NULL. */
33 *kraj = NULL;
34 }
35
36 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
37                   Zahtev * zahtev)
38 {
39     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40     Cvor *novi = napravi_cvor(zahtev);
41     if (novi == NULL)
42         return 1;
43
44     /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
45      kraj, to je podjednako efikasno kao dodavanje na pocetak liste
46      */
47     if (*adresa_kraja != NULL) {
48         (*adresa_kraja)->sledeci = novi;
49         *adresa_kraja = novi;
50     } else {
51         /* Ako je red bio ranije prazan */
52         *adresa_pocetka = novi;
53         *adresa_kraja = novi;
54     }
55
56     /* Vraca se indikator uspesnog dodavanja */
57     return 0;
58 }
59
60 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
61                     Zahtev * zahtev)
62 {
63     Cvor *pomocni = NULL;
64
65     /* Ako je red prazan */
66     if (*adresa_pocetka == NULL)
67         return 0;
68
69     /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
70      zahtev koji je na pocetku reda. */
71     if (zahtev != NULL)
72         *zahtev = (*adresa_pocetka)->nalog;
73
74     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
75      na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
76      u redu. */
```

```

77     pomocni = *adresa_pocetka;
78     *adresa_pocetka = (*adresa_pocetka)->sledeci;
79     free(pomocni);
80
81     /* Ukoliko red nakon oslobođanja pocetnog cvora ostane prazan,
82      potrebno je azurirati i vrednost pokazivaca na adresi
83      adresa_kraja na NULL */
84     if (*adresa_pocetka == NULL)
85         *adresa_kraja = NULL;
86
87     return 1;
88 }
89
90 Zahtev *pocetak_reda(Cvor * pocetak)
91 {
92     /* U praznom redu nema zahteva */
93     if (pocetak == NULL)
94         return NULL;
95
96     /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
97     return &(pocetak->nalog);
98 }
99
100 void prikazi_red(Cvor * pocetak)
101 {
102     /* Prikazuje se sadrzaj reda od pocetka prema kraju */
103     for ( ; pocetak != NULL; pocetak = pocetak->sledeci)
104         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
105
106     printf("\n");
107 }
```

*main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "red.h"
5
6 #define VREME_ZA_PAUZU 5
7
8 /* Glavni program */
9 int main(int argc, char **argv)
10 {
11     /* Red je prazan. */
12     Cvor *pocetak = NULL, *kraj = NULL;
13     Zahtev nov_zahtev;
14     Zahtev *sledeci = NULL;
15     char odgovor[3];
16     int broj_usluzenih = 0;
```

## 4 Dinamičke strukture podataka

---

```
18  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
   broja i opisa potrebne usluge. */
20  printf("Sluzbenik evidentira korisnicke zahteve:\n");
21  while (1) {
22
23      /* Ucitava se JMBG */
24      printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
25      if (scanf("%s", nov_zahtev.jmbg) == EOF)
26          break;
27
28      /* 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 */
29      getchar();
30
31      /* Ucitava se opis problema */
32      printf("\tOpis problema: ");
33      fgets(nov_zahtev.opis, MAX - 1, stdin);
34      /* Ako je poslednji karakter nov red, eliminise se */
35      if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
36          nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
37
38      /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
39      if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
40          fprintf(stderr, "Neuspela alokacija za nov cvor\n");
41          oslobodi_red(&pocetak, &kraj);
42          exit(EXIT_FAILURE);
43      }
44
45
46  /* Otvaranje datoteke za dopisivanje izvestaja */
47  FILE *izlaz = fopen("izvestaj.txt", "a");
48  if (izlaz == NULL) {
49      fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
50      exit(EXIT_FAILURE);
51  }
52
53
54  /* Dokle god ima korisnika u redu, treba ih usluziti */
55  while (1) {
56      sledeci = pocetak_reda(pocetak);
57      /* Ako nema nikog vise u redu, prekida se petlja */
58      if (sledeci == NULL)
59          break;
60
61
62      printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
63      printf("i zahtevom: %s\n", sledeci->opis);
64
65      skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
66
67      broj_usluzenih++;
68
69      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
70  }
```

```

70     scanf("%s", odgovor);
72     if (strcmp(odgovor, "Da") == 0)
73         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
74     else
75         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
76                 nov_zahtev.opis);
77
78     if (broj_usluzenih == VREME_ZA_PAZU) {
79         printf("\nDa li je kraj smene? [Da/Ne] ");
80         scanf("%s", odgovor);
81
82         if (strcmp(odgovor, "Da") == 0)
83             break;
84         else
85             broj_usluzenih = 0;
86     }
87
88 /***** Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:
89
90 while (skinii_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
91     printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
92             nov_zahtev.jmbg);
93     printf("sa zahtevom: %s\n", nov_zahtev.opis);
94     broj_usluzenih++;
95
96     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
97     scanf("%s", odgovor);
98     if (strcmp(odgovor, "Da") == 0)
99         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
100    else
101        fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
102                 nov_zahtev.jmbg, nov_zahtev.opis);
103
104    if (broj_usluzenih == VREME_ZA_PAZU) {
105        printf("\nDa li je kraj smene? [Da/Ne] ");
106        scanf("%s", odgovor);
107        if (strcmp(odgovor, "Da") == 0)
108            break;
109        else
110            broj_usluzenih = 0;
111    }
112
113 }
114 *****/
115 /* Datoteka vise nije potrebna i treba je zatvoriti. */
116 fclose(izlaz);
117
118 /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
119 neusluzenih korisnika, u tom slucaju treba osloboediti memoriju
120 koju zauzima red sa neobradjenim zahtevima korisnika. */

```

## 4 Dinamičke strukture podataka

---

```
122     osloboodi_red(&pocetak, &kraj);  
124  
125 }  
126
```

### Rešenje 4.11

*dvostruko\_povezana\_lista.h*

```
1 #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_  
2 #define _DVOSTRUKO_POVEZANA_LISTA_H_  
3  
4 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu  
5    vrednost i pokazivace na sledeci i prethodni cvor liste. */  
6 typedef struct cvor {  
7     int vrednost;  
8     struct cvor *sledeci;  
9     struct cvor *prethodni;  
10 } Cvor;  
11  
12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,  
13    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac  
14    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */  
15 Cvor *napravi_cvor(int broj);  
16  
17 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste  
18    ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na  
19    adresi adresa_kraja. */  
20 void oslobodi_listu(Cvor **adresa_glave, Cvor **adresa_kraja);  
21  
22 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je  
23    bilo greske pri alokaciji memorije, inace vraca 0. */  
24 int dodaj_na_pocetak_liste(Cvor **adresa_glave, Cvor **  
25                             adresa_kraja, int broj);  
26  
27 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske  
28    pri alokaciji memorije, inace vraca 0. */  
29 int dodaj_na_kraj_liste(Cvor **adresa_glave, Cvor **adresa_kraja,  
30                         int broj);  
31  
32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti  
33    novi cvor sa vrednoscu broj. */  
34 Cvor *pronadj_i_mesto_umetanja(Cvor *glava, int broj);  
35  
36 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je  
37    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */  
38 int dodaj_iza(Cvor *tekuci, int broj);  
39  
40 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane  
41    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
```

```

    inace vraca 0. */
43 int dodaj_sortirano(Cvor ** adresa_glove, Cvor ** adresa_kraja, int
                      broj);
45
46 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
47   Vraca pokazivac na cvor liste u kome je sadran trazeni broj ili
48   NULL u slucaju da takav cvor ne postoji u listi. */
49 Cvor *pretrazi_listu(Cvor * glava, int broj);

50 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
51   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
52   neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
53   sadran trazeni broj ili NULL u slucaju da takav cvor ne postoji.
54 */
55 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

56 /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
57   pokazivac glava se nalazi na adresi adresa_glove. */
58 void obrisi_tekuci(Cvor ** adresa_glove, Cvor ** adresa_kraja, Cvor *
                      tekuci);

59 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
60   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
61   obrije stara glava. */
62 void obrisi_cvor(Cvor ** adresa_glove, Cvor ** adresa_kraja, int
                      broj);

63 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
64   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
65   sortirana. Azurira pokazivac na glavu liste, koji moze biti
66   promenjen ukoliko se obrije stara glava liste. */
67 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glove, Cvor **
                                         adresa_kraja, int broj);

68 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
69   liste, razdvojene zapetama i uokvirene zagradama. */
70 void ispisi_listu(Cvor * glava);

71 /* Funkcija prikazuje cvorove liste pocevsi od kraja ka glavi liste.
72   */
73 void ispisi_listu_unazad(Cvor * kraj);

74 #endif

```

*dvostruko\_povezana\_lista.c*

```

2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "dvostruko_povezana_lista.h"
5
6 Cvor *napravi_cvor(int broj)

```

## 4 Dinamičke strukture podataka

---

```
6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;
12
13     /* Inicijalizacija polja strukture */
14     novi->vrednost = broj;
15     novi->sledeci = NULL;
16
17     /* Vraca se adresa novog cvora */
18     return novi;
19 }
20
21 void osloboodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
22 {
23     Cvor *pomocni = NULL;
24
25     /* Ako lista nije prazna, onda treba oslobooditi memoriju */
26     while (*adresa_glave != NULL) {
27         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
28             oslobooditi memoriju cvora koji predstavlja glavu liste */
29         pomocni = (*adresa_glave)->sledeci;
30         free(*adresa_glave);
31         /* Sledeci cvor je nova glava liste */
32         *adresa_glave = pomocni;
33     }
34     /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
35        treba postaviti na NULL */
36     *adresa_kraja = NULL;
37 }
38
39 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
40                               adresa_kraja, int broj)
41 {
42     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43     Cvor *novi = napravi_cvor(broj);
44     if (novi == NULL)
45         return 1;
46
47     /* Sledbenik novog cvora je glava stare liste */
48     novi->sledeci = *adresa_glave;
49
50     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
51        treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
52        krajnji */
53     if (*adresa_glave != NULL)
54         (*adresa_glave)->prethodni = novi;
55     else
56         *adresa_kraja = novi;
```

```

58  /* Novi cvor je nova glava liste */
59  *adresa_glave = novi;
60
61  /* Vraca se indikator uspesnog dodavanja */
62  return 0;
63 }
64
65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
66                         int broj)
67 {
68  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
69  Cvor *novi = napravi_cvor(broj);
70  if (novi == NULL)
71      return 1;
72
73  /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
74   ujedno i cela lista. Azurira se vrednost na koju pokazuju
75   adresa_glave i adresa_kraja */
76  if (*adresa_glave == NULL) {
77      *adresa_glave = novi;
78      *adresa_kraja = novi;
79  } else {
80      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
81       sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
82       liste */
83      (*adresa_kraja)->sledeci = novi;
84      novi->prethodni = (*adresa_kraja);
85      *adresa_kraja = novi;
86  }
87
88  /* Vraca se indikator uspesnog dodavanja */
89  return 0;
90 }

91 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
92 {
93  /* U praznoj listi nema takvog mesta i vraca se NULL */
94  if (glava == NULL)
95      return NULL;
96
97  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
98   pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
99   vrednost vecu ili jednaku od vrednosti novog cvora.
100
101  Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
102  provera da li se doslo do poslednjeg cvora liste pre nego sto se
103  proveri vrednost u sledecem cvoru jer u slucaju poslednjeg,
104  sledeci ne postoji pa ni njegova vrednost. */
105  while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
106      glava = glava->sledeci;
107
108  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do

```

## 4 Dinamičke strukture podataka

---

```
110     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima  
111     vrednost vecu od broj */  
112     return glava;  
113 }  
114  
115 int dodaj_iza(Cvor * tekuci, int broj)  
116 {  
117     /* Kreira se novi cvor i provera se uspesnost kreiranja */  
118     Cvor *novi = napravi_cvor(broj);  
119     if (novi == NULL)  
120         return 1;  
121  
122     novi->sledeci = tekuci->sledeci;  
123     novi->prethodni = tekuci;  
124  
125     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,  
126        a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca  
127        na ispravne adrese */  
128     if (tekuci->sledeci != NULL)  
129         tekuci->sledeci->prethodni = novi;  
130     tekuci->sledeci = novi;  
131  
132     /* Vraca se indikator uspesnog dodavanja */  
133     return 0;  
134 }  
135  
136 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int  
137                      broj)  
138 {  
139     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste  
140        */  
141     if (*adresa_glave == NULL) {  
142         /* Kreira se novi cvor i proverava se uspesnost kreiranja */  
143         Cvor *novi = napravi_cvor(broj);  
144         if (novi == NULL)  
145             return 1;  
146  
147         /* Azuriraju se vrednosti pocetka i kraja liste */  
148         *adresa_glave = novi;  
149         *adresa_kraja = novi;  
150  
151         /* Vraca se indikator uspesnog dodavanja */  
152         return 0;  
153     }  
154  
155     /* Ukoliko je vrednost glave liste veca ili jednaka od nove  
156        vrednosti onda novi cvor treba staviti na pocetak liste */  
157     if ((*adresa_glave)->vrednost >= broj) {  
158         return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);  
159     }  
160  
161     /* Pronazi se cvor iza koga treba uvezati novi cvor */
```

```

162     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
163     /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
164     if (dodaj_iza(pomocni, broj) == 1)
165         return 1;
166     /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
167      novi cvor poslednji u listi. */
168     if (pomocni == *adresa_kraja)
169         *adresa_kraja = pomocni->sledeci;
170
171     return 0;
172 }
173
174 Cvor *pretrazi_listu(Cvor * glava, int broj)
175 {
176     /* Obilaze se cvorovi liste */
177     for (; glava != NULL; glava = glava->sledeci)
178         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
179          se obustavlja */
180         if (glava->vrednost == broj)
181             return glava;
182
183     /* Nema trazenog broja u listi i vraca se NULL */
184     return NULL;
185 }
186
187 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
188 {
189     /* Obilaze se cvorovi liste */
190     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
191     for (; glava != NULL && glava->vrednost <= broj;
192         glava = glava->sledeci)
193         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
194            se obustavlja */
195         if (glava->vrednost == broj)
196             return glava;
197
198     /* Nema trazenog broja u listi i bice vraceno NULL */
199     return NULL;
200 }
201
202 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
203 lako realizovati jer on sadrzi pokazivace na svog sledbenika i
204 prethodnika u listi. U funkciji se bise cvor zadat argumentom
205 tekuci */
206 void obrisi_tekuci(Cvor ** adres_glave, Cvor ** adres_kraja, Cvor *
207                     tekuci)
208 {
209     /* Ako je tekuci NULL pokazivac, nema sta da se brise */
210     if (tekuci == NULL)
211         return;
212
213     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da

```

## 4 Dinamičke strukture podataka

---

```
214     njegov sledbenik bude sledbenik tekuceg cvora */
215     if (tekuci->prethodni != NULL)
216         tekuci->prethodni->sledeci = tekuci->sledeci;
217
218     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
219      treba da bude prethodnik tekuceg cvora */
220     if (tekuci->sledeci != NULL)
221         tekuci->sledeci->prethodni = tekuci->prethodni;
222
223     /* Ako je glava cvor koji se brise, nova glava liste ce biti
224      sledbenik stare glave */
225     if (tekuci == *adresa_glave)
226         *adresa_glave = tekuci->sledeci;
227
228     /* Ako je cvor koji se brise poslednji u listi, azurira se i
229      pokazivac na kraj liste */
230     if (tekuci == *adresa_kraja)
231         *adresa_kraja = tekuci->prethodni;
232
233     /* Oslobadja se dinamicki alociran prostor za cvor tekuci */
234     free(tekuci);
235 }
236
237 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
238 )
239 {
240     Cvor *tekuci = *adresa_glave;
241
242     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
243      cvorovi se brisu iz liste. */
244     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
245         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
246
247 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj)
248 {
249     Cvor *tekuci = *adresa_glave;
250
251     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
252      takvi cvorovi se brisu iz liste. */
253     while ((tekuci =
254             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
255         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
256 }
257
258 void ispisi_listu(Cvor * glava)
259 {
260     putchar('[');
261     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
262      pocetka prema kraju liste */
263     for ( ; glava != NULL; glava = glava->sledeci) {
```

```

264     printf("%d", glava->vrednost);
265     if (glava->sledeci != NULL)
266         printf(", ");
267
268     printf("]\n");
269 }
270
271 void ispisi_listu_unazad(Cvor * kraj)
272 {
273     putchar('[');
274     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
275      prema pocetku liste */
276     for (; kraj != NULL; kraj = kraj->prethodni) {
277         printf("%d", kraj->vrednost);
278         if (kraj->prethodni != NULL)
279             printf(", ");
280     }
281     printf("]\n");
282 }
```

*main\_a.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
5 /* 1) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku */
9     /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
10      da bi operacije poput dodavanja na kraj liste i ispisivanja
11      liste unazad bile efikasne poput dodavanja na pocetak liste i
12      ispisivanja liste od pocetnog do poslednjeg cvora. */
13     Cvor *glava = NULL;
14     Cvor *kraj = NULL;
15     Cvor *trazenii = NULL;
16     int broj;
17
18     /* Testira se funkcija za dodavanja novog broja na pocetak liste */
19     printf("Unesite brojeve: (za kraj CTRL+D)\n");
20     while (scanf("%d", &broj) > 0) {
21         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
22            memorije za novi cvor. Memoriju alociranu za cvorove liste
23            treba osloboditi pre napustanja programa */
24         if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
25             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
26             osloboodi_listu(&glava, &kraj);
27             exit(EXIT_FAILURE);
28     }
```

## 4 Dinamičke strukture podataka

---

```
29     printf("\tLista: ");
30     ispisi_listu(glava);
31 }
32
33 /* Testira se funkcija za pretragu liste */
34 printf("\nUnesite broj koji se trazi u listi: ");
35 scanf("%d", &broj);
36
37 /* Pokazivac trazenih dobija vrednost rezultata pretrage */
38 trazeni = pretrazi_listu(glava, broj);
39 if (trazeni == NULL)
40     printf("Broj %d se ne nalazi u listi!\n", broj);
41 else
42     printf("Trazenih broj %d je u listi!\n", trazeni->vrednost);
43
44 /* Ispisuje se lista unazad */
45 printf("\nLista ispisana u nazad: ");
46 ispisi_listu_unazad(kraj);
47
48 /* Oslobadja se memorija zauzeta za cvorove liste */
49 oslobodi_listu(&glava, &kraj);
50
51 exit(EXIT_SUCCESS);
52 }
```

*main\_b.c*

```
#include <stdio.h>
2 #include <stdlib.h>
# include "dvostruko_povezana_lista.h"
4
/* 2) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku. */
9     Cvor *glava = NULL;
10    Cvor *kraj = NULL;
11    int broj;
12
13    /* Testira se funkcija za dodavanja novog broja na kraj liste */
14    printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
15    while (scanf("%d", &broj) > 0) {
16        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17           memorije za novi cvor. Memoriju alociranu za cvorove liste
18           treba oslobođiti pre napustanja programa */
19        if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21            oslobodi_listu(&glava, &kraj);
22            exit(EXIT_FAILURE);
23        }
24        printf("\tLista: ");
```

```

    ispisi_listu(glava);
26 }

28 /* Testira se funkcija za brisanje elemenata iz liste */
29 printf("\nUnesite broj koji se brise iz liste: ");
30 scanf("%d", &broj);

32 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
   procitanom sa ulaza */
33 obrisi_cvor(&glava, &kraj, broj);

36 printf("Lista nakon brisanja: ");
37 ispisi_listu(glava);

38 /* Ispisuje se lista unazad */
39 printf("\nLista ispisana u nazad: ");
40 ispisi_listu_unazad(kraj);

42 /* Oslobadja se memorija zauzeta za cvorove liste */
43 oslobodi_listu(&glava, &kraj);

46 exit(EXIT_SUCCESS);
}

```

*main\_c.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"

5 /* 3) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku */
9     Cvor *glava = NULL;
10    Cvor *kraj = NULL;
11    Cvor *trazenii = NULL;
12    int broj;

13    /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
14       bude uredjena neopadajuće */
15    printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
16    while (scanf("%d", &broj) > 0) {
17        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za novi cvor. Memoriju alociranu za cvorove liste
           treba oslobođiti pre napustanja programa */
18        if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
19            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
20            oslobodi_listu(&glava, &kraj);
21            exit(EXIT_FAILURE);
22        }
23    }
24 }

```

## 4 Dinamičke strukture podataka

---

```
27     printf("\tLista: ");
28     ispisi_listu(glava);
29 }
30
31 /* Testira se funkcija za pretragu liste */
32 printf("\nUnesite broj koji se trazi u listi: ");
33 scanf("%d", &broj);
34
35 /* Pokazivac trazenih dobija vrednost rezultata pretrage */
36 trazeni = pretrazi_listu(glava, broj);
37 if (trazeni == NULL)
38     printf("Broj %d se ne nalazi u listi!\n", broj);
39 else
40     printf("Trazenih broj %d je u listi!\n", trazeni->vrednost);
41
42 /* Testira se funkcija za brisanje elemenata iz liste */
43 printf("\nUnesite broj koji se brise iz liste: ");
44 scanf("%d", &broj);
45
46 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
47   procitanom sa ulaza */
48 obrisi_cvor_sortirane_liste(&glava, &kraj, broj);
49
50 printf("Lista nakon brisanja: ");
51 ispisi_listu(glava);
52
53 /* Ispisuje se lista unazad */
54 printf("\nLista ispisana u nazad: ");
55 ispisi_listu_unazad(kraj);
56
57 /* Oslobadja se memorija zauzeta za cvorove liste */
58 oslobodi_listu(&glava, &kraj);
59
60 exit(EXIT_SUCCESS);
61 }
```

### Rešenje 4.14

*stabla.h*

```
#ifndef _STABLA_H_
#define _STABLA_H_ 1

/* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
typedef struct cvor {
    int broj;
    struct cvor *levo;
    struct cvor *desno;
} Cvor;
```

```

12 /* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
   funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
   do greske */
18 int dodaj_u_stablo(Cvor ** adresa_korena, int broj);

20 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
22 Cvor *pretrazi_stablo(Cvor * koren, int broj);

24 /* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
26 Cvor *pronadji_najmanji(Cvor * koren);

28 /* f) Funkcija koja pronalazi cvor koji sadrzi najvecu vrednost u
   stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
34 void obrisi_element(Cvor ** adresa_korena, int broj);

36 /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   postablo - Koren - Desno podstablo ) */
38 void ispisi_stablo_infiksno(Cvor * koren);

40 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
42 void ispisi_stablo_prefiksno(Cvor * koren);

44 /* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren ) */
46 void ispisi_stablo_postfiksno(Cvor * koren);

48 /* k) Funkcija koja oslobođava memoriju zauzetu stablom. */
50 void osloboodi_stablo(Cvor ** adresa_korena);

51 #endif

```

*stabla.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 Cvor *napravi_cvor(int broj)
6 {
7     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
8

```

## 4 Dinamičke strukture podataka

---

```
10    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
11    if (novi == NULL)
12        return NULL;
13
14    /* Inicijalizuju se polja novog cvora. */
15    novi->broj = broj;
16    novi->levo = NULL;
17    novi->desno = NULL;
18
19    /* Vraca se adresa novog cvora. */
20    return novi;
21}
22
23int dodaj_u_stablo(Cvor **adresa_korena, int broj)
24{
25    /* Ako je stablo prazno */
26    if (*adresa_korena == NULL) {
27
28        /* Kreira se novi cvor */
29        Cvor *novi_cvor = napravi_cvor(broj);
30
31        /* Proverava se uspesnost kreiranja */
32        if (novi_cvor == NULL) {
33
34            /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
35             */
36            return 1;
37        }
38        /* Inace ... */
39
40        /* Novi cvor se proglašava korenom stabla */
41        *adresa_korena = novi_cvor;
42
43        /* I vraca se indikator uspesnosti kreiranja */
44        return 0;
45    }
46
47    /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati
48     broj */
49
50    /* Ako je zadata vrednost manja od vrednosti korena */
51    if (broj < (*adresa_korena)->broj)
52
53        /* Broj se dodaje u levo podstablo */
54        return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
55
56    else
57        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
58         dodaje u desno podstablo */
59        return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
60}
```

```

Cvor *pretrazi_stablo(Cvor * koren, int broj)
{
    /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
    if (koren == NULL)
        return NULL;

    /* Ako je trazena vrednost sadrzana u korenu */
    if (koren->broj == broj) {

        /* Prekidamo pretragu */
        return koren;
    }

    /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
    if (broj < koren->broj)

        /* Pretraga se nastavlja u levom podstablu */
        return pretrazi_stablo(koren->levo, broj);

    else
        /* U suprotnom, pretraga se nastavlja u desnom podstablu */
        return pretrazi_stablo(koren->desno, broj);
}

Cvor *pronadji_najmanji(Cvor * koren)
{
    /* Ako je stablo prazno, prekida se pretraga */
    if (koren == NULL)
        return NULL;

    /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
     levo od njega */

    /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
     najmanju vrednost */
    if (koren->levo == NULL)
        return koren;

    /* Inace, pretragu treba nastaviti u levom podstablu */
    return pronadji_najmanji(koren->levo);
}

Cvor *pronadji_najveci(Cvor * koren)
{
    /* Ako je stablo prazno, prekida se pretraga */
    if (koren == NULL)
        return NULL;

    /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
     desno od njega */
}

```

## 4 Dinamičke strukture podataka

---

```
114  /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
115   najvecu vrednost */
116  if (koren->desno == NULL)
117      return koren;
118
119  /* Inace, pretragu treba nastaviti u desnom podstablu */
120  return pronadji_najvechi(koren->desno);
121 }
122
123 void obrisi_element(Cvor ** adresu_korena, int broj)
124 {
125     Cvor *pomocni_cvor = NULL;
126
127     /* Ako je stablo prazno, brisanje nije primenljivo */
128     if (*adresu_korena == NULL)
129         return;
130
131     /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132      stabla, ona se eventualno nalazi u levom podstablu, pa treba
133      rekurzivno primeniti postupak na levo podstablo. Koren ovako
134      modifikovanog stabla je nepromenjen. */
135     if (broj < (*adresu_korena)->broj) {
136         obrisi_element(&(*adresu_korena)->levo, broj);
137         return;
138     }
139
140     /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
141      stabla, ona se eventualno nalazi u desnom podstablu pa treba
142      rekurzivno primeniti postupak na desno podstablo. Koren ovako
143      modifikovanog stabla je nepromenjen. */
144     if ((*adresu_korena)->broj < broj) {
145         obrisi_element(&(*adresu_korena)->desno, broj);
146         return;
147     }
148
149     /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150      jednaka broju koji se brise (tj. slucaj kada treba obrisati
151      koren) */
152
153     /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154      prazno stablo (vraca se NULL) */
155     if ((*adresu_korena)->levo == NULL
156         && (*adresu_korena)->desno == NULL) {
157         free(*adresu_korena);
158         *adresu_korena = NULL;
159         return;
160     }
161
162     /* Ako koren ima samo levog sina, tada se brisanje vrsti tako sto se
163      brise koren, a novi koren postaje levi sin */
164     if ((*adresu_korena)->levo != NULL
```

```

166     && (*adresa_korena)->desno == NULL) {
167     pomocni_cvor = (*adresa_korena)->levo;
168     free(*adresa_korena);
169     *adresa_korena = pomocni_cvor;
170     return;
171 }
172 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako što
173  se briše koren, a novi koren postaje desni sin */
174 if ((*adresa_korena)->desno != NULL
175     && (*adresa_korena)->levo == NULL) {
176     pomocni_cvor = (*adresa_korena)->desno;
177     free(*adresa_korena);
178     *adresa_korena = pomocni_cvor;
179     return;
180 }
181 /* Slučaj kada koren ima oba sina - najpre se potrazi sledbenik
182  korena (u smislu poretka) u stablu. To je upravo po vrednosti
183  najmanji cvor u desnom podstablo. On se može pronaci npr.
184  funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
185  vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
186  broj koji se briše). Zatim se prosto rekurzivno pozove funkcija
187  za brisanje na desnou podstabilu. S obzirom da u njemu treba
188  obrisati najmanji element, a on zasigurno ima najviše jednog
189  potomka, jasno je da će njegovo brisanje biti obavljeni na jedan
190  od jednostavnijih nacija koji su gore opisani. */
191 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
192 (*adresa_korena)->broj = pomocni_cvor->broj;
193 pomocni_cvor->broj = broj;
194 obrisi_element(&(*adresa_korena)->desno, broj);
195 }
196
197 void ispisi_stablo_infiksno(Cvor * koren)
198 {
199 /* Ako stablo nije prazno */
200 if (koren != NULL) {
201
202     /* Prvo se ispisuju svi cvorovi levo od korena */
203     ispisi_stablo_infiksno(koren->levo);
204
205     /* Zatim se ispisuje vrednost u korenu */
206     printf("%d ", koren->broj);
207
208     /* Na kraju se ispisuju cvorovi desno od korena */
209     ispisi_stablo_infiksno(koren->desno);
210 }
211 }
212
213 void ispisi_stablo_prefiksno(Cvor * koren)
214 {
215     /* Ako stablo nije prazno */
216

```

## 4 Dinamičke strukture podataka

---

```
188 if (koren != NULL) {
189
190     /* Prvo se ispisuje vrednost u korenu */
191     printf("%d ", koren->broj);
192
193     /* Zatim se ispisuju svi cvorovi levo od korena */
194     ispisi_stablo_prefiksno(koren->levo);
195
196     /* Na kraju se ispisuju svi cvorovi desno od korena */
197     ispisi_stablo_prefiksno(koren->desno);
198 }
199
200 void ispisi_stablo_postfiksno(Cvor * koren)
201 {
202
203     /* Ako stablo nije prazno */
204     if (koren != NULL) {
205
206         /* Prvo se ispisuju svi cvorovi levo od korena */
207         ispisi_stablo_postfiksno(koren->levo);
208
209         /* Zatim se ispisuju svi cvorovi desno od korena */
210         ispisi_stablo_postfiksno(koren->desno);
211
212         /* Na kraju se ispisuje vrednost u korenu */
213         printf("%d ", koren->broj);
214     }
215 }
216
217 void osloboodi_stablo(Cvor ** adresa_korena)
218 {
219     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
220     if (*adresa_korena == NULL)
221         return;
222
223     /* Inace ... */
224     /* Oslobadja se memorija zauzeta levim podstablom */
225     osloboodi_stablo(&(*adresa_korena)->levo);
226
227     /* Oslobadja se memorija zauzetu desnim podstablom */
228     osloboodi_stablo(&(*adresa_korena)->desno);
229
230     /* Oslobadja se memorija zauzeta korenom */
231     free(*adresa_korena);
232
233     /* Proglasava se stablo praznim */
234     *adresa_korena = NULL;
235 }
```

*main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"

5 int main()
{
7     Cvor *koren;
8     int n;
9     Cvor *trazeni_cvor;

11    /* Proglasava se stablo praznim */
12    koren = NULL;

13    /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
14       dodavanja */
15    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
16    while (scanf("%d", &n) != EOF) {
17        if (dodaj_u_stablo(&koren, n) == 1) {
18            fprintf(stderr, "Neuspeslo dodavanje broja %d\n", n);
19            oslobodi_stablo(&koren);
20            return 0;
21        }
22    }

25    /* Generisu se trazeni ispis: */
26    printf("\nInfiksni ispis: ");
27    ispis_i_stablo_infiksno(koren);
28    printf("\nPrefiksni ispis: ");
29    ispis_i_stablo_prefiksno(koren);
30    printf("\nPostfiksni ispis: ");
31    ispis_i_stablo_postfiksno(koren);

33    /* Demonstrira se rad funkcije za pretragu */
34    printf("\nTrazi se broj: ");
35    scanf("%d", &n);
36    trazeni_cvor = pretrazi_stablo(koren, n);
37    if (trazeni_cvor == NULL)
38        printf("Broj se ne nalazi u stablu!\n");
39
40    else
41        printf("Broj se nalazi u stablu!\n");

43    /* Demonstrira se rad funkcije za brisanje */
44    printf("Brise se broj: ");
45    scanf("%d", &n);
46    obrisi_element(&koren, n);
47    printf("Rezultujuce stablo: ");
48    ispis_i_stablo_infiksno(koren);
49    printf("\n");

51    /* Oslobadja se memorija zauzeta stablom */

```

```
53     osloboidi_stablo(&koren);
54
55 }
```

### Rešenje 4.15

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

6 #define MAX 50

8 /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
11     char *rec;
12     int brojac;
13     struct cvor *levo;
14     struct cvor *desno;
15 } Cvor;

16 /* Funkcija koja kreira novi cvora stabla */
17 Cvor *napravi_cvor(char *rec)
18 {
19     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
20     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
21     if (novi_cvor == NULL)
22         return NULL;
23
24     /* Alocira se memorija za zadatu rec: potrebno je rezervisati
       memoriju za svaki karakter reci ukljuccujuci i terminirajuci nulu
       */
25     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
26     if (novi_cvor->rec == NULL) {
27         free(novi_cvor);
28         return NULL;
29     }
30
31     /* Inicijalizuju se polja u novom cvoru */
32     strcpy(novi_cvor->rec, rec);
33     novi_cvor->brojac = 1;
34     novi_cvor->levo = NULL;
35     novi_cvor->desno = NULL;
36
37     /* Vraca se adresa novog cvora */
38     return novi_cvor;
39 }
40
41 }
```

```

46  /* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
47    uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
48 */
49 int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
50 {
51   /* Ako je stablo prazno */
52   if (*adresa_korena == NULL) {
53     /* Kreira se cvor koji sadrzi zadatu rec */
54     Cvor *novi_cvor = napravi_cvor(rec);
55     /* Proverava se uspesnost kreiranja novog cvora */
56     if (novi_cvor == NULL) {
57       /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
58       */
59       return 1;
60     }
61     /* Inace... */
62     /* Novi cvor se proglašava korenom stabla */
63     *adresa_korena = novi_cvor;
64
65     /* I vraca se indikator uspesnog dodavanja */
66     return 0;
67   }
68
69   /* Ako stablo nije prazno, trazi odgovarajuca pozicija za novu rec
70   */
71
72   /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
73    levo podstablo */
74   if (strcmp(rec, (*adresa_korena)->rec) < 0)
75     return dodaj_u_stablo(&(*adresa_korena)->levo, rec);
76
77   else
78     /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
79      desno podstablo */
80   if (strcmp(rec, (*adresa_korena)->rec) > 0)
81     return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
82
83   else
84     /* Ako je rec jednaka reci u korenu, uvecava se njen broj
85      pojavljivanja */
86     (*adresa_korena)->brojac++;
87
88   /* Funkcija koja oslobadja memoriju zauzetu stablom */
89 void osloboodi_stablo(Cvor ** adresa_korena)
90 {
91   /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
92   if (*adresa_korena == NULL)
93     return;
94
95   /* Inace ... */
96   /* Oslobadja se memorija zauzeta levim podstablom */

```

## 4 Dinamičke strukture podataka

---

```
96    oslobodi_stablo(&(*adresa_korena)->levo);

98    /* Oslobadja se memorija zauzeta desnim podstablom */
oslobodi_stablo(&(*adresa_korena)->desno);

100   /* Oslobadja se memorija zauzeta korenom */
102   free((*adresa_korena)->rec);
103   free(*adresa_korena);

104   /* Stablo se proglašava praznim */
106   *adresa_korena = NULL;
107 }

108  /* Funkcija koja pronađe cvor koji sadrži najfrekventniju rec (rec
109   sa najvećim brojem pojavljivanja) */
Cvor *nadji_najfrekventniju_rec(Cvor * koren)
110 {
111     Cvor *max, *max_levo, *max_desno;

112     /* Ako je stablo prazno, prekida se sa pretragom */
113     if (koren == NULL)
114         return NULL;

115     /* Pronađe se najfrekventnija rec u levom podstablu */
max_levo = nadji_najfrekventniju_rec(koren->levo);

116     /* Pronađe se najfrekventnija rec u desnem podstablu */
max_desno = nadji_najfrekventniju_rec(koren->desno);

117     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
       podstabla, korena i desnog podstabla */
max = koren;
118     if (max_levo != NULL && max_levo->brojac > max->brojac)
119         max = max_levo;
120     if (max_desno != NULL && max_desno->brojac > max->brojac)
121         max = max_desno;

122     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
123     return max;
124 }

125  /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
       pravene brojem pojavljivanja */
void prikazi_stablo(Cvor * koren)
126 {
127     /* Ako je stablo prazno, završava se sa ispisom */
128     if (koren == NULL)
129         return;

130     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
       podstabla */
prikazi_stablo(koren->levo);
```

```

148     /* Zatim rec iz korena */
150     printf("%s: %d\n", koren->rec, koren->brojac);

152     /* I nastavlja se sa ispisom reci iz desnog podstabla */
153     prikazi_stablo(koren->desno);
154 }

156 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
157 niz rec. Maksimalna duzina reci je odredjena argumentom max.
158 Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
159 suprotnom. Rec je niz malih ili velikih slova. */
160 int procitaj_rec(FILE * f, char rec[], int max)
161 {
162     /* Karakter koji se cita */
163     int c;
164
165     /* Indeks pozicije na koju se smesta procitani karakter */
166     int i = 0;
167
168     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
169      nije stiglo do kraja datoteke... */
170     while (i < max - 1 && (c = fgetc(f)) != EOF) {
171         /* Proverava se da li je procitani karakter slovo */
172         if (isalpha(c))
173             /* Ako jeste, smesta se u niz - pritom se vrsti konverzija u
174              mala slova jer program treba da bude neosetljiv na razliku
175              izmedju malih i velikih slova */
176             rec[i++] = tolower(c);
177
178         else
179             /* Ako nije, proverava se da li je procitano barem jedno slovo
180              nove reci */
181             /* Ako jeste, prekida se sa citanjem */
182             if (i > 0)
183                 break;
184
185         /* U suprotnom se ide na sledecu iteraciju */
186     }
187
188     /* Dodaje se na rec terminirajuca nula */
189     rec[i] = '\0';
190
191     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
192     return i > 0 ? 0 : EOF;
193 }
194
195 int main(int argc, char **argv)
196 {
197     Cvor *koren = NULL, *max;
198     FILE *f;
199     char rec[MAX];

```

## 4 Dinamičke strukture podataka

---

```
200  /* Provera da li je navedeno ime datoteke prilikom pokretanja
201   programa */
202  if (argc < 2) {
203      fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
204      exit(EXIT_FAILURE);
205  }
206
207  /* Priprema datoteke za citanje */
208  if ((f = fopen(argv[1], "r")) == NULL) {
209      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
210             argv[1]);
211      exit(EXIT_FAILURE);
212  }
213
214  /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
215   pretrage uz proveru uspesnosti dodavanja */
216  while (procitaj_rec(f, rec, MAX) != EOF) {
217      if (dodaj_u_stablo(&koren, rec) == 1) {
218          fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
219          osloboodi_stablo(&koren);
220          exit(EXIT_FAILURE);
221      }
222  }
223
224  /* Posto je citanjem reci zavrseno, zatvara se datoteka */
225  fclose(f);
226
227  /* Prikazuju se sve reci iz teksta i brojevi njihovih
228   pojavljivanja. */
229  prikazi_stablo(koren);
230
231  /* Pronalazi se najfrekventnija rec */
232  max = nadji_najfrekventniju_rec(koren);
233
234  /* Ako takve reci nema... */
235  if (max == NULL)
236
237      /* Ispisuje se odgovarajuce obavestenje */
238      printf("U tekstu nema reci!\n");
239
240  else
241      /* Inace, ispisuje se broj pojavljivanja reci */
242      printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
243             max->rec, max->brojac);
244
245  /* Oslobadja se dinamicki alociran prostor za stablo */
246  osloboodi_stablo(&koren);
247
248  exit(EXIT_SUCCESS);
249 }
```

## Rešenje 4.16

```

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

6 #define MAXIME_DATOTEKE 50
7 #define MAX_CIFARA 13
8 #define MAXIME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrži ime i prezime, broj
11    telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
13     char ime_i_prezime[MAXIME_I_PREZIME];
14     char telefon[MAX_CIFARA];
15     struct cvor *levo;
16     struct cvor *desno;
17 } Cvor;
18
19 /* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
21 {
22     /* Alocira se memorija za novi cvor i proverava se uspesnost
23        alokacije. */
24     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
25     if (novi_cvor == NULL)
26         return NULL;
27
28     /* Inicijalizuju se polja novog cvora */
29     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30     strcpy(novi_cvor->telefon, telefon);
31     novi_cvor->levo = NULL;
32     novi_cvor->desno = NULL;
33
34     /* Vraca se adresa novog cvora */
35     return novi_cvor;
36 }
37
38 /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
39    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
40    povratna vrednost je 1 */
41 int
42 dodaj_u_stablo(Cvor **adresa_korena, char *ime_i_prezime,
43                  char *telefon)
44 {
45     /* Ako je stablo prazno */
46     if (*adresa_korena == NULL) {
47         /* Kreira se novi cvor */
48         Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
49         /* Proverava se uspesnost kreiranja novog cvora */
50         if (novi_cvor == NULL) {
51             /* Pogreska - ne moguce da se dodigne novi cvor */
52             return 1;
53         }
54         /* Ukoliko je stablo prazno, novi cvor postaje koren */
55         *adresa_korena = novi_cvor;
56     }
57     /* Ako je stablo neprazno, poziva se funkcija za dodavanje u stablo */
58     else
59         dodaj_u_stablo(&(novi_cvor->desno), ime_i_prezime, telefon);
60
61     return 0;
62 }

```

## 4 Dinamičke strukture podataka

---

```
52     /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
53     */
54     return 1;
55 }
56 /* Inace...
57 /* Novi cvor se proglašava korenom stabla */
58 *adresa_korena = novi_cvor;
59
60     /* I vraca se indikator uspesnog dodavanja */
61     return 0;
62 }
63
64 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
65    unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
66    treba da bude pretrazivacko po ovom polju */
67
68 /* Ako je zadato ime i prezime leksikografski manje od imena i
69    prezimena sadrzanog u korenju, podaci se dodaju u levo podstablo
70    */
71 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72     < 0)
73     return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                           telefon);
75
76 else
77     /* Ako je zadato ime i prezime leksikografski vece od imena i
78        prezimena sadrzanog u korenju, podaci se dodaju u desno
79        podstablo */
80 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
81     return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
82                           telefon);
83
84 /* Funkcija koja oslobadja memoriju zauzetu stablom */
85 void oslobodi_stablo(Cvor ** adresu_korena)
86 {
87     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
88     if (*adresa_korena == NULL)
89         return;
90
91     /* Inace ...
92     /* Oslobadja se memorija zauzeta levim podstablom */
93     oslobodi_stablo(&(*adresa_korena)->levo);
94
95     /* Oslobadja se memorija zauzeta desnim podstablom */
96     oslobodi_stablo(&(*adresa_korena)->desno);
97
98     /* Oslobadja se memorija zauzeta korenom */
99     free(*adresa_korena);
100
101    /* Stablo se proglašava praznim */
102    *adresa_korena = NULL;
```

```

102  }
104  /* Funkcija koja ispisuje imenik u leksikografskom poretku */
105  /* Napomena: ova funkcija nije trazena u zadatku ali se moze
106   koristiti za proveru da li je stablo lepo kreirano ili ne */
107  void prikazi_stablo(Cvor * koren)
108  {
109      /* Ako je stablo prazno, zavrsava se sa ispisom */
110     if (koren == NULL)
111         return;
112
113     /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
114      podstabla */
115     prikazi_stablo(koren->levo);
116
117     /* Zatim se ispisuju podaci iz korena */
118     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
119
120     /* I nastavlja se sa ispisom podataka iz desnog podstabla */
121     prikazi_stablo(koren->desno);
122 }
123
124 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje ime
125    i prezime i broj telefona u odgovarajuce nizove. Maksimalna duzina
126    imena i prezimena odredjena je konstantom MAX_IME_PREZIME, a
127    maksimalna duzina broja telefona konstantom MAX_CIFARA. Funkcija
128    vraca EOF ako nema vise kontakata ili 0 u suprotnom. */
129 int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
130 {
131     /* Karakter koji se cita */
132     int c;
133
134     /* Indeks pozicije na koju se smesta procitani karakter */
135     int i = 0;
136
137     /* Linije datoteke koje se obradjuju su formata Ime Prezime
138       BrojTelefona */
139
140     /* Preskacu se eventualne praznine sa pocetka linije datoteke */
141     while ((c = fgetc(f)) != EOF && isspace(c));
142
143     /* Prvo procitano slovo upisuje se u ime i prezime */
144     if (!feof(f))
145         ime_i_prezime[i++] = c;
146
147     /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
148       cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
149       Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
150       procitanog karaktera i da se slucajno ne dodje do kraja datoteke
151       */
152     while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
153         if (!isdigit(c))

```

## 4 Dinamičke strukture podataka

---

```
154     ime_i_prezime[i++] = c;
156
157     else if (i > 0)
158         break;
159
160     /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
161        blanko karaktera */
162     ime_i_prezime[--i] = '\0';
163
164     /* I pocinje se sa citanjem broja telefona */
165     i = 0;
166
167     /* Upisuje se cifra koja je vec procitana */
168     telefon[i++] = c;
169
170     /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
171        karaktera cije prisustvo nije dozvoljeno u broju telefona */
172     while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
173         if (c == '/') || c == '-' || isdigit(c))
174             telefon[i++] = c;
175         else
176             break;
177
178     /* Upisuje se terminirajuca nula */
179     telefon[i] = '\0';
180
181     /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
182     return !feof(f) ? 0 : EOF;
183 }
184
185 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
186 */
187 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188 {
189     /* Ako je imenik prazan, zavrsava se sa pretragom */
190     if (koren == NULL)
191         return NULL;
192
193     /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
194        zavrsava sa pretragom */
195     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
196         return koren;
197
198     /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
199        korenu pretraga se nastavlja levo */
200     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
201         return pretrazi_imenik(koren->levo, ime_i_prezime);
202
203     else
204         /* u suprotnom, pretraga se nastavlja desno */
205         return pretrazi_imenik(koren->desno, ime_i_prezime);
```

```

206 }
207
208 int main(int argc, char **argv)
209 {
210     char ime_datoteke[MAX_IME_DATOTEKE];
211     Cvor *koren = NULL;
212     Cvor *trazeni;
213     FILE *f;
214     char ime_i_prezime[MAX_IME_I_PREZIME];
215     char telefon[MAX_CIFARA];
216     char c;
217     int i;
218
219     /* Ucitava se ime datoteke i vrsti se njena priprema za citanje */
220     printf("Unesite ime datoteke: ");
221     scanf("%s", ime_datoteke);
222     getchar();
223     if ((f = fopen(ime_datoteke, "r")) == NULL) {
224         fprintf(stderr, "Greska: Neuspjesno otvaranje datoteke %s.\n",
225                 ime_datoteke);
226         exit(EXIT_FAILURE);
227     }
228
229     /* Citaju se podaci iz datoteke i smestanju u binarno stablo
230      pretrage uz proveru uspesnosti dodavanja */
231     while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF) {
232         if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
233             fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
234                     ime_i_prezime);
235             osloboidi_stablo(&koren);
236             exit(EXIT_FAILURE);
237         }
238
239         /* Zatvara se datoteka */
240         fclose(f);
241
242         /* Omogucava se pretraga imenika */
243         while (1) {
244             /* Ucitavaju se ime i prezime */
245             printf("Unesite ime i prezime: ");
246             i = 0;
247             while ((c = getchar()) != '\n')
248                 ime_i_prezime[i++] = c;
249             ime_i_prezime[i] = '\0';
250
251             /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
252                funkcionalnost */
253             if (strcmp(ime_i_prezime, "KRAJ") == 0)
254                 break;
255
256             /* Inace se ispisuje rezultat pretrage */
257             trazeni = pretrazi_imenik(koren, ime_i_prezime);

```

## 4 Dinamičke strukture podataka

---

```
258     if (trazeni == NULL)
259         printf("Broj nije u imeniku!\n");
260     else
261         printf("Broj je: %s \n", trazeni->telefon);
262     }
263
264     /* Oslobadja se memorija zauzeta imenikom */
265     osloboди_stablo(&koren);
266
267     exit(EXIT_SUCCESS);
268 }
```

### 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 uspeh, uspeh iz matematike, uspeh iz maternjeg
9    jezika i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX];
12     char prezime[MAX];
13     double uspeh;
14     double matematika;
15     double jezik;
16     struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
18 } Cvor;
19
20 /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
22                     double matematika, double jezik)
23 {
24     /* Alocira se memorija za novi cvor i proverava se uspesnost
25        alokacije. */
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizuju se 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;
```

```

39  /* Vraca se adresa kreiranog cvora */
40  return novi;
41 }

43 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
44    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
45    povratna vrednost je 1 */
46 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47                      double uspeh, double matematika, double jezik)
48 {
49     /* Ako je stablo prazno */
50     if (*koren == NULL) {
51         /* Kreira se novi cvor */
52         Cvor *novi_cvor =
53             napravi_cvor(ime, prezime, uspeh, matematika, jezik);
54         /* Proverava se uspesnost kreiranja novog cvora */
55         if (novi_cvor == NULL) {
56             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57             */
58             return 1;
59         }
60         /* Inace... */
61         /* Novi cvor se proglašava korenom stabla */
62         *koren = novi_cvor;
63
64         /* I vraca se indikator uspesnog dodavanja */
65         return 0;
66     }
67
68     /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69      sortirano po ukupnom broju poena */
70     if (uspeh + matematika + jezik >
71         (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
72         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
73                               matematika, jezik);
74     else
75         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
76                               matematika, jezik);
77 }

79 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
80 void osloboodi_stablo(Cvor ** koren)
81 {
82     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
83     if (*koren == NULL)
84         return;
85
86     /* Inace ... */
87     /* Oslobadja se memorija zauzeta levim podstablom */
88     osloboodi_stablo(&(*koren)->levo);
89 }
```

```

91  /* Oslobadja se memorija zauzeta desnim podstabom */
93  oslobodi_stablo(&(*koren)->desno);
95
97  /* Oslobadja se memorija zauzeta korenom */
98  free(*koren);
99
100
101 /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
102  polozili jednaka 0 ispisuju se informacije o ucenicima koji nisu
103  polozili prijemni, a ako je vrednost argumenta razlicita od nule,
104  ispisuju se informacije o ucenicima koji su polozili prijemni */
105 void stampaj(Cvor * koren, int polozili)
106 {
107     /* Stablo je prazno - prekida se sa ispisom */
108     if (koren == NULL)
109         return;
110
111     /* Stampaju se informacije iz levog podstabla */
112     stampaj(koren->levo, polozili);
113
114     /* Stampaju se informacije iz korenog cvora */
115     if (polozili && koren->matematika + koren->jezik >= 10)
116         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
117               koren->prezime, koren->uspeh, koren->matematika,
118               koren->jezik,
119               koren->uspeh + koren->matematika + koren->jezik);
120     else if (!polozili && koren->matematika + koren->jezik < 10)
121         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
122               koren->prezime, koren->uspeh, koren->matematika,
123               koren->jezik,
124               koren->uspeh + koren->matematika + koren->jezik);
125
126     /* Stampaju se informacije iz desnog podstabla */
127     stampaj(koren->desno, polozili);
128 }
129
130 /* Funkcija koja određuje koliko studenata nije polozilo prijemni
131  ispit */
132 int nisu_polozili(Cvor * koren)
133 {
134     /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
135     if (koren == NULL)
136         return 0;
137
138     /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov za
139      polaganje nije ispunjen za koreni cvor, broj studenata se
140      uvećava za 1 */
141

```

```

143     if (koren->matematika + koren->jezik < 10)
144         return 1 + nisu_polozili(koren->levo) +
145             nisu_polozili(koren->desno);
146
147     return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
148 }
149
150 int main(int argc, char **argv)
151 {
152     FILE *in;
153     Cvor *koren;
154     char ime[MAX], prezime[MAX];
155     double uspeh, matematika, jezik;
156
157     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
158     in = fopen("prijemni.txt", "r");
159     if (in == NULL) {
160         fprintf(stderr,
161                 "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
162         exit(EXIT_FAILURE);
163     }
164
165     /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
166      dodavanja */
167     koren = NULL;
168     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
169                   &matematika, &jezik) != EOF) {
170         if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik
171                           )
172             == 1) {
173             fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
174                     prezime);
175             osloboidi_stablo(&koren);
176             exit(EXIT_FAILURE);
177         }
178     }
179
180     /* Zatvaranje datoteke */
181     fclose(in);
182
183     /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
184     stampaj(koren, 1);
185
186     /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
187      prijemni */
188     if (nisu_polozili(koren) != 0)
189         printf("-----\n");
190
191     /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
192     stampaj(koren, 0);
193
194     /* Oslobadja se memorija zauzeta stablom */

```

## 4 Dinamičke strukture podataka

---

```
193     osloboidi_stablo(&koren);
195
196     exit(EXIT_SUCCESS);
197 }
```

### Rešenje 4.18

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_NISKA 51
6
7 /* Struktura koja opisuje jedan cvor stabla: sadrži ime i prezime
8    osobe, dan i mesec rođenja i redom pokazivace na levo i desno
9    podstabla */
10 typedef struct cvor_stabla {
11     char ime[MAX_NISKA];
12     char prezime[MAX_NISKA];
13     int dan;
14     int mesec;
15     struct cvor_stabla *levo;
16     struct cvor_stabla *desno;
17 } Cvor;
18
19 /* Funkcija koja kreira novi cvor */
20 Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
22     /* Alocira se memorija */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizuju se polja strukture */
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->dan = dan;
31     novi->mesec = mesec;
32     novi->levo = NULL;
33     novi->desno = NULL;
34
35     /* Vraca se adresa novog cvora */
36     return novi;
37 }
38
39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
40    uređeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
41    dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
42    vrednost je 1 */
43 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
44                     int dan, int mesec)
```

```

45 {
46     /* Ako je stablo prazno */
47     if (*koren == NULL) {
48
49         /* Kreira se novi cvor */
50         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51         /* Proverava se uspesnost kreiranja novog cvora */
52         if (novi_cvor == NULL) {
53             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
54             */
55             return 1;
56         }
57         /* Inace... */
58         /* Novi cvor se proglašava korenom stabla */
59         *koren = novi_cvor;
60
61         /* I vraca se indikator uspesnog dodavanja */
62         return 0;
63     }
64
65     /* Stablo se uređuje po mesecu, a zatim po danu u okviru istog
66      meseca */
67     if (mesec < (*koren)->mesec)
68         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
69     else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
70         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
71     else
72         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec)
73     ;
74 }
75
76 /* Funkcija vrši pretragu stabla i vraca cvor sa traženim datumom */
77 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
78 {
79     /* Stablo je prazno, obustavlja se pretraga */
80     if (koren == NULL)
81         return NULL;
82
83     /* Ako je traženi datum u korenu */
84     if (koren->dan == dan && koren->mesec == mesec)
85         return koren;
86
87     /* Ako je mesec traženog datuma manji od meseca sadržanog u korenu
88      ili ako su meseci isti ali je dan traženog datuma manji od
89      aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
90      svakako proverava da li leva grana postoji - ako ne postoji
91      treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
92      */
93     if (mesec < koren->mesec
94         || (mesec == koren->mesec && dan < koren->dan)) {
95         if (koren->levo == NULL)
96             return koren;

```

## 4 Dinamičke strukture podataka

---

```
95     else
96         return pretrazi(koren->levo, dan, mesec);
97     }
98
99     /* Inace se nastavlja pretraga u desnom delu */
100    return pretrazi(koren->desno, dan, mesec);
101}
102
103/* Funkcija koja pronalazi najmanji datum u stablu */
104Cvor *pronadji_najmanji_datum(Cvor * koren)
105{
106    /* Stablo je prazno, obustavlja se pretraga */
107    if (koren == NULL)
108        return NULL;
109
110    /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
111     sadrzi najmanji datum */
112    if (koren->levo == NULL)
113        return koren;
114    else
115        /* Inace, trazimo manji datum u levom podstablu */
116        return pronadji_najmanji_datum(koren->levo);
117}
118
119/* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
120 */
121void datum_u_nisku(int dan, int mesec, char datum[])
122{
123    if (dan < 10) {
124        datum[0] = '0';
125        datum[1] = dan + '0';
126    } else {
127        datum[0] = dan / 10 + '0';
128        datum[1] = dan % 10 + '0';
129    }
130    datum[2] = '.';
131
132    if (mesec < 10) {
133        datum[3] = '0';
134        datum[4] = mesec + '0';
135    } else {
136        datum[3] = mesec / 10 + '0';
137        datum[4] = mesec % 10 + '0';
138    }
139    datum[5] = '.';
140    datum[6] = '\0';
141}
142
143/* Funkcija koja oslobadja memoriju zauzetu stablom */
144void oslobodi_stablo(Cvor ** adresa_korena)
145{
146    /* Stablo je prazno */
```

```

147     if (*adresa_korena == NULL)
148         return;
149
150     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
151     if ((*adresa_korena)->levo)
152         oslobodi_stablo(&(*adresa_korena)->levo);
153
154     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
155     if ((*adresa_korena)->desno)
156         oslobodi_stablo(&(*adresa_korena)->desno);
157
158     /* Oslobadja se memorija zauzeta korenom */
159     free(*adresa_korena);
160
161     /* Proglasava se stablo praznim */
162     *adresa_korena = NULL;
163 }
164
165 int main(int argc, char **argv)
166 {
167     FILE *in;
168     Cvor *koren;
169     Cvor *slavljenik;
170     char ime[MAX_NISKA], prezime[MAX_NISKA];
171     int dan, mesec;
172     char datum[7];
173
174     /* Provera da li je zadato ime ulazne datoteke */
175     if (argc < 2) {
176         /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
177            programa */
178         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
179         exit(EXIT_FAILURE);
180     }
181
182     /* Inace, priprema se datoteka za citanje */
183     in = fopen(argv[1], "r");
184     if (in == NULL) {
185         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
186                 argv[1]);
187         exit(EXIT_FAILURE);
188     }
189
190     /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
191        */
192     koren = NULL;
193     while (fscanf(
194             (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
195             if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
196                 fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
197                         prezime);
198                 oslobodi_stablo(&koren);
199             }
200         }
201     }
202 }
```

## 4 Dinamičke strukture podataka

---

```
197         exit(EXIT_FAILURE);
198     }
199
200     /* Datoteka se zatvara */
201     fclose(in);
202
203     /* Omogucuje se pretraga podataka */
204     while (1) {
205
206         /* Ucitava se novi datum */
207         printf("Unesite datum: ");
208         if (scanf("%d.%d.", &dan, &mesec) == EOF)
209             break;
210
211         /* Pretrazuje se stablo */
212         slavljenik = pretrazi(koren, dan, mesec);
213
214         /* Ispisuju se pronadjeni podaci */
215
216         /* Ako slavljenik nije pronadjen, to moze znaci da: */
217         /* 1. drvo je prazno */
218         if (slavljenik == NULL && koren == NULL) {
219             printf("Nema podataka o ovom ni o sledecem rođendanu.\n");
220             continue;
221         }
222         /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223         ovom slučaju se pretraga vrši pocevsi od naredne godine i
224         ispisuje se najmanji datum */
225         if (slavljenik == NULL) {
226             slavljenik = pronadji_najmanji_datum(koren);
227             datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
228             printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229                   slavljenik->prezime, datum);
230             continue;
231         }
232
233         /* Ako je slavljenik pronadjen, razlikuju se slučajevi:
234         /* 1. Pronadjeni su tacni podaci */
235         if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
236             printf("Slavljenik: %s %s\n", slavljenik->ime,
237                   slavljenik->prezime);
238             continue;
239         }
240
241         /* 2. Pronadjeni su podaci o prvom sledecem rođendanu */
242         datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
243         printf("Slavljenik: %s %s %s\n", slavljenik->ime,
244               slavljenik->prezime, datum);
245     }
246
247     /* Oslobadja se memorija zauzeta stablom */
248     osloboodi_stablo(&koren);
```

```

249     exit(EXIT_SUCCESS);
251 }
```

## Rešenje 4.19

```

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja proverava da li su dva stabla koja sadrže cele
brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
odnosno 0 ako nisu */
int identitet(Cvor * koren1, Cvor * koren2)
{
    /* Ako su oba stabla prazna, jednaka su */
    if (koren1 == NULL && koren2 == NULL)
        return 1;

    /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednakih */
    if (koren1 == NULL || koren2 == NULL)
        return 0;

    /* Ako su oba stabla neprazna i u korenu se nalaze različite
vrednosti, može se zaključiti da se razlikuju */
    if (koren1->broj != koren2->broj)
        return 0;

    /* Inace, proverava se da li vazi i jednakost levih i desnih
podstabala */
    return (identitet(koren1->levo, koren2->levo)
            && identitet(koren1->desno, koren2->desno));
}

int main()
{
    int broj;
    Cvor *koren1, *koren2;

    /* Ucitavaju se elementi prvog stabla */
    koren1 = NULL;
    printf("Prvo stablo: ");
    scanf("%d", &broj);
    while (broj != 0) {
        if (dodaj_u_stablo(&koren1, broj) == 1) {
            fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
            osloboodi_stablo(&koren1);
            return 0;
    }
}
```

## 4 Dinamičke strukture podataka

---

```
46     scanf("%d", &broj);
47 }
48
49 /* Ucitavaju se elementi drugog stabla */
50 koren2 = NULL;
51 printf("Drugo stablo: ");
52 scanf("%d", &broj);
53 while (broj != 0) {
54     if (dodaj_u_stablo(&koren2, broj) == 1) {
55         fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
56         oslobodi_stablo(&koren2);
57         return 0;
58     }
59     scanf("%d", &broj);
60 }
61
62 /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
63 njen rezultat. */
64 if (identitet(koren1, koren2))
65     printf("Stabla jesu identicna.\n");
66 else
67     printf("Stabla nisu identicna.\n");
68
69 /* Oslobadja se memorija zauzeta stablima */
70 oslobodi_stablo(&koren1);
71 oslobodi_stablo(&koren2);
72
73 return 0;
74 }
```

### Rešenje 4.20

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

4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija kreira novo stablo identично stablu koje je dato korenom.
8 Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
9 odnosno 1 ukoliko je doslo do greske */
10 int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
11 {
12     /* Izlaz iz rekurzije */
13     if (koren == NULL) {
14         *duplikat = NULL;
15         return 0;
16     }
17
18     /* Duplira se koren stabla i postavlja da bude koren novog stabla
19     */
```

```

1 *duplikat = napravi_cvor(koren->broj);
2 if (*duplikat == NULL) {
3     return 1;
4 }
5
6 /* Rekurzivno se dupliraju levo i desno podstabla i njihove adrese
7    se cuvaju redom u pokazivacima na levo i desno podstabla korena
8    duplikata */
9 int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
10 int kopija_desno =
11     kopiraj_stablo(koren->desno, &(*duplikat)->desno);
12
13 /* Ako je uspesno duplirano i levo i desno podstabla */
14 if (kopija_levo == 0 && kopija_desno == 0)
15     /* Uspesno je duplirano i celo stablo */
16     return 0;
17 /* Inace, prijavljuje se da je doslo do greske */
18 return 1;
19
20 }
21
22 /* Funkcija izracunava uniju dva skupa predstavljena stablima -
23    rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
24    Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
25    uspesno, odnosno 1 ukoliko je doslo do greske */
26 int kreiraj_uniju(Cvor ** adresu_korenai, Cvor * koren2)
27 {
28     /* Ako drugo stablo nije prazno */
29     if (koren2 != NULL) {
30         /* 1. Dodaje se njegov koren u prvo stablo */
31         if (dodaj_u_stablo(adresa_korenai, koren2->broj) == 1) {
32             return 1;
33         }
34
35         /* 2. Rekurzivno se racuna unija levog i desnog podstabala drugog
36            stabla sa prvim stablom */
37         int unija_levo = kreiraj_uniju(adresa_korenai, koren2->levo);
38         int unija_desno = kreiraj_uniju(adresa_korenai, koren2->desno);
39
40         /* Ako je unija podstabala uspesno kreirana */
41         if (unija_levo == 0 && unija_desno == 0)
42             /* Uspesno je kreirana i unija stabala */
43             return 0;
44
45         /* U suprotnom se prijavljuje da je doslo do greske */
46         return 1;
47     }
48
49     /* Ako je drugo stablo prazno, nista se ne preduzima */
50     return 0;
51 }
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

## 4 Dinamičke strukture podataka

---

```
72  /* Funkcija izracunava presek dva skupa predstavljana stablima -  
73   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.  
74   Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka  
75   uspesno, odnosno 1 ukoliko je doslo do greske */  
76  int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)  
77  {  
78      /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */  
79      if (*adresa_korena1 == NULL)  
80          return 0;  
81  
82      /* Inace... */  
83      /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.  
84         iz levog i desnog podstabla prvog stabla brisu se svi oni  
85         elementi koji ne postoje u drugom stablu */  
86      int presek_levo = kreiraj_presek(&(*adresa_korena1)->levo, koren2);  
87      int presek_desno =  
88          kreiraj_presek(&(*adresa_korena1)->desno, koren2);  
89      if (presek_levo == 0 && presek_desno == 0) {  
90          /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on  
91             uklanja iz prvog stabla */  
92          if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)  
93              obrisi_element(adresa_korena1, (*adresa_korena1)->broj);  
94  
95          /* Presek stabala je uspesno kreiran */  
96          return 0;  
97      }  
98      /* Inece, prijavljuje se da je doslo do greske */  
99      return 1;  
100 }  
101  
102 /* Funkcija izracunava razliku dva skupa predstavljana stablima -  
103 rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.  
104 Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike  
105 uspesno, odnosno 1 ukoliko je doslo do greske */  
106 int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)  
107 {  
108     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */  
109     if (*adresa_korena1 == NULL)  
110         return 0;  
111  
112     /* Inace... */  
113     /* Kreira se razlika levog i desnog podstabla sa drugim stablom,  
114        tj. iz levog i desnog podstabla prvog stabla se brisu svi oni  
115        elementi koji postoje i u drugom stablu */  
116     int razlika_levo =  
117         kreiraj_razliku(&(*adresa_korena1)->levo, koren2);  
118     int razlika_desno =  
119         kreiraj_razliku(&(*adresa_korena1)->desno, koren2);  
120     if (razlika_levo == 0 && razlika_desno == 0) {  
121         /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on  
122            uklanja se iz prvog stabla */  
123         if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
```

```

124     obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
125
126     /* Razlika stabala je uspesno kreirana */
127     return 0;
128 }
129
130 /* Ineće, prijavljuje se da je doslo do greske */
131 return 1;
132 }
133
134 int main()
135 {
135     Cvor *skup1;
136     Cvor *skup2;
137     Cvor *pomocni_skup = NULL;
138     int n;
139
140     /* Ucitavaju se elementi prvog skupa */
141     skup1 = NULL;
142     printf("Prvi skup: ");
143     while (scanf("%d", &n) != EOF) {
144         if (dodaj_u_stablo(&skup1, n) == 1) {
145             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
146             osloboidi_stablo(&skup1);
147             return 0;
148         }
149     }
150
151     /* Ucitavaju se elementi drugog skupa */
152     skup2 = NULL;
153     printf("Drugi skup: ");
154     while (scanf("%d", &n) != EOF) {
155         if (dodaj_u_stablo(&skup2, n) == 1) {
156             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
157             osloboidi_stablo(&skup2);
158             return 0;
159         }
160     }
161
162     /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
163     bi se isti mogao iskoristiti i za preostale operacije */
164     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
165         osloboidi_stablo(&skup1);
166         osloboidi_stablo(&pomocni_skup);
167         return 0;
168     }
169     if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
170         osloboidi_stablo(&pomocni_skup);
171         osloboidi_stablo(&skup2);
172         return 0;
173     }
174     printf("Unija: ");

```

## 4 Dinamičke strukture podataka

---

```
176     ispisi_stablo_infiksno(pomocni_skup);
177     putchar('\n');

178     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
179      operacije */
180     oslobodi_stablo(&pomocni_skup);

182     /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
183      kako bi se isti mogao iskoristiti i za preostale operacije */
184     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
185         oslobodi_stablo(&skup1);
186         oslobodi_stablo(&pomocni_skup);
187         return 0;
188     }
189     if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190         oslobodi_stablo(&pomocni_skup);
191         oslobodi_stablo(&skup2);
192         return 0;
193     }
194     printf("Presek: ");
195     ispisi_stablo_infiksno(pomocni_skup);
196     putchar('\n');

198     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
199      operacije */
200     oslobodi_stablo(&pomocni_skup);

202     /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
203      kako bi se isti mogao iskoristiti i za preostale operacije */
204     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
205         oslobodi_stablo(&skup1);
206         oslobodi_stablo(&pomocni_skup);
207         return 0;
208     }
209     if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
210         oslobodi_stablo(&pomocni_skup);
211         oslobodi_stablo(&skup2);
212         return 0;
213     }
214     printf("Razlika: ");
215     ispisi_stablo_infiksno(pomocni_skup);
216     putchar('\n');

218     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
219      operacije */
220     oslobodi_stablo(&pomocni_skup);

222     /* Oslobadja se memorija zauzeta polaznim skupovima */
223     oslobodi_stablo(&skup1);
224     oslobodi_stablo(&skup2);

226     return 0;
```

}

**Rešenje 4.21**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 #define MAX 50
8
9 /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
10    cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11    su smestene u niz. */
12 int kreiraj_niz(Cvor * koren, int a[])
13 {
14     int r, s;
15
16     /* Stablo je prazno - u niz je smesteno 0 elemenata */
17     if (koren == NULL)
18         return 0;
19
20     /* Dodaju se u niz elementi iz levog podstabla */
21     r = kreiraj_niz(koren->levo, a);
22
23     /* Tekuća vrednost promenljive r je broj elemenata koji su upisani
24       u niz i na osnovu nje se može odrediti indeks novog elementa */
25
26     /* Smesta se vrednost iz korena */
27     a[r] = koren->broj;
28
29     /* Dodaju se elementi iz desnog podstabla */
30     s = kreiraj_niz(koren->desno, a + r + 1);
31
32     /* Racuna se indeks na koji treba smestiti naredni element */
33     return r + s + 1;
34 }
35
36 /* Funkcija sortira niz tako što najpre elemente niza smesti u
37    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
38    desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
39    kreiran i sortiran, a 1 ukoliko je doslo do greske.
40
41 Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
42 sto je to slučaj sa ostalim opisanim algoritmima sortiranja jer se
43 sortiranje vrši u pomocnoj dinamickoj strukturi, a ne razmenom
44 elemenata niza. */
45 int sortiraj(int a[], int n)
46 {
47     int i;

```

## 4 Dinamičke strukture podataka

---

```
49     Cvor *koren;
50
51     /* Kreira se stablo smestanjem elemenata iz niza u stablo */
52     koren = NULL;
53     for (i = 0; i < n; i++) {
54         if (dodaj_u_stablo(&koren, a[i]) == 1) {
55             oslobodi_stablo(&koren);
56             return 1;
57         }
58     }
59     /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
60      a */
61     kreiraj_niz(koren, a);
62
63     /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
64      */
65     oslobodi_stablo(&koren);
66
67     /* Vraca se indikator uspesnog sortiranja */
68     return 0;
69 }
70
71 int main()
72 {
73     int a[MAX];
74     int n, i;
75
76     /* Ucitavaju se dimenzija i elementi niza */
77     printf("n: ");
78     scanf("%d", &n);
79     if (n < 0 || n > MAX) {
80         printf("Greska: pogresna dimenzija niza!\n");
81         return 0;
82     }
83
84     printf("a: ");
85     for (i = 0; i < n; i++)
86         scanf("%d", &a[i]);
87
88     /* Poziva se funkcija za sortiranje */
89     if (sortiraj(a, n) == 0) {
90         /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
91         for (i = 0; i < n; i++)
92             printf("%d ", a[i]);
93         printf("\n");
94     } else {
95         /* Inace, obavestava se korisnik da je doslo do greske */
96         printf("Greska: problem prilikom sortiranja niza!\n");
97     }
98
99     return 0;
}
```

## Rešenje 4.22

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* a) Funkcija koja izracunava broj cvorova stabla */
8 int broj_cvorova(Cvor * koren)
9 {
10    /* Ako je stablo prazno, broj cvorova je nula */
11    if (koren == NULL)
12        return 0;
13
14    /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15     levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
16     zato sto treba racunati i koren */
17    return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18}
19
20 /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
22 {
23    /* Ako je stablo prazno, broj listova je nula */
24    if (koren == NULL)
25        return 0;
26
27    /* Proverava se da li je tekuci cvor list */
28    if (koren->levo == NULL && koren->desno == NULL)
29        /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
30         uvecati broj listova za 1 */
31        return 1;
32
33    /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
34     */
35    return broj_listova(koren->levo) + broj_listova(koren->desno);
36}
37
38 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39 void pozitivni_listovi(Cvor * koren)
40 {
41    /* Slucaj kada je stablo prazno */
42    if (koren == NULL)
43        return;
44
45    /* Ako je cvor list i sadrzi pozitivnu vrednost */
46    if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47        /* Stampa se */
48        printf("%d ", koren->broj);
49
50    /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */

```

## 4 Dinamičke strukture podataka

---

```
52     pozitivni_listovi(koren->levo);
53     pozitivni_listovi(koren->desno);
54 }
55
56 /* d) Funkcija koja izracunava zbir cvorova stabla */
57 int zbir_svih_cvorova(Cvor * koren)
58 {
59     /* Ako je stablo prazno, zbir cvorova je 0 */
60     if (koren == NULL)
61         return 0;
62
63     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
64      elemenata u podstablima */
65     return koren->broj + zbir_svih_cvorova(koren->levo) +
66            zbir_svih_cvorova(koren->desno);
67 }
68
69 /* e) Funkcija koja izracunava najveci element stabla */
70 Cvor *najveci_element(Cvor * koren)
71 {
72     /* Ako je stablo prazno, obustavlja se pretraga */
73     if (koren == NULL)
74         return NULL;
75
76     /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena se
77      nalaze u desnom podstablu */
78
79     /* Ako desnog podstabla nema */
80     if (koren->desno == NULL)
81         /* Najveca vrednost je koren */
82         return koren;
83
84     /* Inace, najveca vrednost se trazi desno */
85     return najveci_element(koren->desno);
86 }
87
88 /* f) Funkcija koja izracunava dubinu stabla */
89 int dubina_stabla(Cvor * koren)
90 {
91     /* Dubina praznog stabla je 0 */
92     if (koren == NULL)
93         return 0;
94
95     /* Izracunava se dubina levog podstabla */
96     int dubina_levo = dubina_stabla(koren->levo);
97
98     /* Izracunava se dubina desnog podstabla */
99     int dubina_desno = dubina_stabla(koren->desno);
100
101    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
102       jer se racuna i koren */
103    return dubina_levo >
```

```

104     dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
105 }
106 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
107 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
108 {
109     /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
110        nivoa */
111
112     /* Ako nema vise cvorova, nema spustanja niz stablo */
113     if (koren == NULL)
114         return 0;
115
116     /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
117        rekurzivnih poziva uvecati broj cvorova za 1 */
118     if (i == 0)
119         return 1;
120
121     /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
122        */
123     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
124         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
125 }
126
127 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
128 void ispis_nivo(Cvor * koren, int i)
129 {
130     /* Ideja je slicna ideji iz prethodne funkcije */
131
132     /* Nema vise cvorova, nema spustanja kroz stablo */
133     if (koren == NULL)
134         return;
135
136     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
137     if (i == 0) {
138         printf("%d ", koren->broj);
139         return;
140     }
141     /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
142        desnom podstablu */
143     ispis_nivo(koren->levo, i - 1);
144     ispis_nivo(koren->desno, i - 1);
145 }
146
147 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
148    stabla */
149 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
150 {
151     /* Ako je stablo prazno, obustavlja se pretraga */
152     if (koren == NULL)
153         return NULL;

```

## 4 Dinamičke strukture podataka

---

```
154 /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
155 if (i == 0)
156     return koren;
157
158 /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
159 Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
160
161 /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
162 Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
163
164 /* Trazi se i vraca maksimum izracunatih vrednosti */
165 if (a == NULL && b == NULL)
166     return NULL;
167 if (a == NULL)
168     return b;
169 if (b == NULL)
170     return a;
171     return a->broj > b->broj ? a : b;
172 }
173
174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
175 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
177     /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)
179         return 0;
180
181     /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
182     if (i == 0)
183         return koren->broj;
184
185     /* Inace, spustanje se nastavlja za jedan nivo nize i traze se sume
186      iz levog i desnog podstabla */
187     return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
189 }
190
191
192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
193    manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
195 {
196     /* Ako je stablo prazno, zbir je nula */
197     if (koren == NULL)
198         return 0;
199
200     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
201        prirode pretrazivackog stabla treba obici i levo i desno
202        podstablo */
203     if (koren->broj <= x)
204         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
205             zbir_manjih_od_x(koren->desno, x);
```

```

206     /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
207      medju njima jedino moze biti onih koje zadovoljavaju uslov */
208      return zbir_manjih_od_x(koren->levo, x);
209  }
210
211 int main(int argc, char **argv)
212 {
213     /* Analiza argumenata komandne linije */
214     if (argc != 3) {
215         fprintf(stderr,
216                 "Greska! Program se poziva sa: ./a.out nivo
217                 broj_za_pretragu\n");
218         return 1;
219     }
220     int i = atoi(argv[1]);
221     int x = atoi(argv[2]);
222
223     /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
224      */
225     Cvor *koren = NULL;
226     int broj;
227     while (scanf("%d", &broj) != EOF) {
228         if (dodaj_u_stablo(&koren, broj) == 1) {
229             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
230             oslobodi_stablo(&koren);
231             return 0;
232         }
233     }
234
235     /* ispisuju se rezultati rada funkcija */
236     printf("Broj cvorova: %d\n", broj_cvorova(koren));
237     printf("Broj listova: %d\n", broj_listova(koren));
238     printf("Pozitivni listovi: ");
239     pozitivni_listovi(koren);
240     printf("\n");
241     printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
242     if (najveci_element(koren) == NULL)
243         printf("Najveci element: ne postoji\n");
244     else
245         printf("Najveci element: %d\n", najveci_element(koren)->broj);
246
247     printf("Dubina stabla: %d\n", dubina_stabla(koren));
248
249     printf("Broj cvorova na %d. nivou: %d\n", i,
250           broj_cvorova_na_itom_nivou(koren, i));
251     printf("Elementi na %d. nivou: ", i);
252     ispis_nivo(koren, i);
253     printf("\n");
254     if (najveci_element_na_itom_nivou(koren, i) == NULL)
255         printf("Nema elemenata na %d. nivou!\n", i);
256     else

```

```
258     printf("Maksimalni element na %d. nivou: %d\n", i,
259            najveci_element_na_itom_nivou(koren, i)->broj);
260
260     printf("Zbir elemenata na %d. nivou: %d\n", i,
261            zbir_cvorova_na_itom_nivou(koren, i));
262     printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
263            zbir_manjih_od_x(koren, x));
264
264     /* Oslobadja se memorija zauzeta stablom */
265     osloboodi_stablo(&koren);
266
268     return 0;
269 }
```

### Rešenje 4.23

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
    /* Dubina praznog stabla je 0 */
    if (koren == NULL)
        return 0;

    /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

    /* Izracunava se dubina desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
       jer se racuna i koren */
    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
{
    /* Ideja je slicna ideji iz prethodne funkcije */
    /* Nema vise cvorova, nema spustanja niz stablo */
    if (koren == NULL)
        return;

    /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
    if (i == 0) {
```

```

36     printf("%d ", koren->broj);
37     return;
38 }
39 /* Inace, vrsti se spustanje za jedan nivo nize i u levom i u desnom
40    podstablu */
41 ispisi_nivo(koren->levo, i - 1);
42 ispisi_nivo(koren->desno, i - 1);
43 }
44
45 /* Funkcija koja ispisuje stablo po nivoima */
46 void ispisi_stablo_po_nivoima(Cvor * koren)
47 {
48     int i;
49
50     /* Prvo se izracunava dubina stabla */
51     int dubina;
52     dubina = dubina_stabla(koren);
53
54     /* Ispisuje se nivo po nivo stabla */
55     for (i = 0; i < dubina; i++) {
56         printf("%d. nivo: ", i);
57         ispisi_nivo(koren, i);
58         printf("\n");
59     }
60 }
61
62 int main(int argc, char **argv)
63 {
64     Cvor *koren;
65     int broj;
66
67     /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
68        uspesnosti dodavanja */
69     koren = NULL;
70     while (scanf("%d", &broj) != EOF) {
71         if (dodaj_u_stablo(&koren, broj) == 1) {
72             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
73             osloboidi_stablo(&koren);
74             return 0;
75         }
76     }
77
78     /* Ispisuje se stablo po nivoima */
79     ispisi_stablo_po_nivoima(koren);
80
81     /* Oslobadja se memorija zauzeta stablom */
82     osloboidi_stablo(&koren);
83
84     return 0;
85 }
```

### Rešenje 4.25

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija koja izracunava dubinu stabla */
8 int dubina_stabla(Cvor * koren)
9 {
10     /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
12         return 0;
13
14     /* Izracunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izracunava se dubina desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
21        jer se racuna i koren */
22     return dubina_levo >
23             dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }
25
26 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
27   stablo */
28 int avl(Cvor * koren)
29 {
30     int dubina_levo, dubina_desno;
31
32     /* Ako je stablo prazno, zaustavlja se brojanje */
33     if (koren == NULL) {
34         return 0;
35     }
36
37     /* Izracunava se dubina levog podstabla korena */
38     dubina_levo = dubina_stabla(koren->levo);
39
40     /* Izracunava se dubina desnog podstabla korena */
41     dubina_desno = dubina_stabla(koren->desno);
42
43     /* Ako je uslov za AVL stablo ispunjen */
44     if (abs(dubina_desno - dubina_levo) <= 1) {
45         /* Racuna se broj AVL cvorova u levom i desnom podstablu i
46            uvecava za jedan iz razloga sto koren ispunjava uslov */
47         return 1 + avl(koren->levo) + avl(koren->desno);
48     } else {
49         /* Inace, racuna se samo broj AVL cvorova u podstablima */
50         return avl(koren->levo) + avl(koren->desno);
51     }
52 }
```

```

51     }
52 }
53 int main(int argc, char **argv)
54 {
55     Cvor *koren;
56     int broj;
57
58     /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
59      uspesnosti dodavanja */
60     koren = NULL;
61     while (scanf("%d", &broj) != EOF) {
62         if (dodaj_u_stablo(&koren, broj) == 1) {
63             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
64             osloboodi_stablo(&koren);
65             return 0;
66         }
67     }
68
69     /* Racuna se i ispisuje broj AVL cvorova */
70     printf("%d\n", avl(koren));
71
72     /* Oslobadja se memorija zauzeta stablom */
73     osloboodi_stablo(&koren);
74
75     return 0;
76 }
77 }
```

### Rešenje 4.26

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
8  brojeva hip. Ideja koja ce biti implementirana u osnovi ima
9  pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
10 desnog podstabla - ako je vrednost u korenu veca od izracunatih
11 vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
12 ce funkcija vracati maksimalne vrednosti iz uocenog podstabala ili
13 vrednost -1 ukoliko se zakljuci da stablo nije hip. */
14 int heap(Cvor * koren)
15 {
16     int max_levo, max_desno;
17
18     /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
19      pozitivan broj */
20     if (koren == NULL) {
21         return 0;
22     }
23
24     max_levo = heap(koren->levi);
25     max_desno = heap(koren->desni);
26
27     if (max_levo >= koren->vrednost &amp; max_desno >= koren->vrednost)
28         return koren->vrednost;
29     else
30         return -1;
31 }
```

## 4 Dinamičke strukture podataka

---

```
22    }
23    /* Ukoliko je stablo list... */
24    if (koren->levo == NULL && koren->desno == NULL) {
25        /* Vraca se njegova vrednost */
26        return koren->broj;
27    }
28    /* Inace... */

29    /* Proverava se svojstvo za levo podstablo */
30    max_levo = heap(koren->levo);

31    /* Proverava se svojstvo za desno podstablo */
32    max_desno = heap(koren->desno);

33    /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
34     ni celo stablo */
35    if (max_levo == -1 || max_desno == -1) {
36        return -1;
37    }

38    /* U suprotnom proverava se da li svojstvo vazi za uoceni cvor */
39    if (koren->broj > max_levo && koren->broj > max_desno) {
40        /* Ako vazi, vraca se vrednost korena */
41        return koren->broj;
42    }

43    /* U suprotnom zaključuje se da stablo nije hip */
44    return -1;
45}

46 int main(int argc, char **argv)
47 {
48     Cvor *koren;
49     int hip_indikator;

50     /* Kreira se stablo prema zadatoj slici */
51     koren = NULL;
52     koren = napravi_cvor(100);
53     koren->levo = napravi_cvor(19);
54     koren->levo->levo = napravi_cvor(17);
55     koren->levo->levo->levo = napravi_cvor(2);
56     koren->levo->levo->desno = napravi_cvor(7);
57     koren->levo->desno = napravi_cvor(3);
58     koren->desno = napravi_cvor(36);
59     koren->desno->levo = napravi_cvor(25);
60     koren->desno->desno = napravi_cvor(1);

61     /* Poziva se funkcija kojom se proverava da li je stablo hip */
62     hip_indikator = heap(koren);

63     /* Ispisuje se rezultat */
64     if (hip_indikator == -1) {
```

```
74     printf("Zadato stablo nije hip!\n");
75 } else {
76     printf("Zadato stablo je hip!\n");
77 }
78 /* Oslobadja se memorija zauzeta stablom */
79 oslobodi_stablo(&koren);
80
81     return 0;
82 }
```



# Dodatak A

## Ispitni rokovi

### A.1 Praktični deo ispita, jun 2015.

**Zadatak A.1** Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu za grešku ispisati vrednost  $-1$  i prekinuti izvršavanje programa.

*Test 1*

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
5
Programiranje
Matematika
12345
dInAmicNArEc
Ispit

IZLAZ:
Ispit
Matematika
Programiranje
```

*Test 2*

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
2
maksimalano
poena

IZLAZ:
```

## A Ispitni rokovi

---

Test 3

```
|| POKRETANJE: ./a.out ulaz.txt
|| DATOTEKA ULAZ.TXT NE POSTOJI
|| IZLAZ ZA GREŠKU:
|| -1
```

Test 4

```
|| POKRETANJE: ./a.out
|| IZLAZ ZA GREŠKU:
|| -1
```

[Rešenje A.1]

**Zadatak A.2** Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumiraj_n (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 `sumiraj_n` za broj  $n$  i tako kreirano stablo. U slučaju greške na standardni izlaz za greške ispisati  $-1$ .

NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka [4.14](#).

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 A.2]

**Zadatak A.3** Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice  $A$ , a zatim i elementi matrice  $A$ . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost  $-1$  na standardni izlaz za greške.

*Test 1*

```
||| ULAZ:
 4 5
1 2 3 4 5
-1 2 -3 4 -5
-5 -4 -3 -2 1
-1 0 0 0 0
IZLAZ:
 0
```

*Test 2*

```
||| ULAZ:
 2 3
 0 0 -5
 1 2 -4
IZLAZ:
 2
```

*Test 3*

```
||| ULAZ:
 -2
IZLAZ ZA GREŠKU:
 -1
```

[Rešenje A.3]

## A.2 Praktični deo ispita, jul 2015.

**Zadatak A.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. Prepostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

*Test 1*

```
||| POKRETANJE: ./a.out ulaz.txt test
ULAZ.TXT
  Ovo je test primer.
  U njemu se rec test javlja
  vise puta. testtesttest
IZLAZ:
 5
```

*Test 2*

```
||| POKRETANJE: ./a.out
IZLAZ ZA GREŠKU:
 -1
```

*Test 3*

```
||| POKRETANJE: ./a.out ulaz.txt foo
DATOTEKA ULAZ.TXT NE POSTOJI
IZLAZ ZA GREŠKU:
 -1
```

*Test 4*

```
||| POKRETANJE: ./a.out ulaz.txt .
DATOTEKA ULAZ.TXT JE PRAZNA
IZLAZ:
 0
```

## A Ispitni rokovi

---

[Rešenje A.4]

**Zadatak A.5** Na početku datoteke `trouglovi.txt` nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitava trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati  $-1$  na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Test 1

```
|| TROUGLOVI.TXT
|| 4
|| 0 0 0 1.2 1 0
|| 0.3 0.3 0.5 0.5 0.9 1
|| -2 0 0 0 0 1
|| -2 0 0 0 0 1
|| IZLAZ:
|| 2 0 2 2 -1 -1
|| -2 0 0 0 0 1
|| 0 0 0 1.2 1 0
|| 0.3 0.3 0.5 0.5 0.9 1
```

Test 2

```
|| TROUGLOVI.TXT
|| 3
|| 1.2 3.2 1.1 4.3
|| IZLAZ ZA GREŠKU:
|| -1
```

Test 3

```
|| DATOTEKA TROUGLOVI.TXT NE POSTOJI
|| IZLAZ ZA GREŠKU:
|| -1
```

Test 4

```
|| TROUGLOVI.TXT
|| 0
|| IZLAZ:
```

[Rešenje A.5]

**Zadatak A.6** Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Napisati funkciju

```
int prebroj_n(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.

NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka 4.14.

Test 1

```
|| ULAZ:
|| 1 5 3 6 1 4 7 9
|| IZLAZ:
|| 1
```

Test 2

```
|| ULAZ:
|| 2 5 3 6 1 0 4 7 9
|| IZLAZ:
|| 2
```

Test 3

```
|| ULAZ:
|| 0 4 2 5
|| IZLAZ:
|| 0
```

### A.3 Praktični deo ispita, septembar 2015.

Test 4

ULAZ:	3
IZLAZ:	0

Test 5

ULAZ:	-1 4 5 1 7
IZLAZ:	0

[Rešenje A.6]

## A.3 Praktični deo ispita, septembar 2015.

**Zadatak A.7** Sa standardnog ulaza se učitavaju neoznačeni celi brojevi  $x$  i  $n$ . Na standardni izlaz ispisati neoznačen ceo broj koji se dobija od broja  $x$  kada se njegov binarni zapis rotira za  $n$  mesta udesno (na primer, ako je binarni zapis broja  $x$  jednak 000000000000000000000000000000001111, i ako je  $n = 1$  tada na standardni izlaz treba ispisati neočačen broj čiji je binarni zapis jednak 10000000000000000000000000000000111).

Test 1

ULAZ:	6 1
IZLAZ:	3

Test 2

ULAZ:	15 3
IZLAZ:	3758096385

Test 3

ULAZ:	31 100
IZLAZ:	4026531841

Test 4

ULAZ:	4 0
IZLAZ:	4

Test 5

ULAZ:	0 5
IZLAZ:	0

[Rešenje A.7]

**Zadatak A.8** Data je biblioteka za rad sa listama. Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

## A Ispitni rokovi

---

Test 1

ULAZ:	
	1 2 3 4 5
IZLAZ:	1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

Test 2

ULAZ:	
	12
IZLAZ:	12.00

Test 3

ULAZ:	
	prazna lista
IZLAZ:	

Test 4

ULAZ:	
	13.3 15.8
IZLAZ:	13.30 14.55

[Rešenje A.8]

**Zadatak A.9** Sa standardnog ulaza se učitava dimenzija  $n$  kvadratne celobrojne matrice  $A$  ( $n > 0$ ), a zatim i elementi matrice  $A$ . Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje su sume brojeva u svim redovima i kolonama međusobno jednake). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati "-". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1 na standardni izlaz za greške. NAPOMENA: Rešenje koristi biblioteku za rad sa matricama iz zadatka 2.19.

Test 1

ULAZ:	
	4
	1 2 3 4
	2 1 4 3
	3 4 2 1
	4 3 1 2
IZLAZ:	10

Test 2

ULAZ:	
	3
	1 1 1
	1 1 1
	1 1 1
IZLAZ:	3

Test 3

ULAZ:	
	2
	1 1
	2 2
IZLAZ:	-

Test 4

ULAZ:	
	2
	1 2
	1 2
IZLAZ:	-

Test 5

ULAZ:	
	1
	5
IZLAZ:	5

Test 6

ULAZ:	
	0
IZLAZ ZA GREŠKU:	-1

[Rešenje A.9]

## A.4 Praktični deo ispita, januar 2016.

**Zadatak A.10** Napisati funkciju `unsigned int zamena(unsigned int x)` koja u datom broju `x` menja mesta prvom i četvrtom bajtu. Prvi bajt je sačinjen od 8 bitova najmanje težine. Napisati program koji testira funkciju `zamena` za ceo broj `x` unet sa standardnog ulaza. U slučaju da je uneti broj `x` negativan na standardni izlaz za greške program ispisuje `-1`, a inače ispisaje na standardni izlaz broj dobijen primenom funkcije `zamena`.

Test 1

ULAZ:	285278344
IZLAZ:	2281766929

Test 2

ULAZ:	1024
IZLAZ:	1024

Test 3

ULAZ:	1
IZLAZ:	16777216

Test 4

ULAZ:	0
IZLAZ:	0

Test 5

ULAZ:	-63
IZLAZ ZA GREŠKU:	-1

[Rešenje [A.10](#)]

**Zadatak A.11** Data je biblioteka za rad sa binarnim pretraživackim stablima celih brojeva. Napisati funkciju `int najduzi_put (Cvor * koren)` koja za dato stablo izračunava dužinu najdužeg puta od korena do nekog lista. Ako je stablo prazno, povratna vrednost funkcije je `-1`. Ako stablo ima samo koren dužina najdužeg puta je `0`. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

NAPOMENA: Koristiti biblioteku za rad s binarnim pretraživackim stablima celih brojeva iz zadatka [4.14](#).

Test 1

ULAZ:	10 5 15 3 2 4 30 12 14 13
IZLAZ:	4

Test 2

ULAZ:	3
IZLAZ:	0

## A Ispitni rokovi

---

Test 3

ULAZ:	
5 6	
IZLAZ:	
1	

Test 4

ULAZ:	
7 5 8	
IZLAZ:	
1	

Test 5

ULAZ:	
5 7 8	
IZLAZ:	
2	

[Rešenje A.11]

**Zadatak A.12** Sa standardnog ulaza zadaje se ime datoteke u kojoj se nalazi matrica realnih brojeva jednostrukе tačnosti i jedan realan broj. Napisati program koji iz datoteke učitava matricu realnih brojeva, a zatim pronađe i na standardni izlaz ispisuje indeks vrste matrice u kojoj se uneti realan broj pojavljuje najmanje puta. Ako postoji više takvih vrsta, ispisati indeks prve vrste. U datoteci su prvo navedena dva cela broja koja predstavljaju dimenzije matrice, redom broj vrsta i broj kolona, a zatim i elementi matrice vrstu po vrstу. U slučaju greške ispisati -1 na standardni izlaz za greške. Prepostaviti da ime datoteke neće biti duže od 30 karaktera. NAPOMENA: *U zadatku treba koristiti dinamičku alokaciju matrice.*

Test 1

ULAZ:	brojevi.txt 0
BROJEVI.TXT	
4 4	
0 0 0 1.2	
1 0 0.3 0.3	
0.5 0.5 0.9 -1	
-2 0 0 0	
IZLAZ:	
2	

Test 2

ULAZ:	in.txt 2
IN.TXT	
3 3	
2 0 2	
-1 2 -1	
2 5 3	
IZLAZ:	
1	

Test 3

ULAZ:	matrica.txt 7
MATRICA.TXT	
3 2	
1.1 -5.31	
-3.7 35.24	
1.4 2.09	
IZLAZ:	
0	

Test 4

ULAZ:	brojevi.txt 12
BROJEVI.TXT	
IZLAZ ZA GREŠKU:	
-1	

[Rešenje A.12]

## A.5 Rešenja

### Rešenje A.1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAKS 50

5 /* Funkcija vrši dinamicku alokaciju memorije potrebne n linija tj.
6   n niski od kojih nijedna nije duza od MAKS karaktera. */
7 char **alociranje_memorije(int n)
8 {
9     char **linije = NULL;
10    int i, j;
11    /* Alocira se prostor za niz vrsti matrice */
12    linije = (char **) malloc(n * sizeof(char *));
13    /* U slučaju neuspesnog otvaranja ispisuje se -1 na stderr i
14      program završava. */
15    if (linije == NULL)
16        return NULL;
17    /* Alocira se prostor za svaku vrstu matrice. Niska nije duza od
18      MAKS karaktera, a 1 se dodaje zbog terminirajuće nule. */
19    for (i = 0; i < n; i++) {
20        linije[i] = malloc((MAKS + 1) * sizeof(char));
21        /* Ako alokacija nije prosla uspesno, oslobadjavaju se svi
22          prethodno alocirani resursi, i povratna vrednost je NULL */
23        if (linije[i] == NULL) {
24            for (j = 0; j < i; j++) {
25                free(linije[j]);
26            }
27            free(linije);
28            return NULL;
29        }
30    }
31    return linije;
32 }
33
34 /* Funkcija oslobadjava dinamicki alociranu memoriju */
35 char **oslobadjanje_memorije(char **linije, int n)
36 {
37     int i;
38     /* Oslobadja se prostor rezervisan za svaku vrstu */
39     for (i = 0; i < n; i++) {
40         free(linije[i]);
41     }
42     /* Oslobadja se memorija za niz pokazivaca na vrste */
43     free(linije);

44     /* Matrica postaje prazna, tj. neallocirana */
45 }
```

```
    return NULL;
48 }

50 int main(int argc, char *argv[])
{
52     FILE *ulaz;
53     char **linije;
54     int i, j, n;

56     /* Proverava argumenata komandne linije. */
57     if (argc != 2) {
58         fprintf(stderr, "-1\n");
59         exit(EXIT_FAILURE);
60     }

62     /* Otvaranje datoteke cije ime je navedeno kao argument komandne
63      linije neposredno nakon imena programa koji se poziva. U slucaju
64      neuspesnog otvaranja ispisuje se -1 na stderr i program zavrsava
65      izvrsavanje. */
66     ulaz = fopen(argv[1], "r");
67     if (ulaz == NULL) {
68         fprintf(stderr, "-1\n");
69         exit(EXIT_FAILURE);
70     }
71     /* Ucitavanje broja linija. */
72     fscanf(ulaz, "%d", &n);

73     /* Alociranje memorije na osnovu ucitanog broja linija. */
74     linije = alociranje_memorije(n);

75     /* U slucaju neuspesne alokacije ispisuje se -1 na stderr i program
76      zavrsava. */
77     if (linije == NULL) {
78         fprintf(stderr, "-1\n");
79         exit(EXIT_FAILURE);
80     }

81     /* Ucitavanje svih n linija iz datoteke. */
82     for (i = 0; i < n; i++) {
83         fscanf(ulaz, "%s", linije[i]);
84     }

85     /* Ispisivanje u odgovarajucem poretku ucitane linije koje
86      zadovoljavaju kriterijum. */
87     for (i = n - 1; i >= 0; i--) {
88         if (isupper(linije[i][0])) {
89             printf("%s\n", linije[i]);
90         }
91     }
92     /* Oslobadjanje memorije koja je dinamicki alocirana. */
93     linije = oslobadjanje_memorije(linije, n);
94 }
```

```

100    /* Zatvaranje datoteku. */
101    fclose(ulaz);
102
103    exit(EXIT_SUCCESS);
104 }
```

### Rešenje A.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

*main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 int sumiraj_n(Cvor * koren, int n)
6 {
7     /* Ako je stablo prazno, suma je nula */
8     if (koren == NULL)
9         return 0;
10    /* Inace ... */
11    /* Ako je n jednako nula, vraca se broj iz korena */
12    if (n == 0)
13        return koren->broj;
14    /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
15       kao i suma na (n-1)-om nivou u desnom podstablu i vraca se zbir
16       te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
17       n-tom nivou u pocetnom stablu */
18    return sumiraj_n(koren->levo, n - 1) + sumiraj_n(koren->desno, n -
19               1);
20}
21
22 int main()
23 {
24     Cvor *koren = NULL;
25     int n;
26     int nivo;
27
28     /* Ucitava se vrednost nivoa */
29     scanf("%d", &nivo);
30     while (1) {
31         scanf("%d", &n);
32         /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
33         if (n == 0)
34             break;
35         /* Ako nije, dodaje se procitani broj u stablo. */
36         if (dodaj_u_stablo(&koren, n) == 1) {
37             fprintf(stderr, "-1\n", n);
```

## A Ispitni rokovi

---

```
38         oslobodi_stablo(&koren);
39         exit(EXIT_FAILURE);
40     }
41
42     /* Ispisuje se rezultat rada trazene funkcije */
43     printf("%d\n", sumiraj_n(koren, nivo));
44
45     /* Oslobadja se memorija */
46     oslobodi_stablo(&koren);
47
48     exit(EXIT_SUCCESS);
49 }
```

### Rešenje A.3

```
#include <stdio.h>
#include <stdlib.h>
#define MAKS 50

/* Funkcija ucitava elemenate matrice sa standardnog ulaza */
void ucitaj_matricu(int m[][MAKS], int v, int k)
{
    int i, j;
    for (i = 0; i < v; i++) {
        for (j = 0; j < k; j++) {
            scanf("%d", &m[i][j]);
        }
    }
}

/* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
   koja ima v vrsta */
int broj_negativnih_u_koloni(int m[][MAKS], int v, int k)
{
    int broj_negativnih = 0;
    int i;
    for (i = 0; i < v; i++) {
        if (m[i][k] < 0)
            broj_negativnih++;
    }
    return broj_negativnih;
}

/* Funkcija vraca indeks kolone matrice m u kojoj se nalazi najvise
   negativnih elemenata */
int maks_indeks(int m[][MAKS], int v, int k)
{
    int i, j;
    int broj_negativnih;
    /* Inicijalizacija na nulu indeksa kolone sa maksimalnim brojem
```

```

36     negativnih (maks_indeks_kolone), kao i maksimalnog broja
37     negativnih elemenata u nekoj koloni (maks_broj_negativnih) */
38 int maks_indeks_kolone = 0;
39 int maks_broj_negativnih = 0;
40
41 for (j = 0; j < k; j++) {
42     /* Racuna se broj negativnih u j-oj koloni */
43     broj_negativnih = broj_negativnih_u_koloni(m, v, j);
44     /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
45        maksimalnog, azurira se trenutni maksimalni broj negativnih
46        kao i indeks kolone sa maksimalnim brojem negativnih */
47     if (maks_broj_negativnih < broj_negativnih) {
48         maks_indeks_kolone = j;
49         maks_broj_negativnih = broj_negativnih;
50     }
51 }
52 return maks_indeks_kolone;
53
54 int main()
55 {
56     int m[MAKS][MAKS];
57     int v, k;
58
59     /* Ucitavanje broja vrsta matrice */
60     scanf("%d", &v);
61
62     /* Provera validnosti broja vrsta */
63     if (v < 0 || v > MAKS) {
64         fprintf(stderr, "-1\n");
65         exit(EXIT_FAILURE);
66     }
67
68     /* Ucitavanje broja kolona matrice */
69     scanf("%d", &k);
70
71     /* Provera validnosti broja kolona */
72     if (k < 0 || k > MAKS) {
73         fprintf(stderr, "-1\n");
74         exit(EXIT_FAILURE);
75     }
76     /* Ucitavanje elemenata matrice */
77     ucitaj_matricu(m, v, k);
78
79     /* Pronalazenje kolone koja sadrzi najveci broj negativnih
80        elemenata i ispisivanje trazenog rezultata */
81     printf("%d\n", maks_indeks(m, v, k));
82
83     exit(EXIT_SUCCESS);
84 }

```

Rešenje A.4

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAKS 128
5
6 int main(int argc, char **argv)
7 {
8     FILE *f;
9     int brojac = 0;
10    char linija[MAKS], *p;
11
12    /* Provera da li je broj argumenata komandne linije 3 */
13    if (argc != 3) {
14        fprintf(stderr, "-1\n");
15        exit(EXIT_FAILURE);
16    }
17    /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18       linije */
19    if ((f = fopen(argv[1], "r")) == NULL) {
20        fprintf(stderr, "-1\n");
21        exit(EXIT_FAILURE);
22    }
23    /* Ucitavanje iz otvorene datoteke - liniju po liniju */
24    while (fgets(linija, MAKS, f) != NULL) {
25        p = linija;
26        while (*p) {
27            p = strstr(p, argv[2]);
28
29            /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
30            if (p == NULL)
31                break;
32            /* Inace se uvecava brojac */
33            brojac++;
34            /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
35               linije nakon uocene podniske */
36            p = p + strlen(argv[2]);
37        }
38    }
39
40    /* Zatvaranje datoteke */
41    fclose(f);
42
43    /* Ispisivanje vrednosti brojaca */
44    printf("%d\n", brojac);
45
46    exit(EXIT_SUCCESS);
47 }
```

## Rešenje A.5

```

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

5 /* Struktura trougao */
6 typedef struct _trougao {
7     double xa, ya, xb, yb, xc, yc;
8 } trougao;

10 /* Funkcija racuna duzinu duzi */
11 double duzina(double x1, double y1, double x2, double y2)
12 {
13     return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
14 }

16 /* Funkcija racuna povrsinu trougla koristeci Heronov obrazac */
17 double povrsina(trougao t)
18 {
19     /* Racunanje duzina stranica trougla */
20     double a = duzina(t.xb, t.yb, t.xc, t.yc);
21     double b = duzina(t.xa, t.ya, t.xc, t.yc);
22     double c = duzina(t.xa, t.ya, t.xb, t.yb);
23     /* Poluobim */
24     double s = (a + b + c) / 2;
25     /* Primena Heronovog obrasca */
26     return sqrt(s * (s - a) * (s - b) * (s - c));
27 }

29 /* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
30 prvi argument funkcije manja od povrsine trougla koji je drugi
31 element funkcije funkcija vraca 1, ukoliko je veca -1, a ukoliko
32 su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
33 napisana tako da se moze proslediti funkciji qsort da se niz
34 trouglova sortira po povrsini opadajuce. */
35 int poredi(const void *a, const void *b)
36 {
37     trougao x = *(trougao *) a;
38     trougao y = *(trougao *) b;
39     double xp = povrsina(x);
40     double yp = povrsina(y);
41     if (xp < yp)
42         return 1;
43     if (xp > yp)
44         return -1;
45     return 0;
46 }
47
48 int main()
49 {
    FILE *f;

```

## A Ispitni rokovi

---

```
51 int n, i;
52 trougao *niz;
53
54 /* Otvaranje datoteke ciji je naziv trouglovi.txt */
55 if ((f = fopen("trouglovi.txt", "r")) == NULL) {
56     fprintf(stderr, "-1\n");
57     exit(EXIT_FAILURE);
58 }
59
60 /* Ucitavanje podataka o broju trouglova iz datoteke */
61 if (fscanf(f, "%d", &n) != 1) {
62     fprintf(stderr, "-1\n");
63     exit(EXIT_FAILURE);
64 }
65
66 /* Dinamicka alokacija memotije za niz trouglova duzine n */
67 if ((niz = malloc(n * sizeof(trougao))) == NULL) {
68     fprintf(stderr, "-1\n");
69     exit(EXIT_FAILURE);
70 }
71
72 /* Ucitavanje podataka u niz iz otvorene datoteke */
73 for (i = 0; i < n; i++) {
74     if (fscanf(f, "%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
75                &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
76     {
77         fprintf(stderr, "-1\n");
78         exit(EXIT_FAILURE);
79     }
80 }
81
82 /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
83 poredi */
84 qsort(niz, n, sizeof(trougao), &poredi);
85
86 /* Ispisivanje sortiranog niza na standardni izlaz */
87 for (i = 0; i < n; i++)
88     printf("%g %g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
89            niz[i].yb, niz[i].xc, niz[i].yc);
90
91 /* Oslobadjanje dinamicki alocirane memorije */
92 free(niz);
93
94 /* Zatvranje datoteke */
95 fclose(f);
96
97 exit(EXIT_SUCCESS);
98 }
```

### Rešenje A.6

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

*main.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 /* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
6    stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa u
7    stablo, a inace 0. */
8 int ucitaj_stablo(Cvor ** koren)
{
9
10    *koren = NULL;
11    int x;
12    /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
13       dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
14       je i povratna vrednost iz funkcije ucitaj_stablo 1. */
15    while (scanf("%d", &x) == 1)
16        if (dodaj_u_stablo(koren, x) == 1)
17            return 1;
18    return 0;
19}
20
21 /* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
22 int prebroj_n(Cvor * koren, int n)
{
23
24    /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
25       negativan broj, na tom nivou nema cvorova (rezultat je nula). */
26    if (koren == NULL || n < 0)
27        return 0;
28    /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima jednog
29       potomka funkcija vraca 1, inace 0 */
30    if (n == 0) {
31        if (koren->levo == NULL && koren->desno != NULL)
32            return 1;
33        if (koren->levo != NULL && koren->desno == NULL)
34            return 1;
35        return 0;
36    }
37    /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
38       (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
39       levog podstabla */
40    return prebroj_n(koren->levo, n - 1) + prebroj_n(koren->desno, n -
41               1);
42}
43
44 int main()
{
    Cvor *koren;

```

## A Ispitni rokovi

---

```
46     int n;
47     scanf("%d", &n);
48
49     /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
50      oslobođa se alocirana memorija i izlazi se iz programa. */
51     if (ucitaj_stablo(&koren) == 1) {
52         fprintf(stderr, "-1\n");
53         osloboodi_stablo(&koren);
54         exit(EXIT_FAILURE);
55     }
56
57     /* Ispisivanje rezultata */
58     printf("%d\n", prebroj_n(koren, n));
59
60     /* Oslobadjanje dinamicki alociranog stabla */
61     osloboodi_stablo(&koren);
62
63     exit(EXIT_SUCCESS);
64 }
```

### Rešenje A.7

```
#include <stdio.h>
1
2     /* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
3        zapis argumenta x rotira za n mesta udesno */
4     unsigned int rotiraj(unsigned int x, unsigned int n)
5     {
6         int i;
7         unsigned int maska = 1;
8         /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
9            izgledala: 000...00001111 */
10        /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
11           nastavku je drugi nacin. */
12        for (i = 1; i < n; i++)
13            maska = (maska << 1) | 1;
14        /* Kada se x poremeli za n mesta udesno x >> n, poslednjih n bitova
15           binarne reprezentacije broja x ce "ispasti". Za rotaciju je
16           potrebno da se tih n bitova postavi na pocetak broja. Kreirana
17           maska omogucava da se tih n bitova izdvoji sa (maska & x), a
18           zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta uлево
19           tih n bitova postavlja na pocetak. Primenom logicke disjunkcije
20           dobija se rotirani broj. */
21        return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
22    }
23
24    int main()
25    {
26        unsigned int x, n;
27
28        /* Ucitavanje brojeva sa standardnog ulaza */
```

```

30    scanf("%u%u", &x, &n);
31
32    /* Ispisivanje rezultata */
33    printf("%u\n", rotiraj(x, n));
34    return 0;
}

```

### Rešenje A.8

*liste.h*

```

1 #ifndef _LISTE_H_
2 #define _LISTE_H_ 1
3
4 /* Struktura koja predstavlja cvor liste */
5 typedef struct cvor {
6     double vrednost;
7     struct cvor *sledeci;
8 }
9 Cvor;
10
11 /* Pomocna funkcija koja kreira cvor. */
12 Cvor * napravi_cvor(double broj);
13
14 /* Funkcija oslobođa dinamicku memoriju zauzetu za elemente
15    liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
16 void osloboodi_listu(Cvor ** adresa_glave);
17
18 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
19    ili NULL kao je lista prazna */
20 Cvor * pronadji_poslednji(Cvor * glava);
21
22 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
23    greske pri alokaciji memorije,inace vraca 0. */
24 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);
25
26 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
27    liste. */
28 void ispisi_listu(Cvor * glava);
29
30#endif

```

*liste.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "liste.h"
4

```

```
/* Pomocna funkcija koja kreira cvor. */
6 Cvor *napravi_cvor(double broj)
{
8     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10    if (novi == NULL)
11        return NULL;
12    /* inicializacija polja u novom cvoru */
13    novi->vrednost = broj;
14    novi->sledeci = NULL;
15    return novi;
16}

18 /* Funkcija oslobođa dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glove. */
20 void osloboodi_listu(Cvor ** adresa_glove)
{
22     Cvor *pomocni = NULL;
23     while (*adresa_glove != NULL) {
24         pomocni = (*adresa_glove)->sledeci;
25         free(*adresa_glove);
26         *adresa_glove = pomocni;
27     }
28}
29

32 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
34 Cvor *pronadji_poslednji(Cvor * glava)
{
35     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slučaju
       funkcija vraca NULL. */
36     if (glava == NULL)
37         return NULL;
38     while (glava->sledeci != NULL)
39         glava = glava->sledeci;
40     return glava;
41}

46 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
48 int dodaj_na_kraj_liste(Cvor ** adresa_glove, double broj)
{
50     Cvor *novi = napravi_cvor(broj);
51     if (novi == NULL)
52         return 1;
53     if (*adresa_glove == NULL) {
54         *adresa_glove = novi;
55         return 0;
56     }
```

```
    }
58 Cvor *poslednji = pronadji_poslednji(*adresa_glave);
  poslednji->sledeci = novi;
60 }
62 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
   */
64 void ispisi_listu(Cvor * glava)
{
66     for ( ; glava != NULL; glava = glava->sledeci)
67         printf("%.2lf ", glava->vrednost);
68     putchar('\n');
69 }
70 }
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "liste.h"

/* Funkcija umece novi cvor iza tekuceg u listi */
void dodaj_iza(Cvor * tekuci, Cvor * novi)
{
    /* Novi cvor se dodaje iza tekuceg cvora. */
    novi->sledeci = tekuci->sledeci;
    tekuci->sledeci = novi;
}

/* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
   Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
   0. */
int dopuni_listu(Cvor ** adresa_glave)
{
    Cvor *tekuci;
    Cvor *novi;
    double aritmeticka_sredina;
    /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
    if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
        return 1;
    /* Promenljiva tekuci se inicijalizuje da pokazuje na pocetni
       cvor */
    tekuci = *adresa_glave;
    /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
       vrednosti u susednim cvorovima i kreira cvor sa tom vrednoscu. U
       slucaju neupeake alokacije novog cvora, funkcija vraca 1. Inace,
       novi cvor se umece izmedju dva cvora za koje racunata
       aritmeticka sredina */
    while (tekuci->sledeci != NULL) {
```

## A Ispitni rokovi

---

```
34     aritmeticka_sredina =
35         ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
36     novi = napravi_cvor(aritmeticka_sredina);
37     if (novi == NULL)
38         return 1;
39     /* Poziva se funkcija koja umece novi cvor iza tekuceg cvora */
40     dodaj_iza(tekuci, novi);
41     /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
42      cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
43      polazne liste */
44     tekuci = tekuci->sledeci;
45     tekuci = tekuci->sledeci;
46 }
47 return 0;
48 }

50 int main()
{
51     Cvor *glava = NULL;
52     double broj;
53
54     /* Ucitavanje se vrsi do kraja ulaza. Elementi se dodaju na kraj
55      liste! */
56     while (scanf("%lf", &broj) > 0) {
57         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
58            memorije za nov cvor. Memoriju alociranu za cvorove liste
59            treba osloboziti. */
60         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
61             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
62             oslobodi_listu(&glava);
63             exit(EXIT_FAILURE);
64         }
65     }
66
67     /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
68       onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
69       alociranu za cvorove liste treba osloboziti. */
70     if (dopuni_listu(&glava) == 1) {
71         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
72         oslobodi_listu(&glava);
73         exit(EXIT_FAILURE);
74     }
75
76     /* Ispisivanje liste */
77     ispisi_listu(glava);
78
79     /* Oslobozjanje liste */
80     oslobodi_listu(&glava);
81
82     exit(EXIT_SUCCESS);
83 }
```

## Rešenje A.9

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 /* Funkcija racuna zbir elemenata v-te vrste */
6 int zbir_vrste(int **M, int n, int v)
7 {
8     int j, zbir = 0;
9     for (j = 0; j < n; j++)
10        zbir += M[v][j];
11    return zbir;
12 }
13
14 /* Funkcija racuna zbir elemenata k-te kolone */
15 int zbir_kolone(int **M, int n, int k)
16 {
17     int i, zbir = 0;
18     for (i = 0; i < n; i++)
19        zbir += M[i][k];
20    return zbir;
21 }
22
23 /* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
24 argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
25 Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
26 kolone ukoliko je kvadrat magican. */
27 int magicni_kvadrat(int **M, int n, int *zbirmagicnog)
28 {
29     int i, j;
30     int zbir = 0, zbir_pom;
31     /* Promenljivu zbir inicijalizujemo na zbir 0-te vreste */
32     zbir = zbir_vrste(M, n, 0);
33
34     /* Racunaju se zbirovi u ostalim vrstama i ako neki razlikuje od
35      vrednosti promeljive zbir funkcija vraca 1 */
36     for (i = 1; i < n; i++) {
37         zbir_pom = zbir_vrste(M, n, i);
38         if (zbir_pom != zbir)
39             return 0;
40     }
41     /* Racunaju se zbirovi u svim kolonama i ako neki razlikuje od
42      vrednosti promeljive zbir funkcija vraca 1 */
43     for (j = 0; j < n; j++) {
44         zbir_pom = zbir_kolone(M, n, j);
45         if (zbir_pom != zbir)
46             return 0;
47     }
48     /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
49      vresnost u zbirmagicnog i funkcija vraca 1 */
50     *zbirmagicnog = zbir;

```

## A Ispitni rokovi

---

```
    return 1;
52 }

54 int main()
{
55     int n, i, j;
56     int **matrica = NULL;
57     int zbir = 0, zbirmagicnog = 0;
58     scanf("%d", &n);

59     /* Provera da li je n strogo pozitivan */
60     if (n <= 0) {
61         printf("-1\n");
62         exit(EXIT_FAILURE);
63     }

64     /* Dinamicka alokacija matrice dimenzije nxn */
65     matrica = alociraj_matricu(n);
66     if (matrica == NULL) {
67         printf("-1\n");
68         exit(EXIT_FAILURE);
69     }

70     /* Ucitavanje elemenata matrice sa standardnog ulaza */
71     ucitaj_matricu(matrica, n);

72     /* Ispisivanje rezultata na osnovu funkcije magicni_kvadrat */
73     if (magicni_kvadrat(matrica, n, &zbirmagicnog)) {
74         printf("%d\n", zbir);
75     } else {
76         printf("-\n");
77     }

78     /* Oslobođjanje dinamicki alocirane memorije */
79     matrica = dealociraj_matricu(matrica, n);

80     exit(EXIT_SUCCESS);
81 }
```

### Rešenje A.10

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

4 /* Funkcija u datom broju x menja mesta prvom i četvrtom bajtu */
5 unsigned int zamena(unsigned int x){
6     /* Deklaracija promenljivih za odgovarajuće maske i pomocne
7      * promenljive*/
8     unsigned maska_prvi_bajt, maska_cetvrti_bajt;
9     unsigned maska_prvi_bajt_komplement, maska_cetvrti_bajt_komplement;
10    unsigned prvi_bajt, cetvrti_bajt;
11    unsigned i;
```

```

12  /*
13   * Maska prvi bajt odgovara broju cija je binarna reprezentacija
14   * 00000....000001111111 (8 bitova najmanje tezine su jedinice,
15   * a ostalo su nule) */
16  maska_prvi_bajt = 1;
17  for(i=0;i<7;i++)
18      maska_prvi_bajt=maska_prvi_bajt<<1|1;

19  /*
20   * Maska cetvrti bajt odgovara broju cija je binarna reprezentacija
21   * 111111100000....00000 (8 bitova najveće tezine su jedinice,
22   * a ostalo su nule) */
23  maska_cetvrti_bajt=maska_prvi_bajt<<(sizeof(unsigned)*8-8);

24  /*
25   * Primenom operatorka ~ na maska_prvi_bajt dobija se broj cija je
26   * binarna reprezentacija 1111....111100000000 (8 bitova najmanje
27   * tezine su nule, a ostalo su jedinice) */
28  maska_prvi_bajt_komplement=~maska_prvi_bajt;
29  /*
30   * Primenom operatorka ~ na maska_prvi_bajt dobija se broj cija je
31   * binarna reprezentacija 000000001111....1111 (8 bitova najveće
32   * tezine su nule, a ostalo su jedinice) */
33  maska_cetvrti_bajt_komplement=~maska_cetvrti_bajt;

34  /*
35   * U promenljivu prvi_bajt smestamo broj koji se dobija kada se
36   * bitovi prvog bajta broja x pomere uлево, tako da budu na
37   * poziciji cetvrtoog bajta */
38  prvi_bajt=(maska_prvi_bajt&x)<<(sizeof(unsigned)*8-8);
39  /*
40   * U promenljivu cetvrti_bajt smestamo broj koji se dobija kada se
41   * bitovi cetvrtoog bajta broja x pomere uдесно, tako da budu na
42   * poziciji prvog bajta */
43  cetvrti_bajt=(maska_cetvrti_bajt&x)>>(sizeof(unsigned)*8-8);

44  /*
45   * Na nule se postavlja 8 bitova najmanje tezine, a ostali bitovi
46   * ostaju nepromenjeni */
47  x=x&maska_prvi_bajt_komplement;

48  /*
49   * Na nule se postavlja 8 bitova najveće tezine, a ostali bitovi
50   * ostaju nepromenjeni */
51  x=x&maska_cetvrti_bajt_komplement;

52  /*
53   * Na bitove na poziciji cetvrtoog bajta se postavljaju bitovi
54   * iz prvog bajta */
55  x=x|prvi_bajt;

56  /*
57   * Na bitove na poziciji cetvrtoog bajta se postavljaju bitovi
58   * iz prvog bajta */
59  x=x|cetvrti_bajt;

60  return x;
61 }

62 int main () {
63     int x;

```

## A Ispitni rokovi

---

```
64    /* Sa standardnog ulaza se ucitava ceo broj */
65    scanf("%d", &x);

66    /* Provera da li je uneti broj negativan */
67    if(x<0){
68        fprintf(stderr, "-1\n");
69        exit(EXIT_FAILURE);
70    }

71    /* Ispisivanje rezultata primene funkcije zamena na uneti broj x */
72    printf("%u\n", zamena(x));

73    exit(EXIT_SUCCESS);
74 }
```

**Rešenje A.11** NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

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

4 /* Funkcija racuna duzinu najduzeg puta od korena do nekog lista */
5 int najduzi_put(Cvor *koren) {
6     /* Pomocne promenljive */
7     int najduzi_u_levom,najduzi_u_desnom;

8     /* Ako je stablo prazno, povratna vrednost je -1 */
9     if(koren==NULL)
10         return -1;

11     /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
12     najduzi_u_levom=najduzi_put(koren->levo);

13     /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
14     najduzi_u_desnom=najduzi_put(koren->desno);

15     /* Veca od prethodno izracunatih vrednosti za podstabla se uvecava
16      za 1 i vraca kao konacan rezultat */
17     return 1 + (najduzi_u_levom > najduzi_u_desnom ? najduzi_u_levom :
18                 najduzi_u_desnom);
19 }

20 int main() {
21     Cvor *stablo = NULL;
22     int x;

23     /* U svakoj iteraciji se procitani broj dodaje u stablo. */
24     while (scanf("%d", &x) == 1)
```

```

32     if (dodaj_u_stablo(&stablo, x) == 1) {
33         fprintf(stderr, "-1\n");
34         exit(EXIT_FAILURE);
35     }
36
37     /* Ispisuje se rezultat rada trazene funkcije */
38     printf("%d\n", najduzi_put(stablo));
39
40     /* Oslobadja se memorija */
41     osloboidi_stablo(&stablo);
42
43     exit(EXIT_SUCCESS);
44 }
```

### Rešenje A.12

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 /* Ime datoteke nije duze od 30 karaktera */
4 #define MAX 31
5
6 /* Funkcija alocira memorijski prostor za matricu sa n vrsta i m
   kolona */
7 float **alociraj_matricu(int n, int m)
8 {
9     float **matrica = NULL;
10    int i, j;
11
12    /* Alocira se prostor za niz vrsta matrice */
13    matrica = (float **) malloc(n * sizeof(float *));
14    /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
       biti NULL, sto mora biti provereno u main funkciji */
15    if (matrica == NULL)
16        return NULL;
17
18    /* Alocira se prostor za svaku vrstu matrice */
19    for (i = 0; i < n; i++) {
20        matrica[i] = (float *) malloc(m * sizeof(float));
21        /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
           prethodno alocirani resursi, i povratna vrednost je NULL */
22        if (matrica[i] == NULL) {
23            for (j = 0; j < i; j++)
24                free(matrica[j]);
25            free(matrica);
26            return NULL;
27        }
28    }
29    return matrica;
30 }
31
32 /* Funkcija oslobadja alocirani memorijski prostor */
33 }
```

```
36 float **deallociraj_matricu(float **matrica, int n)
{
38     int i;
/* Oslobadja se prostor rezervisan za svaku vrstu */
40     for (i = 0; i < n; i++)
        free(matrica[i]);
42
/* Oslobadja se memorija za niz pokazivaca na vrste */
44     free(matrica);
46
/* Matrica postaje prazna, tj. nealocirana */
    return NULL;
48 }

/* Funkcija vraca indeks vrste matrice A u kojoj se realan broj x
   pojavljuje najmanje puta */
50 int f(float x, float **A, int n, int m){
52     /* Indeks vrste sa minimalnim brojem pojavljivanja broja x */
54     int min;
/* Broj pojavljivanja broja x u vrsti sa indeksom min */
56     int min_broj=0;
/* Promenljiva u kojoj ce se racunati broj pojavljivanja broja x u
58 tekucnoj vrsti */
59     int broj_u_vrsti;
/* Pomocne promenljive */
60     int i,j;
62
/* Promenljiva min se inicijalizuje na nulu, a min_broj na broj
64 pojavljivanja broja x u nultoj vrsti */
65     min=0;
66     for(j=0;j<m;j++){
67         if(A[0][j]==x)
68             min_broj++;
69     }
70
/* Za svaku vrstu (osim nulte) se racuna broj pojavljivanja broja
   x u njoj, pa ukoliko je taj broj manji od trenutno najmanjeg
   azuriraju se promenljive min i min_broj */
71     for(i=1;i<n;i++){
72         broj_u_vrsti=0;
73         for(j=0;j<m;j++){
74             if(A[i][j]==x)
75                 broj_u_vrsti++;
76         }
77         if(broj_u_vrsti<min_broj){
78             min_broj=broj_u_vrsti;
79             min=i;
80         }
81     }
82
/* Funkcija vraca odgovarajuci indeks vrste */
83     return min;
84 }
```

```

88 }
89
90 int main () {
91     FILE *in;
92     char datoteka[MAX];
93     float broj;
94     float **A=NULL;
95     int i,j;
96     int m,n;
97
98     /* Sa standardnog ulaza se ucitava ime datoteke i realan broj*/
99     scanf("%s",datoteka);
100    scanf("%f",&broj);
101
102    /* Otvaranje datoteke za citanje */
103    in=fopen(datoteka,"r");
104
105    /* Provera da li je datoteka uspesno otvorena */
106    if(in==NULL){
107        fprintf(stderr,"-1\n");
108        exit(EXIT_FAILURE);
109    }
110
111    /* Dimenzije matrice se ucitavaju iz datoteke (prva dva cela broja
112       u datoteci). U slucaju neuspelog ucitavanja, na standardni
113       izlaz za greske se ispisuje -1 i prekida se program. */
114    if(fscanf(in,"%d %d",&n,&m)==EOF){
115        fprintf(stderr,"-1\n");
116        exit(EXIT_FAILURE);
117    }
118
119    /* Provera da li su ucitani brojevi m i n pozitivni */
120    if(n<=0 || m<=0){
121        fprintf(stderr,"-1\n");
122        exit(EXIT_FAILURE);
123    }
124
125    /* Alokacija matrice */
126    A = alociraj_matricu(n, m);
127
128    /*Provera da li je alokacija uspela */
129    if(A == NULL){
130        fprintf(stderr,"-1\n");
131        exit(EXIT_FAILURE);
132    }
133
134    /* Ucitavanje elemenata matrice iz datoteke */
135    for(i=0; i<n; i++){
136        for(j=0; j<m; j++)
137            fscanf(in, "%f", &A[i][j]);
138    }

```

## A Ispitni rokovi

---

```
140  /* Zatvaranje datoteke */
141  fclose(in);
142
143  /* Ispisivanje rezultata poziva funkcije */
144  printf("%d\n", f(broj, A, n, m));
145
146  /* Dealokacija matrice */
147  A = dealociraj_matricu(A, n);
148
149  exit(EXIT_SUCCESS);
150 }
```