

## **PROGRAMIRANJE 2**



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

**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

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Anđelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu. Studentski trg 16, 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 i crteži: *autori*. Dizajn korica: *Anđelka Zečević*

Štampa: Copy Centar, Beograd. Tiraž 200.

СIP Каталогизација у публикацији

Народна библиотека Србије, Београд

004.4(075.8)(076)

004.432.2C(075.8)(076)

PROGRAMIRANJE 2 : zbirka zadataka sa rešenjima / Milena Vujošević

Jančić ... [et al.]. - Beograd : Matematički fakultet, 2016

(Beograd : Copy Centar). - VII, 361 str. ; 24 cm

Tiraž 200.

ISBN 978-86-7589-107-9

1. Вујошевић Јаничић, Милена 1980- [аутор]

а) Програмирање - Задаци б) Програмски језик "C"- Задаци

COBISS.SR-ID 221508876

©2016. Milena Vujošević Jančić, Jelena Graovac, Nina Radojičić, Ana Spasić, Mirko Spasić, Anđelka Zečević

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</b>	<b>Uvodni zadaci</b>	<b>1</b>
1.1	Podela koda po datotekama . . . . .	1
1.2	Algoritmi za rad sa bitovima . . . . .	5
1.3	Rekurzija . . . . .	10
1.4	Rešenja . . . . .	18
<b>2</b>	<b>Pokazivači</b>	<b>67</b>
2.1	Pokazivačka aritmetika . . . . .	67
2.2	Višedimenzioni nizovi . . . . .	71
2.3	Dinamička alokacija memorije . . . . .	75
2.4	Pokazivači na funkcije . . . . .	81
2.5	Rešenja . . . . .	83
<b>3</b>	<b>Algoritmi pretrage i sortiranja</b>	<b>125</b>
3.1	Algoritmi pretrage . . . . .	125
3.2	Algoritmi sortiranja . . . . .	130
3.3	Bibliotečke funkcije pretrage i sortiranja . . . . .	140
3.4	Rešenja . . . . .	144
<b>4</b>	<b>Dinamičke strukture podataka</b>	<b>217</b>
4.1	Liste . . . . .	217
4.2	Stabla . . . . .	227
4.3	Rešenja . . . . .	236
<b>A</b>	<b>Ispitni rokovi</b>	<b>333</b>
A.1	Praktični deo ispita, jun 2015. . . . .	333
A.2	Praktični deo ispita, jul 2015. . . . .	335
A.3	Praktični deo ispita, septembar 2015. . . . .	336
A.4	Praktični deo ispita, januar 2016. . . . .	338
A.5	Rešenja . . . . .	340



# Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanja 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 propratna rešenja u elektronskom formatu, dostupna su besplatno u okviru strane kursa [www.programiranje2.matf.bg.ac.rs](http://www.programiranje2.matf.bg.ac.rs) u skladu sa navedenom licencom.

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 akademske 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 savlađivanje koncepata koji se obrađuju u okviru kursa. Takođe, formulacije zadataka smo obogatili primerima koji upotpunjuju razumevanje zahteva zadataka i koji omogućavaju čitaocu zbirke da proveriti sopstvena rešenja. Primeri su dati u obliku testova i interakcija sa programom. Testovi su svedene prirode i obuhvataju samo jednostavne ulaze i izlaze iz programa. Interakcija sa programom obuhvata naizmeničnu interakciju čovek-računar u kojoj su ulazi i izlazi isprepletani. U zadacima koji zahtevaju

rad sa argumentima komandne linije, navedeni su i primeri poziva programa, a u zadacima koji demonstriraju rad sa datotekama, i primeri ulaznih ili izlaznih datoteka. Test primeri koji su navedeni uz ispitne zadatke u dodatku su oni koji su korišćeni za početno testiranje (koje prethodi ocenjivanju) studentskih radova na ispitima.

Neizmerno 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 uobličavanju ovog materijala.

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

*Autori*

---

<sup>1</sup>Adrese autora su: milena, jgraovac, nina, aspasic, mirko, andjelkaz, 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) Definirati strukturu `KompleksanBroj` koja opisuje kompleksan broj zadan 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 imaginaran_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  $z1$  i  $z2$ .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća proizvod dva kompleksna broja  $z1$  i  $z2$ .
- (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  $z1$  i  $z2$ , a zatim ispisati realni deo, imaginarni deo, moduo, konjugovano-kompleksan broj i argument broja koji se dobija kao zbir, razlika ili proizvod brojeva  $z1$  i  $z2$  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
```

**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
```

**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. Ipisati 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 datoj tački `x` koristeći Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja vraća zbir dva polinoma `p` i `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja vraća proizvod dva polinoma `p` i `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 najvećeg stepena do nultog):
3 1.2 3.5 2.1 4.2
Unesite polinom q (prvo stepen, pa zatim koeficijente od najvećeg 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
```

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

- (a) Definirati 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 vraća brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraća imenilac razlomka `r`.
- (f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.
- (g) Napisati funkciju `double recipročna_vrednost(const Razlomak * r)` koja vraća recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skрати(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: 0xf4f4f4f
Binarna reprezentacija: 01001111010011110100111101001111
Unesite broj tipa short: 0xf4f
Binarna reprezentacija: 0100111101001111
Unesite broj tipa char: 0xf
Binarna reprezentacija: 01001111
```

**Zadatak 1.6** Napisati funkcije `_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:

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

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

**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:
  00000000000000000000000001111111

```

### Test 2

```

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

```

### Test 3

```

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

```

### Test 4

```

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

```

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

- 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.
- 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.
- 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.
- 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`.
- 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`.



## 1 Uvodni zadaci

na poziciju najmanje težine. Analogno, rotiranje bitova udesno podrazumeva pomeranje svih bitova za jednu poziciju udesno, s tim što se bit sa pozicije najmanje težine pomera na poziciju najveće težine.

- (a) Napisati funkciju `unsigned rotiraj_ulevo(unsigned x, unsigned n)` koja 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 heksadekaskom 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 neoznaceni broj x: ba11a7
Unesite neoznaceni 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) = 11000111111101000010111101110010
```

**Zadatak 1.10** Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća broj čiji binarni zapis predstavlja sliku u ogledalu binarnog zapisa 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:
00000000000000000000000000000000
10101010010000000000000000000000
```

### Test 2

```
ULAZ:
-15
IZLAZ:
11111111111111111111111111110101
11010111111111111111111111111111
```



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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 10	ULAZ: 2147377146	ULAZ: 1111111115
IZLAZ: 0	IZLAZ: 1	IZLAZ: 0

**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.*

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

\* **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 `+`, `-`, `/`, `*`, `%`.

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
POKRETANJE: ./a.out 1 2	POKRETANJE: ./a.out 1 2	POKRETANJE: ./a.out 12 12
INTERAKCIJA SA PROGRAMOM:	INTERAKCIJA SA PROGRAMOM:	INTERAKCIJA SA PROGRAMOM:
ULAZ: 11	ULAZ: 1024	ULAZ: 12345
IZLAZ: 13	IZLAZ: 1024	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 prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksade-

## 1 Uvodni zadaci

---

kadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1	Test 2	Test 3
ULAZ: 11 IzLAZ: 0000000B	ULAZ: 1024 IzLAZ: 00000400	ULAZ: 12345 IzLAZ: 00003039

\* **Zadatak 1.15** Napisati funkciju koja za data 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 nepromenjeni. Napisati program koji testira tu funkciju za brojeve koji se zadaju sa standardnog ulaza.

Test 1	Test 2	Test 3
ULAZ: 123 10 IzLAZ: 4294967285	ULAZ: 3251 0 IzLAZ: 4294967295	ULAZ: 12541 1024 IzLAZ: 4294966271

**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
ULAZ: 123 IzLAZ: 0	ULAZ: 3245 IzLAZ: 2	ULAZ: 100328 IzLAZ: 1

## 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,
- (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 standardni 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
```

**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.
```

**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
```

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

## 1 Uvodni zadaci

---

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:

$$F(0) = 0$$

$$F(1) = 1$$

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

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 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:
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:
1 - iterativna
2 - rekurzivna
3 - rekurzivna napredna
3
Unesite koeficijente: 4 2
Unesite koji clan niza se racuna: 8
F(8) = 31360
```

**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.

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

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

- (a) sabirajući elemente počev od početka niza ka kraju niza,
- (b) 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.

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

**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

```

**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

```

**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  $x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do kraja ulaza. Pretpostaviti 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

```

**Zadatak 1.27** Napisati rekurzivnu funkciju kojom se proverava da li su tri data cela broja uzastopni članovi datog celobrojnog niza. Sa standardnog ulaza

```
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.
```

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

```
ULAZ:
    0x7F
IZLAZ:
    7
```

```
ULAZ:
    0x00FF00FF
IZLAZ:
    16
```

```
ULAZ:
    0xFFFFFFFF
IZLAZ:
    32
```

```
ULAZ:
    10
IZLAZ:
    000000000000000000000000000000001010
```

```
ULAZ:  
0  
IZLAZ:  
00000000000000000000000000000000
```

## 1 Uvodni zadaci

---

Test 1	Test 2	Test 3
ULAZ: 5    IZLAZ: 5	ULAZ: 125    IZLAZ: 7	ULAZ: 8    IZLAZ: 1

**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	Test 2	Test 3
ULAZ: 5    IZLAZ: 5	ULAZ: 16    IZLAZ: 1	ULAZ: 18    IZLAZ: 2

**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. Pretpostaviti da niska neće imati više od 31 karaktera.

Test 1	Test 2	Test 3
ULAZ: a    IZLAZ: da	ULAZ: aa    IZLAZ: da	ULAZ: aba    IZLAZ: da

Test 4	Test 5
ULAZ: programiranje    IZLAZ: ne	ULAZ: anavolimilovana    IZLAZ: da

\* **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

```

ULAZ:
  2
IZLAZ:
  1 2
  2 1

```

Test 2

```

ULAZ:
  3
  1 2 3
  1 3 2
  2 1 3
  2 3 1
  3 1 2
  3 2 1

```

Test 3

```

ULAZ:
  -5
  Duzina
  permutacije
  mora biti
  broj iz
  intervala
  [0, 15]!

```

\* **Zadatak 1.34** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbira 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.

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

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

Test 1

```

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

```

Test 2

```

ULAZ:
  6 5
IZLAZ:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
32

```

\* **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
```

\* **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 premestiti 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 premestiti diskove na drugi štap tako da budu u istom redosledu, premestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg. 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

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

```
10 } KomplexsanBroj;

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

22 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
    obliku (x + i y). Ovoj funkciji se argument prenosi po vrednosti
    jer se u samoj funkciji ne menja njegova vrednost */
24 void ispisi_kompleksan_broj(KompleksanBroj z)
26 {
    /* Zapocinje se sa ispisom */
28     printf("(");

30     /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
    razlicit od nule: tada se realni deo ispisuje na standardni
    izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li
    je imaginarni deo pozitivan ili negativan, a potom i apsolutna
    vrednost imaginarnog dela kompleksnog broja 2) realni deo
    kompleksnog broja je nula: tada se samo ispisuje imaginaran
    deo, s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
    decimalnih mesta */
32     if (z.real != 0) {
        printf("%.2f", z.real);

34         if (z.imag > 0)
            printf(" + %.2f i", z.imag);
        else if (z.imag < 0)
            printf(" - %.2f i", -z.imag);
36     } else {
        if (z.imag == 0)
            printf("0");
        else
            printf("%.2f i", z.imag);
38     }

40     /* Završava se sa ispisom */
42     printf(")");
44 }

46 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
48 float realan_deo(KompleksanBroj z)
50 {
    return z.real;
52 }
54 }
56
58
60
```

```
62  /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
64  float imaginaran_deo(KompleksanBroj z)
65  {
66      return z.imag;
67  }
68
69  /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
70  float moduo(KompleksanBroj z)
71  {
72      return sqrt(z.real * z.real + z.imag * z.imag);
73  }
74
75  /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
76     odgovara kompleksnom broju argumentu */
77  KompleksanBroj konjugovan(KompleksanBroj z)
78  {
79      /* Konjugovano kompleksan broj z se dobija tako sto se promeni
80         znak imaginarnom delu kompleksnog broja */
81      KompleksanBroj z1 = z;
82      z1.imag *= -1;
83
84      return z1;
85  }
86
87  /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
88     argumenata funkcije */
89  KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
90  {
91      /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
92         broj ciji je realan deo zbir realnih delova kompleksnih brojeva
93         z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
94         brojeva z1 i z2 */
95      KompleksanBroj z = z1;
96      z.real += z2.real;
97      z.imag += z2.imag;
98
99      return z;
100 }
101
102 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
103     argumenata funkcije */
104 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
105 {
106     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
107        broj ciji je realan deo razlika realnih delova kompleksnih
108        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
109        kompleksnih brojeva z1 i z2 */
110     KompleksanBroj z = z1;
111     z.real -= z2.real;
112     z.imag -= z2.imag;
```

```

114     return z;
115 }
116
117 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
118    argumenata funkcije */
119 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
120 {
121     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
122        broj ciji se realan i imaginaran deo racunaju po formuli za
123        mnozenje kompleksnih brojeva z1 i z2 */
124     KompleksanBroj z;
125     z.real = z1.real * z2.real - z1.imag * z2.imag;
126     z.imag = z1.real * z2.imag + z1.imag * z2.real;
127
128     return z;
129 }
130
131 /* Funkcija vraca argument zadatog kompleksnog broja */
132 float argument(KompleksanBroj z)
133 {
134     /* Argument kompleksnog broja z se racuna pozivanjem funkcije
135        atan2 iz biblioteke math.h */
136     return atan2(z.imag, z.real);
137 }
138
139 int main()
140 {
141     char c;
142
143     /* Deklaracija 3 promenljive tipa KompleksanBroj */
144     KompleksanBroj z1, z2, z;
145
146     /* Ucitava se prvi kompleksni broj, koji se potom ispisuje na
147        standardni izlaz */
148     ucitaj_kompleksan_broj(&z1);
149     ispisi_kompleksan_broj(z1);
150     printf("\n");
151
152     /* Ucitava se drugi kompleksni broj, koji se potom ispisuje na
153        standardni izlaz */
154     ucitaj_kompleksan_broj(&z2);
155     ispisi_kompleksan_broj(z2);
156     printf("\n");
157
158     /* Ucitavase znak na osnovu koga korisnik bira aritmeticku
159        operaciju koja ce se izvorsiti nad kompleksnim brojevima, a
160        zatim se vrsi provera da li je unet neki od dozvoljenih
161        aritmetickih znakova. */
162     getchar();
163     printf("Unesite znak (+,-,*): ");
164     scanf("%c", &c);
165     if (c != '+' && c != '-' && c != '*') {

```

## 1 Uvodni zadaci

```
166     fprintf(stderr, "Greska: Nedozvoljena vrednost operatora.\n");
167     exit(EXIT_FAILURE);
168 }
169
170 /* Analizira se uneti operator */
171 if (c == '+') {
172     /* Racuna se zbir */
173     z = saberi(z1, z2);
174 } else if (c == '-') {
175     /* Racuna se razlika */
176     z = oduzmi(z1, z2);
177 } else {
178     /* Racuna se proizvod */
179     z = mnozi(z1, z2);
180 }
181
182 /* Ispisuje se rezultat */
183 ispisi_kompleksan_broj(z1);
184 printf(" %c ", c);
185 ispisi_kompleksan_broj(z2);
186 printf(" = ");
187 ispisi_kompleksan_broj(z);
188
189 /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
190    broja */
191 printf("\nRealni_deo: %.f\nImaginarni_deo: %.f\nModuo: %.f\n",
192        realan_deo(z), imaginaran_deo(z), moduo(z));
193
194 /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
195    kompleksnog broja */
196 printf("Konjugovano kompleksan broj: ");
197 ispisi_kompleksan_broj(konjugovan(z));
198 printf("\n");
199
200 /* Testira se funkcija koja racuna argument kompleksnog broja */
201 printf("Argument kompleksnog broja: %.f\n", argument(z));
202
203 exit(EXIT_SUCCESS);
204 }
```

### Rešenje 1.2

*kompleksan\_broj.h*

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

```
7  /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i
   onemogućava se da se sadržaj zaglavlja više puta uključi. Niska
9  posle ključne reči ifndef je proizvoljna, ali treba da se ponovi
   u narednoj pretprocesorskoj define direktivi. */
11 #ifndef _KOMPLEKSAN_BROJ_H
   #define _KOMPLEKSAN_BROJ_H
13
13  /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
15  koje se koriste u definicijama funkcija navedenim u
   kompleksan_broj.c */
17 #include <stdio.h>
   #include <math.h>
19
19  /* Struktura KompleksanBroj */
21 typedef struct {
   float real;
23   float imag;
   } KompleksanBroj;
25
25  /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
27  definisane u kompleksan_broj.c */
29
29  /* Funkcija učitava sa standardnog ulaza realan i imaginarni deo
   kompleksnog broja i smesta ih u strukturu čija je adresa argument
31  funkcije */
   void ucitaj_kompleksan_broj(KompleksanBroj * z);
33
33  /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
35  obliku (x + i y) */
   void ispisi_kompleksan_broj(KompleksanBroj z);
37
37  /* Funkcija vraća vrednosti realnog dela kompleksnog broja */
39 float realan_deo(KompleksanBroj z);
41
41  /* Funkcija vraća vrednosti imaginarnog dela kompleksnog broja */
   float imaginarni_deo(KompleksanBroj z);
43
43  /* Funkcija vraća vrednost modula zadanog kompleksnog broja */
45 float moduo(KompleksanBroj z);
47
47  /* Funkcija vraća vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju argumentu */
49 KompleksanBroj konjugovan(KompleksanBroj z);
51
51  /* Funkcija vraća kompleksan broj čija vrednost je jednaka zbiru
   argumenata funkcije */
53 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
55
55  /* Funkcija vraća kompleksan broj čija vrednost je jednaka razlici
   argumenata funkcije */
57 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
```

## 1 Uvodni zadaci

---

```
59 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
61 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);

63 /* Funkcija vraca argument zadatog kompleksnog broja */
    float argument(KompleksanBroj z);
65
    /* Kraj zakljucanog dela */
67 #endif
```

*kompleksan\_broj.c*

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

7  void ucitaj_kompleksan_broj(KompleksanBroj * z)
    {
9      /* Ucitavanje vrednosti sa standardnog ulaza */
        printf("Unesite realan i imaginaran deo kompleksnog broja: ");
11     scanf("%f", &z->real);
        scanf("%f", &z->imag);
13     }

15 void ispisi_kompleksan_broj(KompleksanBroj z)
    {
17     /* Zapocinje se sa ispisom */
        printf("(");

19
        /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
        21     razlicit od nule: tada se realni deo ispisuje na standardni
            izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li
        23     je imaginarni deo pozitivan ili negativan, a potom i apsolutna
            vrednost imaginarnog dela kompleksnog broja 2) realni deo
        25     kompleksnog broja je nula: tada se samo ispisuje imaginaran
            deo, s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
        27     decimalnih mesta */
        if (z.real != 0) {
29             printf("%.2f", z.real);

31             if (z.imag > 0)
                printf(" + %.2f i", z.imag);
33             else if (z.imag < 0)
                printf(" - %.2f i", -z.imag);
35         } else {
            if (z.imag == 0)
37                 printf("0");
            else
39                 printf("%.2f i", z.imag);
```



```
41     }
42
43     /* Završava se sa ispisom */
44     printf("\n");
45 }
46
47 float realan_deo(KompleksanBroj z)
48 {
49     /* Vraca se vrednost realnog dela kompleksnog broja */
50     return z.real;
51 }
52
53 float imaginaran_deo(KompleksanBroj z)
54 {
55     /* Vraca se vrednost imaginarnog dela kompleksnog broja */
56     return z.imag;
57 }
58
59 float moduo(KompleksanBroj z)
60 {
61     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
62     return sqrt(z.real * z.real + z.imag * z.imag);
63 }
64
65 KompleksanBroj konjugovan(KompleksanBroj z)
66 {
67     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
68        se promeni znak imaginarnom delu kompleksnog broja */
69     KompleksanBroj z1 = z;
70     z1.imag *= -1;
71     return z1;
72 }
73
74 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
75 {
76     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
77        broj ciji je realan deo zbir realnih delova kompleksnih brojeva
78        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
79        brojeva z1 i z2 */
80     KompleksanBroj z = z1;
81     z.real += z2.real;
82     z.imag += z2.imag;
83
84     return z;
85 }
86
87 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
88 {
89     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
90        broj ciji je realan deo razlika realnih delova kompleksnih
91        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
92        kompleksnih brojeva z1 i z2 */
93 }
```

## 1 Uvodni zadaci

---

```

    KomplexanBroj z = z1;
93  z.real -= z2.real;
    z.imag -= z2.imag;
95
    return z;
97 }

99 KomplexanBroj mnozi(KomplexanBroj z1, KomplexanBroj z2)
{
101  /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
    broj ciji se realan i imaginaran deo racunaju po formuli za
103  mnozenje kompleksnih brojeva z1 i z2 */
    KomplexanBroj z;
105  z.real = z1.real * z2.real - z1.imag * z2.imag;
    z.imag = z1.real * z2.imag + z1.imag * z2.real;
107
    return z;
109 }

111 float argument(KomplexanBroj z)
{
113  /* Argument kompleksnog broja z se racuna pozivanjem funkcije
    atan2 iz biblioteke math.h */
115  return atan2(z.imag, z.real);
}

```

*main.c*

```

/*****
2  Ovaj program koristi korektno definisanu biblioteku kompleksnih
   brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
4  kompleksnog broja i popis deklaracija podrzanih funkcija, a u
   kompleksan_broj.c se nalaze njihove definicije.
6
   Kompilacija programa se najjednostavnije postize naredbom
8  gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c

10  Kompilacija se moze uraditi i na sledeci nacin:
   gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
12  gcc -Wall -c -o main.o main.c
   gcc -lm -o kompleksan_broj kompleksan_broj.o main.o
14
   Napomena: Prethodne komande se koriste kada se sva tri navedena
16  dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
   kompleksan_broj.c kompleksan_broj.h) nalazi u direktorijumu sa imenom
18  header_dir prevodjenje se vrši dodavanjem opcije opcije -I header_dir
   gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c
20  main.c
   *****/
22

```

```

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

28 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
   polarni oblik */
30 int main()
{
32     KompleksanBroj z;

34     /* Učitava se kompleksan broj */
    učitaj_kompleksan_broj(&z);

36     /* Ispisuje se polarni oblik kompleksnog broja */
38     printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
           moduo(z), argument(z));

40     return 0;
42 }

```

### Rešenje 1.3

*polinom.h*

```

1  #ifndef _POLINOM_H
   #define _POLINOM_H
3
   #include <stdio.h>
5   #include <stdlib.h>

7   /* Maksimalni stepen polinoma */
   #define MAKS_STEPEN 20
9
11  /* Polinomi se predstavljaju strukturom koja čuva koeficijente
   (koef[i] je koeficijent uz član  $x^i$ ) i stepen polinoma */
13  typedef struct {
       double koef[MAKS_STEPEN + 1];
15     int stepen;
   } Polinom;
17
   /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
19     obliku. Polinom se prenosi po adresi da bi se uštedela memorija:
       ne kopira se cela struktura, već se samo prenosi adresa na kojoj
21     se nalazi polinom koji ispisujemo */
   void ispisi(const Polinom * p);
23
   /* Funkcija koja učitava polinom sa tastature */
25 Polinom učitaj();

```

## 1 Uvodni zadaci

---

```
27 /* Funkcija racuna i vraca vrednost polinoma p u tacki x Hornerovim
    algoritmom */
29 double izracunaj(const Polinom * p, double x);

31 /* Funkcija koja sabira dva polinoma */
    Polinom saberi(const Polinom * p, const Polinom * q);
33
    /* Funkcija koja mnozi dva polinoma p i q */
35 Polinom pomnozi(const Polinom * p, const Polinom * q);

37 /* Funkcija koja racuna izvod polinoma p */
    Polinom izvod(const Polinom * p);
39
    /* Funkcija koja racuna n-ti izvod polinoma p */
41 Polinom n_izvod(const Polinom * p, int n);
    #endif
```

*polinom.c*

```
#include <stdio.h>
2 #include <stdlib.h>
    #include "polinom.h"
4
    void ispisi(const Polinom * p)
6 {
    int nulaPolinom = 1;
    int i;
    /* Ispisivanje polinoma pocinje od najviseg stepena ka najnižem da
10     bi polinom bio ispisan na prirodan nacin. Ispisuju se samo oni
        koeficijenti koji su razliciti od nule. Ispred pozitivnih
12     koeficijenata je potrebno ispisati znak + (osim u slucaju
        koeficijenta uz najvisi stepen). */
14     for (i = p->stepen; i >= 0; i--) {

16         if (p->koef[i]) {
            /* Polinom nije nula polinom, cim je neki od koeficijenata
18             razlicit od nule */
            nulaPolinom = 0;
20             if (p->koef[i] >= 0 && i != p->stepen)
                putchar('+');
22             if (i > 1)
                printf("%.2fx^%d", p->koef[i], i);
24             else if (i == 1)
                printf("%.2fx", p->koef[i]);
26             else
                printf("%.2f", p->koef[i]);
28         }
    }
30 /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada se
    ispisuje nula. */
```

```

32     if (nulaPolinom)
33         printf("0");
34     putchar('\n');
35 }
36
37 Polinom ucitaj()
38 {
39     int i;
40     Polinom p;
41
42     /* Ucitava se stepena polinoma */
43     scanf("%d", &p.stepen);
44
45     /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
46     dovoljenog opsega */
47     while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
48         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
49         scanf("%d", &p.stepen);
50     }
51
52     /* Unose se koeficijenti polinoma */
53     for (i = p.stepen; i >= 0; i--)
54         scanf("%lf", &p.koef[i]);
55
56     /* Vraca se procitani polinom */
57     return p;
58 }
59
60 double izracunaj(const Polinom * p, double x)
61 {
62     /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
63     svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
64     vrednost odgovarajuceg koeficijenta */
65
66     /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
67      $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1$  */
68
69     double rezultat = 0;
70     int i = p->stepen;
71     for (; i >= 0; i--)
72         rezultat = rezultat * x + p->koef[i];
73     return rezultat;
74 }
75
76 Polinom saberi(const Polinom * p, const Polinom * q)
77 {
78     Polinom rez;
79     int i;
80
81     /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
82     stepenom */
83     rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

```

```

84  /* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
86  sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
88  sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
90  od stepena nekog od polaznih polinoma podrazumeva se da je
92  koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
94  polinoma */
96  for (i = 0; i <= rez.stepen; i++)
98  rez.koef[i] =
100      (i > p->stepen ? 0 : p->koef[i]) +
102      (i > q->stepen ? 0 : q->koef[i]);
104
106  /* Vraca se dobijeni polinom */
108  return rez;
110 }
112
114 Polinom pomnozi(const Polinom * p, const Polinom * q)
116 {
118     int i, j;
120     Polinom r;
122
124     /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
126     */
128     r.stepen = p->stepen + q->stepen;
130     if (r.stepen > MAKS_STEPEN) {
132         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
134         exit(EXIT_FAILURE);
136     }
138
140     /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na 0 */
142     for (i = 0; i <= r.stepen; i++)
144         r.koef[i] = 0;
146
148     /* U svakoj iteraciji odgovarajuci koeficijent rezultata se
150     uvecava za proizvod odgovarajucih koeficijenata iz polaznih
152     polinoma */
154     for (i = 0; i <= p->stepen; i++)
156         for (j = 0; j <= q->stepen; j++)
158             r.koef[i + j] += p->koef[i] * q->koef[j];
160
162     /* Vraca se dobijeni polinom */
164     return r;
166 }
168
170 Polinom izvod(const Polinom * p)
172 {
174     int i;
176     Polinom r;
178
180     /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
182     polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
184     je rezultujuci polinom nula (izvod od konstante je nula). */

```

```

136     if (p->stepen > 0) {
138         r.stepen = p->stepen - 1;

138         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
139            polaznog polinoma */
140         for (i = 0; i <= r.stepen; i++)
142             r.koef[i] = (i + 1) * p->koef[i + 1];
144     } else
146         r.koef[0] = r.stepen = 0;

144     /* Vraca se dobijeni polinom */
146     return r;
148 }

148 Polinom n_izvod(const Polinom * p, int n)
150 {
152     int i;
154     Polinom r;

154     /* Provera da li je n nenegativna vrednost */
156     if (n < 0) {
158         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
160         exit(EXIT_FAILURE);
162     }

160     /* Multi izvod je bas taj polinom */
162     if (n == 0)
164         return *p;

164     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove
165        funkcija za racunanje prvog izvoda polinoma */
166     r = izvod(p);
168     for (i = 1; i < n; i++)
170         r = izvod(&r);

170     /* Vraca se dobijeni polinom */
172     return r;
172 }

```

*main.c*

```

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

4 int main(int argc, char **argv)
5 {
6     Polinom p, q, z, r;
7     double x;
8     int n;

10     /* Unosi se polinom p */

```

```
printf
12     ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
    najveceg stepena do nultog):\n");
p = ucitaj();

14

/* Ispisuje se polinom p */
16 ispisi(&p);

18 /* Unosi se polinom q */
printf
20     ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
    od najveceg stepena do nultog):\n");
q = ucitaj();

22

/* Polinomi se sabiraju i ispisuje se izracunati zbir */
24 z = saberi(&p, &q);
printf("Zbir polinoma je polinom z:\n");
26 ispisi(&z);

28 /* Polinomi se mnoze i ispisuje se izracunati proizvod */
r = pomnozi(&p, &q);
30 printf("Prozvod polinoma je polinom r:\n");
ispisi(&r);

32

/* Ispisuje se vrednost polinoma u unetoj tacki */
34 printf("Unesite tacku u kojoj racunate vrednost polinoma z:\n");
scanf("%lf", &x);
36 printf("Vrednost polinoma z u tacki %.2f je %.2f\n", x,
    izracunaj(&z, x));

38

/* Racuna se n-ti izvoda polinoma i ispisuje se dobijeni polinoma
    */
40 printf("Unesite izvod polinoma koji zelite:\n");
scanf("%d", &n);
42 r = n_izvod(&r, n);
printf("%d. izvod polinoma r je: ", n);
44 ispisi(&r);

46     exit(EXIT_SUCCESS);
}
```

### Rešenje 1.5

stampanje\_bitova.h

```
1 #ifndef _STAMPANJE_BITOVA_H
2 #define _STAMPANJE_BITOVA_H
3
4 #include <stdio.h>
5
```



```

7  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji. Bitove koji predstavljaju binarnu
   reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
9  najveće težine ka bitu najmanje težine */
void stampaj_bitove(unsigned x);
11
13 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja tipa 'short' u memoriji. */
void stampaj_bitove_short(short x);
15
17 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   karaktera u memoriji. */
void stampaj_bitove_char(char x);
19
#endif

```

*stampanje\_bitova.c*

```

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

4  void stampaj_bitove(unsigned x)
   {
6     /* Broj bitova celog broja */
     unsigned velicina = sizeof(unsigned) * 8;

8     /* Maska koja se koristi za "ocitavanje" bitova celog broja */
     unsigned maska;

12    /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
       na mestu bita najveće težine sadrži jedinicu, a na svim ostalim
       mestima sadrži nulu. U svakoj iteraciji maska se menja tako sto
14    se jedini bit jedinica pomera udesno, kako bi se odredio naredni
       bit broja x koji je argument funkcije. Zatim se odgovarajuća
       cifra, ('0' ili '1'), ispisuje na standardnom izlazu. Neophodno
16    je da promenljiva maska bude deklarirana kao neoznačen ceo broj
       kako bi se pomeranjem u desno vrsilo logicko pomeranje
       (popunjavanje nulama), a ne aritmeticko pomeranje (popunjavanje
       znakom broja). */
20    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
       putchar(x & maska ? '1' : '0');

22    putchar('\n');
24 }

26 void stampaj_bitove_short(short x)
   {
30    /* Broj bitova celog broja tipa short */
     unsigned velicina = sizeof(short) * 8;

32    /* Maska koja se koristi za "ocitavanje" bitova broja tipa short */

```

## 1 Uvodni zadaci

---

```
34 unsigned short maska;

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

38 putchar('\n');
40 }

42 void stampaj_bitove_char(char x)
43 {
44     /* Broj bitova karaktera */
45     unsigned velicina = sizeof(char) * 8;

46     /* Maska koja se koristi za "ocitavanje" bitova jednog karaktera */
47     unsigned char maska;

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

52     putchar('\n');
54 }
```

*main.c*

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

4 int main()
5 {
6     int broj_int;
7     short broj_short;
8     char broj_char;

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

14     /* Ispisuje se binarna reprezentacija unetog broja */
15     printf("Binarna reprezentacija: ");
16     stampaj_bitove(broj_int);

18     /* Ucitava se broj tipa short */
19     printf("Unesite broj tipa short: ");
20     scanf("%hx", &broj_short);

22     /* Ispisuje se binarna reprezentacija unetog broja */
23     printf("Binarna reprezentacija: ");
24     stampaj_bitove_short(broj_short);

26     /* Ucitava se broj tipa char */
27     printf("Unesite broj tipa char: ");
```

```

28     scanf("%hbx", &broj_char);

30     /* Ispisuje se binarna reprezentacija unetog broja */
    printf("Binarna reprezentacija: ");
32     stampaj_bitove_char(broj_char);

34     return 0;
}

```

## Rešenje 1.6

```

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

3

5  /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   kreiranjem odgovarajuće maske i njenim pomeranjem */
   int prebroj_bitove_1(int x)
7  {
   int br = 0;
9   unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;

11  /* Formiranje se maska čija binarna reprezentacija izgleda
   100000...0000000, koja služi za očitavanje bita najveće težine.
13   U svakoj iteraciji maska se pomera u desno za 1 mesto, i očitava
   se sledeći bit. Petlja se završava kada više nema jedinica tj.
15   kada maska postane nula. */
   unsigned maska = 1 << broj_pomeranja;
17   for (; maska != 0; maska >>= 1)
       x & maska ? br++ : 1;

19   return br;
21 }

23 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   formiranjem odgovarajuće maske i pomeranjem promenljive x */
   int prebroj_bitove_2(int x)
25 {
   int br = 0;
27   unsigned broj_pomeranja = sizeof(int) * 8 - 1;

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

35   unsigned maska = 1 << broj_pomeranja;
   for (; x != 0; x <<= 1)
37       x & maska ? br++ : 1;
39

```

## 1 Uvodni zadaci

---

```
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 odgovarajuci rezultat na osnovu unesenog rednog
58        broja funkcije */
59     if (i == 1) {
60         printf("Poziva se funkcija prebroj_bitove_1\n");
61         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
62     } else if (i == 2) {
63         printf("Poziva se funkcija prebroj_bitove_2\n");
64         printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
65     } else {
66         fprintf(stderr, "Greska: Neodgovarajuci redni broj funkcije.\n");
67         exit(EXIT_FAILURE);
68     }
69
70     exit(EXIT_SUCCESS);
71 }
```

### 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 najveći neoznačeni broj sastavljen od istih bitova
5     koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
6  unsigned najveći(unsigned x)
7  {
8      unsigned velicina = sizeof(unsigned) * 8;
9
10     /* Formira se maska 100000...0000000 */
11     unsigned maska = 1 << (velicina - 1);
12
13     /* Rezultat se inicijalizuje vrednošću 0 */
14     unsigned rezultat = 0;
15 }
```

```
17  /* Promenljiva x se pomera u levo sve dok postoje jedinice u
    njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
    razlicita od nule). */
19  for (; x != 0; x <= 1) {
    /* Za svaku jedinicu koja se koriscenjem maske detektuje na
    21  poziciji najvece tezine u binarnoj reprezentaciji promenjive
    x, potiskuje se jedna nova jedinicu sa leva u rezultat */
    23  if (x & maska) {
        rezultat >>= 1;
    25  rezultat |= maska;
    }
    27  }

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

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

        41  /* Promenljiva x se pomera u desno sve dok postoje jedinice u
        njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
        43  razlicita od nule). */
        for (; x != 0; x >= 1) {
            45  /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
            detektuje na poziciji najmanje tezine u binarnoj
            47  reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
            sa desna u rezultat */
            49  if (x & 1) {
                rezultat <= 1;
            51  rezultat |= 1;
            }
            53  }

            55  /* Vraca se dobijena vrednost */
            return rezultat;
            57  }

    59  int main()
    {
        61  int broj;

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

        65  /* Ispisuju se, redom, najveći i najmanji broj formirani od bitova
        67  unetog broja */
```

## 1 Uvodni zadaci

```
printf("Najveci:\n");
69 stampaj_bitove(najveci(broj));

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

73 return 0;
75 }
```

### Rešenje 1.8

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

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

4 /* Funkcija postavlja na nulu n bitova pocev od pozicije p. */
unsigned postavi_0(unsigned x, unsigned n, unsigned p)
6 {
8     /******
     Formira se maska cija binarna reprezentacija ima n bitova
     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
     1111 1111 1111 1111 1111 1000 0011 1111
12     To se postize na sledeci nacin:
     ~0                1111 1111 1111 1111 1111 1111 1111 1111
14     (~0 << n)         1111 1111 1111 1111 1111 1111 1110 0000
     ~(~0 << n)         0000 0000 0000 0000 0000 0000 0001 1111
16     ~(~0 << n) << (p-n+1) 0000 0000 0000 0000 0000 0111 1100 0000
     ~(~0 << n) << (p-n+1) 1111 1111 1111 1111 1111 1000 0011 1111
18     *****/
     unsigned maska = ~(~0 << n) << (p - n + 1);

20     return x & maska;
22 }

24 /* Funkcija postavlja na jedinicu n bitova pocev od pozicije p. */
unsigned postavi_1(unsigned x, unsigned n, unsigned p)
26 {
28     /******
     Formira se maska kod koje je samo n bitova pocev od pocev od
30     pozicije p jednako 1, a ostali su 0.
     Na primer, za n=5 i p=10 formira se maska oblika
32     0000 0000 0000 0000 0000 0111 1100 0000
     *****/
34     unsigned maska = ~(~0 << n) << (p - n + 1);

36     return x | maska;
38 }
```

```

40  /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
    predstavlja n bitova pocev od pozicije p u binarnoj
    reprezentaciji broja x. */
42  unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
    {
44      /******
46      Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
        Na primer, za n=5
48      0000 0000 0000 0000 0000 0000 0001 1111
        *****/
50      unsigned maska = ~(~0 << n);

52      /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
        polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
54      zeljenih n i funkcija vraca tako dobijenu vrednost */
        return maska & (x >> (p - n + 1));
56  }

58  /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
    postavljeni na vrednosti n bitova najmanje tezine binarne
    reprezentacije broja y */
60  unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
62      unsigned y)
    {
64      /* Kreira se maska kod koje su poslednjih n bitova 1, a ostali
        su 0. */
66      unsigned poslednjih_n_1 = ~(~0 << n);

68      /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima n
        bitova postavljenih na 0 pocevsi od pozicije p, dok su ostali
70      bitovi 1. */
        unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));

72      /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se u
        binarnoj reprezentaciji vrednosti promenljive x postavi na 0 n
        bitova na pozicijama pocev od p */
74      unsigned x_postavi_0 = x & srednjih_n_0;

76      /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
        binarne reprezentacije vrednosti promenljive y cijih je n
        bitova najnize tezine pomera tako da stoje pocev od pozicije p.
        Ostali bitovi su nule. Sa (y & poslednjih_n_1) postave na 0 svi
82      bitovi osim najnizih n */
        unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);

84      return x_postavi_0 ^ y_pomeri_srednje;
86  }

88  /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
    njih n */
90  unsigned invertuj(unsigned x, unsigned n, unsigned p)

```

```
{
92  /* Formira se maska sa n jedinica pocev od pozicije p. */
   unsigned maska = ~(~0 << n) << (p - n + 1);
94
   /* Operator ekskluzivno ili invertuje sve bitove gde je
96      odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
   return maska ^ x;
98 }

100 int main()
   {
102     unsigned x, p, n, y;

104     /* Ucitavaju se vrednosti sa standardnog ulaza */
     printf("Unesite neoznaceni broj x:\n");
106     scanf("%u", &x);
     printf("Unesite neoznaceni broj n:\n");
108     scanf("%u", &n);
     printf("Unesite neoznaceni broj p:\n");
110     scanf("%u", &p);
     printf("Unesite neoznaceni broj y:\n");
112     scanf("%u", &y);

114     /* Ispisuju se binarne reprezentacije broja x i broja koji se
        dobije kada se primeni funkcija postavi_0 za x, n i p */
116     printf("x = %10u %36s = ", x, "");
        stampaj_bitove(x);
118     printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
        stampaj_bitove(postavi_0(x, n, p));
120     printf("\n");

122     /* Ispisuju se binarne reprezentacije broja x i broja koji se
        dobije kada se primeni funkcija postavi_1 za x, n i p */
124     printf("x = %10u %36s = ", x, "");
        stampaj_bitove(x);
126     printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
        stampaj_bitove(postavi_1(x, n, p));
128     printf("\n");

130     /* Ispisuju se binarne reprezentacije broja x i broja koji se
        dobije kada se primeni funkcija vrati_bitove za x, n i p */
132     printf("x = %10u %36s = ", x, "");
        stampaj_bitove(x);
134     printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
        stampaj_bitove(vrati_bitove(x, n, p));
136     printf("\n");

138     /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
        dobije kada se primeni funkcija postavi_1_n_bitova za x, n, p */
140     printf("x = %10u %36s = ", x, "");
        stampaj_bitove(x);
142     printf("y = %10u %36s = ", y, "");
```



```

144     stampaj_bitove(y);
145     printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
146     stampaj_bitove(postavi_1_n_bitova(x, n, p, y));
147     printf("\n");
148
149     /* Ispisuju se binarne reprezentacije broja x i broja koji se
150        dobije kada se primeni funkcija invertuj za x, n i p */
151     printf("x = %10u %36s = ", x, "");
152     stampaj_bitove(x);
153     printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
154     stampaj_bitove(invertuj(x, n, p));
155
156     return 0;
157 }

```

### 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_u_levo(int x, unsigned n)
6  {
7      unsigned bit_najvece_tezine;
8
9      /* Maska koja ima samo bit na poziciji najveće težine postavljen
10         na 1 je neophodna da bi pre pomeranja u levo za 1 bit na
11         poziciji najveće težine bio sacuvan */
12      unsigned bit_najvece_tezine_maska =
13          1 << (sizeof(unsigned) * 8 - 1);
14      int i;
15
16      /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
17         se odredi bit na poziciji najveće težine, a potom se pomera
18         binarna reprezentacija trenutne vrednosti promenljive x u levo
19         za 1. Nakon toga, bit na poziciji najmanje težine se postavlja
20         na vrednost koju je imao bit na poziciji najveće težine 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
27      /* Vraca se dobijena vrednost */
28      return x;
29  }
30
31  /* Funkcija neoznaceni broj x rotira u desno za n mesta. */
32  unsigned rotiraj_u_desno(unsigned x, unsigned n)

```

```
{
34  unsigned bit_najmanje_tezine;
35  int i;
36
37  /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
38     iteraciji se odredjuje bit na poziciji najmanje tezine broja x,
39     zatim tako odredjeni bit se pomera u levo tako da bit na
40     poziciji najmanje tezine dodje do pozicije najvece tezine.
41     Zatim, nakon pomeranja binarne reprezentacije trenutne vrednosti
42     promenljive x za 1 u desno, bit na poziciji najvece tezine se
43     postavlja na vrednost vec zapamcenog bita koji je bio na
44     poziciji
45     najmanje tezine. */
46  for (i = 0; i < n; i++) {
47      bit_najmanje_tezine = x & 1;
48      x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
49  }
50
51  /* Vraca se dobijena vrednost */
52  return x;
53 }
54
55 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
56    argument funkcije x oznaceni ceo broj */
57 int rotiraj_udesno_oznaceni(int x, unsigned n)
58 {
59     unsigned bit_najmanje_tezine;
60     int i;
61
62     /* U svakoj iteraciji se odredjuje bit na poziciji najmanje tezine
63        i smesta u promenljivu bit_najmanje_tezine. Kako je x oznacen
64        ceo broj, tada se prilikom pomeranja u desno vrši aritmeticko
65        pomeranje i cuva se znak broja. Dakle, razlikuju se dva slucaja
66        u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
67        izvrši pre petlje, s obzirom da rotiranjem u desno na poziciju
68        najvece tezine moze doći i 0 i 1, nezavisno od pocetnog znaka
69        broja smestenog u promenljivu x. */
70     for (i = 0; i < n; i++) {
71         bit_najmanje_tezine = x & 1;
72
73         if (x < 0)
74             /******
75              Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
76              na poziciji najvece tezine. Na primer ako je x
77              1010 1011 1100 1101 1110 0001 0010 001b
78              (sa b je oznacen ili 1 ili 0 na poziciji najmanje tezine)
79              Onda je sadržaj promenljive bit_najmanje_tezine:
79              0000 0000 0000 0000 0000 0000 0000 000b
80              Nakon siftovanja sadržaja promenljive x za 1 u desno
80              1101 0101 1110 0110 1111 0000 1001 0001
81              Kako bi umesto 1 na poziciji najvece tezine u trenutnoj binarnoj
82              reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
```

```

84     poziciju najveće težine jer bi se time dobile 0, a u ovom slučaju
85     su potrebne jedinice zbog bitovskog & zato se prvo vrši
86     komplementiranje, a zatim pomeranje
87     ~bit_najmanje_tezine << (sizeof(int)*8 -1)
88     B000 0000 0000 0000 0000 0000 0000 0000
89     gde B označava ~b.
90     Potom se ponovo vrši komplementiranje kako bi se b nalazilo na
91     poziciji najveće težine i sve jedinice na ostalim pozicijama
92     ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
93     b111 1111 1111 1111 1111 1111 1111 1111
94     *****/
95     x = (x >> 1) & ~(~bit_najmanje_tezine <<
96         (sizeof(int) * 8 - 1));
97
98     else
99         x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
100 }
101
102 /* Vraća se dobijena vrednost */
103 return x;
104 }
105
106 int main()
107 {
108     unsigned x, n;
109
110     /* Učitavaju se vrednosti sa standardnog ulaza */
111     printf("Unesite neoznačen ceo broj x:");
112     scanf("%x", &x);
113     printf("Unesite neoznačen ceo broj n:");
114     scanf("%x", &n);
115
116     /* Ispisuje se binarna reprezentacija broja x */
117     printf("x\t\t\t\t\t= ");
118     stampaj_bitove(x);
119
120     /* Testiraju se napisane funkcije */
121     printf("rotiraj_ulevo(%x,%u)\t\t= ", x, n);
122     stampaj_bitove(rotiraj_ulevo(x, n));
123
124     printf("rotiraj_udesno(%x,%u)\t\t= ", x, n);
125     stampaj_bitove(rotiraj_udesno(x, n));
126
127     printf("rotiraj_udesno_oznaceni(%x,%u)\t\t= ", x, n);
128     stampaj_bitove(rotiraj_udesno_oznaceni(x, n));
129
130     printf("rotiraj_udesno_oznaceni(-%x,%u)\t\t= ", x, n);
131     stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));
132
133     return 0;
134 }

```

### 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 tekuće
13        vrednosti broja x se određuje i pamti u promenljivoj
14        najnizi_bit, nakon čega 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
20            prethodno postavljeni bitovi dobijaju veću poziciju. Novi bit
21            se postavlja na najnižu 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
44     return 0;
45 }
```

## Rešenje 1.11

```
1  #include <stdio.h>
2
3  /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n
4     broj jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
5  int broj_01(unsigned int n)
6  {
7      int broj_nula, broj_jedinica;
8      unsigned int maska;
9
10     broj_nula = 0;
11     broj_jedinica = 0;
12
13     /* Maska je inicijalizovana tako da moze da analizira bit najvece
14        tezine */
15     maska = 1 << (sizeof(unsigned int) * 8 - 1);
16
17     /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
18        iteraciji pomera u desno pa ce jedini bit koji je postavljen na
19        1 biti na svim pozicijama u binarnoj reprezentaciji maske */
20     while (maska != 0) {
21         /* Provera da li se na poziciji koju odredjuje maska nalazi 0
22            ili 1 i uveca se odgovarajuci 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
38 int main()
39 {
40     unsigned int n;
41
42     /* Ucitava se broj */
43     scanf("%u", &n);
44
45     /* Ispisuje se rezultat */
46     printf("%d\n", broj_01(n));
47
48     return 0;
49 }
```

### Rešenje 1.12

```
1  #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 procita 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         /* Pomerava se broj u desnu stranu da bi se u narednoj iteraciji
27            proveravao sledeci par bitova. Pomeranjem u desno bit najvece
28            tezine se popunjava nulom jer je x neoznaceni broj */
29         x = x >> 1;
30     }
31
32     /* Vraca se dobijena vrednost */
33     return broj_parova;
34 }
35
36 int main()
37 {
38     unsigned int x;
39
40     /* Ucitava se broj */
41     scanf("%u", &x);
42
43     /* Ispisuje se rezultat */
44     printf("%d\n", broj_parova(x));
45
46     return 0;
47 }
```

## Rešenje 1.14

```

#include <stdio.h>

/* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
   su za svaku heksadekadnu cifru potrebne 4 binarne cifre i jedna
   dodatna pozicija za terminirajucu nulu. Prethodni izraz se moze
   zapisati kao sizeof(unsigned int)*2+1. */
#define MAKS_DUZINA sizeof(unsigned int)*2 +1

/* Funkcija za neoznacen broj x formira nisku s koja sadrzi njegov
   heksadekadni zapis */
void prevod(unsigned int x, char s[])
{
    int i;
    unsigned int maska;
    int vrednost;

    /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
       maska za citanje 4 uzastopne cifre */
    maska = 15;

    /******
       Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
       tezine. Na primer za broj cija je binarna reprezentacija
       00000000001101000100001111010101
       u prvom koraku se citaju bitovi izdvojeni sa <...>:
       0000000000110100010000111101<0101>
       u drugom koraku:
       000000000011010001000011<1101>0101
       u trecem koraku:
       00000000001101000100<0011>11010101 i tako redom...

       Indeks i oznacava poziciju na koju se smesta vrednost.
       *****/
    for (i = MAKS_DUZINA - 2; i >= 0; i--) {
        /* Vrednost izdvojene cifre */
        vrednost = x & maska;

        /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
           dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega
           od 10 do 15 odgovarajuci karakter se dobija tako sto se prvo
           oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
           dobijenu vrednost doda ASCII kod 'A' (time se dobija
           odgovarajuce slovo 'A', 'B', ... 'F') */
        if (vrednost < 10) {
            s[i] = vrednost + '0';
        } else {
            s[i] = vrednost - 10 + 'A';
        }

        /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u

```

## 1 Uvodni zadaci

```

    narednoj iteraciji posmatraju sledeca 4 bita */
52     x = x >> 4;
    }
54
    /* Upisuje se terminirajuca nula */
56     s[MAKS_DUZINA - 1] = '\0';
    }
58
59 int main()
60 {
    unsigned int x;
62     char s[MAKS_DUZINA];

64     /* Ucitava se broj sa ulaza */
    scanf("%u", &x);

66
    /* Poziva se funkcija za prevodjenje */
68     prevod(x, s);

70     /* I stampa se dobijena niska */
    printf("%s\n", s);

72
73     return 0;
74 }
```

### Rešenje 1.17

```

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

4  /*****
    Resenje linearne slozenosti:
6     x^0 = 1
     x^k = x * x^(k-1)
8  *****/
int stepen(int x, int k)
10 {
    if (k == 0)
12         return 1;

14     return x * stepen(x, k - 1);
    /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
16 }

18 /*****
    Resenje logaritamske slozenosti:
20     x^0 =1;
     x^k = x * (x^2)^(k/2) , za neparno k
22     x^k = (x^2)^(k/2) , za parno k
     Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
24     se doslo do rezultata, i stoga je efikasnije.
```



```

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      /* Ucitavanje rednog broja funkcije koja ce se primeniti */
44      printf("Unesite redni broj funkcije (1/2):\n");
45      scanf("%d", &ind);
46
47      /* Ucitavanje vrednosti za x i k */
48      printf("Unesite broj x:\n");
49      scanf("%d", &x);
50      printf("Unesite broj k:\n");
51      scanf("%d", &k);
52
53      /* Ispisuje se vrednost koju vraca odgovarajuca 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, "Greska: 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
4   (posrednu) rekurziju */
5
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

```

## 1 Uvodni zadaci

---

```
    paran. */
10 unsigned neparan(unsigned n);

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

22 /* Funkcija neparan vraca 1 ako broj n ima neparan broj cifara,
    inace vraca 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);

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

42     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 faktoriijela. Kada dodje do izlaza iz
7   rekurzije iz rekurzije potrebno je da vratimo result. */
8 int faktoriyel_repna(int n, int result)
9 {
10     if (n == 0)
11         return result;
```

```
13     return faktorijel_repna(n - 1, n * result);
14 }
15
16 /* U sledecim funkcijama je prikazan postupak oslobadjanja od repne
17    rekurzije koja postoji u funkciji faktorijel_repna. */
18
19 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
20    naredbama kojima se vrednost argumenta funkcije postavlja na
21    vrednost koja bi se prosledjivala rekurzivnom pozivu i
22    navodjenjem goto naredbe za vracanje na pocetak tela funkcije. */
23 int faktorijel_repna_v1(int n, int result)
24 {
25     pocetak:
26         if (n == 0)
27             return result;
28
29     result = n * result;
30     n = n - 1;
31     goto pocetak;
32 }
33
34 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
35    programerska praksa i prethodna funkcija se koristi samo kao
36    medjukorak. Sledi iterativno resenje bez bezuslovnih skokova */
37 int faktorijel_repna_v2(int n, int result)
38 {
39     while (n != 0) {
40         result = n * result;
41         n = n - 1;
42     }
43
44     return result;
45 }
46
47 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
48    broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
49    ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
50    udobnosti korisnika, jer je sasvim prirodno da za faktorijel
51    zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
52    sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
53    faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
54    funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
55    poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
56    pozivom funkcije faktorijel_repna_v2. */
57 int faktorijel(int n)
58 {
59     return faktorijel_repna(n, 1);
60 }
61
62 int main()
63 {
64     int n;
```

## 1 Uvodni zadaci

---

```
65  /* Ucitava se ceo broj */
67  printf("Unesite n (<= 12): ");
    scanf("%d", &n);
69  if (n > 12) {
    fprintf(stderr, "Greska: Nedozvoljena vrednost za n.\n");
71      exit(EXIT_FAILURE);
    }
73
    /* Ispisuje se rezultat */
75  printf("%d! = %d\n", n, faktorijel(n));
77  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.21

```
1  #include <stdio.h>

3  /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
    int f_iterativna(int n, int a, int b)
5  {
    int i;
7    int f_0 = 0;
    int f_1 = 1;
9    int tmp;

11   if (n == 0)
        return 0;

13   for (i = 2; i <= n; i++) {
15       tmp = a * f_1 + b * f_0;
        f_0 = f_1;
17       f_1 = tmp;
    }

19   return f_1;
21 }

23 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
    int f_rekurzivna(int n, int a, int b)
25 {
    /* Izlaz iz rekurzije */
27   if (n == 0 || n == 1)
        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 slucaju 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.
    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 dimenzija ce se
    resavati jos veci broj puta. Resenje za ovaj problem je
41    kombinacija rekurzije sa dinamicnim programiranjem. Podproblemi se
    resavaju samo jednom, a njihova resenja se pamte u memoriji
43    (obicno u nizovima ili matricama), odakle se koriste ako tokom
    resavanja ponovo budu potrebni.

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

51 /* c) Funkcija racuna n-ti broj niza F - efikasnija 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,
57        elemente celobrojnog niza inicijalizuje na 0 */
58     static int f[20];

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

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

69     /* Rekurzivni pozivi */
70     return f[n] =
71         a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
72 }

75 int main()
76 {
77     int n, a, b, ind;

79     /* Unosi se redni broj funkcije koja ce se primeniti */
80     printf("Unesite redni broj funkcije:\n");
81     printf("1 - iterativna\n");
82     printf("2 - rekurzivna\n");
83     printf("3 - rekurzivna napredna\n");
84     scanf("%d", &ind);

85     /* Ucitaju se koeficijenti a i b */

```

## 1 Uvodni zadaci

---

```
87  printf("Unesite koeficijente:\n");
    scanf("%d%d", &a, &b);
89
    /* Ucitava se broj n */
91  printf("Unesite koji clan niza se racuna:\n");
    scanf("%d", &n);
93
    /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
95     funkcije f_iterativna, f_rekurzivna ili f_napredna */
    if (ind == 0)
97         printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
    else if (ind == 1)
99         printf("F(%d) = %d\n", n, f_rekurzivna(n, a, b));
    else
101         printf("F(%d) = %d\n", n, f_napredna(n, a, b));
103
    return 0;
}
```

### Rešenje 1.22

```
#include <stdio.h>
2
/* Funkcija odredjuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
{
6     /* Izlazak iz rekurzije: ako je broj jednocifren */
    if (x < 10)
8         return x;

10     /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
        poslednje cifre + poslednja cifra tog broja */
12     return zbir_cifara(x / 10) + x % 10;
}
14
16 int main()
{
    unsigned int x;
18
    /* Ucitava se ceo broj */
20     scanf("%u", &x);

22     /* Ispisuje se zbir cifara ucitanog broja */
    printf("%d\n", zbir_cifara(x));
24
    return 0;
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:\n");
38     scanf("%d", &n);
39
40     /* Ucitava se n elemenata niza. */
41     printf("Unesite elemente niza:\n");
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, odnosno suma_niza_2 */
47     if (ind == 1)
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));
```

```
52     else {  
53         fprintf(stderr, "Greska: Neodgovarajuci redni broj funkcije.\n");  
54         exit(EXIT_FAILURE);  
55     }  
56     exit(EXIT_SUCCESS);  
57 }
```

### Rešenje 1.24

```
2  #include <stdio.h>  
3  #define MAKS_DIM 256  
4  /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz  
   * dimenzije n */  
6  int maksimum_niza(int niz[], int n)  
7  {  
8      /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je  
       * ujedno i jedini element niza */  
10     if (n == 1)  
11         return niz[0];  
12  
13     /* Resava se problem manje dimenzije */  
14     int max = maksimum_niza(niz, n - 1);  
15  
16     /* Na osnovu poznatog resenja problema dimenzije n-1, resava se  
       * problem dimenzije n */  
18     return niz[n - 1] > max ? niz[n - 1] : max;  
19 }  
20  
21 int main()  
22 {  
23     int brojevi[MAKS_DIM];  
24     int n;  
25  
26     /* Sve dok se ne stigne do kraja ulaza, brojevi 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. */  
30     int i = 0;  
32     while (scanf("%d", &brojevi[i]) != EOF) {  
33         i++;  
34         if (i == MAKS_DIM)  
35             break;  
36     }  
37     n = i;  
38  
39     /* Stampa se maksimum unetog niza brojeva */  
40     printf("%d\n", maksimum_niza(brojevi, n));
```



```

42     return 0;
    }

```

### Rešenje 1.25

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #define MAKS_DIM 256

5  /* Funkcija koja izracunava skalarni proizvod dva data vektora */
   int skalarno(int a[], int b[], int n)
7  {
   /* Izlazak iz rekurzije: vektori su duzine 0 */
9   if (n == 0)
       return 0;

11  /* Na osnovu resenja problema dimenzije n-1, resava se problem
   dimenzije n primenom definicije skalarnog proizvoda a*b =
13  a[0]*b[0] + a[1]*b[1] + ... + a[n-2]*a[n-2] + a[n-1]*a[n-1]
   Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
15  na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
   polazna dva vektora otklanjanjem poslednjih elemenata, doda
17  proizvod poslednja dva elementa polaznih vektora. */
19  else
       return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
21 }

23 int main()
   {
25     int i, a[MAKS_DIM], b[MAKS_DIM], n;

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

31     /* Provera da li je dimenzija niza odgovarajuca */
       if (n < 0 || n > MAKS_DIM) {
33         fprintf(stderr,
           "Greska: Dimenzija mora biti prirodan broj <= %d!\n",
35             MAKS_DIM);
           exit(EXIT_FAILURE);
37     }

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

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

```

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

51  exit(EXIT_SUCCESS);
}
```

### 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;
}
```

```

44  /* Ispisuje se broj pojavljivanja */
    printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
46
    return 0;
48 }

```

### Rešenje 1.27

```

1  #include <stdio.h>
   #define MAKS_DIM 256
3
   /* Funkcija koja proverava da li su tri zadata broja uzastopni
   5   clanovi niza */
   int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
7   {
   /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i
   9   vraca se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi
      niza */
11  if (n < 3)
      return 0;
13
   /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
15   dovoljno je da su oni poslednja tri clana niza ili da se oni
      rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
17  return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
      || tri_uzastopna_clana(x, y, z, a, n - 1);
19 }

21 int main()
   {
23   int x, y, z, a[MAKS_DIM];
      int n;
25
   /* Ucitavaju se tri cela broja za koje se ispituje da li su
27   uzastopni clanovi niza */
   printf("Unesite tri cela broja:");
29   scanf("%d%d%d", &x, &y, &z);

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

```

## 1 Uvodni zadaci

```
43  n = i;

45  /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana
    ispisuje se odgovarajuca poruka */
47  if (tri_uzastopna_clana(x, y, z, a, n))
    printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
49  else
    printf("Uneti brojevi nisu uzastopni clanovi niza.\n");

51  return 0;
53 }
```

### Rešenje 1.28

```
#include <stdio.h>

2
/* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
{
6     /* Izlaz iz rekurzije */
    if (x == 0)
8         return 0;

10     /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x
        je postavljen na 1. Koriscenjem odgovarajuce maske proverava se
12     vrednost bita na poziciji najvece tezine i na osnovu toga se
        razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
14     broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
        x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
16     na poziciji najvece tezine jedinica, rezultat dobijen
        rekurzivnim
        pozivom funkcije za x<<1 treba uvecati za jedan. Za rekurzivni
18     poziv se salje vrednost koja se dobija kada se x pomeri u levo.
        Napomena: argument funkcije x je oznacen ceo broj, usled cega se
20     ne koristi pomeranje udesno, jer funkciji moze biti prosledjen i
        negativan broj. Iz tog razloga, odlucujemo se da proveramo
22     najvisi, umesto najnizeg bita */
    if (x & (1 << (sizeof(x) * 8 - 1)))
24         return 1 + prebroj(x << 1);
    else
26         return prebroj(x << 1);
    /*
    Krace zapisano
    return ((x & (1<<(sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
    */
30 }

32 int main()
34 {
    int x;

36 }
```

```

38  /* Ucitava se ceo broj */
    scanf("%x", &x);

40  /* Ispisuje se rezultat */
    printf("%d\n", prebroj(x));

42

44  return 0;
}

```

### Rešenje 1.30

```

#include <stdio.h>

2
/* Rekurzivna funkcija za odredjivanje najveće oktalne cifre */
4 int maks_oktalna_cifra(unsigned x)
{
6     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
        vrednost najveće oktalne cifre u broju 0 */
8     if (x == 0)
        return 0;

10
12     /* Odredjuje se poslednja oktalna cifra u broju */
    int poslednja_cifra = x & 7;

14     /* Odredjuje se maksimalna oktalna cifra u broju kada se iz njega
        izbrise poslednja oktalna cifra */
16     int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);

18     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
        : maks_bez_poslednje_cifre;
20 }

22 int main()
{
24     unsigned x;

26     /* Ucitava se neoznaceni ceo broj */
    scanf("%u", &x);

28
30     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
    printf("%d\n", maks_oktalna_cifra(x));

32
    return 0;
}

```

### Rešenje 1.31

```

#include <stdio.h>

2

```

## 1 Uvodni zadaci

---

```
/* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre */
4 int maks_heksadekadna_cifra(unsigned x)
{
6     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
       vrednost najveće heksadekadne cifre u broju 0 */
8     if (x == 0)
        return 0;
10
12     /* Odredjuje se poslednja heksadekadna cifra u broju */
    int poslednja_cifra = x & 15;

14     /* Odredjuje se maksimalna heksadekadna cifra broja kada se iz
       njega izbrise poslednja heksadekadna cifra */
16     int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);

18     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
        : maks_bez_poslednje_cifre;
20 }

22 int main()
{
24     unsigned x;

26     /* Ucitava se neoznaceni ceo broj */
    scanf("%u", &x);

28
30     /* Ispisivanje vrednosti najveće heksadekadne cifre unetog broja */
    printf("%d\n", maks_heksadekadna_cifra(x));

32     return 0;
}
```

### Rešenje 1.32

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

4 /* Niska može imati najviše 31 karaktera + 1 za terminalnu nulu */
#define MAKS_DIM 32

6 /* Funkcija ispituje da li je zadata niska dužine n palindrom */
8 int palindrom(char s[], int n)
{
10     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
       palindrom */
12     if ((n == 1) || (n == 0))
        return 1;
14
16     /* 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 */
}
```

```

18     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
19 }
20
21 int main()
22 {
23     char s[MAKS_DIM];
24     int n;
25
26     /* Ucitava se niska sa standardnog ulaza */
27     scanf("%s", s);
28
29     /* Odredjuje se duzina niske */
30     n = strlen(s);
31
32     /* Ispisuje se da li je niska palindrom ili nije */
33     if (palindrom(s, n))
34         printf("da\n");
35     else
36         printf("ne\n");
37
38     return 0;
39 }

```

### 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 vraca 1 ako se broj x nalazi u nizu a duzine n, a
16    inace 0 */
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

```

```
/* Zakljucuje se da broj nije pronadjen */
28 return 0;
}

30
/* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n}
32 dobija kao argument niz a[] u koji se smesta permutacija, broj m
   oznacava da se u okviru tog poziva funkcije na m-tu poziciju u
34 permutaciji smesta jedan od preostalih celih brojeva, n je
   velicina skupa koji se permutuje. Funkciju se inicijalno poziva
36 sa argumentom m = 1, jer formiranje permutacije pocinje od
   pozicije broj 1. Stoga, a[0] se ne koristi. */
38 void permutacija(int a[], int m, int n)
{
40     int i;

42     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
       premasila velicinu skupa, onda se svi brojevi vec nalaze u
44 permutaciji i ispisuje se permutacija. */
       if (m > n) {
46             ispisi_niz(a, n);
               return;
48     }

50     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
       mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
52 Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
       ovako postavljenom pocetku permutacije. Kada se to zavrshi, vrsi
54 se provera da li postoji jos neki broj koji moze da se stavi na
       m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
56 funkcija zavrшава sa radom. Ukoliko takav broj postoji, onda se
       ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
58 ali sada sa drugacije postavljanim prefiksom. */
       for (i = 1; i <= n; i++) {
60             /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
               pozicije, onda se on postavlja na poziciju m i poziva se
62 ponovo funkcija da dopuni ostatak permutacije posle
               upisivanja i na poziciju m. Inace, nastavlja se dalje,
64 trazeci broj koji se nije pojavio do sada u permutaciji. */
               if (!koriscen(a, m - 1, i)) {
66                     a[m] = i;
                               permutacija(a, m + 1, n);
68             }
       }
70 }

72 int main(void)
{
74     int n;
       int a[MAKS_DUZINA_PERMUTACIJE + 1];

76     /* Ucitava se broj n iz odgovarajuceg opsega */
78     scanf("%d", &n);
```



```

80     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
        fprintf(stderr,
            "Duzina permutacije mora biti broj iz intervala [0, %d]!\n"
            "n",
82             MAKS_DUZINA_PERMUTACIJE);
        exit(EXIT_FAILURE);
84     }

86     /* Ispisuju se permutacije duzine n */
    permutacija(a, 1, n);
88
    exit(EXIT_SUCCESS);
90 }

```

### Rešenje 1.34

```

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

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

16 /******
    Iterativno izracunavanje datog binomnog koeficijenta
18
    int binomni_koeficijent (int n, int k) {
20         int i, j, b;

22         for (b=i=1, j=n; i<=k; b =b * j-- / i++);

24         return b;
    }

26
    Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
28     resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
    *****/

30
    /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
32     zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
    ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
34     veca od sume elemenata prethodne hipotenuze. */
    int suma_elemenata_hipotenuze(int n)

```

```
36 {
37     return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
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     /* Proverava se 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         );
62         exit(EXIT_FAILURE);
63     }
64
65     /* Ispisuje se suma elemenata hipotenuze */
66     printf("%d\n", suma_elemenata_hipotenuze(r));
67
68     exit(EXIT_SUCCESS);
69 }
```

## 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
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuca dimenzija niza.
```

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

- (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 učitano 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.
```

**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
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuća dimenzija niza.
```

### Primer 4

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju niza: 101
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuća dimenzija niza.
```

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

- (a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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:
  .
```

**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
  koji su palindromi je 4.
```

### Primer 2

```
POKRETANJE: ./a.out

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

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

### 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 2

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
  IZLAZ ZA GREŠKE:
  Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

### Primer 3

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

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

INTERAKCIJA SA PROGRAMOM:
IZLAZ ZA GREŠKE:
Greska: Nedovoljan broj
argumenata komandne linije.
Program se poziva sa
./a.out ime_dat br_karaktera
```

**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 završavaju na ke

INTERAKCIJA SA PROGRAMOM:
Broj reci koje se završavaju 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:
IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje
datoteke ulaz.txt.
```

### Primer 4

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

ULAZ.TXT
Ovo je sadrzaj ulaza.

INTERAKCIJA SA PROGRAMOM:
IZLAZ ZA GREŠKE:
Greska: Nedovoljan broj argumenata
komandne linije.
Program se poziva sa
./a.out ime_dat suf/pref -s/-p
```

## 2.2 Višedimenzioni nizovi

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

- 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 učitane matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrice: 3
Unesite elemente matrice po vrstama:
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
IZLAZ ZA GREŠKE:
Greska: Neodgovarajuća dimenzija
matrice.
```

**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 po vrstama:
1 2 3
1 2 3
1 2 3
Unesite elemente druge matrice po vrstama:
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
```

**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.

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

Napisati program koji učitava matricu iz datoteke čije se ime zadaje kao prvi argument komandne linije. U prvoj liniji datoteke nalazi se 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

```

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice 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 najvećim 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 najvećim elementom je 0.
Broj negativnih elemenata matrice je 16.
```

**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 učitanoj matrici.

### Primer 1

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

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

- (a) Napisati funkciju koja učitava elemente matrice sa standardnog ulaza

- (b) Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice, 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  $m$  ( $0 < m \leq 10$ ) 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 po vrstama:
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 po vrstama:
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
```

**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 po vrstama:
1 2 3
4 5 6
7 8 9
Unesite stepen koji se racuna: 8
8. stepen matrice je:
510008400 626654232 743300064
1154967822 1419124617 1683281412
1799927244 2211595002 2623262760
```

## 2.3 Dinamička alokacija memorije

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

### Primer 1

```
INTERAKCIJA 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
Greska: Neuspesna alokacija memorije.
```

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

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

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
```

**Zadatak 2.17** Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera (pretpostaviti da niske nisu duže od 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
```

**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:
2 3
Unesite elemente matrice po vrstama:
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:
2 2
Unesite elemente matrice po vrstama:
-0.1 -0.2
-0.3 -0.4
Trag unete matrice je -0.50.
```

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

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

## 2 Pokazivači

- (j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzije  $n \times m$  u već otvorenu datoteku `f`. U slučaju neuspešnog 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 neuspešnog 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:
Unesite broj vrsta matrice: 3
Unesite broj kolona matrice: 4
Unesite 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:
Unesite broj vrsta matrice: 5
Unesite broj kolona matrice: 0
IZLAZ ZA GREŠKE:
Greska: Broj vrsta i broj kolona ne mogu
biti negativni brojevi.
```

- (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 ZA GREŠKE:
Greska: Neispravan pocetak
datoteke.
```

### Test 3

```
POKRETANJE: ./a.out

IZLAZ:
Koriscenje programa:
./a.out datoteka
```

**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 za rad sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

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

**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 za rad sa celobrojnim matricama iz zadatka 2.19.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta i broj kolona:
2 3
Unesite elemente matrice po vrstama:
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:
2 4
Unesite elemente matrice po vrstama:
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.

## 2 Pokazivači

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
```

**Zadatak 2.23** Napisati program koji na osnovu dve realne matrice dimenzije  $m \times n$  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 datoteke se nalaze broj vrsta  $m$  i broj kolona  $n$  matrica, 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 ulevo. Napisati program koji testira ovu funkciju. Rezultujuću matricu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad 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`  
 Pomozite 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: *Koristiti `realloc()` funkciju za realokaciju memorije.*

### Primer 1

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

### INTERAKCIJA SA PROGRAMOM:

```
Petru ukupno nedostaje 7 sličica.
Reprezentacija za koju je sakupio
najmanji broj sličica je Brazil.
```

\* **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
-1 -1
1 -1
1 1
-1 1
```

### INTERAKCIJA SA PROGRAMOM:

```
U datoteci su zadata temena
cetvougla.
Obim je 8.
Povrsina je 4.
```

### Primer 2

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

### INTERAKCIJA SA PROGRAMOM:

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

## 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  $[a, b]$ . Re-

## 2 Pokazivači

alni brojevi  $a$  i  $b$  ( $a < b$ ) kao i ceo broj  $n$  ( $n \geq 2$ ) se učitavaju sa standardnog ulaza. Ime funkcije se zadaje kao argument komandne linije (`sin`, `cos`, `tan`, `atan`, `acos`, `asin`, `exp`, `log`, `log10`, `sqrt`, `floor`, `ceil`, `sqr`).

### Primer 1

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

### Primer 2

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

**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 }

```

## 2 Pokazivači

---

```
16 }
18 /* Funkcija obrne 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, cime pokazivac "poslednji" pokazuje na
38            element koji mu prethodi u nizu */
39         *poslednji-- = t;
40     }
41
42     /******
43     Drugi nacin za obrtanje niza
44
45     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
46          prvi++, poslednji--) {
47
48         int t = *prvi;
49         *prvi = *poslednji;
50         *poslednji = t;
51     }
52     *****/
53 }
54
55 int main()
56 {
57     /* Deklarise se niz od najvise MAX elemenata */
58     int a[MAX];
59
60     /* Broj elemenata niza a */
61     int n;
62
63     /* Pokazivac na elemente niza */
64     int *p;
65
66     printf("Unesite dimenziju niza: ");
67     scanf("%d", &n);
```

```

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

74

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

78

    obrni_niz_v1(a, n);

80

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

82

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

86

    obrni_niz_v2(a, n);

88

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

90

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

94

    exit(EXIT_SUCCESS);
96 }

```

## Rešenje 2.2

```

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

4  #define MAX 100

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

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

14   return s;
    }

16

/* Funkcija izracunava proizvod elemenata niza */
18 double proizvod(double *a, int n)

```

```
{
20  double p = 1;

22  for (; n; n--)
    p *= (a + n - 1);

24  return p;
26 }

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

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

40  return min;
42 }

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

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

56  return max;
58 }

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

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

68  /* Proverava se da li je doslo do prekoračenja ograničenja
70     dimenzije */
    if (n <= 0 || n > MAX) {
```

```

72     fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
    exit(EXIT_FAILURE);
}

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

```

### Rešenje 2.3

```

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

4  #define MAX 100

6  /* Funkcija povecava za jedan sve elemente u prvoj polovini niza a
   smanjuje za jedan sve elemente u drugoj polovini niza. Ukoliko
   niz ima neparan broj elemenata, srednji element ostaje
   nepromenjen */
10 void povecaj_smanji(int *a, int n)
   {
12     int *prvi = a;
     int *poslednji = a + n - 1;
14
     while (prvi < poslednji) {
16
18         /* Uvecava se element na koji pokazuje pokazivac "prvi" */
         (*prvi)++;

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

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

26         /* Pokazivac "poslednji" se pomera na prethodni element */
28         poslednji--;
   }
30
   /******
32   Drugi nacin:

```

```
34     while (prvi < poslednji) {
35         (*prvi++)++;
36         (*poslednji--)--;
37     }
38     /* **** */
39 }
40 int main()
41 {
42     int a[MAX];
43     int n;
44     int *p;
45
46     printf("Unesite dimenziju niza: ");
47     scanf("%d", &n);
48
49     /* Proverava se da li je doslo do prekoračenja ograničenja
50        dimenzije */
51     if (n <= 0 || n > MAX) {
52         fprintf(stderr, "Greska: Neodgovarajuća dimenzija niza.\n");
53         exit(EXIT_FAILURE);
54     }
55
56     printf("Unesite elemente niza:\n");
57     for (p = a; p - a < n; p++)
58         scanf("%d", p);
59
60     povecaj_smanji(a, n);
61
62     printf("Transformisan niz je:\n");
63     for (p = a; p - a < n; p++)
64         printf("%d ", *p);
65     printf("\n");
66
67     exit(EXIT_SUCCESS);
68 }
```

### 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);
```



```

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     argc = i;
35 }
36
37 printf("Pocetna slova argumenata komandne linije:\n");
38 if (tip_ispisa == 'I') {
39     /* koristeci indeksnu sintaksu */
40     for (i = 0; i < argc; i++)
41         printf("%c ", argv[i][0]);
42     printf("\n");
43 } else if (tip_ispisa == 'P') {
44     /* koristeci pokazivacku sintaksu */
45     for (i = 0; i < argc; i++)
46         printf("%c ", **argv++);
47     printf("\n");
48 }
49
50 return 0;
51 }

```

## Rešenje 2.5

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX 100
5
6 /* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
7    cita spreda i odpozadi */
8 int palindrom(char *niska)
9 {

```

```
11  int i, j;
    for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
        if (*(niska + i) != *(niska + j))
13      return 0;
    return 1;
15 }

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

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

25     printf("Broj argumenata koji su palindromi je %d.\n", n);

27     return 0;
29 }
```

### Rešenje 2.6

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

3

5  #define MAX_KARAKTERA 100

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

13

15 int main(int argc, char **argv)
{
    char rec[MAX_KARAKTERA + 1];
17    int br = 0, n;
    FILE *in;

19

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

```

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

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

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

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

49  /* Zatvara se datoteka */
    fclose(in);
51
    exit(EXIT_SUCCESS);
53 }

```

## Rešenje 2.7

```

1  #include <stdio.h>
    #include <stdlib.h>
3
    #define MAX_KARAKTERA 100
5
    /* Implementacija funkcije strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
    {
9      int i;
        for (i = 0; *(src + i); i++)
11         *(dest + i) = *(src + i);
    }
13
    /* Implementacija funkcije strcmp() iz standardne biblioteke */
15  int poredjenje_niski(char *s, char *t)
    {
17      int i;
        for (i = 0; *(s + i) == *(t + i); i++)
19         if (*(s + i) == '\0')
            return 0;
21         return *(s + i) - *(t + i);
    }
23

```

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

31
/* Funkcija ispituje da li je niska zadata drugim argumentom
33 funkcije sufixs niske zadate prvi argumentom funkcije */
int sufixs_niske(char *niska, char *sufiks)
35 {
    int duzina_sufiksa = duzina_niske(sufiks);
37     int duzina_niske_pom = duzina_niske(niska);
    if (duzina_sufiksa <= duzina_niske_pom &&
39         poredjenje_niski(niska + duzina_niske_pom -
                           duzina_sufiksa, sufixs) == 0)
41         return 1;
    return 0;
43 }

45
/* Funkcija ispituje da li je niska zadata drugim argumentom
    funkcije prefiks niske zadate prvi argumentom funkcije */
47 int prefiks_niske(char *niska, char *prefiks)
{
49     int i;
    int duzina_prefiksa = duzina_niske(prefiks);
51     int duzina_niske_pom = duzina_niske(niska);
    if (duzina_prefiksa <= duzina_niske_pom) {
53         for (i = 0; i < duzina_prefiksa; i++)
            if (*(prefiks + i) != *(niska + i))
55                 return 0;
        return 1;
57     } else
        return 0;
59 }

61 int main(int argc, char **argv)
{
63     /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
        greska */
65     if (argc < 4) {
        fprintf(stderr, "Greska: ");
67         fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
        fprintf(stderr, "Program se poziva sa\n");
69         fprintf(stderr, "%s ime_dat suf/pref -s/-p\n", argv[0]);
        exit(EXIT_FAILURE);
71     }

73     FILE *in;
    int br = 0;
75     char rec[MAX_KARAKTERA + 1];
```

```

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

83
    /* Provera se opcija kojom je pozvan program a zatim se učitavaju
85     reci iz datoteke i broji se koliko njih zadovoljava trazeni
        uslov */
87     if (!(poredjenje_niski(*(argv + 3), "-s"))) {
        while (fscanf(in, "%s", rec) != EOF)
89         br += sufiks_niske(rec, *(argv + 2));
        printf("Broj reci koje se završavaju na %s je %d.\n",
91             *(argv + 2), br);
    } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
93         while (fscanf(in, "%s", rec) != EOF)
            br += prefiks_niske(rec, *(argv + 2));
95         printf("Broj reci koje počinju na %s je %d.\n", *(argv + 2), br);
    }

97     fclose(in);

99     exit(EXIT_SUCCESS);
101 }

```

## Rešenje 2.8

```

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

5  #define MAX 100

7  /* Funkcija izracunava trag matrice */
    int trag(int M[][MAX], int n)
9  {
        int trag = 0, i;
11     for (i = 0; i < n; i++)
            trag += M[i][i];
13     return trag;
    }

15
    /* Funkcija izracunava euklidsku normu matrice */
17     double euklidska_norma(int M[][MAX], int n)
    {
19         double norma = 0.0;
            int i, j;
21
            for (i = 0; i < n; i++)

```

```

23     for (j = 0; j < n; j++)
        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()
44 {
45     int A[MAX][MAX];
46     int i, j, n;
47
48     printf("Unesite broj vrsta matrice: ");
49     scanf("%d", &n);
50
51     /* Provera prekoracenja dimenzije matrice */
52     if (n > MAX || n <= 0) {
53         fprintf(stderr, "Greska: Neodgovarajuca dimenzija matrice.\n");
54         exit(EXIT_FAILURE);
55     }
56
57     printf("Unesite elemente matrice po vrstama:\n ");
58     for (i = 0; i < n; i++)
59         for (j = 0; j < n; j++)
60             scanf("%d", &A[i][j]);
61
62     /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
63     for (i = 0; i < n; i++) {
64         /* Ispis elemenata i-te vrste */
65         for (j = 0; j < n; j++)
66             printf("%d ", A[i][j]);
67         printf("\n");
68     }
69
70     /******
71     Ispisuju se elementi 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 slede
       svi elementi te vrste i tako dalje redom.
77
       for( i = 0; i < n ; i++) {
79         for ( j=0 ; j<n ; j++)
           printf("%d ", *((A+i)+j));
81         printf("\n");
       }
83     *****/

85     /* Ispisuje se rezultat na standardni izlaz */
     int tr = trag(A, n);
87     printf("Trag matrice je %d.\n", tr);

89     printf("Euklidska norma matrice je %.2f.\n",
           euklidska_norma(A, n));
91     printf("Vandijagonalna norma matrice je = %d.\n",
           gornja_vandijagonalna_norma(A, n));
93
     exit(EXIT_SUCCESS);
95 }

```

## Rešenje 2.9

```

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

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

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

```

```

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

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

45 /* Funkcija izracunava zbir dve kvadratne matrice */
void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
47 {
    int i, j;
49
    for (i = 0; i < n; i++)
51         for (j = 0; j < n; j++)
            c[i][j] = a[i][j] + b[i][j];
53 }

55 /* Funkcija izracunava proizvod dve kvadratne matrice */
void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
57 {
    int i, j, k;
59
    for (i = 0; i < n; i++)
61         for (j = 0; j < n; j++) {
            /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
63             c[i][j] = 0;
            for (k = 0; k < n; k++)
65                 c[i][j] += a[i][k] * b[k][j];
67         }
69 }

int main()
71 {
    /* Matrice ciji se elementi zadaju sa ulaza */
73     int a[MAX][MAX], b[MAX][MAX];

75     /* Matrice zbira i proizvoda */
    int zbir[MAX][MAX], proizvod[MAX][MAX];
77
    /* Dimenzija kvadratnih matrica */
79     int n;
```



```

81 printf("Unesite broj vrsta matrica:\n");
   scanf("%d", &n);
83
   /* Proverava se da li je doslo do prekoracenja */
85 if (n > MAX || n <= 0) {
   fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
87 fprintf(stderr, "matrica.\n");
   exit(EXIT_FAILURE);
89 }

91 printf("Unesite elemente prve matrice po vrstama:\n");
   ucitaj_matricu(a, n);
93 printf("Unesite elemente druge matrice po vrstama:\n");
   ucitaj_matricu(b, n);
95
   /* Izracunava se zbir i proizvod matrica */
97 saberi(a, b, zbir, n);
   pomnozi(a, b, proizvod, n);
99
   /* Ispisuje se rezultat */
101 if (jednake_matrice(a, b, n) == 1)
   printf("Matrice su jednake.\n");
103 else
   printf("Matrice nisu jednake.\n");
105
   printf("Zbir matrica je:\n");
107 ispisi_matricu(zbir, n);

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

```

### Rešenje 2.10

```

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

4 #define MAX 64

6 /* Funkcija proverava da li je relacija refleksivna. Relacija je
   refleksivna ako je svaki element u relaciji sa sobom, odnosno ako
8 se u matrici relacije na glavnoj dijagonali nalaze jedinice */
int refleksivnost(int m[][MAX], int n)
10 {
   int i;
12
   for (i = 0; i < n; i++) {
14     if (m[i][i] != 1)

```

```
        return 0;
16    }

18    return 1;
19 }

20 /* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono je
22    odredjeno matricom koja sadrzi sve elemente polazne matrice
    dopunjene jedinicama na glavnoj dijagonali */
24 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
25 {
26     int i, j;

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

32     /* Na glavnoj dijagonali se postavljaju jedinice */
    for (i = 0; i < n; i++)
        zatvorenje[i][i] = 1;
36 }

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

    /* Obilaze se elementi ispod glavne dijagonale matrice i
48    uporeduju se sa njima simetricnim elementima */
    for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
            if (m[i][j] != m[j][i])
52                return 0;

54    return 1;
55 }

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

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

```

        zatvorenje[i][j] = m[i][j];
68
    for (i = 0; i < n; i++)
69        for (j = 0; j < n; j++)
70            if (zatvorenje[i][j] == 1)
71                zatvorenje[j][i] = 1;
72    }
73
74
75
76    /* Funkcija proverava da li je relacija tranzitivna. Relacija je
77       tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
78       relaciji sa elementom "j" i element "j" u relaciji sa elementom
79       "k", onda je i element "i" u relaciji sa elementom "k" */
80    int tranzitivnost(int m[][MAX], int n)
81    {
82        int i, j, k;
83
84        for (i = 0; i < n; i++)
85            for (j = 0; j < n; j++)
86                /* Ispituje se da li postoji element koji narušava *
87                   tranzitivnost */
88                for (k = 0; k < n; k++)
89                    if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
90                        return 0;
91
92        return 1;
93    }
94
95
96    /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
97       relacije koriscenjem Varsalovog algoritma */
98    void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][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                zatvorenje[i][j] = m[i][j];
106
107        /* Odredjuje se refleksivno zatvorenje matrice */
108        for (i = 0; i < n; i++)
109            zatvorenje[i][i] = 1;
110
111        /* Primenom Varsalovog algoritma odredjuje se tranzitivno
112           zatvorenje matrice */
113        for (k = 0; k < n; k++)
114            for (i = 0; i < n; i++)
115                for (j = 0; j < n; j++)
116                    if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
117                        && (zatvorenje[i][j] == 0))
118                        zatvorenje[i][j] = 1;

```

```
120 }
121
122 /* Funkcija ispisuje elemente matrice */
123 void pisi_matricu(int m[][MAX], int n)
124 {
125     int i, j;
126
127     for (i = 0; i < n; i++) {
128         for (j = 0; j < n; j++)
129             printf("%d ", m[i][j]);
130         printf("\n");
131     }
132 }
133
134 int main(int argc, char *argv[])
135 {
136     FILE *ulaz;
137     int m[MAX][MAX];
138     int pomocna[MAX][MAX];
139     int n, i, j;
140
141     /* Provera da li korisnik nije uneo trazene argumente */
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]);
```

```

172  /* Ispisuje se rezultat */
    printf("Relacija %s refleksivna.\n",
           refleksivnost(m, n) == 1 ? "jeste" : "nije");
174
    printf("Relacija %s simetricna.\n",
           simetricnost(m, n) == 1 ? "jeste" : "nije");
176
    printf("Relacija %s tranzitivna.\n",
           tranzitivnost(m, n) == 1 ? "jeste" : "nije");
178
    printf("Refleksivno zatvorenje relacije:\n");
    ref_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
180
    printf("Simetricno zatvorenje relacije:\n");
    sim_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
182
    printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
    ref_tran_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
184
    /* Zatvara se datoteka */
    fclose(ulaz);
186
    exit(EXIT_SUCCESS);
188
}
190
192
194
196

```

## Rešenje 2.11

```

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

4  #define MAX 32

6  /* Funkcija izracunava najveći element na sporednoj dijagonali. Za
   elemente sporedne dijagonale vazi da je zbir indeksa vrste i
   indeksa kolone jednak n-1 */
8  int max_sporedna_dijagonala(int m[][MAX], int n)
10 {
    int i;
    int max_na_sporednoj_dijagonali = m[0][n - 1];
12
    for (i = 1; i < n; i++)
        if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)
14             max_na_sporednoj_dijagonali = m[i][n - 1 - i];
16
    return max_na_sporednoj_dijagonali;
18
}
20

/* Funkcija izracunava indeks kolone najmanjeg elementa */

```

```
22 int indeks_min(int m[][MAX], int n)
23 {
24     int i, j;
25     int min = m[0][0], indeks_kolone = 0;
26
27     for (i = 0; i < n; i++)
28         for (j = 0; j < n; j++)
29             if (m[i][j] < min) {
30                 min = m[i][j];
31                 indeks_kolone = j;
32             }
33
34     return indeks_kolone;
35 }
36
37 /* Funkcija izracunava indeks vrste najveceg elementa */
38 int indeks_max(int m[][MAX], int n)
39 {
40     int i, j;
41     int max = m[0][0], indeks_vrste = 0;
42
43     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 }
51
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
58     for (i = 0; i < n; i++)
59         for (j = 0; j < n; j++)
60             if (m[i][j] < 0)
61                 broj_negativnih++;
62
63     return broj_negativnih;
64 }
65
66 int main(int argc, char *argv[])
67 {
68     int m[MAX][MAX];
69     int n;
70     int i, j;
71
72     /* Proverava se broj argumenata komandne linije */
73     if (argc < 2) {
```

```

74     printf("Greska: ");
    printf("Nedovoljan broj argumenata komandne linije.\n");
76     printf("Program se poziva sa %s br_vrsta_mat.\n", argv[0]);
    exit(EXIT_FAILURE);
78 }

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

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

88
/* Ucitava se matrica */
90 printf("Unesite elemente matrice dimenzije %dx%d:\n", n, n);
for (i = 0; i < n; i++)
92     for (j = 0; j < n; j++)
        scanf("%d", &m[i][j]);
94
/* Ispisuju se rezultati izracunavanja */
96 printf("Najveci element sporedne dijagonale je %d.\n",
        max_sporedna_dijagonala(m, n));

98
printf("Indeks kolone sa najmanjim elementom je %d.\n",
100     indeks_min(m, n));

102 printf("Indeks vrste sa najvećim elementom je %d.\n",
        indeks_max(m, n));

104
printf("Broj negativnih elemenata matrice je %d.\n",
106     broj_negativnih(m, n));

108 exit(EXIT_SUCCESS);
}

```

### Rešenje 2.12

```

1  #include <stdio.h>
    #include <stdlib.h>
3
    #define MAX 32
5
/* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
   standardnog ulaza */
7  void ucitaj_matricu(int m[][MAX], int n)
9  {
    int i, j;
11
    for (i = 0; i < n; i++)

```

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

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

29 /* Funkcija proverava da li je zadata matrica ortonormirana,
30    odnosno, da li je normirana i ortogonalna. Matrica je normirana
31    ako je proizvod svake vrste matrice sa samom sobom jednak
32    jedinici. Matrica je ortogonalna, ako je proizvod dve bilo koje
33    razlicite vrste matrice jednak nuli */
34 int ortonormirana(int m[][MAX], int n)
35 {
36     int i, j, k;
37     int proizvod;

38     /* Ispituje se uslov normiranosti */
39     for (i = 0; i < n; i++) {
40         proizvod = 0;
41         for (j = 0; j < n; j++)
42             proizvod += m[i][j] * m[i][j];
43         if (proizvod != 1)
44             return 0;
45     }

46     /* Ispituje se uslov ortogonalnosti */
47     for (i = 0; i < n - 1; i++) {
48         for (j = i + 1; j < n; j++) {
49             proizvod = 0;
50             for (k = 0; k < n; k++)
51                 proizvod += m[i][k] * m[j][k];
52             if (proizvod != 0)
53                 return 0;
54         }
55     }

56     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
57     return 1;
58 }

60 int main()
61 {
62     int i, j;
63     int m[10][10];
```



```

65 {
66     int A[MAX][MAX];
67     int n;

68     printf("Unesite broj vrsta matrice: ");
69     scanf("%d", &n);

70     if (n > MAX || n <= 0) {
71         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
72         fprintf(stderr, "matrice.\n");
73         exit(EXIT_FAILURE);
74     }

75     printf("Unesite elemente matrice po vrstama:\n");
76     ucitaj_matricu(A, n);

77     printf("Matrica %s ortonormirana.\n",
78           ortonormirana(A, n) ? "je" : "nije");

79     exit(EXIT_SUCCESS);
80 }

```

### Rešenje 2.13

```

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

#define MAX 32

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

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

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

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

```

```
28 }

30 /* Funkcija proverava da li je zadata matrica ortonormirana,
31    odnosno, da li je normirana i ortogonalna. Matrica je normirana
32    ako je proizvod svake vrste matrice sa samom sobom jednak
33    jedinici. Matrica je ortogonalna, ako je proizvod dve bilo koje
34    razlicite vrste matrice jednak nuli */
35 int ortonormirana(int m[][MAX], int n)
36 {
37     int i, j, k;
38     int proizvod;

40     /* Ispituje se uslov normiranosti */
41     for (i = 0; i < n; i++) {
42         proizvod = 0;
43         for (j = 0; j < n; j++)
44             proizvod += m[i][j] * m[i][j];
45         if (proizvod != 1)
46             return 0;
47     }

48     /* Ispituje se uslov ortogonalnosti */
49     for (i = 0; i < n - 1; i++) {
50         for (j = i + 1; j < n; j++) {
51             proizvod = 0;
52             for (k = 0; k < n; k++)
53                 proizvod += m[i][k] * m[j][k];
54             if (proizvod != 0)
55                 return 0;
56         }
57     }

58     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
59     return 1;
60 }

62 }

64 int main()
65 {
66     int A[MAX][MAX];
67     int n;

68     printf("Unesite broj vrsta matrice: ");
69     scanf("%d", &n);

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

76     printf("Unesite elemente matrice po vrstama:\n");
77     ucitaj_matricu(A, n);
```

```

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

```

### Rešenje 2.15

```

1  #include <stdio.h>
   #include <stdlib.h>
3
4  int main()
5  {
6      int *p = NULL;
7      int i, n;
8
9      printf("Unesite dimenziju niza: ");
10     scanf("%d", &n);
11
12     /* Rezervise se prostor za n celih brojeva */
13     if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
14         fprintf(stderr, "Greska: Neupesna alokacija memorije.\n");
15         exit(EXIT_FAILURE);
16     }
17
18     printf("Unesite elemente niza: ");
19     for (i = 0; i < n; i++)
20         scanf("%d", &p[i]);
21
22     printf("Niz u obrnutom poretku je: ");
23     for (i = n - 1; i >= 0; i--)
24         printf("%d ", p[i]);
25     printf("\n");
26
27     /* Oslobadja se prostor rezervisan funkcijom malloc() */
28     free(p);
29
30     exit(EXIT_SUCCESS);
31 }

```

### Rešenje 2.16

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define KORAK 10
5
6  int main()

```

```
{
8  /* Adresa prvog alociranog bajta */
   int *a = NULL;
10
   /* Velicina alocirane memorije */
12  int alocirano;

   /* Broj elemenata niza */
14  int n;

16
   /* Broj koji se učitava sa ulaza */
18  int x;
   int i;
20  int *b = NULL;
   char realokacija;
22

   /* Inicijalizacija */
24  alocirano = n = 0;

26  printf("Unesite zeljeni nacin realokacije (M ili R):\n");
   scanf("%c", &realokacija);
28

   printf("Unesite brojeve, nulu za kraj:\n");
   scanf("%d", &x);
30

32  while (x != 0) {
       if (n == alocirano) {
34         alocirano = alocirano + KORAK;

36         if (realokacija == 'M') {
             /* Vrsi se realokacija memorije sa novom velicinom */
38             b = (int *) malloc(alocirano * sizeof(int));

40             if (b == NULL) {
                 fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
42                 free(a);
                 exit(EXIT_FAILURE);
44             }

46             /* Svih n elemenata koji pocinju na adresi a prepisuju se na
               novu adresu b */
48             for (i = 0; i < n; i++)
                 b[i] = a[i];

50             free(a);

52             /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
               bloka cija je adresa prilikom alokacije zapamcena u
               promenljivoj b */
54             a = b;
56         } else if (realokacija == 'R') {
58
```

```

60      /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
        "a" bude inicijalizovano na NULL */
        a = (int *) realloc(a, alocirano * sizeof(int));
62      if (a == NULL) {
        fprintf(stderr,
64          "Greska: Neuspesna realokacija memorije.\n");
        exit(EXIT_FAILURE);
66      }
    }
68    }
    a[n++] = x;
70    scanf("%d", &x);
}
72    printf("Niz u obrnutom poretku je: ");
    for (n--; n >= 0; n--)
74        printf("%d ", a[n]);
    printf("\n");
76
    /* Oslobadja se dinamicki alocirana memorija */
78    free(a);
80
    exit(EXIT_SUCCESS);
}

```

### Rešenje 2.17

```

#include <stdio.h>
2  #include <stdlib.h>
#include <string.h>
4
#define MAX 1000
6
/* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8  nadovezivanja niski. Adresa kreiranog niza se vraca kao povratna
   vrednost. */
10 char *nadovezi(char *s, char *t)
   {
12     char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
        * sizeof(char));
14
    /* Proverava se da li je memorija uspesno alocirana */
16     if (p == NULL) {
        fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
18         exit(EXIT_FAILURE);
    }
20
    /* Kopiraju se i nadovezuju niske karaktera */
22     strcpy(p, s);
    strcat(p, t);
24
    return p;

```

```
26 }
27
28 int main()
29 {
30     char *s = NULL;
31     char s1[MAX], s2[MAX];
32
33     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 matrice */
16     double trag = 0;
17
18     printf("Unesite broj vrsta i broj kolona:\n ");
19     scanf("%d%d", &n, &m);
20
21     /* Dinamicki se rezervise prostor za niz vrsta matrice */
22     A = (double **) malloc(sizeof(double *) * n);
23
24     /* Provera se da li je uspelo rezervisanje memorije */
25     if (A == NULL) {
26         fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
27     }
28 }
```

```

27     exit(EXIT_FAILURE);
28 }
29
30 /* Dinamicki se rezervise prostor za elemente u vrstama */
31 for (i = 0; i < n; i++) {
32     A[i] = (double *) malloc(sizeof(double) * m);
33
34     /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
35        potrebno je osloboditi svih i-1 prethodno alociranih vrsta, i
36        alociran niz pokazivaca */
37     if (A[i] == NULL) {
38         for (j = 0; j < i; j++)
39             free(A[j]);
40         free(A);
41         exit(EXIT_FAILURE);
42     }
43 }
44
45 printf("Unesite elemente matrice po vrstama:\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 /* Oslobadja 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 dimenzije n x m */

```

## 2 Pokazivači

```
5 int **alociraj_matricu(int n, int m);
7 /* Funkcija dinamički alokira memoriju za kvadratnu matricu
   dimenzije n x n */
9 int **alociraj_kvadratnu_matricu(int n);
11 /* Funkcija oslobadja memoriju za matricu sa n vrsta */
   int **dealociraj_matricu(int **matrica, int n);
13
15 /* Funkcija učitava već alociranu matricu dimenzije n x m sa
   standardnog ulaza */
   void učitaj_matricu(int **matrica, int n, int m);
17
19 /* Funkcija učitava već alociranu kvadratnu matricu dimenzije n sa
   standardnog ulaza */
   void učitaj_kvadratnu_matricu(int **matrica, int n);
21
23 /* Funkcija ispisuje matricu dimenzije n x m na standardnom izlazu */
   void ispisi_matricu(int **matrica, int n, int m);
25
27 /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
   izlazu */
   void ispisi_kvadratnu_matricu(int **matrica, int n);
29
31 /* Funkcija učitava već alociranu matricu dimenzije n x m iz
   datoteke f */
   int učitaj_matricu_iz_datoteke(int **matrica, int n, int m,
                                   FILE * f);
33
35 /* Funkcija učitava već alociranu kvadratnu matricu dimenzije n x n
   iz datoteke f */
   int učitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
                                             FILE * f);
37
39 /* Funkcija upisuje matricu dimenzije n x m u datoteku f */
   int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);
41
43 /* Funkcija upisuje kvadratnu matricu dimenzije n x n u datoteku f */
   int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
                                           FILE * f);
45
   #endif
```

*matrica.c*

```
1 #include <stdio.h>
2 #include <stdlib.h>
   #include "matrica.h"
4
   int **alociraj_matricu(int n, int m)
6 {
```



```

8   int **matrica = NULL;
   int i, j;

10  /* Alocira se prostor za niz vrsta matrice */
   matrica = (int **) malloc(n * sizeof(int *));
12  /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije
      ce biti NULL, sto mora biti provereno u main funkciji */
14  if (matrica == NULL)
      return NULL;

16
   /* Alocira se prostor za svaku vrstu matrice */
18  for (i = 0; i < n; i++) {
      matrica[i] = (int *) malloc(m * sizeof(int));
20  /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
      prethodno alocirani resursi, i povratna vrednost je NULL */
22  if (matrica[i] == NULL) {
      for (j = 0; j < i; j++)
24      free(matrica[j]);
      free(matrica);
26      return NULL;
      }
28  }
   return matrica;
30 }

32 int **alociraj_kvadratnu_matricu(int n)
   {
34   /* Alociranje matrice dimenzije n x n */
      return alociraj_matricu(n, n);
36 }

38 int **dealociraj_matricu(int **matrica, int n)
   {
40   int i;
      /* Oslobadja se prostor rezervisan za svaku vrstu */
42   for (i = 0; i < n; i++)
      free(matrica[i]);
44   /* Oslobadja se memorija za niz pokazivaca na vrste */
      free(matrica);

46   /* Matrica postaje prazna, tj. nealocirana */
48   return NULL;
   }

50
   void ucitaj_matricu(int **matrica, int n, int m)
52   {
      int i, j;
54   /* Elementi matrice se ucitavaju po vrstama */
      for (i = 0; i < n; i++)
56         for (j = 0; j < m; j++)
            scanf("%d", &matrica[i][j]);
58 }

```

```
60 void ucitaj_kvadratnu_matricu(int **matrica, int n)
61 {
62     /* Ucitavanje matrice n x n */
63     ucitaj_matricu(matrica, n, n);
64 }
65
66 void ispisi_matricu(int **matrica, int n, int m)
67 {
68     int i, j;
69     /* Ispis po vrstama */
70     for (i = 0; i < n; i++) {
71         for (j = 0; j < m; j++)
72             printf("%d ", matrica[i][j]);
73         printf("\n");
74     }
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 učitavaju po vrstama */
87     for (i = 0; i < n; i++)
88         for (j = 0; j < m; j++)
89             /* Ako je nemoguće učitati sledeći element, povratna vrednost
90              funkcije je 1, kao indikator neuspešnog učitavanja */
91             if (fscanf(f, "%d", &matrica[i][j]) != 1)
92                 return 1;
93
94     /* Uspesno učitana matrica */
95     return 0;
96 }
97
98 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
99                                           FILE * f)
100 {
101     /* Ucitavanje matrice n x n iz datoteke */
102     return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
103 }
104
105 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
106 {
107     int i, j;
108     /* Ispis po vrstama */
109     for (i = 0; i < n; i++) {
110         for (j = 0; j < m; j++)
```

```

112     /* Ako je nemoguće ispisati sledeći element, povratna vrednost
        funkcije je 1, kao indikator neuspešnog ispisa */
114     if (fprintf(f, "%d ", matrica[i][j]) <= 0)
        return 1;
116     fprintf(f, "\n");
    }

118     /* Uspešno upisana matrica */
    return 0;
120 }

122 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
                                         FILE * f)
124 {
    /* Ispis matrice n x n u datoteku */
126     return upisi_matricu_u_datoteku(matrica, n, n, f);
}

```

*main\_a.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "matrica.h"

5  int main()
   {
7      int **matrica = NULL;
       int n, m;
9      FILE *f;

11     /* Učitava se broj vrsta i broj kolona matrice */
       printf("Unesite broj vrsta matrice: ");
13     scanf("%d", &n);
       printf("Unesite broj kolona matrice: ");
15     scanf("%d", &m);

17     /* Provera dimenzija matrice */
       if (n <= 0 || m <= 0) {
19         fprintf(stderr, "Greska: Broj vrsta i broj kolona ne mogu biti
            negativni brojevi.\n");
           exit(EXIT_FAILURE);
21     }

23     /* Rezervise se memorijski prostor za matricu i proverava se da li
        je memorijski prostor uspešno rezervisan */
25     matrica = alociraj_matricu(n, m);
       if (matrica == NULL) {
27         fprintf(stderr, "Greska: Neuspešna alokacija matrice.\n");
           exit(EXIT_FAILURE);
29     }
}

```

```
31  /* Ucitava se matrica sa standardnog ulaza */
    printf("Unesite elemente matrice po vrstama:\n");
33  ucitaj_matricu(matrica, n, m);

35  /* Otvara se datoteka za upis matrice */
    if ((f = fopen("matrica.txt", "w")) == NULL) {
37      fprintf(stderr, "Greska: Neuspesno otvaranje datoteke.\n");
        matrica = dealociraj_matricu(matrica, n);
39      exit(EXIT_FAILURE);
    }

41  /* Upis matrice u datoteku */
43  if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
        fprintf(stderr, "Greska: Neuspesno upisivanje matrice u datoteku
        .\n");
45      matrica = dealociraj_matricu(matrica, n);
        exit(EXIT_FAILURE);
47  }

49  /* Zatvara se datoteka */
    fclose(f);

51  /* Oslobadja se memorija koju je zauzimala matrica */
53  matrica = dealociraj_matricu(matrica, n);

55  exit(EXIT_SUCCESS);
}
```

*main\_b.c*

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

4
    int main(int argc, char **argv)
    {
6        int **matrica = NULL;
        int n;
        FILE *f;

10
        /* Provera argumenata komandne linije */
12        if (argc != 2) {
            fprintf(stderr, "Greska: Koriscenje programa: %s datoteka\n",
            argv[0]);
14            exit(EXIT_FAILURE);
        }

16
        /* Otvara se datoteka za citanje */
18        if ((f = fopen(argv[1], "r")) == NULL) {
            fprintf(stderr, "Greska: Neuspesno otvaranje datoteke.\n");
20            exit(EXIT_FAILURE);
        }
    }
```

```

22     }

23     /* Ucitava se dimenzija matrice */
24     if (fscanf(f, "%d", &n) != 1) {
25         fprintf(stderr, "Greska: Neispravan pocetak datoteke.\n");
26         exit(EXIT_FAILURE);
27     }

28     /* Provera se dimenzija matrice */
29     if (n <= 0) {
30         fprintf(stderr, "Greska: Neodgovarajca dimenzija matrice.\n");
31         exit(EXIT_FAILURE);
32     }

33     /* Rezervise se memorijski prostor za matricu i vrsi se provera */
34     matrica = alociraj_kvadratnu_matricu(n);
35     if (matrica == NULL) {
36         fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
37         exit(EXIT_FAILURE);
38     }

39     /* Ucitava se matrica iz datoteke */
40     if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
41         fprintf(stderr, "Greska: Neuspesno ucitavanje matrice iz datoteke\n");
42         matrica = dealociraj_matricu(matrica, n);
43         exit(EXIT_FAILURE);
44     }

45     /* Zatvara se datoteka */
46     fclose(f);

47     /* Ispis matrice na standardni izlaz */
48     ispisi_kvadratnu_matricu(matrica, n);

49     /* Oslobadja se memorija koju je zauzimala matrica */
50     matrica = dealociraj_matricu(matrica, n);

51     exit(EXIT_SUCCESS);
52 }

```

## Rešenje 2.20

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "matrica.h"
5
6  /* Funkcija ispisuje elemente matrice ispod glavne dijagonale */
7  void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
8  {

```

```
9   int i, j;

11  for (i = 0; i < n; i++) {
12      for (j = 0; j <= i; j++)
13          printf("%d ", M[i][j]);
14      printf("\n");
15  }
16 }

17
18 int main()
19 {
20     int m, n;
21     int **matrica = NULL;

22
23     printf("Unesite broj vrsta i broj kolona:\n ");
24     scanf("%d %d", &n, &m);

25
26     /* Rezervise se memorija za matricu */
27     matrica = alociraj_matricu(n, m);
28     /* Provera alokacije */
29     if (matrica == NULL) {
30         fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
31         exit(EXIT_FAILURE);
32     }

33
34     printf("Unesite elemente matrice po vrstama:\n");
35     ucitaj_matricu(matrica, n, m);

36
37     printf("Elementi ispod glavne dijagonale matrice:\n");
38     ispisi_elemente_ispod_dijagonale(matrica, n, m);

39
40     /* Oslobadja se memorija */
41     matrica = dealociraj_matricu(matrica, n);

42
43     exit(EXIT_SUCCESS);
44 }
```

### Rešenje 2.22

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

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

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

```
        a[i][j] /= 2;
14     else if (i > j)
        a[i][j] *= 2;
16 }

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

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

32     return zbir;
}

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

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

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

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

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

64 }
```

```

    m = (float **) malloc(n * sizeof(float *));
66  if (m == NULL) {
    fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
68  exit(EXIT_FAILURE);
    }

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

    if (m[i] == NULL) {
74  fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
76  for (j = 0; j < i; j++)
        free(m[j]);
78  free(m);
        exit(EXIT_FAILURE);
80  }
    }
82  return m;
}

84  /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
86  n x n */
void oslobodi_memoriju(float **m, int n)
88 {
    int i;
90
    for (i = 0; i < n; i++)
92  free(m[i]);
    free(m);
94 }

96 int main(int argc, char *argv[])
97 {
98  FILE *ulaz;
99  float **a;
100  int n;

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

110
    /* Otvara se datoteka za citanje */
112  ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
114  fprintf(stderr, "Greska: ");
    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
116  exit(EXIT_FAILURE);
    }
}
```



```

118     }
119
120     /* Cita se dimenzija matrice */
121     fscanf(ulaz, "%d", &n);
122
123     /* Rezervise se memorija */
124     a = alociraj_memoriju(n);
125
126     /* Ucitavaju se elementi matrice */
127     ucitaj_matricu(ulaz, a, n);
128
129     float zbir = zbir_ispod_sporedne_dijagonale(a, n);
130
131     /* Poziva se funkcija za transformaciju matrice */
132     izmeni(a, n);
133
134     /* Ispisuju se rezultati */
135     printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
136     printf("je %.2f.\n", zbir);
137
138     printf("Transformisana matrica je:\n");
139     ispisi_matricu(a, n);
140
141     /* Oslobadja se memorija */
142     oslobodi_memoriju(a, n);
143
144     /* Zatvara se datoteka */
145     fclose(ulaz);
146
147     exit(EXIT_SUCCESS);
148 }

```

### 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++) {

```

```

    x = a + i * (b - a) / (n - 1);
19     printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
    }
21     printf("-----\n");
    }
23
24     double sqr(double a)
25     {
26         return a * a;
27     }
28
29     int main(int argc, char *argv[])
30     {
31         double a, b;
32         int n;
33
34         char ime_funkcije[6];
35
36         /* Pokazivac na funkciju koja ima jedan argument tipa double i
37            povratnu vrednost istog tipa */
38         double (*fp) (double);
39
40         /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
41            greska */
42         if (argc < 2) {
43             fprintf(stderr, "Greska: ");
44             fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
45             fprintf(stderr,
46                 "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 komandnoj liniji */
53         strcpy(ime_funkcije, argv[1]);
54
55         /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
56         if (strcmp(ime_funkcije, "sin") == 0)
57             fp = &sin;
58         else if (strcmp(ime_funkcije, "cos") == 0)
59             fp = &cos;
60         else if (strcmp(ime_funkcije, "tan") == 0)
61             fp = &tan;
62         else if (strcmp(ime_funkcije, "atan") == 0)
63             fp = &atan;
64         else if (strcmp(ime_funkcije, "acos") == 0)
65             fp = &acos;
66         else if (strcmp(ime_funkcije, "asin") == 0)
67             fp = &asin;
68         else if (strcmp(ime_funkcije, "exp") == 0)
69             fp = &exp;
```

```
71     else if (strcmp(ime_funkcije, "log") == 0)
72         fp = &log;
73     else if (strcmp(ime_funkcije, "log10") == 0)
74         fp = &log10;
75     else if (strcmp(ime_funkcije, "sqrt") == 0)
76         fp = &sqrt;
77     else if (strcmp(ime_funkcije, "floor") == 0)
78         fp = &floor;
79     else if (strcmp(ime_funkcije, "ceil") == 0)
80         fp = &ceil;
81     else if (strcmp(ime_funkcije, "sqr") == 0)
82         fp = &sqr;
83     else {
84         fprintf(stderr, "Greska");
85         fprintf(stderr,
86             "Program jos uvek ne podrzava trazenu funkciju!\n");
87         exit(EXIT_FAILURE);
88     }
89
90     printf("Unesite krajeve intervala:\n");
91     scanf("%lf %lf", &a, &b);
92
93     printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
94     printf("(ukljucujuci krajeve intervala)?\n");
95     scanf("%d", &n);
96
97     /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
98        uneti broj veci od 2 */
99     if (n < 2) {
100         fprintf(stderr, "Greska: Broj tacaka mreze mora biti bar 2!\n");
101         exit(EXIT_FAILURE);
102     }
103
104     /* Ispisuje se ime funkcije */
105     printf("      x %10s(x)\n", ime_funkcije);
106
107     /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
108        komandne linije */
109     tabela(a, b, n, fp);
110
111     exit(EXIT_SUCCESS);
112 }
```



## 3

# Algoritmi pretrage i sortiranja

## 3.1 Algoritmi pretrage

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

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

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

### 3 Algoritmi pretrage i sortiranja

---

#### Test 1

```
POKRETANJE: ./a.out 1000000 23542
```

```
VREMENA.TXT
```

```
IZLAZ:
Linearna pretraga:
Element nije u nizu
Binarna pretraga:
Element nije u nizu
Interpolaciona pretraga:
Element nije u nizu
```

```
Dimenzija niza: 1000000
Linearna: 3615091 ns
Binarna: 1536 ns
Interpolaciona: 558 ns
```

#### Test 2

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

```
VREMENA.TXT
```

```
IZLAZ:
Linearna pretraga:
Element nije u nizu
Binarna pretraga:
Element nije u nizu
Interpolaciona pretraga:
Element nije u nizu
```

```
Dimenzija niza: 1000000
Linearna: 3615091 ns
Binarna: 1536 ns
Interpolaciona: 558 ns
```

```
Dimenzija niza: 100000
Linearna: 360803 ns
Binarna: 1187 ns
Interpolaciona: 628 ns
```

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

#### Primer 1

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

#### Primer 2

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

**Zadatak 3.3** Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više

studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti `-indeks` ili `-prezime`. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složeno-sti. 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
```

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

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

#### Test 1

```
POKRETANJE: ./a.out dat.txt -x

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

IZLAZ:
-0.3 23
```

#### Test 2

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

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

IZLAZ:
-1 2.1
```

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

**Zadatak 3.7** Napisati funkciju koja metodom polovljenja intervala određuje nulu izabrane funkcije na proizvoljnom 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
```



**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.

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

**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.

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

**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.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati pozitivan ceo broj  $a$  na standardni izlaz ispisati njegov logaritam.

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

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

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 3</i>
<pre> INTERAKCIJA SA PROGRAMOM: Unesite broj tacaka: 2 Unesite koordinate tacaka:   2 0 2 1   1 2 2 2   26.57           </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Unesite broj tacaka: 2 Unesite koordinate tacaka:   1 0 1 1   0 1 1 1   45           </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Unesite broj tacaka: 3 Unesite koordinate tacaka:   1 0 1 1   2 0 2 1   1 2 2 2   26.57           </pre>

## 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 poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za 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  $n$ .

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

**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.

<p><i>Primer 1</i></p> <pre>    INTERAKCIJA SA PROGRAMOM:    Unesite prvu nisku anagram    Unesite drugu nisku ramgana    jesu </pre>	<p><i>Primer 2</i></p> <pre>    INTERAKCIJA SA PROGRAMOM:    Unesite prvu nisku anagram    Unesite drugu nisku anagrm    nisu </pre>	<p><i>Primer 3</i></p> <pre>    INTERAKCIJA SA PROGRAMOM:    Unesite prvu nisku test    Unesite drugu nisku tset    jesu </pre>
---	--	---

**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.

### 3 Algoritmi pretrage i sortiranja

---

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

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

Test 1	Test 2	Test 3
ULAZ: 4 23 5 2 4 6 7 34 6 4 5	ULAZ: 2 4 6 2 6 7 99 1	ULAZ: 123
IzLAZ: 4	IzLAZ: 2	IzLAZ: 123

**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	Primer 2	Primer 3
INTERAKCIJA SA PROGRAMOM: Unesite trazeni zbir: 34 Unesite elemente niza: 134 4 1 6 30 23 da	INTERAKCIJA SA PROGRAMOM: Unesite trazeni zbir: 12 Unesite elemente niza: 53 1 43 3 56 13 ne	INTERAKCIJA SA PROGRAMOM: Unesite trazeni zbir: 52 Unesite elemente niza: 52 ne

**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 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
```

**Zadatak 3.18** Napisati program koji čita sadržaj dve datoteke od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima, a u slučaju istog imena po prezimenima, i kreira jedinstven spisak studenata sortiranih takođe po istom kriterijumu. Program dobija nazive datoteka iz komandne linije i jedinstveni spisak upisuje u datoteku `ceo-tok.txt`. Pretpostaviti da 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
Andrija Petrovic
Anja Ilic
Ivana Markovic
Lazar Micic
Nenad Brankovic
Sofija Filipovic
Uros Milic
Vladimir Savic

DRUGI-DEO.TXT
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Marija Stankovic
Ognjen Peric

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

**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.

#### 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
```

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

IZLAZ:
3
```

#### Test 2

```
BIRACKI-SPISAK.TXT
Milan Milicevic

IZLAZ:
1
```

#### Test 3

```
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI

IZLAZ ZA GREŠKE:
Neupesno otvaranje
datoteke za citanje
```

**Zadatak 3.21** Definirati strukturu koja čuva imena, prezimena i godišta dece. Pretpostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

### Test 1

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

### Test 1

```
NISKE.TXT
```

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

```
IZLAZ:
```

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

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

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

UKUPNO: 270.00 dinara.

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

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

UKUPNO: 230.00 dinara.

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

Kraj rada kase!
```

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



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

### Test 1

AKTIVNOSTI.TXT

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

DAT1.TXT

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

DAT2.TXT

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

DAT3.TXT

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

**Zadatak 3.25** U datoteci `pesme.txt` nalaze se informacije o gledanosti pesama na Youtube-u. Svaki red datoteke sadrži informacije o gledanosti pesama u formatu **izvođač - naslov, broj gledanja**. Napisati program koji učitava informacije o pesmama i vrši sortiranje pesama u zavisnosti od argumenata komandne linije na sledeći način:

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

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

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

**\* 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. Optimizovati program, ukoliko se zna da neće biti više od 500 različitih unetih brojeva. *NAPOMENA: Brojeve čuvati u sortiranom nizu i svaki naredni element umetati na svoje mesto.*

#### Primer 1

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

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

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

2    2\_\_ 5    5

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

*Test 1*

```

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

```

**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 za rad 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

```

**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 za rad 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. Pretraživanje niza vršiti bibliotečkom funkcijom `bsearch`. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

#### Test 1

```
ULAZ:
R

IZLAZ:
Dusko Radovic
```

#### Test 2

```
ULAZ:
E

IZLAZ:
Dobrica Eric
```

#### Test 3

```
ULAZ:
X

IZLAZ:
-1
```

**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

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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 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!
```

**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:
```

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

(a) leksikografski,

(b) po dužini.

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

#### Primer 1

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

INTERAKCIJA 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
```

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

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

#### 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
opadajuće, pa po prezimenu rastuće:
Bojan Golubovic 30
Dragan Markovic 25
Ivana Stankovic 25
Filip Dukic 20
Lazar Micic 20
Ognjen Peric 20
Nenad Brankovic 15
Aleksandra Cvetic 15
Marija Stankovic 15
Uros Milic 10
Anja Ilic 5
Ivana Markovic 5
Andrija Petrovic 0
Unesite broj bodova: 20
Pronadjen je student sa unetim
brojem bodova: Filip Dukic 20
Unesite prezime: Markovic
Pronadjen je student sa unetim
prezimenom: Dragan Markovic 25
```

### 3.3 Bibliotečke funkcije pretrage i sortiranja

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

**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
```

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

#### Test 1

```
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
```

**Zadatak 3.39** Definirati strukturu `Datum`. Napisati funkciju koja poredi dva

### 3 Algoritmi pretrage i sortiranja

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:
Unesite sledeci datum: 13.12.2016.
postoji
Unesite sledeci datum: 10.5.2015.
ne postoji
Unesite sledeci 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 trazeni
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,
13  kao 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 vrši 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 }
```



```

27 /* Binarna pretraga: Funkcija trazi u sortiranom nizu a[] duzine n
    broj x. Pretraga koristi osobinu sortiranosti niza i u svakoj
29 iteraciji polovi interval pretrage. */
int binarna_pretraga(int a[], int n, int x)
31 {
    int levi = 0;
33     int desni = n - 1;
    int srednji;
35     /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
37         /* Srednji indeks je njihova aritmeticka sredina */
        srednji = (levi + desni) / 2;
39         /* Ako je element sa sredisnjim indeksom veci od x, tada se x
            mora nalaziti u levom delu niza */
41         if (x < a[srednji])
            desni = srednji - 1;
43         /* Ako je element sa sredisnjim indeksom manji od x, tada se x
            mora nalaziti u desnom delu niza */
45         else if (x > a[srednji])
            levi = srednji + 1;
47         else
            /* Ako je element sa sredisnjim indeksom jednak x, tada je
49             broj x pronadjen na poziciji srednji */
            return srednji;
51     }
    /* Ako element x nije pronadjen, vraca se -1 */
53     return -1;
}

55 /* Interpolaciona pretraga: Funkcija trazi u sortiranom nizu a[]
    duzine n broj x. Pretraga koristi osobinu sortiranosti niza i
57 zasniva se na linearnoj interpolaciji vrednosti koja se trazi
    vrednostima na krajevima prostora pretrage. */
int interpolaciona_pretraga(int a[], int n, int x)
61 {
    int levi = 0;
63     int desni = n - 1;
    int srednji;
65     /* Dokle god je indeks levi levo od indeksa desni... */
    while (levi <= desni) {
67         /* Ako je trazeni element manji od pocetnog ili veci od
            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
            dogodilo da se prilikom izracunavanja indeksa srednji izadje
71             izvan opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
73             return -1;
        /* 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 trazeni broj
            x jednak ovim vrednostima, pa se vraca indeks levi (ili
77             indeks desni). Ova provera je neophodna, jer bi se u
            suprotnom prilikom izracunavanja indeksa srednji pojavilo

```

```
79         deljenje nulom. */
80     else if (a[levi] == a[desni])
81         return levi;
82     /* Racuna se sredisnji indeks */
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        poređiti sa pretragom rečnika: ako neko trazi rec na slovo
91        'B', sigurno nece da otvori rečnik na polovini, vec verovatno
92        negde blize pocetku. */
93     /* Ako je element sa indeksom srednji veci od trazenog, tada se
94        trazeni 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        trazeni element mora nalaziti u desnoj polovini niza */
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 završava 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             "Greska: Program se poziva sa %s dim_niza broj\n",
120             argv[0]);
121         exit(EXIT_FAILURE);
122     }
123
124     /* Dimenzija niza */
125     n = atoi(argv[1]);
126     if (n > MAX || n <= 0) {
127         fprintf(stderr, "Greska: Dimenzija niza neodgovarajuca\n");
128         exit(EXIT_FAILURE);
129     }
130 }
```

```

131  /* Broj koji se trazi */
      x = atoi(argv[2]);
133  /* Elementi niza se generisu slucajno, tako da je svaki sledeci
      veci od prethodnog. Funkcija srandom() inicijalizuje pocetnu
135  vrednost sa kojom se krece u izracunavanje sekvence
      pseudo-slucajnih brojeva. Kako generisani niz ne bi uvek bio
137  isti, ova vrednost se postavlja na tekuce vreme u sekundama od
      Nove godine 1970, tako da je za svako sledece pokretanje
139  programa (u vremenskim intervalima vecim od jedne sekunde) ove
      vrednost drugacija. random()%100 vraca brojeve izmedju 0 i 99 */
141  srandom(time(NULL));
      for (i = 0; i < n; i++)
143      a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
      /* Linearna pretraga */
145  printf("Linearna pretraga:\n");
      /* Vreme proteklo od Nove godine 1970 */
147  clock_gettime(CLOCK_REALTIME, &vreme1);
      i = linearna_pretraga(a, n, x);
149  /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
      linearnu pretragu */
151  clock_gettime(CLOCK_REALTIME, &vreme2);
      if (i == -1)
153      printf("Element nije u nizu\n");
      else
155      printf("Element je u nizu na poziciji %d\n", i);
      /* Binarna pretraga */
157  printf("Binarna pretraga:\n");
      clock_gettime(CLOCK_REALTIME, &vreme3);
159  i = binarna_pretraga(a, n, x);
      clock_gettime(CLOCK_REALTIME, &vreme4);
161  if (i == -1)
      printf("Element nije u nizu\n");
163  else
      printf("Element je u nizu na poziciji %d\n", i);
165  /* Interpolaciona pretraga */
      printf("Interpolaciona pretraga:\n");
167  clock_gettime(CLOCK_REALTIME, &vreme5);
      i = interpolaciona_pretraga(a, n, x);
169  clock_gettime(CLOCK_REALTIME, &vreme6);
      if (i == -1)
171      printf("Element nije u nizu\n");
      else
173      printf("Element je u nizu na poziciji %d\n", i);
      /* Podaci o izvršavanju programa bivaju upisani u log */
175  if ((f = fopen("vremena.txt", "a")) == NULL) {
      fprintf(stderr, "Greska: Neuspesno otvaranje log datoteke.\n");
177      exit(EXIT_FAILURE);
      }

179

181  fprintf(f, "Dimenzija niza: %d\n", n);
      fprintf(f, "\tLinearna:%10ld ns\n",
              (vreme2.tv_sec - vreme1.tv_sec) * 1000000000 +

```

### 3 Algoritmi pretrage i sortiranja

```
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     /* Zatvara se datoteka */
191     fclose(f);
192
193     exit(EXIT_SUCCESS);
194 }
```

#### Rešenje 3.2

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

4 #define MAX 1024

6 /* Rekurzivna linearna pretraga od pocetka niza */
7 int linearna_pretraga_r1(int a[], int n, int x)
8 {
9     int tmp;
10    /* Izlaz iz rekurziije */
11    if (n <= 0)
12        return -1;
13    /* Ako je prvi element trazeni */
14    if (a[0] == x)
15        return 0;
16    /* Pretraga ostatka niza */
17    tmp = linearna_pretraga_r1(a + 1, n - 1, x);
18    return tmp < 0 ? tmp : tmp + 1;
19 }

20 /* Rekurzivna linearna pretraga od kraja niza */
21 int linearna_pretraga_r2(int a[], int n, int x)
22 {
23    /* Izlaz iz rekurziije */
24    if (n <= 0)
25        return -1;
26    /* Ako je poslednji element trazeni */
27    if (a[n - 1] == x)
28        return n - 1;
29    /* Pretraga ostatka niza */
30    return linearna_pretraga_r2(a, n - 1, x);
31 }

32 /* Rekurzivna binarna pretraga */
33 int binarna_pretraga_r(int a[], int l, int d, int x)
34 {
35 }
```

```

38     int srednji;
    if (l > d)
        return -1;
40    /* Sredisnja pozicija na kojoj se trazi vrednost x */
    srednji = (l + d) / 2;
42    /* Ako je element na sredisnjoj poziciji trazeni */
    if (a[srednji] == x)
44        return srednji;
    /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
46    pretrazuje se desna polovina niza */
    if (a[srednji] < x)
48        return binarna_pretraga_r(a, srednji + 1, d, x);
    /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
50    pretrazuje se leva polovina niza */
    else
52        return binarna_pretraga_r(a, l, srednji - 1, x);
}

54    /* Rekurzivna interpolaciona pretraga */
56    int interpolaciona_pretraga_r(int a[], int l, int d, int x)
    {
58        int p;
        /* Ako je trazeni element manji od prvog ili veci od poslednjeg */
60        if (x < a[l] || x > a[d])
            return -1;
62        /* Ako je ostao jedan element u delu niza koji se pretrazuje */
        if (a[d] == a[l])
64            return l;
        /* Pozicija na kojoj se trazi vrednost x */
66        p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
        if (a[p] == x)
68            return p;
        /* Pretraga sufiksa niza */
70        if (a[p] < x)
            return interpolaciona_pretraga_r(a, p + 1, d, x);
72        /* Pretraga prefiksa niza */
        else
74            return interpolaciona_pretraga_r(a, l, p - 1, x);
    }

76    int main()
78    {
        int a[MAX];
80        int x;
        int i, indeks;

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

86        /* Ucitavaju se elemenati niza sve do kraja ulaza - ocekuje se da
88        korisnik pritisne CTRL+D za naznaku kraja */

```

### 3 Algoritmi pretrage i sortiranja

```

i = 0;
90 printf("Unesite sortiran niz elemenata: ");
while (i < MAX && scanf("%d", &a[i]) == 1) {
92     if (i > 0 && a[i] < a[i - 1]) {
        fprintf(stderr, "Greska: Elementi moraju biti uneseni ");
94         fprintf(stderr, "u neopadajućem poretku\n");
        exit(EXIT_FAILURE);
96     }
    i++;
98 }

/* Rezultati linearnih pretraga */
printf("Linearna pretraga\n");
indeks = linearna_pretraga_r1(a, i, x);
102 if (indeks == -1)
104     printf("Element se ne nalazi u nizu.\n");
else
106     printf("Pozicija elementa je %d.\n", indeks);
indeks = linearna_pretraga_r2(a, i, x);
108 if (indeks == -1)
110     printf("Element se ne nalazi u nizu.\n");
else
112     printf("Pozicija elementa je %d.\n", indeks);

/* Rezultati binarna pretrage */
114 printf("Binarna pretraga\n");
indeks = binarna_pretraga_r(a, 0, i - 1, x);
116 if (indeks == -1)
118     printf("Element se ne nalazi u nizu.\n");
else
120     printf("Pozicija elementa je %d.\n", indeks);

/* Rezultati interpolacione pretrage */
122 printf("Interpolaciona pretraga\n");
indeks = interpolaciona_pretraga_r(a, 0, i - 1, x);
124 if (indeks == -1)
126     printf("Element se ne nalazi u nizu.\n");
else
128     printf("Pozicija elementa je %d.\n", indeks);

130 exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.3

```

#include <stdio.h>
2  #include <stdlib.h>
   #include <string.h>
4
   #define MAX_STUDENATA 128
6  #define MAX_DUZINA 16
```

```
8  /* 0 svakom studentu postoje 3 informacije i one su objedinjene u
   strukturi kojom se predstavlja svaki student. */
10 typedef struct {
   /* Indeks mora biti tipa long jer su podaci u datoteci preveliki
   za int, npr. 20140123 */
12     long indeks;
14     char ime[MAX_DUZINA];
   char prezime[MAX_DUZINA];
16 } Student;

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

22 /* 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. */
24 int binarna_pretraga(Student a[], int n, long x)
   {
26     int levi = 0;
   int desni = n - 1;
30     int srednji;
   /* Dokle god je indeks levi levo od indeksa desni */
32     while (levi <= desni) {
   /* Racuna se srednja pozicija */
34     srednji = (levi + desni) / 2;
   /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
   se trazeni indeks mora nalaziti u levoj polovini niza */
36     if (x < a[srednji].indeks)
   desni = srednji - 1;
   /* Ako je pak manji od trazenog, tada se on mora nalaziti u
40     desnoj polovini niza */
   else if (x > a[srednji].indeks)
   levi = srednji + 1;
   else
44     /* Ako je jednak trazenom indeksu x, tada je pronadjen student
   sa trazenom indeksom na poziciji srednji */
   return srednji;
46     }
   /* Ako nije pronadjen, vraca se -1 */
48     return -1;
50 }

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

```
        return i;
60     return -1;
    }

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

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

94         /* Otvara se datoteka */
        fin = fopen(argv[1], "r");
96         if (fin == NULL) {
98             fprintf(stderr,
                "Greska: Neuspesno otvaranje datoteke %s za citanje\n",
100                 argv[1]);
            exit(EXIT_FAILURE);
102         }

104         /* Cita se sve dok postoji red sa informacijama o studentu */
        i = 0;
106         while (1) {
            if (i == MAX_STUDENATA)
108                 break;
            if (fscanf
110                 (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
```



```

        dosije[i].prezime) != 3)
112     break;
    i++;
114 }
    br_studenata = i;

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

120 /* Pretraga po indeksu */
122 if (bin_pretraga) {
    /* Unos indeksa koji se binarno trazi u nizu */
    printf("Unesite indeks studenta cije informacije zelite: ");
124     scanf("%ld", &trazen_indeks);
    i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
126     /* Rezultat binarne pretrage */
    if (i == -1)
128         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
130         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
132 }
    /* Pretraga po prezimenu */
134 else {
    /* Unos prezimena koje se linearno trazi u nizu */
    printf("Unesite prezime studenta cije informacije zelite: ");
136     scanf("%s", trazeno_prezime);
    i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
138     /* Rezultat linearne pretrage */
    if (i == -1)
140         printf("Ne postoji student sa prezimenom %s\n",
            trazeno_prezime);
    else
142         printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
144 }
146 }

148 exit(EXIT_SUCCESS);
}

```

### Rešenje 3.4

```

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

5  #define MAX_STUDENATA 128
   #define MAX_DUZINA 16
7
   typedef struct {
9     long indeks;

```

### 3 Algoritmi pretrage i sortiranja

---

```
char ime[MAX_DUZINA];
11 char prezime[MAX_DUZINA];
} Student;

13
15 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
    long x)
{
17     /* Ako je pozicija elementa na levom kraju veca od pozicije
        elementa na desnom kraju dela niza koji se pretrazuje, onda se
19     zapravo pretrazuje prazan deo niza. U praznom delu niza nema
        trazenog elementa pa se vraca -1 */
21     if (levi > desni)
        return -1;
23     /* Racuna se pozicija srednjeg elementa */
    int srednji = (levi + desni) / 2;
25     /* Da li je srednji bas onaj trazeni */
    if (a[srednji].indeks == x) {
27         return srednji;
    }
29     /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
        poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
31     je poznato da je niz sortiran po indeksu u rastucem poretku. */
    if (x < a[srednji].indeks)
33         return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
    /* Inace ga treba traziti u desnoj polovini */
35     else
        return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }

39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
{
41     /* Ako je niz prazan, vraca se -1 */
    if (n == 0)
43         return -1;
    /* Kako se trazi prvi student sa trazanim prezimenom, pocinje se
45     sa prvim studentom u nizu. */
    if (strcmp(a[0].prezime, x) == 0)
47         return 0;
    int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
49     return i >= 0 ? 1 + i : -1;
}

51
53 int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
{
    /* Ako je niz prazan, vraca se -1 */
55     if (n == 0)
        return -1;
57     /* Ako se trazi poslednji student sa trazanim prezimenom, pocinje
        se sa poslednjim studentom u nizu. */
    if (strcmp(a[n - 1].prezime, x) == 0)
59         return n - 1;
61     return linearna_pretraga_rekurzivna(a, n - 1, x);
}
```

```
}
63
64 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
65    prosledjuju kao argumenti komandne linije */
66 int main(int argc, char *argv[])
67 {
68     Student dosije[MAX_STUDENATA];
69     FILE *fin = NULL;
70     int i;
71     int br_studenata = 0;
72     long trazen_indeks = 0;
73     char trazeno_prezime[MAX_DUZINA];
74     int bin_pretraga;
75
76     /* Provera da li je korisnik prilikom poziva programa prosledio
77        ime datoteke sa informacijama o studentima i opciju pretrage */
78     if (argc != 3) {
79         fprintf(stderr,
80             "Greska: Program se poziva sa %s ime_datoteke opcija\n",
81             argv[0]);
82         exit(EXIT_FAILURE);
83     }
84
85     /* Provera prosledjene opcije */
86     if (strcmp(argv[2], "-indeks") == 0)
87         bin_pretraga = 1;
88     else if (strcmp(argv[2], "-prezime") == 0)
89         bin_pretraga = 0;
90     else {
91         fprintf(stderr,
92             "Greska: Opcija mora biti -indeks ili -prezime\n");
93         exit(EXIT_FAILURE);
94     }
95
96     /* Otvara se datoteka */
97     fin = fopen(argv[1], "r");
98     if (fin == NULL) {
99         fprintf(stderr,
100             "Greska: Neuspesno otvaranje datoteke %s za citanje\n",
101             argv[1]);
102         exit(EXIT_FAILURE);
103     }
104
105     /* Cita se sve dok postoji red sa informacijama o studentu */
106     i = 0;
107     while (1) {
108         if (i == MAX_STUDENATA)
109             break;
110         if (fscanf
111             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
112              dosije[i].prezime) != 3)
113             break;
```

### 3 Algoritmi pretrage i sortiranja

```

    i++;
115 }
    br_studenata = i;
117
119 /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
    fclose(fin);
121
123 /* Pretraga po indeksu */
    if (bin_pretraga) {
125         /* Unos indeksa koji se binarno trazi u nizu */
        printf("Unesite indeks studenta cije informacije želite: ");
127         scanf("%ld", &trazen_indeks);
        i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
                                         trazen_indeks);
129         /* Rezultat binarne pretrage */
        if (i == -1)
            printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
131         else
            printf("Indeks: %ld, Ime i prezime: %s %s\n",
133                  dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
    }
135 /* Pretraga po prezimenu */
    else {
137         /* Unos prezimena koje se linearno trazi u nizu */
        printf("Unesite prezime studenta cije informacije želite: ");
139         scanf("%s", trazeno_prezime);
        i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
                                             trazeno_prezime);
141         /* Rezultat linearne pretrage */
        if (i == -1)
            printf("Ne postoji student sa prezimenom %s\n",
143                  trazeno_prezime);
145         else
            printf("Indeks: %ld, Ime i prezime: %s %s\n",
147                  dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
149     }
151     exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.5

```

1 #include <stdio.h>
  #include <string.h>
3 #include <math.h>
  #include <stdlib.h>
5
  #define MAX 1024
7
  /* Struktura koja opisuje tacku u ravni */
9 typedef struct Tacka {
```

```
11     float x;
12     float y;
13 } Tacka;

14
15 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
16    pocetka (0,0) */
17 float rastojanje(Tacka A)
18 {
19     return sqrt(A.x * A.x + A.y * A.y);
20 }

21 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u
22    nizu zadatih tacaka t dimenzije n */
23 Tacka najbliza_koordinatnom(Tacka t[], int n)
24 {
25     Tacka najbliza;
26     int i;
27     najbliza = t[0];
28     for (i = 1; i < n; i++) {
29         if (rastojanje(t[i]) < rastojanje(najbliza)) {
30             najbliza = t[i];
31         }
32     }
33     return najbliza;
34 }

35
36 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
37    tacaka t dimenzije n */
38 Tacka najbliza_x_osi(Tacka t[], int n)
39 {
40     Tacka najbliza;
41     int i;
42     najbliza = t[0];
43     for (i = 1; i < n; i++) {
44         if (fabs(t[i].x) < fabs(najbliza.x)) {
45             najbliza = t[i];
46         }
47     }
48     return najbliza;
49 }

50
51 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
52    tacaka t dimenzije n */
53 Tacka najbliza_y_osi(Tacka t[], int n)
54 {
55     Tacka najbliza;
56     int i;
57     najbliza = t[0];
58     for (i = 1; i < n; i++) {
59         if (fabs(t[i].y) < fabs(najbliza.y)) {
60             najbliza = t[i];
61         }
62     }
63 }
```

```
    }
63     return najbliza;
64 }
65
66 int main(int argc, char *argv[])
67 {
68     FILE *ulaz;
69     Tacka tacke[MAX];
70     Tacka najbliza;
71     int i, n;
72
73     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
       ime datoteke sa tackama */
74     if (argc < 2) {
75         fprintf(stderr,
76             "Greska: Programa se poziva sa %s ime_datoteke\n",
77             argv[0]);
78         exit(EXIT_FAILURE);
79     }
80
81     /* Otvara se datoteka za citanje */
82     ulaz = fopen(argv[1], "r");
83     if (ulaz == NULL) {
84         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s!\n",
85             argv[1]);
86         exit(EXIT_FAILURE);
87     }
88
89     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
       tackama; i predstavlja indeks tekuce tacke */
90     i = 0;
91     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
92         i++;
93     }
94     n = i;
95
96     /* Proverava se koji su dodatni argumenti komandne linije. Ako
       nema dodatnih argumenata */
97     if (argc == 2)
98         /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
99         najbliza = najbliza_koordinatnom(tacke, n);
100     /* Inace proverava se koji je dodatni argument prosledjen. Ako je
       u pitanju opcija -x */
101     else if (strcmp(argv[2], "-x") == 0)
102         /* Racuna se rastojanje u odnosu na x osu */
103         najbliza = najbliza_x_osi(tacke, n);
104     /* Ako je u pitanju opcija -y */
105     else if (strcmp(argv[2], "-y") == 0)
106         /* Racuna se rastojanje u odnosu na y osu */
107         najbliza = najbliza_y_osi(tacke, n);
108     else {
109         /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
110         111
112         113
```

```

    korisnika i prekida se izvršavanje programa */
115     fprintf(stderr, "Greska: Pogresna opcija\n");
    exit(EXIT_FAILURE);
117 }

/* Stampaju se koordinate trazene tacke */
119 printf("%g %g\n", najbliza.x, najbliza.y);
121
/* Zatvara se datoteka */
123 fclose(ulaz);

125 exit(EXIT_SUCCESS);
}

```

### Rešenje 3.6

```

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

/* Tacnost */
4  #define EPS 0.001

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

10
    /* Kod intervala [0, 2] leva granica je 0, a desna 2 */
12    l = 0;
    d = 2;

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

32
    /* Stampa se vrednost trazene tacke */
34    printf("%g\n", s);

```

```
36     return 0;
    }
```

#### Rešenje 3.7

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

6 int main(int argc, char **argv)
{
8     double l, d, s, epsilon;

10     char ime_funkcije[6];

12     /* Pokazivac na funkciju koja ima jedan argument tipa double i
        povratnu vrednost istog tipa */
14     double (*fp) (double);

16     /* Ako korisnik nije uneo argument, prijavljuje se greska */
    if (argc != 2) {
18         fprintf(stderr, "Greska: Program se poziva sa %s ime_funkcije\n",
            argv[0]);
20         exit(EXIT_FAILURE);
    }

22     /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
        u komandnoj liniji */
24     strcpy(ime_funkcije, argv[1]);

26     /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
28     if (strcmp(ime_funkcije, "sin") == 0)
        fp = &sin;
30     else if (strcmp(ime_funkcije, "cos") == 0)
        fp = &cos;
32     else if (strcmp(ime_funkcije, "tan") == 0)
        fp = &tan;
34     else if (strcmp(ime_funkcije, "atan") == 0)
        fp = &atan;
36     else if (strcmp(ime_funkcije, "asin") == 0)
        fp = &asin;
38     else if (strcmp(ime_funkcije, "log") == 0)
        fp = &log;
40     else if (strcmp(ime_funkcije, "log10") == 0)
        fp = &log10;
42     else {
        fprintf(stderr,
44             "Greska: Program ne podrzava trazenu funkciju!\n");
        exit(EXIT_SUCCESS);
    }

46 }
```



```

48 printf("Unesite krajeve intervala: ");
   scanf("%lf %lf", &l, &d);
50
   if ((*fp) (l) * (*fp) (d) >= 0) {
52     fprintf(stderr, "Greska: %s na intervalu ", ime_funkcije);
       fprintf(stderr, "[%g, %g] ne zadovoljava uslove\n", l, d);
54     exit(EXIT_FAILURE);
   }
56
   printf("Unesite preciznost: ");
58   scanf("%lf", &epsilon);
60
   /* Sve dok se ne pronadje trazena vrednost argumenta */
   while (1) {
62     /* Polovi se interval */
       s = (l + d) / 2;
64     /* Ako je apsolutna vrednost trazene funkcije u ovoj tacki manja
       od zadate tacnosti, prekida se pretraga */
66     if (fabs((*fp) (s)) < epsilon) {
       break;
68     }
       /* Ako je nula u levom delu intervala, nastavlja se pretraga na
70     [l, s] */
       if ((*fp) (l) * (*fp) (s) < 0)
72         d = s;
       else
74         /* Inace, na intervalu [s, d] */
           l = s;
76   }
78
   /* Stampa se vrednost trazene tacke */
   printf("%g\n", s);
80
   exit(EXIT_SUCCESS);
82 }

```

### Rešenje 3.8

```

#include <stdio.h>
2  #include <stdlib.h>
4
#define MAX 256
6
int prvi_veci_od_nule(int niz[], int n)
{
8   /* Granice pretrage */
   int l = 0, d = n - 1;
10  int s;
   /* Sve dok je leva manja od desne granice */
12  while (l <= d) {

```

### 3 Algoritmi pretrage i sortiranja

```
14      /* Racuna se sredisnja pozicija */
15      s = (l + d) / 2;
16      /* Ako je broj na toj poziciji veci od nule, a eventualni njegov
17       prethodnik manji ili jednak nuli, pretraga je završena */
18      if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
19          return s;
20      /* U slučaju broja manjeg ili jednakog nuli, pretražuje se desna
21       polovina niza */
22      if (niz[s] <= 0)
23          l = s + 1;
24      /* A inace, leva polovina */
25      else
26          d = s - 1;
27  }
28  return -1;
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      /* Stampa se rezultat */
40      printf("%d\n", prvi_veci_od_nule(niz, n));
41
42      return 0;
43  }
```

#### 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 0, a eventualni njegov
16          prethodnik veci ili jednak 0, pretraga se završava */
17         if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
```

```

    return s;
19  /* Ako je broj veci ili jednak 0, pretrazuje se desna polovina
    niza */
21  if (niz[s] >= 0)
    l = s + 1;
23  /* A inace leva */
    else
25      d = s - 1;
    }
27  return -1;
}

29
int main()
31 {
    int niz[MAX];
33     int n = 0;

35     /* Unos niza */
    while (scanf("%d", &niz[n]) == 1)
37         n++;

39     /* Stampa se rezultat */
    printf("%d\n", prvi_manji_od_nule(niz, n));
41
    return 0;
43 }

```

### Rešenje 3.10

```

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

3
   unsigned int logaritam_a(unsigned int x)
5  {
    /* Izlaz iz rekurzije */
7   if (x == 1)
        return 0;
9   /* Rekurzivni korak */
    return 1 + logaritam_a(x >> 1);
11 }

13 unsigned int logaritam_b(unsigned int x)
   {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
        najveće važnosti, tj. najlevlja. Pretragu se vrši od pozicije 0
        do 31 */
17     int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
    /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
        /* Racuna se sredisnja pozicija */

```

### 3 Algoritmi pretrage i sortiranja

```
23     s = (l + d) / 2;
    /* Proverava se da li je na toj poziciji trazena jedinica */
25     if ((l << s) <= x && (l << (s + 1)) > x)
        return s;
27     /* Pretraga desne polovine binarnog zapisa */
    if ((l << s) > x)
29         l = s - 1;
    /* Pretraga leve polovine binarnog zapisa */
31     else
        d = s + 1;
33 }
    return s;
35 }

37 int main()
{
39     unsigned int x;

41     /* Unos podatka */
    scanf("%u", &x);
43
    /* Provera da li je uneti broj pozitivan */
45     if (x == 0) {
        fprintf(stderr, "Greska: Logaritam od nule nije definisan\n");
47         exit(EXIT_FAILURE);
    }
49
    /* Ispis povratnih vrednosti funkcija */
51     printf("%u %u\n", logaritam_a(x), logaritam_b(x));

53     exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.12

*sort.h*

```
#ifndef _SORT_H_
2 #define _SORT_H_ 1

4 /* Selection sort: Funkcija sortira niz celih brojeva metodom
   sortiranja izborom. Ideja algoritma je sledeca: U svakoj
6   iteraciji pronalazi se najmanji element i premesta se na pocetak
   niza. Dakle, u prvoj iteraciji, pronalazi se najmanji element, i
8   dovodi na nultu mesto u nizu. U i-toj iteraciji najmanjih i-1
   elemenata su vec na svojim pozicijama, pa se od elemenata sa
10   indeksima od i do n-1 trazi najmanji, koji se dovodi na i-tu
   poziciju. */
12 void selection_sort(int a[], int n);
```

```

14  /* Insertion sort: Funkcija sortira niz celih brojeva metodom
15     sortiranja umetanjem. Ideja algoritma je sledeca: 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 medju prvih i elemenata tako da se
19     dobije niz duzine i+1 koji je sortiran. Ovo se radi tako sto se
20     i-ti element najpre uporedi sa njegovim prvim levim susedom
21     (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i niz
22     a[0],a[1],...,a[i] je sortiran, pa se moze preci na sledecu
23     iteraciju. Ako je a[i-1] vece, tada se zamenjuju a[i] i a[i-1], a
24     zatim se proverava da li je potrebno dalje potiskivanje elementa u
25     levo, poredeci ga sa njegovim novim levim susedom. Ovim uzastopnim
26     premestanjem se a[i] umece na pravo mesto u nizu. */
27 void insertion_sort(int a[], int n);
28
29 /* Bubble sort: Funkcija sortira niz celih brojeva metodom mehurica.
30     Ideja algoritma je sledeca: prolazi se kroz niz redom poredeci
31     susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
32     poretku. Ovim se najveći element poput mehurica istiskuje na
33     "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je potrebno
34     ovaj postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih
35     n-1 elemenata niza bez poslednjeg koji je postavljen na pravu
36     poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
37     kracim prefiksima niza, cime se jedan po jedan istiskuju
38     elementi na svoje prave pozicije. */
39 void bubble_sort(int a[], int n);
40
41 /* 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
44     prolazi vise puta; u prvom prolazu, umesto koraka 1 uzima se neki
45     korak h koji je manji od n (sto omogucuje razmenu udaljenih
46     elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
47     elementi na rastojanju h sortirani, mada susedni elementi to ne
48     moraju biti. U drugom prolazu kroz isti algoritam sprovodi se
49     isti postupak ali za manji korak h. Sa prolazima se nastavlja sve
50     do koraka h = 1, u kome se dobija potpuno sortirani niz. Izbor
51     pocetne vrednosti za h, i nacina njegovog smanjivanja menja u
52     nekim slucajevima brzinu algoritma, ali bilo koja vrednost ce
53     rezultovati ispravnim sortiranjem, pod uslovom da je u poslednjoj
54     iteraciji h imalo vrednost 1. */
55 void shell_sort(int a[], int n);
56
57 /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
58     Sortiranje se vrši 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 rekurzivno
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
        algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u

```

### 3 Algoritmi pretrage i sortiranja

---

```
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
70 l i d. Njena ideja sortiranja je izbor jednog elementa niza, koji
71 se naziva pivot, i koji se dovodi na svoje mesto. Posle ovog
72 koraka, svi elementi levo od njega bice manji, a svi desno bice
73 veci od njega. Kako je pivot doveden na svoje mesto, da bi niz
74 bio kompletno sortiran, potrebno je sortirati elemente levo
75 (manje) od njega, i elemente desno (vece). Kako su dimenzije ova
76 dva podniza manje od dimenzije pocetnog niza koji je trebalo
77 sortirati, ovaj deo moze se uraditi rekurzivno. */
78 void quick_sort(int a[], int l, int d);
79
80 #endif
```

sort.c

```
#include "sort.h"
2
#define MAX 1000000
4
void selection_sort(int a[], int n)
6 {
    int i, j;
    int min;
    int pom;
10
    /* U svakoj iteraciji ove petlje pronalazi se najmanji element
12 medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
    poziciju i, dok se element na poziciji i premesta na poziciju
14 min, na kojoj se nalazio najmanji od navedenih elemenata. */
    for (i = 0; i < n - 1; i++) {
16         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
            najmanji od elemenata a[i],...,a[n-1]. */
18         min = i;
            for (j = i + 1; j < n; j++)
20             if (a[j] < a[min])
                min = j;
22
23         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo
24 ako su (i) i min razliciti, inace je nepotrebno. */
        if (min != i) {
26             pom = a[i];
            a[i] = a[min];
28             a[min] = pom;
        }
30     }
}
32
void insertion_sort(int a[], int n)
```

```

34 {
35     int i, j;
36
37     /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
38        sortiran */
39     for (i = 1; i < n; i++) {
40         /* U ovoj petlji se redom potiskuje element a[i] ulevo 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 završava 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         /* Poput "mehurica" potiskuje se najveći element medju
60            elementima od a[0] do a[i-1] na poziciju i-1 upoređujući
61            susedne elemente niza i potiskujući veci u desno */
62         for (j = 0, ind = 0; j < i - 1; j++)
63             if (a[j] > a[j + 1]) {
64                 int temp = a[j];
65                 a[j] = a[j + 1];
66                 a[j + 1] = temp;
67                 /* Promenljiva ind registruje da je bilo premestanja. Samo u
68                    tom slucaju ima smisla ici na sledecu iteraciju, jer ako
69                    nije bilo premestanja, znaci da su svi elementi vec u
70                    dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
71                    niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
72                    ispravno, ali manje efikasano, jer bi se cesto nepotrebno
73                    vrsila mnoga uporedjivanja, kada je vec jasno da je
74                    sortiranje završeno. */
75                 ind = 1;
76             }
77 }
78
79 void shell_sort(int a[], int n)
80 {
81     int h = n / 2, i, j;
82     while (h > 0) {
83         /* Insertion sort sa korakom h */
84         for (i = h; i < n; i++) {
85             int temp = a[i];

```

```
86     j = i;
87     while (j >= h && a[j - h] > temp) {
88         a[j] = a[j - h];
89         j -= h;
90     }
91     a[j] = temp;
92 }
93 h = h / 2;
94 }
95 }
96
97 void merge_sort(int a[], int l, int d)
98 {
99     int s;
100     static int b[MAX];          /* Pomocni niz */
101     int i, j, k;
102
103     /* Izlaz iz rekurziije */
104     if (l >= d)
105         return;
106
107     /* Odredjivanje sredisnjeg indeksa */
108     s = (l + d) / 2;
109
110     /* Rekurzivni pozivi */
111     merge_sort(a, l, s);
112     merge_sort(a, s + 1, d);
113
114     /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
115        niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
116        prolazi kroz pomocni niz b[] */
117     i = l;
118     j = s + 1;
119     k = 0;
120
121     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
122     while (i <= s && j <= d) {
123         if (a[i] < a[j])
124             b[k++] = a[i++];
125         else
126             b[k++] = a[j++];
127     }
128
129     /* U slucaju da se prethodna petlja zavrsla izlaskom promenljive
130        j iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
131        polovine niza */
132     while (i <= s)
133         b[k++] = a[i++];
134
135     /* U slucaju da se prethodna petlja zavrsla izlaskom promenljive
136        i iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
137        polovine niza */
```



```

138     while (j <= d)
139         b[k++] = a[j++];
140
141     /* Prepisuje se "ucesljani" niz u originalni niz */
142     for (k = 0, i = 1; i <= d; i++, k++)
143         a[i] = b[k];
144 }
145
146 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
147 void swap(int a[], int i, int j)
148 {
149     int tmp = a[i];
150     a[i] = a[j];
151     a[j] = tmp;
152 }
153
154 void quick_sort(int a[], int l, int d)
155 {
156     int i, pivot_pozicija;
157
158     /* Izlaz iz rekurzije -- prazan niz */
159     if (l >= d)
160         return;
161
162     /* Partitionisanje niza. Svi elementi na pozicijama levo od
163     pivot_pozicija (izuzev same pozicije l) su strogo manji od
164     pivota. Kada se pronadje neki element manji od pivota, uvecava
165     se promenljiva pivot_pozicija i na tu poziciju se premesta
166     nadjeni element. Na kraju ce pivot_pozicija zaista biti
167     pozicija na koju treba smestiti pivot, jer ce svi elementi levo
168     od te pozicije biti manji a desno biti veci ili jednaki od
169     pivota. */
170     pivot_pozicija = l;
171     for (i = l + 1; i <= d; i++)
172         if (a[i] < a[l])
173             swap(a, ++pivot_pozicija, i);
174
175     /* Postavljanje pivota na svoje mesto */
176     swap(a, l, pivot_pozicija);
177
178     /* Rekurzivno sortiranje elementa manjih od pivota */
179     quick_sort(a, l, pivot_pozicija - 1);
180     /* Rekurzivno sortiranje elementa vecih od pivota */
181     quick_sort(a, pivot_pozicija + 1, d);
182 }

```

*main.c*

```

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

```

### 3 Algoritmi pretrage i sortiranja

```
4 #include "sort.h"

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

8
10 int main(int argc, char *argv[])
{
12     /******
13      tip_sortiranja == 0 => selectionsort, (podrazumevano)
14      tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
15      tip_sortiranja == 2 => bubblesort, -b opcija komandne linije
16      tip_sortiranja == 3 => shellsort, -s opcija komandne linije
17      tip_sortiranja == 4 => mergesort, -m opcija komandne linije
18      tip_sortiranja == 5 => quicksort, -q opcija komandne linije
19      *****/
20     int tip_sortiranja = 0;
21     /******
22      tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
23      tip_niza == 1 => rastuce sortirani nizovi, -r opcija
24      tip_niza == 2 => opadajuće soritrani nizovi, -o opcija
25      *****/
26     int tip_niza = 0;

28     /* Dimenzija niza koji se sortira */
29     int dimenzija;
30     int i;
31     int niz[MAX];

32     /* Provera argumenata komandne linije */
33     if (argc < 2) {
34         fprintf(stderr, "Greska: Program zahteva bar 2 ");
35         fprintf(stderr, "argumenta komandne linije!\n");
36         exit(EXIT_FAILURE);
37     }

38
39     /* Ocitavaju se opcije i argumenati komandne linije */
40     for (i = 1; i < argc; i++) {
41         /* Ako je u pitanju opcija... */
42         if (argv[i][0] == '-') {
43             switch (argv[i][1]) {
44                 case 'i':
45                     tip_sortiranja = 1;
46                     break;
47                 case 'b':
48                     tip_sortiranja = 2;
49                     break;
50                 case 's':
51                     tip_sortiranja = 3;
52                     break;
53                 case 'm':
54                     tip_sortiranja = 4;
55                     break;
```

```

56     case 'q':
57         tip_sortiranja = 5;
58         break;
59     case 'r':
60         tip_niza = 1;
61         break;
62     case 'o':
63         tip_niza = 2;
64         break;
65     default:
66         fprintf(stderr, "Greska: Pogresna opcija -%c\n", argv[i][1]);
67         exit(EXIT_SUCCESS);
68         break;
69     }
70 }
71 /* Ako je u pitanju argument, onda je to duzina niza koji treba
72    da se sortira */
73 else {
74     dimenzija = atoi(argv[i]);
75     if (dimenzija <= 0 || dimenzija > MAX) {
76         fprintf(stderr, "Greska: Dimenzija niza neodgovarajuca!\n");
77         exit(EXIT_FAILURE);
78     }
79 }
80 }
81
82 /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
83    niza dobijenom iz komandne linije. srand() funkcija obezbedjuje
84    novi seed za pozivanje rand funkcije, i kako generisani niz ne
85    bi uvek bio isti seed je postavljen na tekuce vreme u sekundama
86    od Nove godine 1970. rand()%100 daje brojeve izmedju 0 i 99 */
87 srand(time(NULL));
88 if (tip_niza == 0)
89     for (i = 0; i < dimenzija; i++)
90         niz[i] = rand();
91 else if (tip_niza == 1)
92     for (i = 0; i < dimenzija; i++)
93         niz[i] = i == 0 ? rand() % 100 : niz[i - 1] + rand() % 100;
94 else
95     for (i = 0; i < dimenzija; i++)
96         niz[i] = i == 0 ? rand() % 100 : niz[i - 1] - rand() % 100;
97
98 /* Ispisuju se elemenati niza */
99 /******
100    Ovaj deo je iskomentaran jer sledeci ispis ne treba da se nadje
101    na standardnom izlazu. Njegova svrha je samo bila provera da li
102    je niz generisan u skladu sa opcijama komandne linije.
103
104    printf("Niz koji sortiramo je:\n");
105    for (i = 0; i < dimenzija; i++)
106        printf("%d\n", niz[i]);
107    *****/

```

### 3 Algoritmi pretrage i sortiranja

```
108  /* Sortira se niz na odgovarajuci nacin */
110  if (tip_sortiranja == 0)
        selection_sort(niz, dimenzija);
112  else if (tip_sortiranja == 1)
        insertion_sort(niz, dimenzija);
114  else if (tip_sortiranja == 2)
        bubble_sort(niz, dimenzija);
116  else if (tip_sortiranja == 3)
        shell_sort(niz, dimenzija);
118  else if (tip_sortiranja == 4)
        merge_sort(niz, 0, dimenzija - 1);
120  else
        quick_sort(niz, 0, dimenzija - 1);

122
124  /* Ispisuju se elementi niza */
124  /*****
        Ovaj deo je iskomentaran jer vreme potrebno za njegovo
126  izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
        programom time. Takodje, kako je svrha ovog programa da prikaze
128  vremena razlicitih algoritama sortiranja, dimenzije nizova ce
        biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
130  od toliko elemenata. Ovaj deo je koriscen u razvoju programa
        zarad testiranja korektnosti.

132  printf("Sortiran niz je:\n");
134  for (i = 0; i < dimenzija; i++)
        printf("%d\n", niz[i]);
136  *****/

138  exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.13

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

4  #define MAX_DIM 128

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

```
18     pom = s[i];
19     s[i] = s[min];
20     s[min] = pom;
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     /* Sortiraju se karakteri */
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     /* Ucitavaju se niske 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

NAPOMENA: Rešenje koristi biblioteku za sortiranje celih brojeva iz zadatka 3.12.

```
1 #include <stdio.h>
2 #include "sort.h"
3
4 #define MAX 256
5
6 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
7    sortiranom nizu celih brojeva */
8 int najmanje_rastojanje(int a[], int n)
9 {
10     int i, min;
11     /* Postavlja se najmanje rastojanje na razliku prvog i drugog
12        elementa niza */
13     min = a[1] - a[0];
14     /* Za sve ostale susedne elemente proverava se njigova razlika */
15     for (i = 2; i < n; i++)
16         if (a[i] - a[i - 1] < min)
17             min = a[i] - a[i - 1];
18     return min;
19 }
20
21 int main()
22 {
23     int i, a[MAX];
24
25     /* Ucitavaju se elementi niza sve do kraja ulaza */
26     i = 0;
27     while (scanf("%d", &a[i]) != EOF)
28         i++;
29
30     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
31        sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
32        se selection sort. */
33     selection_sort(a, i);
34
35     /* Ispisuje se rezultat */
36     printf("%d\n", najmanje_rastojanje(a, i));
37
38     return 0;
39 }
```

#### Rešenje 3.15

NAPOMENA: Rešenje koristi biblioteku za sortiranje celih brojeva iz zadatka 3.12.

```
1 #include <stdio.h>
2 #include "sort.h"
3
4 #define MAX_DIM 256
```

```

5  /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
7     najviše puta pojavio u tom nizu */
8  int najvise_puta(int a[], int n)
9  {
10     int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
11     /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
12     for (i = 0; i < n; i = j) {
13         br_pojava = 1;
14         for (j = i + 1; j < n && a[i] == a[j]; j++)
15             br_pojava++;
16         /* Ispituje se da li se do tog trenutka i-ti element pojavio
17            najviše puta u nizu */
18         if (br_pojava > max_br_pojava) {
19             max_br_pojava = br_pojava;
20             i_max_pojava = i;
21         }
22     }
23     /* Vraca se element koji se najviše puta pojavio u nizu */
24     return a[i_max_pojava];
25 }

27 int main()
28 {
29     int a[MAX_DIM], i;

31     /* Ucitavaju se elemenati niza sve do kraja ulaza */
32     i = 0;
33     while (scanf("%d", &a[i]) != EOF)
34         i++;

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);

41     /* Odredjuje se broj koji se najviše puta pojavio u nizu */
42     printf("%d\n", najvise_puta(a, i));

43     return 0;
44 }

```

### Rešenje 3.16

NAPOMENA: Rešenje koristi biblioteku za sortiranje celih brojeva iz zadatka 3.12.

```

1  #include <stdio.h>
2  #include "sort.h"
3
4  #define MAX_DIM 256
5

```

### 3 Algoritmi pretrage i sortiranja

---

```
7  /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
   u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
   poretku */
9  int binarna_pretraga(int a[], int n, int x)
   {
11     int levi = 0, desni = n - 1, srednji;

13     while (levi <= desni) {
        srednji = (levi + desni) / 2;
15         if (a[srednji] == x)
            return 1;
17         else if (a[srednji] > x)
            desni = srednji - 1;
19         else if (a[srednji] < x)
            levi = srednji + 1;
21     }
        return 0;
23 }

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

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

33     /* Ucitavaju se elementi niza sve do kraja ulaza */
        i = 0;
35     printf("Unesite elemente niza: ");
        while (scanf("%d", &a[i]) != EOF)
37         i++;
        n = i;
39

41     /* Za sortiranje niza moze se koristiti bilo koja od funkcija
        sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
        se quick sort. */
43     quick_sort(a, 0, n - 1);

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

55     return 0;
   }
```



## 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 slucaju 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 prepisuje se u treci */
    while (i < dim1)
        niz3[k++] = niz1[i++];

    /* Ostatak drugog niza prepisuje se 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];
    int i = 0, j = 0, k, dim3;

    /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
       Pretpostavka je da na ulazu nema vise od MAX_DIM elemenata */
    printf("Unesite elemente prvog niza: ");
    while (1) {
        scanf("%d", &niz1[i]);
        if (niz1[i] == 0)
            break;
        i++;
    }
    printf("Unesite elemente drugog niza: ");
    while (1) {
        scanf("%d", &niz2[j]);
        if (niz2[j] == 0)

```

### 3 Algoritmi pretrage i sortiranja

---

```
        break;
52     j++;
    }

54     /* Poziv trazene funkcije */
56     dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

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

62     return 0;
64 }
```

#### Rešenje 3.18

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

4
int main(int argc, char *argv[])
6 {
    FILE *fin1 = NULL, *fin2 = NULL;
    FILE *fout = NULL;
    char ime1[11], ime2[11];
10    char prezime1[16], prezime2[16];
    int kraj1 = 0, kraj2 = 0;

12
    /* Ako nema dovoljno argumenata komandne linije */
14    if (argc < 3) {
        fprintf(stderr,
16             "Greska: Program se poziva sa %s datoteka1 datoteka2\n",
                argv[0]);
18        exit(EXIT_FAILURE);
    }

20
    /* Otvara se datoteka zadata prvim argumentom komandne linije */
22    fin1 = fopen(argv[1], "r");
    if (fin1 == NULL) {
24        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s\n",
                argv[1]);
26        exit(EXIT_FAILURE);
    }

28
    /* Otvara se datoteka zadata drugim argumentom komandne linije */
30    fin2 = fopen(argv[2], "r");
    if (fin2 == NULL) {
32        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s\n",
                argv[2]);
34        exit(EXIT_FAILURE);
    }
}
```

```

36     }
37
38     /* Otvara se datoteka za upis rezultata */
39     fout = fopen("ceo-tok.txt", "w");
40     if (fout == NULL) {
41         fprintf(stderr,
42             "Greska: Neuspesno otvaranje datoteke ceo-tok.txt\n");
43         exit(EXIT_FAILURE);
44     }
45
46     /* Cita se prvi student iz prve datoteke */
47     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
48         kraj1 = 1;
49
50     /* Cita se prvi student iz druge datoteke */
51     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
52         kraj2 = 1;
53
54     /* Sve dok nije dostignut kraj neke datoteke */
55     while (!kraj1 && !kraj2) {
56         int tmp = strcmp(ime1, ime2);
57         if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
58             /* Ime i prezime iz prve datoteke je leksikografski ranije, i
59                biva upisano u izlaznu datoteku */
60             fprintf(fout, "%s %s\n", ime1, prezime1);
61             /* Cita se naredni student iz prve datoteke */
62             if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
63                 kraj1 = 1;
64         } else {
65             /* Ime i prezime iz druge datoteke je leksikografski ranije, i
66                biva upisano u izlaznu datoteku */
67             fprintf(fout, "%s %s\n", ime2, prezime2);
68             /* Cita se naredni student iz druge datoteke */
69             if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
70                 kraj2 = 1;
71         }
72     }
73
74     /* Ako se iz prethodne petlje izašlo zato što je dostignut kraj
75        druge datoteke, onda ima još studenata u prvoj datoteci, koje
76        treba prepisati u izlaznu, redom, jer su već sortirani po
77        imenu. */
78     while (!kraj1) {
79         fprintf(fout, "%s %s\n", ime1, prezime1);
80         if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
81             kraj1 = 1;
82     }
83
84     /* Ako se iz prve petlje izašlo zato što je dostignut kraj prve
85        datoteke, onda ima još studenata u drugoj datoteci, koje treba
86        prepisati u izlaznu, redom, jer su već sortirani po imenu. */
87     while (!kraj2) {

```

### 3 Algoritmi pretrage i sortiranja

---

```
    fprintf(fout, "%s %s\n", ime2, prezime2);
88     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;
90 }

92 /* Zatvaraju se datoteke */
    fclose(fin1);
94     fclose(fin2);
    fclose(fout);

96     exit(EXIT_SUCCESS);
98 }
```

#### Rešenje 3.19

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

27     for (i = 0; i < n - 1; i++) {
        min = i;
29         for (j = i + 1; j < n; j++) {
            if (rastojanje(t[j]) < rastojanje(t[min])) {
31                 min = j;
            }
33         }
        if (min != i) {
35             tmp = t[i];
            t[i] = t[min];
            t[min] = tmp;
        }
    }
}
```

```
37         t[min] = tmp;
38     }
39 }
40 }
41
42 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
43 void sortiraj_po_x(Tacka t[], int n)
44 {
45     int min, i, j;
46     Tacka tmp;
47
48     for (i = 0; i < n - 1; i++) {
49         min = i;
50         for (j = i + 1; j < n; j++) {
51             if (abs(t[j].x) < abs(t[min].x)) {
52                 min = j;
53             }
54         }
55         if (min != i) {
56             tmp = t[i];
57             t[i] = t[min];
58             t[min] = tmp;
59         }
60     }
61 }
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         }
76         if (min != i) {
77             tmp = t[i];
78             t[i] = t[min];
79             t[min] = tmp;
80         }
81     }
82 }
83
84 int main(int argc, char *argv[])
85 {
86     FILE *ulaz;
87     FILE *izlaz;
88     Tacka tacke[MAX_BR_TACAKA];
```

```
89  int i, n;

91  /* Proverava se broj argumenata komandne linije: ocekuje se ime
    izvrnog programa, opcija, ime ulazne datoteke i ime izlazne
93  datoteke, tj. 4 argumenta */
    if (argc != 4) {
95      fprintf(stderr,
          "Greska: Program se poziva sa %s opcija ulaz izlaz\n",
97      argv[0]);
      exit(EXIT_FAILURE);
99  }

101 /* Otvara se datoteka u kojoj su zadate tacke */
    ulaz = fopen(argv[2], "r");
103 if (ulaz == NULL) {
    fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s!\n",
105      argv[2]);
    exit(EXIT_FAILURE);
107 }

109 /* Otvara se datoteka u koju treba upisati rezultat */
    izlaz = fopen(argv[3], "w");
111 if (izlaz == NULL) {
    fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s!\n",
113      argv[3]);
    exit(EXIT_FAILURE);
115 }

117 /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
    koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
119 brojacem i. */
    i = 0;
121 while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
    i++;
123 }

125 /* Ukupan broj procitanih tacaka */
    n = i;
127

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

```

141     case 'o':
142         /* Sortira se po udaljenosti od koordinatnog pocetka */
143         sortiraj_po_rastojanju(tacke, n);
144         break;
145     }

147     /* Upisuje se dobijeni niz u izlaznu datoteku */
148     for (i = 0; i < n; i++) {
149         fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
150     }

151     /* Zatvaraju se otvorene datoteke */
152     fclose(ulaz);
153     fclose(izlaz);

154     exit(EXIT_SUCCESS);
155 }
156
157 }

```

### Rešenje 3.20

```

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

#define MAX 1000
#define MAX_DUZINA 16

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

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

    for (i = 0; i < n - 1; i++) {
        /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
           najmanji od elemenata a[i].ime,...,a[n-1].ime. */
        min = i;
        for (j = i + 1; j < n; j++)
            if (strcmp(a[j].ime, a[min].ime) < 0)
                min = j;
        /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo
           ako su (i) i min razliciti, inace je nepotrebno. */
        if (min != i) {
            pom = a[i];
            a[i] = a[min];
            a[min] = pom;
        }
    }
}

```

### 3 Algoritmi pretrage i sortiranja

---

```
32     a[i] = a[min];
33     a[min] = pom;
34 }
35 }
36 }

38 /* Funkcija sortira niz gradjana rastuce po prezimenima */
39 void sort_prezime(Gradjanin a[], int n)
40 {
41     int i, j;
42     int min;
43     Gradjanin pom;

44     for (i = 0; i < n - 1; i++) {
45         /* Unutrasnja petlja pronalazi 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
52            ako 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 }

62 /* Pretraga niza gradjana */
63 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64 {
65     int i;
66     for (i = 0; i < n; i++)
67         if (strcmp(a[i].ime, x->ime) == 0
68             && strcmp(a[i].prezime, x->prezime) == 0)
69             return i;
70     return -1;
71 }

72

74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;

80     /* Otvara se datoteka */
81     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
82         fprintf(stderr,
```



```

84         "Greska: Neupesno otvaranje datoteke za citanje.\n");
      exit(EXIT_FAILURE);
86     }

88     /* Cita se sadrzaj */
    for (i = 0;
90         fscanf(fp, "%s %s", spisak1[i].ime,
                 spisak1[i].prezime) != EOF; i++)
92         spisak2[i] = spisak1[i];
    n = i;

94     /* Zatvara se datoteka */
96     fclose(fp);

98     sort_ime(spisak1, n);

100    /*****
      Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
102      sortiranih nizova. Koriscen je samo u fazi testiranja programa.

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

108     sort_prezime(spisak2, n);

110    /*****
      Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
112      sortiranih nizova. Koriscen je samo u fazi testiranja programa.

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

120    /* Linearno se pretrazuju nizovi */
    for (i = 0; i < n; i++)
122        if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
            isti_rbr++;

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

132    /* Ispisuje se rezultat */
134    printf("%d\n", isti_rbr);

```

```
136     exit(EXIT_SUCCESS);  
    }
```

#### Rešenje 3.22

```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  #include <string.h>  
4  #include <ctype.h>  
5  
6  #define MAX_BR_RECII 128  
7  #define MAX_DUZINA_RECII 32  
8  
9  /* Funkcija koja izracunava broj suglasnika u reci */  
10 int broj_suglasnika(char s[])  
11 {  
12     char c;  
13     int i;  
14     int suglasnici = 0;  
15     /* Prolazi karakter po karakter kroz zadatu nisku */  
16     for (i = 0; s[i]; i++) {  
17         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio  
18            pokriven slucaj i malih i velikih suglasnika. */  
19         if (isalpha(s[i])) {  
20             c = toupper(s[i]);  
21             /* Ukoliko slovo nije samoglasnik uvecava se brojac. */  
22             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')  
23                 suglasnici++;  
24         }  
25     }  
26     /* Vraca se izracunata vrednost */  
27     return suglasnici;  
28 }  
29  
30 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o  
31    duzini reci se mora proslediti zbog pravilnog upravljanja  
32    memorijom */  
33 void sortiraj_reci(char reci[][MAX_DUZINA_RECII], int n)  
34 {  
35     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,  
36         duzina_j, duzina_min;  
37     char tmp[MAX_DUZINA_RECII];  
38     for (i = 0; i < n - 1; i++) {  
39         min = i;  
40         for (j = i; j < n; j++) {  
41             /* Prvo se upoređuje broj suglasnika */  
42             broj_suglasnika_j = broj_suglasnika(reci[j]);  
43             broj_suglasnika_min = broj_suglasnika(reci[min]);  
44             if (broj_suglasnika_j < broj_suglasnika_min)  
45                 min = j;  
46             else if (broj_suglasnika_j == broj_suglasnika_min) {  
47                 /* Ako su brojevi suglasnika jednaki, onda se  
48                    sortira po duzini reči. */  
49                 duzina_j = strlen(reci[j]);  
50                 duzina_min = strlen(reci[min]);  
51                 if (duzina_j < duzina_min)  
52                     min = j;  
53                 else if (duzina_j == duzina_min)  
54                     /* Ako su i brojevi suglasnika i duzine reči  
55                        jednake, onda se sortira po leksikografskom  
56                        poretku. */  
57                     if (strcmp(reci[j], reci[min]) < 0)  
58                         min = j;  
59                 }  
60             }  
61         }  
62         /* Zamena reči na mestima i i min */  
63         strcpy(tmp, reci[i]);  
64         strcpy(reci[i], reci[min]);  
65         strcpy(reci[min], tmp);  
66     }  
67 }
```

```

47      /* Zatim, recima koje imaju isti broj suglasnika upoređuju
        se duzine */
49      duzina_j = strlen(reci[j]);
        duzina_min = strlen(reci[min]);

51
53      if (duzina_j < duzina_min)
        min = j;
        else {
55          /* Ako reci imaju i isti broj suglasnika i iste duzine,
            upoređuju se leksikografski */
57          if (duzina_j == duzina_min
              && strcmp(reci[j], reci[min]) < 0)
            min = j;
        }
61    }
    }
63    if (min != i) {
        strcpy(tmp, reci[min]);
65        strcpy(reci[min], reci[i]);
        strcpy(reci[i], tmp);
67    }
    }
69 }

71 int main()
    {
73     FILE *ulaz;
        int i = 0, n;

75
77     /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
        a drugi maksimalnu duzinu pojedinačne reci */
        char reci[MAX_BR_RECII][MAX_DUZINA_RECII];

79
81     /* Otvara se datoteka niske.txt za citanje */
        ulaz = fopen("niske.txt", "r");
        if (ulaz == NULL) {
83             fprintf(stderr,
                "Greska: Neuspesno otvaranje datoteke niske.txt!\n");
85             exit(EXIT_FAILURE);
        }

87
89     /* Sve dok se moze procitati sledeca rec */
        while (fscanf(ulaz, "%s", reci[i]) != EOF) {
        /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
            prekida se ucitavanje */
91         if (i == MAX_BR_RECII)
            break;
93         /* Priprema brojaca za narednu iteraciju */
        i++;
95    }

97
    /* n je duzina niza reci i predstavlja poslednju vrednost

```

### 3 Algoritmi pretrage i sortiranja

---

```
99     koriscenog brojaca */
    n = i;
101 /* Poziva se funkcija za sortiranje reci */
    sortiraj_reci(reci, n);
103
    /* Ispis sortiranog niza reci */
105 for (i = 0; i < n; i++) {
    printf("%s ", reci[i]);
107 }
    printf("\n");
109
    /* Zatvara se datoteka */
111 fclose(ulaz);
113
    exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.23

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

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

22    /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
24        /* Racuna se srednji indeks */
        int srednji = (levi + desni) / 2;
26        /* Ako je srednji element veci od traženog, tada se traženi
            mora nalaziti u levoj polovini niza */
28        if (x < a[srednji].kod)
            desni = srednji - 1;
30        /* Ako je srednji element manji od traženog, tada se traženi
            mora nalaziti u desnoj polovini niza */
32        else if (x > a[srednji].kod)
```

```

    levi = srednji + 1;
34  else
    /* Ako je sredisnji element jednak trazenom, tada je artikal
36     sa bar kodom x pronadjen na poziciji srednji */
    return srednji;
38  }
    /* Ako nije pronadjen artikal za trazanim bar kodom, vraca se -1 */
40  return -1;
}

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

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

62
64  int main()
{
    Artikal asortiman[MAX_ARTIKALA];
66     long kod;
    int i, n;
68     float racun;
    FILE *fp = NULL;

70

    /* Otvara se datoteka */
72     if ((fp = fopen("artikli.txt", "r")) == NULL) {
        fprintf(stderr,
74             "Greska: Neuspesno otvaranje datoteke artikli.txt.\n");
        exit(EXIT_FAILURE);
76     }

78     /* Ucitavaju se artikali */
    i = 0;
80     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
        asortiman[i].naziv, asortiman[i].proizvodjac,
82         &asortiman[i].cena) == 4)
        i++;
84

```

### 3 Algoritmi pretrage i sortiranja

```
/* Zatvara se datoteka */
86 fclose(fp);

88 n = i;

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

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

114 kod = 0;
while (1) {
116     printf("-----\n");
117     printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
118     printf("- Za nov racun unesite kod artikla:\n\n");
119     /* Unos bar koda provog artikla sledeceg kupca */
120     if (scanf("%ld", &kod) == EOF)
121         break;
122     /* Trenutni racun novog kupca */
123     racun = 0;
124     /* Za sve artikle trenutnog kupca */
125     while (1) {
126         /* Vrsi se njihov pronalazak u nizu */
127         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
128             printf("\tGreska: Ne postoji proizvod sa trazanim kodom!\n");
129         } else {
130             printf("\tTrazili ste:\t%s %s %12.2f\n",
131                 asortiman[i].naziv, asortiman[i].proizvodjac,
132                 asortiman[i].cena);
133             /* I dodaje se cena na ukupan racun */
134             racun += asortiman[i].cena;
135         }
136     }
}
```

```

136     /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako
        on nema vise artikla */
138     printf("Unesite kod artikla [ili 0 za prekid]: \t");
        scanf("%ld", &kod);
140     if (kod == 0)
        break;
142 }
    /* Stampa se ukupan racun trenutnog kupca */
144     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
}

146     printf("Kraj rada kase!\n");
148     exit(EXIT_SUCCESS);
150 }

```

### Rešenje 3.24

```

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

5  #define MAX 500

7  /* Struktura sa svim informacijama o pojedinacnom studentu */
    typedef struct {
9      char ime[21];
        char prezime[26];
11     int prisustvo;
        int zadaci;
13 } Student;

15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
        rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
    {
19     int i, j;
        int min;
21     Student pom;

23     for (i = 0; i < n - 1; i++) {
        min = i;
25         for (j = i + 1; j < n; j++)
            if (strcmp(niz[j].ime, niz[min].ime) < 0)
27             min = j;

29         if (min != i) {
            pom = niz[min];
31             niz[min] = niz[i];
                niz[i] = pom;
33         }
    }

```

### 3 Algoritmi pretrage i sortiranja

---

```
    }
35 }

37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
    zadatka opadajuće, a ukoliko neki studenti imaju isti broj
39 uradjenih zadataka sortiraju se po dužini imena rastuće. */
void sort_zadatke_pa_imena(Student niz[], int n)
41 {
    int i, j;
43     int max;
    Student pom;
45     for (i = 0; i < n - 1; i++) {
        max = i;
47         for (j = i + 1; j < n; j++)
            if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
            else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
                max = j;
53         if (max != i) {
            pom = niz[max];
55             niz[max] = niz[i];
            niz[i] = pom;
57         }
    }
59 }

61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
    su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
63 sortiraju se opadajuće po broju uradjenih zadataka, a ukoliko se
    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65 prezimenu opadajuće. */
void sort_prisustvo_pa_zadatke_pa_prezimenama(Student niz[], int n)
67 {
    int i, j;
69     int max;
    Student pom;
71     for (i = 0; i < n - 1; i++) {
        max = i;
73         for (j = i + 1; j < n; j++)
            if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;
            else if (niz[j].prisustvo == niz[max].prisustvo
77                     && niz[j].zadaci > niz[max].zadaci)
                max = j;
            else if (niz[j].prisustvo == niz[max].prisustvo
79                     && niz[j].zadaci == niz[max].zadaci
81                     && strcmp(niz[j].prezime, niz[max].prezime) > 0)
                max = j;
83         if (max != i) {
            pom = niz[max];
85             niz[max] = niz[i];
```



```

    niz[i] = pom;
87     }
    }
89 }

91 int main(int argc, char *argv[])
{
93     Student praktikum[MAX];
    int i, br_studenata = 0;

95     FILE *fp = NULL;

97     /* Otvara se datoteka za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
        fprintf(stderr,
101             "Greska: Neupesno otvaranje datoteke aktivnost.txt.\n");
        exit(EXIT_FAILURE);
103     }

105     /* Ucitava se sadrzaj */
    for (i = 0;
107         fscanf(fp, "%s%d", praktikum[i].ime,
                praktikum[i].prezime, &praktikum[i].prisustvo,
109                 &praktikum[i].zadaci) != EOF; i++);
    /* Zatvara se datoteka */
111     fclose(fp);
    br_studenata = i;

113     /* Kreira se prvi spisak studenata po prvom kriterijumu */
    sort_ime_leksikografski(praktikum, br_studenata);
115     /* Otvara se datoteka za pisanje */
117     if ((fp = fopen("dat1.txt", "w")) == NULL) {
        fprintf(stderr,
119             "Greska: Neupesno otvaranje datoteke dat1.txt.\n");
        exit(EXIT_FAILURE);
121     }
    /* Upisuje se niz u datoteku */
123     fprintf
        (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
125     for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
127                praktikum[i].prezime, praktikum[i].prisustvo,
                praktikum[i].zadaci);
129     /* Zatvara se datoteka */
    fclose(fp);

131     /* Kreira se drugi spisak studenata po drugom kriterijumu */
133     sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvara se datoteka za pisanje */
135     if ((fp = fopen("dat2.txt", "w")) == NULL) {
        fprintf(stderr,
137             "Greska: Neupesno otvaranje datoteke dat2.txt.\n");
    }
```

### 3 Algoritmi pretrage i sortiranja

```
        exit(EXIT_FAILURE);
139 }
    /* Upisuje se niz u datoteku */
141 fprintf(fp, "Studenti sortirani po broju zadataka opadajuće,\n");
    fprintf(fp, "pa po dužini imena rastuće:\n");
143 for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
145                praktikum[i].prezime, praktikum[i].prisustvo,
                praktikum[i].zadaci);
147 /* Zatvara se datoteka */
    fclose(fp);
149
    /* Kreira se treci spisak studenata po trecem kriterijumu */
151 sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum, br_studenata);
    /* Otvara se datoteka za pisanje */
153 if ((fp = fopen("dat3.txt", "w")) == NULL) {
        fprintf(stderr,
155                "Greska: Neuspesno otvaranje datoteke dat3.txt.\n");
        exit(EXIT_FAILURE);
157 }
    /* Upisuje se niz u datoteku */
159 fprintf(fp, "Studenti sortirani po prisustvu opadajuće,\n");
    fprintf(fp, "pa po broju zadataka,\n");
161 fprintf(fp, "pa po prezimenima leksikografski opadajuće:\n");
    for (i = 0; i < br_studenata; i++)
163        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
                praktikum[i].prezime, praktikum[i].prisustvo,
165                praktikum[i].zadaci);
    /* Zatvara se datoteka */
167 fclose(fp);
169
    exit(EXIT_SUCCESS);
}
```

#### Rešenje 3.25

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define KORAK 10
6
/* Struktura koja opisuje jednu pesmu */
8 typedef struct {
    char *izvodjac;
10    char *naslov;
    int broj_gledanja;
12 } Pesma;

14 /* Funkcija za upoređjivanje pesama po broju gledanosti (potrebna za
    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 }
23
24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
25    qsort funkcije) */
26 int uporedi_naslove(const void *pp1, const void *pp2)
27 {
28     Pesma *p1 = (Pesma *) pp1;
29     Pesma *p2 = (Pesma *) pp2;
30
31     return strcmp(p1->naslov, p2->naslov);
32 }
33
34 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
35    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
37 {
38     Pesma *p1 = (Pesma *) pp1;
39     Pesma *p2 = (Pesma *) pp2;
40
41     return strcmp(p1->izvodjac, p2->izvodjac);
42 }
43
44 /* Funkcija koja oslobadja memoriju zauzetu dinamickim nizom pesme
45    dimenzije n */
46 void oslobodi(Pesma * pesme, int n)
47 {
48     int i;
49     for (i = 0; i < n; i++) {
50         free(pesme[i].izvodjac);
51         free(pesme[i].naslov);
52     }
53     free(pesme);
54 }
55
56 int main(int argc, char *argv[])
57 {
58     FILE *ulaz;
59     Pesma *pesme;                                /* Pokazivac na deo memorije za
60                                                    cuvanje pesama */
61
62     int alocirano_za_pesme;                       /* Broj mesta alociranih za pesme */
63     int i;                                         /* Redni broj pesme cije se
64                                                    informacije citaju */
65
66     int n;                                         /* Ukupan broj pesama */
67     int j;
68     char c;
69     int alocirano;                                /* Broj mesta alociranih za
```

### 3 Algoritmi pretrage i sortiranja

```
68                                     propratne informacije o pesmama */
69
70     int broj_gledanja;
71
72     /* Priprema se datoteka za citanje */
73     ulaz = fopen("pesme.txt", "r");
74     if (ulaz == NULL) {
75         fprintf(stderr,
76             "Greska: Neuspesno otvaranje ulazne datoteke!\n");
77         exit(EXIT_FAILURE);
78     }
79
80     /* Citaju se informacije o pesmama */
81     pesme = NULL;
82     alocirano_za_pesme = 0;
83     i = 0;
84
85     while (1) {
86         /* Proverava se da li je dostignut kraj datoteke */
87         c = fgetc(ulaz);
88         if (c == EOF) {
89             /* Nema vise sadrzaja za citanje */
90             break;
91         } else {
92             /* Inace, vraca se procitani karakter nazad */
93             ungetc(c, ulaz);
94         }
95
96         /* Proverava se da li postoji dovoljno vec alocirane memorije za
97            citanje nove pesme */
98         if (alocirano_za_pesme == i) {
99             /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
100                novih KORAK mesta */
101             alocirano_za_pesme += KORAK;
102             pesme =
103                 (Pesma *) realloc(pesme,
104                                     alocirano_za_pesme * sizeof(Pesma));
105
106             /* Proverava se da li je nova memorija uspesno realocirana */
107             if (pesme == NULL) {
108                 /* Ako nije ispisuje se obavestenje */
109                 fprintf(stderr, "Greska: Problem sa alokacijom memorije!\n");
110                 /* I oslobadja sva memorija zauzeta do ovog koraka */
111                 oslobodi(pesme, i);
112                 exit(EXIT_FAILURE);
113             }
114         }
115
116         /* Ako jeste, nastavlja se sa citanjem pesama ... */
117         /* Cita se ime izvodjaca */
118         j = 0;                                     /* Pozicija na koju treba smestiti
119                                                    procitani karakter */
120         alocirano = 0;                             /* Broj alociranih mesta */
121     }
```

```

120     pesme[i].izvodjac = NULL;    /* Memorija za smestanje procitanih
                                   karakterata */
122
123     /* Sve do prve beline u liniji (beline koja se nalazi nakon
124        imena izvodjaca) citaju se karakteri iz datoteke */
125     while ((c = fgetc(ulaz)) != ' ') {
126         /* Provera da li postoji dovoljno memorije za smestanje
127            procitanog karakterata */
128         if (j == alocirano) {
129
130             /* Ako ne, ako je potrosena sva alocirana memorija, alocira
131                se novih KORAK mesta */
132             alocirano += KORAK;
133             pesme[i].izvodjac =
134                 (char *) realloc(pesme[i].izvodjac,
135                                 alocirano * sizeof(char));
136
137             /* Provera da li je nova alokacija uspesna */
138             if (pesme[i].izvodjac == NULL) {
139                 /* Ako nije oslobadja se sva memorija zauzeta do ovog
140                    koraka */
141                 oslobodi(pesme, i);
142                 /* I prekida sa izvršavanjem programa */
143                 exit(EXIT_FAILURE);
144             }
145         }
146         /* Ako postoji dovoljno alocirane memorije, smesta se vec
147            procitani karakter */
148         pesme[i].izvodjac[j] = c;
149         j++;
150         /* I nastavlja se sa citanjem */
151     }
152
153     /* Upis terminirajuće nule na kraj reci */
154     pesme[i].izvodjac[j] = '\0';
155
156     /* Preskace se karakter - */
157     fgetc(ulaz);
158
159     /* Preskace se razmak */
160     fgetc(ulaz);
161
162     /* Cita se naslov pesme */
163     j = 0;                                /* Pozicija na koju treba smestiti
164                                           procitani karakter */
165     alocirano = 0;                        /* Broj alociranih mesta */
166     pesme[i].naslov = NULL;              /* Memorija za smestanje procitanih
167                                           karakterata */
168
169     /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
170        karakteri iz datoteke */
171     while ((c = fgetc(ulaz)) != ',') {

```

```
172     /* Provera da li postoji dovoljno memorije za smestanje
        procitanog karaktera */
174     if (j == alocirano) {
        /* Ako ne, ako je potrosena sva alocirana memorija, alocira
        se novih KORAK mesta */
176         alocirano += KORAK;
178         pesme[i].naslov =
            (char *) realloc(pesme[i].naslov,
180                             alocirano * sizeof(char));

182         /* Provera da li je nova alokacija uspesna */
        if (pesme[i].naslov == NULL) {
184             /* Ako nije, oslobadja se sva memorija zauzeta do ovog
                koraka */
186             free(pesme[i].izvodjac);
                oslobodi(pesme, i);
188             /* I prekida izvršavanje programa */
                exit(EXIT_FAILURE);
190         }
        }
192     /* Smesta se procitani karakter */
        pesme[i].naslov[j] = c;
194     j++;
        /* I nastavlja dalje sa citanjem */
196     }
    /* Upisuje se terminirajuca nula na kraj reci */
198     pesme[i].naslov[j] = '\0';

200     /* Preskace se razmak */
    fgetc(ulaz);

202
204     /* Cita se broj gledanja */
    broj_gledanja = 0;

206     /* Sve do znaka za novi red (kraja linije) citaju se karakteri
        iz datoteke */
208     while ((c = fgetc(ulaz)) != '\n') {
        broj_gledanja = broj_gledanja * 10 + (c - '0');
210     }
    pesme[i].broj_gledanja = broj_gledanja;

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

216
218 /* Informacija o broju procitanih pesama */
    n = i;
    /* Zatvara se datoteka */
220     fclose(ulaz);

222     /* Analiza argumenta komandne linije */
    if (argc == 1) {
```

```

224     /* Nema dodatnih opcija => sortiranje po broju gledanja */
    qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
226 } else {
    if (argc == 2 && strcmp(argv[1], "-n") == 0) {
228         /* Sortira se po naslovu */
        qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
230    } else {
        if (argc == 2 && strcmp(argv[1], "-i") == 0) {
232            /* Sortira se po izvodjacu */
            qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
234        } else {
            fprintf(stderr, "Greska: Nedozvoljeni argumenti!\n");
236            free(pesme);
            exit(EXIT_FAILURE);
238        }
    }
240 }

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

248 /* Oslobadja se memorija */
oslobodi(pesme, n);

250 exit(EXIT_SUCCESS);
252 }

```

### Rešenje 3.28

NAPOMENA: Rešenje koristi biblioteku za rad sa celobrojnim matricama iz zadatka 2.19.

```

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

4
/* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
6  kolona */
int zbir_vrste(int **a, int v, int m)
8  {
    int i, zbir = 0;
10
    for (i = 0; i < m; i++) {
12        zbir += a[v][i];
    }
14    return zbir;
16 }

```

```
18  /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
    osnovu zbira koriscenjem selection sort algoritma */
void sortiraj_vrste(int **a, int n, int m)
20 {
    int i, j, min;
22
    for (i = 0; i < n - 1; i++) {
24         min = i;
        for (j = i + 1; j < n; j++) {
26             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
                min = j;
28             }
        }
30         if (min != i) {
            int *tmp;
32             tmp = a[i];
            a[i] = a[min];
34             a[min] = tmp;
        }
36     }
}

38
int main(int argc, char *argv[])
40 {
    int **a;
42     int n, m;

44     /* Unos dimenzija matrice */
    printf("Unesite dimenzije matrice: ");
46     scanf("%d %d", &n, &m);

48     /* Alokacija memorije */
    a = alociraj_matricu(n, m);
50     if (a == NULL) {
        fprintf(stderr, "Greska: Neuspesna alokacija matrice\n");
52         exit(EXIT_FAILURE);
    }
54

    /* Ucitavaju se elementi matrice */
56     printf("Unesite elemente matrice po vrstama:\n");
    ucitaj_matricu(a, n, m);
58

    /* Poziva se funkcija koja sortira vrste matrice prema zbiru */
60     sortiraj_vrste(a, n, m);

62     /* Ispisuje se rezultujuca matrica */
    printf("Sortirana matrica je:\n");
64     ispisi_matricu(a, n, m);

66     /* Oslobadja se memorija */
    a = deallociraj_matricu(a, n);
68
```



```
    exit(EXIT_SUCCESS);
70 }
```

### Rešenje 3.31

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
7  #define MAX 100
9
10 /* Funkcija poredjenja dva cela broja (neopadajuci poredak) */
11 int poredi_int(const void *a, const void *b)
12 {
13     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
14        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
15     int b1 = *((int *) a);
16     int b2 = *((int *) b);
17
18     /* Zbog moguceg prekoracenja opsega celih brojeva, oduzimanje
19        b1-b2 treba izbegavati */
20     if (b1 > b2)
21         return 1;
22     else if (b1 < b2)
23         return -1;
24     else
25         return 0;
26 }
27
28 /* Funkcija poredjenja dva cela broja (nerastuci poredak) */
29 int poredi_int_nerastuce(const void *a, const void *b)
30 {
31     /* Za obrnuti poredak treba samo promeniti znak vrednosti koju
32        koju vraca prethodna funkcija */
33     return -poredi_int(a, b);
34 }
35
36 int main()
37 {
38     size_t n;
39     int i, x;
40     int a[MAX], *p = NULL;
41
42     /* Unos dimenzije */
43     printf("Uneti dimenziju niza: ");
44     scanf("%ld", &n);
45     if (n > MAX)
46         n = MAX;
47
48     /* Unos elementa niza */
```

### 3 Algoritmi pretrage i sortiranja

```
47  printf("Uneti elemente niza:\n");
48  for (i = 0; i < n; i++)
49      scanf("%d", &a[i]);

51  /* Sortira se niz celih brojeva */
52  qsort(a, n, sizeof(int), &poredi_int);

53
54  /* Prikazuje se sortirani 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  /* Pretrazuje se niz */
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);

81  return 0;
}
```

#### Rešenje 3.32

```
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_delilaca(int x)
10 {
11     int i;
12     int br;
```

```
13  /* 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_delilaca(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = broj_delilaca(ak);
42     int n_d_b = broj_delilaca(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];
52
53     /* Unos dimenzije */
54     printf("Uneti dimenziju niza: ");
55     scanf("%ld", &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     /* Sortira se niz celih brojeva prema broju delilaca */
```

### 3 Algoritmi pretrage i sortiranja

```
65  qsort(a, n, sizeof(int), &poredi_po_broju_delilaca);

67  /* Prikazuje se sortirani niz */
printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
69  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
71  putchar('\n');

73  return 0;
}
```

#### Rešenje 3.33

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

5
   #define MAX_NISKI 1000
7  #define MAX_DUZINA 31

9  /*****
   Niz nizova karaktera ovog potpisa
11  char niske[3][4];
   se moze graficki predstaviti ovako:
13  -----
   | a | b | c | \0 | | d | e | \0 |   | f | g | h | \0 | |
15  -----

   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
17  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
   nalazi na adresi koja je za 4 veka od prve reci, a za 4 manja od
19  adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
   i ona je tipa char*.

21
   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23  koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
   nad njima.

27  *****/
int poredi_leksikografski(const void *a, const void *b)
29  {
   return strcmp((char *) a, (char *) b);
31  }

33  /* Funkcija slicna prethodnoj, osim sto elemente ne uporeduje
   leksikografski, vec po duzini */
int poredi_duzine(const void *a, const void *b)
35  {
   return strlen((char *) a) - strlen((char *) b);
37  }
}
```

```

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     /* Otvara se datoteka */
49     if ((fp = fopen("niske.txt", "r")) == NULL) {
50         fprintf(stderr,
51             "Greska: Neupesno otvaranje datoteke niske.txt.\n");
52         exit(EXIT_FAILURE);
53     }
54
55     /* Cita se sadrzaj datoteke */
56     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
57
58     /* Zatvara se datoteka */
59     fclose(fp);
60     n = i;
61
62     /* Sortiraju se niske leksikografski. Biblioteckoj funkciji qsort
63        prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
64        niske po duzini */
65     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);
66
67     printf("Leksikografski sortirane niske:\n");
68     for (i = 0; i < n; i++)
69         printf("%s ", niske[i]);
70     printf("\n");
71
72     /* Unosi se trazena niska */
73     printf("Uneti trazenu nisku: ");
74     scanf("%s", x);
75
76     /* Binarna pretraga */
77     /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
78        je niz vec sortiran leksikografski. */
79     p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
80                 &poredi_leksikografski);
81
82     if (p != NULL)
83         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84             p, (p - (char *) niske) / MAX_DUZINA);
85     else
86         printf("Niska nije pronadjena u nizu\n");
87
88     /* Sortira se po duzini */
89     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

```

### 3 Algoritmi pretrage i sortiranja

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

97     /* Linearna pretraga */
98     p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
99             &poredi_leksikografski);
100     if (p != NULL)
101         printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
102             p, (p - (char *) niske) / MAX_DUZINA);
103     else
104         printf("Niska nije pronadjena u nizu\n");
105     exit(EXIT_SUCCESS);
107 }
```

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

```

33 int poredi_leksikografski(const void *a, const void *b)
34 {
35     return strcmp(*(char **) a, *(char **) b);
36 }
37
38 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
39    leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
40 {
41     return strlen(*(char **) a) - strlen(*(char **) b);
42 }
43
44 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
45    element u nizu sa kojim se poredi, pa njega treba kastovati na
46    char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
47    ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
48    main funkciji je to x, koji je tipa char*, tako da pokazivac a
49    ovde samo treba kastovati i ne dereferencirati. */
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     /* Otvara se datoteka */
64     if ((fp = fopen("niske.txt", "r")) == NULL) {
65         fprintf(stderr,
66             "Greska: Neuspesno otvaranje datoteke niske.txt.\n");
67         exit(EXIT_FAILURE);
68     }
69
70     /* Cita se sadrzaj datoteke */
71     i = 0;
72     while (fscanf(fp, "%s", x) != EOF) {
73         /* Alocira se dovoljno memorije za i-tu nisku */
74         if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
75         {
76             fprintf(stderr, "Greska: Neuspesna alokacija niske\n");
77             exit(EXIT_FAILURE);
78         }
79         /* Kopira se procitana niska na svoje mesto */
80         strcpy(niske[i], x);
81         i++;
82     }

```

```
83      /* Zatvara se datoteka */
85      fclose(fp);
      n = i;

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

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

97      /* Unosi se trazena niska */
99      printf("Uneti trazenu nisku: ");
      scanf("%s", x);

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

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

117      /* Sortira se po duzini */
119      qsort(niske, n, sizeof(char *), &poredi_duzine);
      printf("Niske sortirane po duzini:\n");
121      for (i = 0; i < n; i++)
          printf("%s ", niske[i]);
123      printf("\n");

125      /* Oslobadja se zauzeta memorija */
127      for (i = 0; i < n; i++)
          free(niske[i]);

129      exit(EXIT_SUCCESS);
}
```



## 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 poredi1(const void *a, const void *b)
19 {
20     Student *prvi = (Student *) a;
21     Student *drugi = (Student *) b;
22
23     if (prvi->bodovi > drugi->bodovi)
24         return -1;
25     else if (prvi->bodovi < drugi->bodovi)
26         return 1;
27     else
28         /* Ako su jednaki po broju bodova, treba ih uporediti po
29            prezimenu */
30         return strcmp(prvi->prezime, drugi->prezime);
31 }
32
33 /* Funkcija za poredjenje koja se koristi u pretrazi po broju
34    bodova. Prvi parametar je ono sto se trazi u nizu (broj bodova),
35    a drugi parametar je element niza ciji se bodovi porede. */
36 int poredi2(const void *a, const void *b)
37 {
38     int bodovi = *(int *) a;
39     Student *s = (Student *) b;
40     return s->bodovi - bodovi;
41 }
42
43 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
45    parametar je element niza cije se prezime poredi. */
46 int poredi3(const void *a, const void *b)
47 {
48     char *prezime = (char *) a;
49     Student *s = (Student *) b;
50     return strcmp(prezime, s->prezime);
```

```

}
52
int main(int argc, char *argv[])
54
{
    Student kolokvijum[MAX];
56
    int i;
    size_t br_studenata = 0;
58
    Student *nadjen = NULL;
    FILE *fp = NULL;
60
    int bodovi;
    char prezime[21];
62

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

72
    /* Otvara se datoteka */
    if ((fp = fopen(argv[1], "r")) == NULL) {
74
        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s\n",
            argv[1]);
76
        exit(EXIT_FAILURE);
    }

78
    /* Ucitava se sadrzaj */
    for (i = 0;
80
        fscanf(fp, "%s%s%d", kolokvijum[i].ime,
            kolokvijum[i].prezime,
82
            &kolokvijum[i].bodovi) != EOF; i++);

84

    /* Zatvara se datoteka */
86
    fclose(fp);
    br_studenata = i;
88

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

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

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

```

```

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

108     if (nadjen != NULL)
        printf
110         ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
            nadjen->ime, nadjen->prezime, nadjen->bodovi);
112     else
        printf("Nema studenta sa unetim brojem bodova\n");

114     /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116        sortiran po bodovima. */
    printf("Unesite prezime: ");
118    scanf("%s", prezime);

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

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

130    exit(EXIT_SUCCESS);
132 }

```

### Rešenje 3.36

```

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

4
#define MAX 128

6
/* Funkcija poredi dva karaktera */
8  int uporedi_char(const void *pa, const void *pb)
{
10     return *(char *) pa - *(char *) pb;
}

12
/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14  int anagrami(char s[], char t[])
{
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
    if (strlen(s) != strlen(t))
18         return 0;
}

```

```
20  /* Sortiraju se karakteri u niskama */
    qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22  qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

24  /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
    suprotnom, nisu */
26  return !strcmp(s, t);
}

28
int main()
30 {
    char s[MAX], t[MAX];

32
    /* Unose se niske */
34  printf("Unesite prvu nisku: ");
    scanf("%s", s);
36  printf("Unesite drugu nisku: ");
    scanf("%s", t);

38
    /* Ispituje se da li su niske anagrami */
40  if (anagrami(s, t))
        printf("jesu\n");
42  else
        printf("nisu\n");
44
    return 0;
46 }
```

#### Rešenje 3.37

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

5  #define MAX 10
    #define MAX_DUZINA 32

7
    /* Funkcija poredjenja */
9  int uporedi_niske(const void *pa, const void *pb)
    {
11     return strcmp((char *) pa, (char *) pb);
    }

13
int main()
15 {
    int i, n;
17     char S[MAX][MAX_DUZINA];

19     /* Unosi se broj niski */
    printf("Unesite broj niski:");
```

```

21  scanf("%d", &n);

23  /* Unosi se niz niski */
printf("Unesite niske:\n");
25  for (i = 0; i < n; i++)
    scanf("%s", S[i]);

27

/* Sortira se niz niski */
29  qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

31  /*****
    Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
33  sortiranih niski. Koriscen je samo u fazi testiranja programa.

    printf("Sortirane niske su:\n");
    for(i = 0; i < n; i++)
37      printf("%s ", S[i]);
    *****/

39

/* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
41  niza biti jedna do druge */
for (i = 0; i < n - 1; i++)
43     if (strcmp(S[i], S[i + 1]) == 0) {
        printf("ima\n");
45         return 0;
    }
47  printf("nema\n");

49  return 0;
}

```

### Rešenje 3.38

```

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

5  #define MAX 21

7  /* Struktura koja predstavlja jednog studenta */
typedef struct student {
9      char nalog[8];
    char ime[MAX];
11     char prezime[MAX];
    int poeni;
13 } Student;

15 /* Funkcija poredi studente prema broju poena, rastuce */
int uporedi_poeni(const void *a, const void *b)
17 {
    Student s = *(Student *) a;

```

### 3 Algoritmi pretrage i sortiranja

---

```
19     Student t = *(Student *) b;
    return s.poeni - t.poeni;
21 }

23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i
    na kraju prema indeksu */
25 int uporedi_nalog(const void *a, const void *b)
{
27     Student s = *(Student *) a;
    Student t = *(Student *) b;
29     /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
        broj indeksa */
31     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
    int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
33     char smer1 = s.nalog[1];
    char smer2 = t.nalog[1];
35     int indeks1 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
37         s.nalog[6] - '0';
    int indeks2 =
39         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
        t.nalog[6] - '0';
41     if (godina1 != godina2)
        return godina1 - godina2;
43     else if (smer1 != smer2)
        return smer1 - smer2;
45     else
        return indeks1 - indeks2;
47 }

49 /* Funkcija poredjenja po nalogu za upotrebu u biblioteckoj funkciji
    bsearch */
51 int uporedi_bsearch(const void *a, const void *b)
{
53     /* Nalog studenta koji se trazi */
    char *nalog = (char *) a;
55     /* Kljuc pretrage */
    Student s = *(Student *) b;
57
    int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
59     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
    char smer1 = nalog[1];
61     char smer2 = s.nalog[1];
    int indeks1 =
63         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + (nalog[6] -
        '0');
65     int indeks2 =
        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
67         (s.nalog[6] - '0');
    if (godina1 != godina2)
69         return godina1 - godina2;
    else if (smer1 != smer2)
```

```
71     return smer1 - smer2;
72 else
73     return indeks1 - indeks2;
74 }
75
76 int main(int argc, char **argv)
77 {
78     Student *nadjen = NULL;
79     char nalog_trazeni[8];
80     Student niz_studenata[100];
81     int i = 0, br_studenata = 0;
82     FILE *in = NULL, *out = NULL;
83
84     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
85        korisnik nije ispravno pokrenuo program. */
86     if (argc != 2 && argc != 3) {
87         fprintf(stderr,
88             "Greska: Program se poziva sa %s -opcija [nalog]\n",
89             argv[0]);
90         exit(EXIT_FAILURE);
91     }
92
93     /* Otvara se datoteka za citanje */
94     in = fopen("studenti.txt", "r");
95     if (in == NULL) {
96         fprintf(stderr,
97             "Greska: Neuspesno otvaranje datoteke studenti.txt!\n");
98         exit(EXIT_FAILURE);
99     }
100
101     /* Otvara se datoteka za pisanje */
102     out = fopen("izlaz.txt", "w");
103     if (out == NULL) {
104         fprintf(stderr,
105             "Greska: Neuspesno otvaranje datoteke izlaz.txt!\n");
106         exit(EXIT_FAILURE);
107     }
108
109     /* Ucitavaju se studenti iz ulazne datoteke sve do njenog kraja */
110     while (fscanf
111         (in, "%s %s %s %d", niz_studenata[i].nalog,
112          niz_studenata[i].ime, niz_studenata[i].prezime,
113          &niz_studenata[i].poeni) != EOF)
114         i++;
115
116     br_studenata = i;
117
118     /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
119     if (strcmp(argv[1], "-p") == 0)
120         qsort(niz_studenata, br_studenata, sizeof(Student),
121             &uporedi_poeni);
122
123     /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
```

### 3 Algoritmi pretrage i sortiranja

---

```
123     else if (strcmp(argv[1], "-n") == 0)
124         qsort(niz_studenata, br_studenata, sizeof(Student),
125             &uporedi_nalog);
126
127     /* Sortirani studenti se ispisuju u izlaznu datoteku */
128     for (i = 0; i < br_studenata; i++)
129         fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
130             niz_studenata[i].ime, niz_studenata[i].prezime,
131             niz_studenata[i].poeni);
132
133     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
134        studenta... */
135     if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
136         strcpy(nalog_trazeni, argv[2]);
137
138         /* ... pronalazi se student sa tim nalogom. */
139         nadjen =
140             (Student *) bsearch(nalog_trazeni, niz_studenata,
141                 br_studenata, sizeof(Student),
142                 &uporedi_bsearch);
143
144         if (nadjen == NULL)
145             printf("Nije nadjen!\n");
146         else
147             printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
148                 nadjen->prezime, nadjen->poeni);
149     }
150
151     /* Zatvaraju se datoteke */
152     fclose(in);
153     fclose(out);
154
155     exit(EXIT_SUCCESS);
156 }
```



## 4

# Dinamičke strukture podataka

## 4.1 Liste

**Zadatak 4.1** Napisati biblioteku za rad sa jednostruko povezanom 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`.

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

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

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

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

*Primer 1*

```

INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
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:
Unesite brojeve (CTRL+D za kraj unosa):
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:
Unesite brojeve (CTRL+D za kraj unosa):
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:
Unesite brojeve (CTRL+D za kraj unosa):
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 poretaku. 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 (CTRL+D za kraj unosa):
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se trazi: 14
Trazeni broj 14 je u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 5, 14]
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CTRL+D za kraj unosa):
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [14, 23, 35]

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

Unesite broj koji se brise: 3
Lista nakon brisanja: [14, 23, 35]
```

**Zadatak 4.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekursivno. NAPOMENA: *Koristiti **main** programe i test primere iz zadatka 4.1.*

**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 smestati 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 DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje
datoteke datoteka.html.
```

*Test 2*

```

POKRETANJE: ./a.out datoteka.html

IZLAZ:
a - 4
br - 1
h1 - 2
body - 2
title - 2
head - 2
html - 2

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>

```

**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: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

*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

```

\* **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.

## 4 Dinamičke strukture podataka

---

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>BROJEVI.TXT 43 12 15 16 4 2 8  IZLAZ: REZULTAT.TXT 12 15 16</pre>	<pre>DATOTEKA BROJEVI.TXT NE POSTOJI.  IZLAZ ZA GREŠKE: Greska: Neuspesno otvaranje datoteke brojevi.txt.</pre>	<pre>DATOTEKA BROJEVI.TXT JE PRAZNA  IZLAZ: REZULTAT.TXT Rezultat.txt ce biti prazna.</pre>

\* **Zadatak 4.6** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

<i>Test 1</i>	<i>Test 2</i>
<pre>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]</pre>	<pre>POKRETANJE: ./a.out dat1.txt dat2.txt  DAT1.TXT 2 4 6 10 15  DATOTEKA DAT2.TXT NE POSTOJI.  IZLAZ ZA GREŠKE: Greska: Neuspesno otvaranje datoteke dat2.txt.</pre>

<i>Test 3</i>	<i>Test 4</i>
<pre>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]</pre>	<pre>POKRETANJE: ./a.out dat1.txt  IZLAZ ZA GREŠKE: Greska: Program se poziva sa: ./a.out dat1.txt dat2.txt!</pre>

\* **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: *Koristiti 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  $\{$ ,  $[$  i  $($ . Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

#### Test 1

```
IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23
IZLAZ:
Zagrade su ispravno uparene.
```

#### Test 2

```
IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}
IZLAZ:
Zagrade su ispravno uparene.
```

#### Test 3

```
IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)
IZLAZ:
Zagrade nisu ispravno uparene.
```

#### Test 4

```
IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)
IZLAZ:
Zagrade nisu ispravno uparene.
```

#### Test 5

```
DATOTEKA IZRAZ.TXT JE PRAZNA
IZLAZ:
Zagrade su ispravno uparene.
```

#### Test 6

```
DATOTEKA IZRAZ.TXT NE POSTOJI.
IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje
datoteke izraz.txt!
```

**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 link='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 GREŠKE:
Greska: Neuspesno otvaranje
datoteke datoteka.html.
```

### Test 6

```
POKRETANJE: ./a.out datoteka.html
```

```
DATOTEKA.HTML JE PRAZNA
```

```
IZLAZ:
Etikete su pravilno uparene!
```

**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 vratate 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 implementiran korišćenjem 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

```

**Zadatak 4.11** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** 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`.
- 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:** *Koristiti test primere iz zadatka 4.1*

**\* 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
<pre> ULAZ: 5 3  IZLAZ: 3 1 5 2 4 </pre>	<pre> ULAZ: 8 4  IZLAZ: 4 8 5 2 1 3 7 6 </pre>	<pre> ULAZ: 3 8  IZLAZ ZA GREŠKE: Greska: n mora biti uvek vece od k, a 3 &lt; 8! </pre>

**\* Zadatak 4.13** Grupa od  $n$  plesača na kostimima ima brojeve od 1 do  $n$ . 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. Odbrojavanje se počinje 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 smeru. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi  $n$ ,  $k$  ( $k < n$ ) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 5 3  IZLAZ: 3 5 4 2 1 </pre>	<pre> ULAZ: 8 4  IZLAZ: 4 8 5 7 6 3 2 1 </pre>	<pre> ULAZ: 5 8  IZLAZ ZA GREŠKE: Greska: n mora biti uvek vece od k, a 5 &lt; 8! </pre>

## 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 alokira 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 stablo 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`. Funk-

cija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili *NULL* ukoliko takav čvor ne postoji.

- (e) Napisati funkciju `Cvor *pronadji_najmanji(Cvor * koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- (f) Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor ** 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 oslobodi_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
Rezultujuce 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
Rezultujuce stablo: -3 -2 6 8 13 24
```

**Zadatak 4.15** Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati na standardni izlaz za grešku poruku *Nedostaje ime ulazne datoteke!*. Može se pretpostaviti 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
sunce: 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

IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

*Test 4*

```
POKRETANJE: ./a.out

IZLAZ ZA GREŠKE:
Greska: Nedostaje ime ulazne datoteke!
```

**Zadatak 4.16** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona međusobno razdvojeni blanko znakom, na primer *Pera Perić* 064/123–4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči *KRAJ*, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

### Primer 1

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858

INTERAKCIJA 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
IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke
imenik1.txt.
```

**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 dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

### Test 1

```
PRIJEMNI.TXT
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6

IZLAZ:
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9
```

### Test 2

```
DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke
prijemni.txt.
```

\* **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 *DD.MM.* formata. Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

*Primer 1*

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

RODJDENDANI.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 RODJDENDANI.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
Greska: Neuspesno otvaranje datoteke
rodjendani.txt.
```

**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.
```

\* **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
```

**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!
```

**Zadatak 4.22** Dato je binarno pretraživačko stablo celih brojeva.

- (a) Napisati funkciju koja izračunava broj čvorova stabla.
- (b) Napisati funkciju koja izračunava broj listova stabla.
- (c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.



- (d) Napisati funkciju koja izračunava zbir čvorova stabla.
- (e) Napisati funkciju koja izračunava najveći element stabla.
- (f) Napisati funkciju koja izračunava dubinu stabla.
- (g) Napisati funkciju koja izračunava broj čvorova na  $i$ -tom nivou stabla.
- (h) Napisati funkciju koja ispisuje sve elemente na  $i$ -tom nivou stabla.
- (i) Napisati funkciju koja izračunava maksimalnu vrednost na  $i$ -tom nivou stabla.
- (j) Napisati funkciju koja izračunava zbir čvorova na  $i$ -tom nivou stabla.
- (k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti  $x$ .

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara  $i$  i  $x$  pročitati kao argumente komandne linije. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1	Test 2
POKRETANJE: ./a.out 2 15	POKRETANJE: ./a.out 3 31
ULAZ: 10 5 15 3 2 4 30 12 14 13	ULAZ: 24 53 61 9 7 55 20 16
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	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

**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.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ:   10 5 15 3 2 4 30 12 14 13  IZLAZ: 0.nivo: 10 1.nivo: 5 15 2.nivo: 3 12 30 3.nivo: 2 4 14 4.nivo: 13           </pre>	<pre> ULAZ:   6 11 8 3 -2  IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8           </pre>	<pre> ULAZ:   24 53 61 9 7 55 20 16  IZLAZ: 0.nivo: 24 1.nivo: 9 53 2.nivo: 7 20 61 3.nivo: 16 55           </pre>

**\* 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.

<i>Primer 1</i>	<i>Primer 2</i>
<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 14 30 1 0 Stabla su slicna kao u ogledalu.           </pre>	<pre> INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 20 15 0 Stabla nisu slicna 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.*

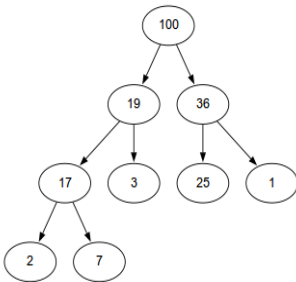
<i>Test 1</i>	<i>Test 2</i>
<pre> ULAZ:   10 5 15 2 11 16 1 13  IZLAZ:   7           </pre>	<pre> ULAZ:   16 30 40 24 10 18 45 22  IZLAZ:   6           </pre>

**Zadatak 4.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int hip(Cvor * koren)`

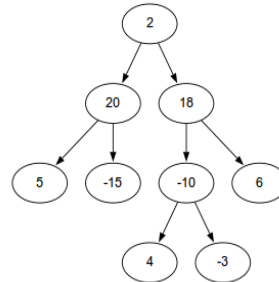
koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `hip` i ispisuje rezultat na standardni izlaz. NAPOMENA: Za alokaciju i oslobađanje memorije koristiti funkcije `napravi_cvor` i `oslobodi_stablo` iz zadatka 4.14.

Test 1

```
|| IZLAZ:
|| Zadato stablo je hip!
```



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

**Zadatak 4.27** Dato je binarno stablo celih brojeva.

- Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.
- Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

Test 1

```
|| IZLAZ:
|| Vrednost u cvoru sa maksimalnim desnim zbirom: 18
|| Vrednost u cvoru sa minimalnim levim zbirom: 18
|| 2 18 -10 4
|| 2 20 -15
```

## 4.3 Rešenja

### Rešenje 4.1

*lista.h*

```
1  #ifndef _LISTA_H_
2  #define _LISTA_H_

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste */
6  typedef struct cvor {
8      int vrednost;
9      struct cvor *sledeci;
10 } Cvor;

12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14 Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicu memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

20 /* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija 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
```

```

46  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slučaju da takav cvor ne postoji u listi. */
48  Cvor *pretrazi_listu(Cvor * glava, int broj);

50  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na činjenicu da je lista koja se pretražuje
52  neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
54  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

56  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj.
    Azurira pokazivac na glavu liste, koji može biti promenjen u
58  slučaju da se obriše stara glava. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj);

60
62  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
64  promenjen ukoliko se obriše stara glava liste. */
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

66
68  /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka
    kraju liste, razdvojene zarezima i uokvirene zagradama. */
    void ispisi_listu(Cvor * glava);
70
    #endif

```

lista.c

```

1  #include <stdio.h>
    #include <stdlib.h>
3  #include "lista.h"

5  Cvor *napravi_cvor(int broj)
    {
7      /* Alokacija memorije za novi cvor uz proveru uspesnosti */
        Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9        if (novi == NULL)
            return NULL;

11
13        /* Inicijalizacija polja strukture */
        novi->vrednost = broj;
        novi->sledeci = NULL;

15
17        /* Vraca se adresa novog cvora */
        return novi;
19    }

    void oslobodi_listu(Cvor ** adresa_glave)

```

```
21 {
22     Cvor *pomocni = NULL;
23
24     /* Ako lista nije prazna, onda treba osloboditi memoriju */
25     while (*adresa_glave != NULL) {
26         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
27            osloboditi cvor koji predstavlja glavu liste */
28         pomocni = (*adresa_glave)->sledeci;
29         free(*adresa_glave);
30
31         /* Sledeci cvor je nova glava liste */
32         *adresa_glave = pomocni;
33     }
34 }
35
36 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
37 {
38     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39     Cvor *novi = napravi_cvor(broj);
40     if (novi == NULL)
41         return 1;
42
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 Cvor *pronadji_poslednji(Cvor * glava)
52 {
53     /* U praznoj listi nema cvorova pa se vraca NULL */
54     if (glava == NULL)
55         return NULL;
56
57     /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
58        glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
59        ce pokazivati na cvor liste koji nema sledbenika, tj. na
60        poslednji cvor liste pa se vraca vrednost pokazivaca glava.
61        Pokazivac glava je argument funkcije i njegove promene nece se
62        odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
63     while (glava->sledeci != NULL)
64         glava = glava->sledeci;
65
66     return glava;
67 }
68
69 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
70 {
71     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
72     Cvor *novi = napravi_cvor(broj);
```

```

73     if (novi == NULL)
74         return 1;
75
76     /* Ako je lista prazna */
77     if (*adresa_glave == NULL) {
78         /* Glava nove liste je upravo novi cvor */
79         *adresa_glave = novi;
80     } else {
81         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi
82            cvor se dodaje na kraj liste kao sledbenik poslednjeg */
83         Cvor *poslednji = pronadji_poslednji(*adresa_glave);
84         poslednji->sledeci = novi;
85     }
86
87     /* Vraca se indikator uspesnog dodavanja */
88     return 0;
89 }
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        pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
99        vecu ili jednaku vrednosti novog cvora.*/
100    /* Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
101       proverava da li se doslo do poslednjeg cvora liste pre nego sto se
102       proveru vrednost u sledecem cvoru, jer u slucaju poslednjeg,
103       sledeci ne postoji, pa ni njegova vrednost. */
104    while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
105        glava = glava->sledeci;
106
107    /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
108       poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci
109       ima vrednost vecu od broj. */
110    return glava;
111 }
112
113 int dodaj_iza(Cvor * tekuci, int broj)
114 {
115     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
116     Cvor *novi = napravi_cvor(broj);
117     if (novi == NULL)
118         return 1;
119
120     /* Novi cvor se dodaje iza tekuceg cvora. */
121     novi->sledeci = tekuci->sledeci;
122     tekuci->sledeci = novi;
123
124     /* Vraca se indikator uspesnog dodavanja */

```

```
125     return 0;
126 }
127
128 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
129 {
130     /* Ako je lista prazna */
131     if (*adresa_glave == NULL) {
132         /* Glava nove liste je novi cvor */
133         /* Kreiranje novog cvora uz proveru uspesnost kreiranja */
134         Cvor *novi = napravi_cvor(broj);
135         if (novi == NULL)
136             return 1;
137
138         *adresa_glave = novi;
139
140         /* Vraca se indikator uspesnog dodavanja */
141         return 0;
142     }
143
144     /* Inace, ako je broj manji ili jednak vrednosti u glavi liste,
145     onda ga treba dodati na pocetak liste. */
146     if ((*adresa_glave)->vrednost >= broj) {
147         return dodaj_na_pocetak_liste(adresa_glave, broj);
148     }
149
150     /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
151     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
152     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
153     return dodaj_iza(pomocni, broj);
154 }
155
156 Cvor *pretrazi_listu(Cvor * glava, int broj)
157 {
158     /* Obilaze se cvorovi liste */
159     for (; glava != NULL; glava = glava->sledeci)
160         /* Ako je vrednost tekućeg cvora jednaka zadatom broju, pretraga
161         se obustavlja */
162         if (glava->vrednost == broj)
163             return glava;
164
165     /* Nema traženog broja u listi i vraća se NULL */
166     return NULL;
167 }
168
169 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
170 {
171     /* Obilaze se cvorovi liste */
172     /* U uslovu ostanka u petlji, bitan je redosled proveru u
173     konjunkciji. */
174     while (glava != NULL && glava->vrednost < broj)
175         glava = glava->sledeci;
```



```

177  /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
179      glava->vrednost veca od traznog broja i tada treba vratiti
181      NULL, jer trazni broj nije nadjen medju manjim brojevima pri
      pocetku sortirane liste, pa se moze zakljuciti da ga nema u
183      listi. Drugi nacini, tako sto se doslo do kraja liste i glava
      je NULL ili tako sto je glava->vrednost == broj. U oba
      poslednja nacina treba vratiti pokazivac glava bilo da je NULL
      ili pokazivac na konkretan cvor. */
185  if (glava->vrednost > broj)
      return NULL;
187  else
      return glava;
189  }

191  void obrisi_cvor(Cvor ** adresa_glave, int broj)
  {
193      Cvor *tekuci = NULL;
      Cvor *pomocni = NULL;
195
      /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
197      broju i azurira se pokazivac na glavu liste */
      while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
      {
199          /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
              adresi adresa_glave */
201          pomocni = (*adresa_glave)->sledeci;
              free(*adresa_glave);
203          *adresa_glave = pomocni;
      }
205
      /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
207      funkcije */
      if (*adresa_glave == NULL)
209          return;

211      /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
          pokazuje na cvor cija je vrednost razlicita od traznog broja.
          Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
213      sledeceg cvora (ako postoji) sa traznim brojem. Cvor se brise
          ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
          Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
215      */
217      tekuci = *adresa_glave;
      while (tekuci->sledeci != NULL)
219          if (tekuci->sledeci->vrednost == broj) {
              /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
221              prvo cuva u promenljivoj pomocni. */
              pomocni = tekuci->sledeci;
223              /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
                  njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
225              sledeci od cvora koji se brise. */
              tekuci->sledeci = pomocni->sledeci;
          }
  }

```

```
227     /* Sada treba osloboditi cvor sa vrednoscu broj. */
    free(pomocni);
229 } else {
    /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
231     pomera na sledeci. */
    tekuci = tekuci->sledeci;
233 }
    return;
235 }

237 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
239     Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;
241
    /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
243     broju i azurira se pokazivac na glavu liste. */
    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
245         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
            adresi adresa_glave. */
247         pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
249         *adresa_glave = pomocni;
    }

251
    /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
253     prekida. Isto se radi i ukoliko glava liste sadrzi vrednost
        koja je veca od broja, jer kako je lista sortirana rastuce nema
255     potrebe broj traziti u repu liste. */
    if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
257         return;

259
    /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci
        pokazuje na cvor cija vrednost je manja od trazenog broja, kao
261     i svim cvorovima levo od njega. Cvor se brise ako je jednak,
        ili, ako je razlicit, prelazi se na sledeci cvor. Ovaj postupak
263     se ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
        cija vrednost je veca od trazenog broja. */
265     tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL
267         && tekuci->sledeci->vrednost <= broj)
        if (tekuci->sledeci->vrednost == broj) {
269             pomocni = tekuci->sledeci;
            tekuci->sledeci = tekuci->sledeci->sledeci;
271             free(pomocni);
        } else {
273             /* Ne treba brisati sledeceg od tekuceg jer je manji od
                trazenog i tekuci se pomera na sledeci cvor. */
275             tekuci = tekuci->sledeci;
        }
277     return;
```

```

}
279 void ispisi_listu(Cvor * glava)
281 {
    /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
283     jer se lista nece menjati */
    putchar('[');
285     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
        pocetka prema kraju liste. */
287     for (; glava != NULL; glava = glava->sledeci) {
        printf("%d", glava->vrednost);
289         if (glava->sledeci != NULL)
            printf(", ");
291     }
    printf("]\n");
293 }

```

*main.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  /* 1) Glavni program */
   int main()
7  {
    /* Lista je prazna na pocetku */
9    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
11   int broj;

13   /* Testiranje funkcije za dodavanje novog broja na pocetak liste */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
15   while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17         memorije za nov cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa. */
19         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
            fprintf(stderr, "Greska: Neuspesna alokacija memorije za cvor.\n");
21             oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
23         }
        printf("\tLista: ");
25         ispisi_listu(glava);
    }

27   /* Testiranje funkcije za pretragu liste */
29   printf("\nUnesite broj koji se trazi: ");
    scanf("%d", &broj);
31

```

## 4 Dinamičke strukture podataka

---

```
trazeni = pretrazi_listu(glava, broj);
33 if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
35 else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
37
/* Oslobadja se memorija zauzeta listom */
39 oslobodi_listu(&glava);

41 exit(EXIT_SUCCESS);
}
```

*main\_b.c*

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

4
/* 2) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
10     int broj;

12     /* Testira se funkcija za dodavanja novog broja na kraj liste */
    printf("Unesite brojeve: (za kraj CTRL+D)\n");
14     while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16         memorije za nov cvor. Memoriju alociranu za cvorove liste
         treba osloboditi pre napustanja programa. */
18         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
            fprintf(stderr, "Greska: Neuspesna alokacija memorije za cvor %
20             d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
22         }
        printf("\tLista: ");
24         ispisi_listu(glava);
    }

26
    /* Testira se funkcije kojom se brise cvor liste */
28     printf("\nUnesite broj koji se brise: ");
    scanf("%d", &broj);
30

    /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
32     procitanom sa ulaza */
    obrisi_cvor(&glava, broj);
34

    printf("Lista nakon brisanja: ");
36     ispisi_listu(glava);
}
```

```

38  /* Oslobadja se memorija zauzeta listom */
    oslobodi_listu(&glava);
40
    exit(EXIT_SUCCESS);
42 }

```

*main.c.c*

```

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

/* 3) Glavni program */
int main()
{
    /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
    int broj;

    /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
       uredjena neopadajuće */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
    while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za nov cvor. Memoriju alociranu za cvorove liste
           treba osloboditi pre napustanja programa. */
        if (dodaj_sortirano(&glava, broj) == 1) {
            fprintf(stderr, "Greska: Neuspesna alokacija memorije za cvor %
d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
        }
        printf("\tLista: ");
        ispisi_listu(glava);
    }

    /* Testira se funkcija za pretragu liste */
    printf("\nUnesite broj koji se trazi: ");
    scanf("%d", &broj);

    trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

    /* Testira se funkcija kojom se brise cvor liste */
    printf("\nUnesite broj koji se brise: ");
    scanf("%d", &broj);

```

```
42      /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
43         procitanom sa ulaza */
44      obrisi_cvor_sortirane_liste(&glava, broj);
45
46      printf("Lista nakon brisanja: ");
47      ispisi_listu(glava);
48
49      /* Oslobadja se memorija zauzeta listom */
50      oslobodi_listu(&glava);
51
52      exit(EXIT_SUCCESS);
53  }
```

### 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 {
7      int vrednost;
8      struct cvor *sledeci;
9  } Cvor;
10
11 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
13    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14 Cvor *napravi_cvor(int broj);
15
16 /* Funkcija oslobadja dinamicu memoriju zauzetu za cvorove liste
17    ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);
19
20 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
21    bilo greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
23
24 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo
25    greske pri alokaciji memorije, inace vraca 0. */
26 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
27
28 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
29    ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
30    memorije, inace vraca 0. */
31 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32
```

```

34  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slučaju da takav cvor ne postoji u listi. */
36  Cvor *pretrazi_listu(Cvor * glava, int broj);

38  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na činjenicu da je lista koja se pretražuje
    neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
42  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

44  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj.
    Azurira pokazivac na glavu liste, koji može biti promenjen u
    slučaju da se obriše stara glava liste. */
46  void obrisi_cvor(Cvor ** adresa_glave, int broj);

48  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
    promenjen ukoliko se obriše stara glava liste. */
52  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

54  /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
    zarezima. */
56  void ispisi_vrednosti(Cvor * glava);

58  /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka
    kraju liste, razdvojene zarezima i uokvirene zagradama. */
60  void ispisi_listu(Cvor * glava);

62  #endif

```

*lista.c*

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  Cvor *napravi_cvor(int broj)
   {
7      /* Alokacija memorije za novi cvor uz proveru uspesnosti */
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9      if (novi == NULL)
          return NULL;

11     /* Inicijalizacija polja strukture */
13     novi->vrednost = broj;
     novi->sledeci = NULL;

15     /* Vraca se adresa novog cvora */

```

```
17     return novi;
18 }
19
20 void oslobodi_listu(Cvor ** adresa_glave)
21 {
22     /* Ako je lista vec prazna */
23     if (*adresa_glave == NULL)
24         return;
25
26     /* Ako lista nije prazna, treba osloboditi memoriju. Treba
27        osloboditi rep liste, pre oslobadjanja memorije za glavu liste
28        */
29     oslobodi_listu(&(*adresa_glave)->sledeci);
30     /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
31        glava u pozivajucoj funkciji tako da odgovara praznoj listi */
32     free(*adresa_glave);
33     *adresa_glave = NULL;
34 }
35
36 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
37 {
38     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39     Cvor *novi = napravi_cvor(broj);
40     if (novi == NULL)
41         return 1;
42
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         /* Azuriranjem vrednosti na koju pokazuje adresa_glave, ujedno
64            se azurira i pokazivacka promenljiva u pozivajucoj funkciji */
65         *adresa_glave = novi;
66
67         /* Vraca se indikator uspesnog dodavanja */
68         return 0;
69     }
```



```

67     }

69     /* Ako lista nije prazna, broj se dodaje u rep liste. */
    /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
71     novi broj se dodaje u rep liste u rekurzivnom pozivu.
    Informaciju o uspesnosti alokacije u rekurzivnom pozivu
73     funkcija prosledjuje visem rekurzivnom pozivu koji tu
    informaciju vraca u rekurzivni poziv iznad, sve dok se ne vrati
75     u main. Tek je iz main funkcije moguće pristupiti pravom
    pocetku liste i osloboditi je celu, ako ima potrebe. Ako je
77     funkcija vratila 0, onda nije bilo greske. */
    return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
79 }

81 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
83     /* Ako je lista prazna */
    if (*adresa_glave == NULL) {
85
87         /* Novi cvor postaje glava liste */
        Cvor *novi = napravi_cvor(broj);

89         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
        */
        if (novi == NULL)
91             return 1;

93         /* Azurira se glava liste */
        *adresa_glave = novi;

95         /* Vraca se indikator uspesnog dodavanja */
        return 0;
97     }

99     /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
    glavi liste, onda se dodaje na pocetak liste */
101     if ((*adresa_glave)->vrednost >= broj)
103         return dodaj_na_pocetak_liste(adresa_glave, broj);

105     /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
    bude sortirana lista. */
107     return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
}

109 Cvor *pretrazi_listu(Cvor * glava, int broj)
111 {
    /* U praznoj listi nema vrednosti */
113     if (glava == NULL)
        return NULL;

115     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
117     if (glava->vrednost == broj)

```

```
        return glava;
119
    /* Inace, pretraga se nastavlja u repu liste */
121    return pretrazi_listu(glava->sledeci, broj);
123
124    Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
125    {
126        /* Trazenog broja nema ako je lista prazna ili ako je broj manji
127           od vrednosti u glavi liste, jer je lista neopadajuće sortirana
128           */
129        if (glava == NULL || glava->vrednost > broj)
130            return NULL;
131
132        /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
133        if (glava->vrednost == broj)
134            return glava;
135
136        /* Inace, pretraga se nastavlja u repu liste */
137        return pretrazi_listu(glava->sledeci, broj);
138
139    void obrisi_cvor(Cvor ** adresa_glave, int broj)
140    {
141        /* U praznoj listi nema cvorova za brisanje. */
142        if (*adresa_glave == NULL)
143            return;
144
145        /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
146        obrisi_cvor(&(*adresa_glave)->sledeci, broj);
147
148        /* Preostaje provera da li glavu liste treba obrisati */
149        if ((*adresa_glave)->vrednost == broj) {
150            /* Pomocni pokazuje na cvor koji treba da se obrise */
151            Cvor *pomocni = *adresa_glave;
152            /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi
153               i brise se cvor koji je bio glava liste. */
154            *adresa_glave = (*adresa_glave)->sledeci;
155            free(pomocni);
156        }
157    }
158
159    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
160    {
161        /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
162           veca od broja, kako je lista sortirana rastuce nema potrebe
163           broj traziti u repu liste i zato se funkcija prekida */
164        if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
165            return;
166
167        /* Brisu se cvorovi iz repa koji imaju vrednost broj */
168        obrisi_cvor(&(*adresa_glave)->sledeci, broj);
```

```

169      /* Preostaje provera da li glavu liste treba obrisati */
171      if ((*adresa_glave)->vrednost == broj) {
          /* Pomocni pokazuje na cvor koji treba da se obrise */
173          Cvor *pomocni = *adresa_glave;
          /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
175             brise se cvor koji je bio glava liste */
          *adresa_glave = (*adresa_glave)->sledeci;
177          free(pomocni);
      }
179  }

181  void ispisi_vrednosti(Cvor * glava)
182  {
183      /* Prazna lista */
184      if (glava == NULL)
185          return;

187      /* Ispisuje se vrednost u glavi liste */
188      printf("%d", glava->vrednost);

189      /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
190         se poziva ista funkcija za ispis ostalih. */
191      if (glava->sledeci != NULL) {
192          printf(", ");
193          ispisi_vrednosti(glava->sledeci);
194      }
195  }

197  void ispisi_listu(Cvor * glava)
198  {
199      /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
200         jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
201         na glavu liste iz pozivajuće funkcije. Ona ispisuje samo
202         zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
203         rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
204         elemente razdvojene zapetom i razmakom. */
205      putchar('[');
206      ispisi_vrednosti(glava);
207      printf("]\n");
209  }

```

### Rešenje 4.3

```

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

4
#define MAX_DUZINA 20

6
/* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,

```

```

8      broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
typedef struct _Cvor {
10     char etiketa[20];
    unsigned broj_pojavljanja;
12     struct _Cvor *sledeci;
} Cvor;

14
/* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
ili NULL ako alokacija nije uspesno izvršena */
16 Cvor *napravi_cvor(unsigned br, char *etiketa)
{
18     /* Alokacija memorije za cvor uz proveru uspesnosti alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
20     if (novi == NULL)
22         return NULL;

24     /* Inicijalizacija polja strukture */
    novi->broj_pojavljanja = br;
26     strcpy(novi->etiketa, etiketa);
    novi->sledeci = NULL;

28
    /* Vraca se adresa novog cvora */
30     return novi;
}

32
/* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
34 void oslobodi_listu(Cvor ** adresa_glave)
{
36     Cvor *pomocni = NULL;

38     /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
azurira se vrednost glave liste */
40     while (*adresa_glave != NULL) {
        pomocni = (*adresa_glave)->sledeci;
42         free(*adresa_glave);
        *adresa_glave = pomocni;
44     }
    /* Pokazivac glava u main funkciji, na adresi adresa_glave, bice
postavljen na NULL vrednost po izlasku iz petlje. */
46 }

48
/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
50 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
52                             char *etiketa)
{
54     /* Kreira se novi cvor i proverava se uspesnost alokacije */
    Cvor *novi = napravi_cvor(br, etiketa);
56     if (novi == NULL)
        return 1;

58
    /* Dodaje se novi cvor na pocetak liste */

```

```

60     novi->sledeci = *adresa_glave;
    *adresa_glave = novi;

62

    /* Vraca se indikator uspesnog dodavanja */
64     return 0;
}

66
/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
68     NULL ako takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
70 {
    Cvor *tekuci;

72

    /* Obilazi se cvor po cvor liste */
74     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
        /* Ako tekuci cvor sadrzi trazenu etiketu, vraca se njegova
76         vrednost */
        if (strcmp(tekuci->etiketa, etiketa) == 0)
78             return tekuci;

80     /* Cvor nije pronadjen */
    return NULL;
82 }

84
/* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Cvor * glava)
86 {
    /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
88     broj njihovog pojavljivanja u HTML datoteci. */
    for (; glava != NULL; glava = glava->sledeci)
90         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
}

92
/* Glavni program */
94 int main(int argc, char **argv)
{
96     /* Proverava se da li je program pozvan sa ispravnim brojem
        argumenata
        komandne linije. */
98     if (argc != 2) {
        fprintf(stderr,
100             "Greska: Program se poziva sa: ./a.out datoteka.html!\n")
        ;
        exit(EXIT_FAILURE);
102     }

104
    /* Priprema datoteke za citanje */
    FILE *in = NULL;
106     in = fopen(argv[1], "r");
    if (in == NULL) {
108         fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke %s!\n", argv[1]);

```

```

110     exit(EXIT_FAILURE);
111 }
112
113 char c;
114 int i = 0;
115 char procitana[MAX_DUZINA];
116 Cvor *glava = NULL;
117 Cvor *trazeni = NULL;
118
119 /* Cita se datoteka, karakter po karakter, dok se ne procita cela
120    */
121 while ((c = fgetc(in)) != EOF) {
122     /* Proverava se da li se pocinje sa citanjem nove etikete */
123     if (c == '<') {
124         /* Proverava se da li se cita zatvarajuca etiketa */
125         if ((c = fgetc(in)) == '/') {
126             i = 0;
127             while ((c = fgetc(in)) != '>')
128                 procitana[i++] = c;
129         }
130         /* Cita se otvarajuca etiketa */
131         else {
132             i = 0;
133             procitana[i++] = c;
134             while ((c = fgetc(in)) != ' ' && c != '>')
135                 procitana[i++] = c;
136         }
137         procitana[i] = '\0';
138
139         /* Trazi se procitana etiketa medju postojećim cvorovima
140            liste. Ukoliko ne postoji, dodaje se novi cvor za ucitanu
141            etiketu sa brojem pojavljivanja 1. Inace se uvecava broj
142            pojavljivanja etikete. */
143         trazeni = pretrazi_listu(glava, procitana);
144         if (trazeni == NULL) {
145             if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
146                 fprintf(stderr,
147                     "Greska: Neuspesna alokacija memorije za nov cvor\n
148                     ");
149                 oslobodi_listu(&glava);
150                 exit(EXIT_FAILURE);
151             }
152             else
153                 trazeni->broj_pojavljivanja++;
154         }
155     }
156 }
157
158 /* Zatvara se datoteka */
159 fclose(in);
160
161 /* Ispisuje se sadrzaj cvorova liste */
162 ispisi_listu(glava);

```

```

160     /* Oslobadja se memorija zauzeta listom */
162     oslobodi_listu(&glava);
164     exit(EXIT_SUCCESS);
}

```

#### Rešenje 4.4

```

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

#define MAX_INDEKS 11
#define MAX_IME_PREZIME 21

/* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
   studentu */
typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
    char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
    struct _Cvor *sledeci;
} Cvor;

/* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
   novi cvor ili NULL ukoliko je doslo do greske */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
{
    /* Alokacija memorije za cvor uz proveru uspesnosti alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;

    /* Inicijalizacija polja strukture */
    strcpy(novi->broj_indeksa, broj_indeksa);
    strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
    novi->sledeci = NULL;

    /* Vraca se adresa novog cvora */
    return novi;
}

/* Funkcija oslobadja memoriju zauzetu cvorovima liste */
void oslobodi_listu(Cvor ** adresa_glave)
{
    /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
    if (*adresa_glave == NULL)
        return;
}

```

```
44  /* Rekurzivnim pozivom se oslobadja rep liste */
    oslobodi_listu(&(*adresa_glave)->sledeci);

46  /* Potom se oslobadja i glava liste */
    free(*adresa_glave);

48

50  /* Proglasava se lista praznom */
    *adresa_glave = NULL;
}

52
/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
54  do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
56                          char *ime, char *prezime)
{
58  /* Kreira se novi cvor i proverava se uspesnost alokacije */
    Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
60  if (novi == NULL)
        return 1;

62

64  /* Dodaje se novi cvor na pocetak liste */
    novi->sledeci = *adresa_glave;
    *adresa_glave = novi;

66

68  /* Vraca se indikator uspesnog dodavanja */
    return 0;
}

70
/* Funkcija ispisuje sadrzaj cvorova liste. */
72 void ispisi_listu(Cvor * glava)
{
74  /* Pocevsi od glave liste */
    for (; glava != NULL; glava = glava->sledeci)
76        printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
                glava->prezime);
78 }

80 /* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj
    indeksa. U suprotnom funkcija vraca NULL */
82 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
{
84  /* Ako je lista prazna, ne postoji trazeni cvor */
    if (glava == NULL)
86        return NULL;

88  /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste
    */
    if (!strcmp(glava->broj_indeksa, broj_indeksa))
90        return glava;

92  /* Ukoliko u glavi liste nije trazeni indeks, pretraga se
    nastavlja u repu liste */
```



```

94     return pretrazi_listu(glava->sledeci, broj_indeksa);
95 }
96
97 /* Glavni program */
98 int main(int argc, char **argv)
99 {
100     /* Argumenti komandne linije su neophodni jer se iz komandne
101        linije dobija ime datoteke sa informacijama o studentima */
102     if (argc != 2) {
103         fprintf(stderr,
104             "Greska: Program se poziva sa: ./a.out ime_datoteke\n");
105         exit(EXIT_FAILURE);
106     }
107
108     /* Priprema datoteke za citanje */
109     FILE *in = NULL;
110     in = fopen(argv[1], "r");
111     if (in == NULL) {
112         fprintf(stderr,
113             "Greska: Neuspesno otvaranje datoteke %s.\n", argv[1]);
114         exit(EXIT_FAILURE);
115     }
116
117     /* Deklaracije pomocnih promenljiva za citanje vrednosti koje
118        treba smestiti u listu */
119     char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
120     char broj_indeksa[MAX_INDEKS];
121     Cvor *glava = NULL;
122     Cvor *trazeni = NULL;
123
124     /* Ucitavanje vrednosti u listu */
125     while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
126     {
127         if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
128             fprintf(stderr,
129                 "Greska: Neuspesna alokacija memorije za nov cvor\n");
130             oslobodi_listu(&glava);
131             exit(EXIT_FAILURE);
132         }
133     }
134
135     /* Datoteka vise nije potrebna i zatvara se. */
136     fclose(in);
137
138     /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
139     while (scanf("%s", broj_indeksa) != EOF) {
140         trazeni = pretrazi_listu(glava, broj_indeksa);
141         if (trazeni == NULL)
142             printf("ne\n");
143         else
144             printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
145     }
146
147     /* Oslobadja se memorija zauzeta za cvorove liste. */

```

```
146     oslobodi_listu(&glava);
148     exit(EXIT_SUCCESS);
}
```

### Rešenje 4.6

NAPOMENA: *Rešenje koristi biblioteku za rad sa listama iz zadatka 4.1.*

```
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 listi 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;
16     Cvor *tekuci = NULL;
17
18     /* Ako su obe liste prazne i rezultat je prazna lista */
19     if (lista1 == NULL && lista2 == NULL)
20         return NULL;
21
22     /* Ako je prva lista prazna, rezultat je druga lista */
23     if (lista1 == NULL)
24         return lista2;
25
26     /* Ako je druga lista prazna, rezultat je prva lista */
27     if (lista2 == NULL)
28         return lista1;
29
30     /* Odredjuje se prvi cvor rezultujuće liste - to je ili prvi cvor
31        liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
32        koji sadrzi manju vrednost */
33     if (lista1->vrednost < lista2->vrednost) {
34         rezultujuca = lista1;
35         lista1 = lista1->sledeci;
36     } else {
37         rezultujuca = lista2;
38         lista2 = lista2->sledeci;
39     }
40     /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste, ne
41        sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
42        koji sadrzi adresu trenutnog cvora rezultujuće liste */
43 }
```

```

43     tekuci = rezultujuca;

45     /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi
46        cvor tako da bude uredjena neopadajuce. Pokazivac na listu iz
47        koje se uzima cvor se azurira tako da pokazuje na sledeci cvor
48        */
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     /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
60        rezultujucu listu treba nadovezati ostatak druge liste */
61     if (lista1 == NULL)
62         tekuci->sledeci = lista2;
63     else
64         /* U suprotnom treba nadovezati ostatak prve liste */
65         tekuci->sledeci = lista1;

67     /* Preko adresa glava polaznih listi vrednosti pokazivaca u
68        pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
69        prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
70        se moze doci prateci pokazivace iz glave rezultujuce liste, tako
71        da stare pokazivace treba postaviti na NULL. */
72     *adresa_glave_1 = NULL;
73     *adresa_glave_2 = NULL;

75     return rezultujuca;
76 }

77
78 int main(int argc, char **argv)
79 {
80     /* Argumenti komandne linije su neophodni */
81     if (argc != 3) {
82         fprintf(stderr,
83             "Greska: Program se poziva sa:\n ./a.out dat1.txt dat2.
84             txt\n");
85         exit(EXIT_FAILURE);
86     }

87     /* Otvaraju se datoteke u kojima se nalaze elementi listi */
88     FILE *in1 = NULL;
89     in1 = fopen(argv[1], "r");
90     if (in1 == NULL) {
91         fprintf(stderr,
92             "Greska: Neuspesno otvaranje datoteke %s.\n", argv[1]);

```

```
93     exit(EXIT_FAILURE);
94 }
95
96 FILE *in2 = NULL;
97 in2 = fopen(argv[2], "r");
98 if (in2 == NULL) {
99     fprintf(stderr,
100         "Greska: Neuspesno otvaranje datoteke %s.\n", argv[2]);
101     exit(EXIT_FAILURE);
102 }
103
104 /* Liste su na pocetku prazne */
105 int broj;
106 Cvor *lista1 = NULL;
107 Cvor *lista2 = NULL;
108
109 /* Ucitavanje listi */
110 while (fscanf(in1, "%d", &broj) != EOF)
111     dodaj_na_kraj_liste(&lista1, broj);
112
113 while (fscanf(in2, "%d", &broj) != EOF)
114     dodaj_na_kraj_liste(&lista2, broj);
115
116 /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
117 fclose(in1);
118 fclose(in2);
119
120 /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
121    objedinjavanjem listi */
122 Cvor *rezultat = objedini(&lista1, &lista2);
123
124 /* Ispis rezultujuce liste. */
125 ispisi_listu(rezultat);
126
127 /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.
128    Njenim oslobadjanjem oslobadja se sva zauzeta memorija. */
129 oslobodi_listu(&rezultat);
130
131 exit(EXIT_SUCCESS);
132 }
```

### Rešenje 4.8

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
5    predstavlja vidjenu zagradu i pokazivac na sledeci cvor liste */
6 typedef struct cvor {
7     char zagrada;
8     struct cvor *sledeci;
9 }
```

```
9 } Cvor;

11 /* Funkcija koja oslobadja memoriju zauzetu stekom */
void oslobodi_stek(Cvor ** stek)
13 {
    Cvor *tekuci;
15     Cvor *pomocni;

17     /* Oslobadja se cvor po cvor steka */
    tekuci = *stek;
19     while (tekuci != NULL) {
        pomocni = tekuci->sledeci;
21         free(tekuci);
        tekuci = pomocni;
23     }

25     /* Stek se proglašava praznim */
    *stek = NULL;
27 }

29 /* Glavni program */
int main()
31 {
    /* Stek je na pocetku prazan */
33     Cvor *stek = NULL;
    FILE *ulaz = NULL;
35     char c;
    Cvor *pomocni = NULL;
37

39     /* Otvaranje datoteke za citanje izraza */
    ulaz = fopen("izraz.txt", "r");
    if (ulaz == NULL) {
41         fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke izraz.txt!\n");
43         exit(EXIT_FAILURE);
    }

45     /* Cita se datoteka, karakter po karakter, dok se ne procita cela
    */
47     while ((c = fgetc(ulaz)) != EOF) {
        /* Ako je učitana otvorena zagrada, stavlja se na stek */
49         if (c == '(' || c == '{' || c == '[') {
            /* Alocira se memorija za novi cvor liste i proverava se
            uspesnost alokacije */
51             pomocni = (Cvor *) malloc(sizeof(Cvor));
            if (pomocni == NULL) {
53                 fprintf(stderr, "Greska: Neuspesna alokacija memorije!\n");
                    /* Oslobadjanje memorije zauzete stekom */
55                 oslobodi_stek(&stek);
                    /* Prekid izvršavanja programa */
57                 exit(EXIT_FAILURE);
            }
59         }
```

```
61      /* Inicijalizacija polja strukture */
        pomocni->zagrada = c;
63
        /* Promena vrha steka */
65      pomocni->sledeci = stek;
        stek = pomocni;
67  }
        /* Ako je učitana zatvorena zagrada, proverava se da li je stek
69      prazan i ako nije, da li se na vrhu steka nalazi odgovarajuća
        otvorena zagrada */
71  else {
        if (c == '(' || c == '[' || c == '{') {
73          if (stek != NULL && ((stek->zagrada == '(' && c == '(')
                                || (stek->zagrada == '[' && c == '[')
                                || (stek->zagrada == '{' && c == '{'))
75          {
                /* Sa vrha steka se uklanja otvorena zagrada */
77          pomocni = stek->sledeci;
                free(stek);
                stek = pomocni;
79          } else {
81          /* Dakle zagrade u izrazu nisu ispravno uparene */
                break;
83          }
        }
85    }
    }
87
    /* Pročitana je cela datoteka i treba je zatvoriti */
89    fclose(ulaz);

91    /* Ako je stek prazan i pročitana je cela datoteka, zagrade su
        ispravno uparene */
93    if (stek == NULL && c == EOF)
        printf("Zagrade su ispravno uparene.\n");
95    else {
        /* U suprotnom se zaključuje da zagrade nisu ispravno uparene */
97        printf("Zagrade nisu ispravno uparene.\n");
        /* Oslobadja se memorija koja je ostala zauzeta stekom */
99        oslobodi_stek(&stek);
    }
101
    exit(EXIT_SUCCESS);
103 }
```

## Rešenje 4.9

stek.h

```

1  #ifndef _STEK_H_
2  #define _STEK_H_

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

14 #define VAN_ETIKETE 0
15 #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2

18 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
   pokazivac na sledeci 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
26    njegovu adresu ili NULL ako alokacija nije bila uspesna */
27 Cvor *napravi_cvor(char *etiketa);

28
29 /* Funkcija oslobadja memoriju zauzetu stekom */
30 void oslobodi_stek(Cvor ** adresa_vrha);

32 /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske
   pri alokaciji za novi cvor funkcija vraca 1, inace vraca 0 */
34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);

36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
   pokazivac razlicit od NULL, tada u niz karaktera na koji on
38   pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok
   u suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan
40   (pa samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
   suprotnom */
42 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);

44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
   steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);

```

## 4 Dinamičke strukture podataka

```
48 /* Funkcija prikazuje stek od vrha prema dnu */
void prikazi_stek(Cvor * vrh);
50
/* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
52 etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
54 procita etiketa. Vraca OTVORENA, ako je procitana otvorena
etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
56 int uzmi_etiketu(FILE * f, char *etiketa);
58 #endif
```

stek.c

```
#include "stek.h"
2
Cvor *napravi_cvor(char *etiketa)
4 {
    /* Alokacija memorije za novi cvor uz proveru uspesnost alokacije
    */
    6 Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
    8     return NULL;

    /* Inicijalizacija polja u novom cvoru */
    10 if (strlen(etiketa) >= MAX) {
    12     fprintf(stderr, "Greska: Etiketa je preduga, bice skracena.\n");
    etiketa[MAX - 1] = '\0';
    14 }
    strcpy(novi->etiketa, etiketa);
    16 novi->sledec = NULL;

    /* Vraca se adresa novog cvora */
    18 return novi;
    20 }

void oslobodi_stek(Cvor ** adresa_vrha)
22 {
    24 Cvor *pomocni;

    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
    26 while (*adresa_vrha != NULL) {
    28     /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
    osloboditi cvor koji predstavlja vrh steka */
    30     pomocni = *adresa_vrha;
    /* Sledec cvor je novi vrh steka */
    32     *adresa_vrha = (*adresa_vrha)->sledec;
    free(pomocni);
    34 }

    36 /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
```



```
    adresa_vrha ce pokazivati na NULL. */
38 }

40 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
41 {
42     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43     Cvor *novi = napravi_cvor(etiketa);
44     if (novi == NULL)
45         return 1;

46     /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
47     novi->sledeci = *adresa_vrha;
48     *adresa_vrha = novi;
49     return 0;
50 }

52 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
53 {
54     Cvor *pomocni;

55     /* Pokušaj skidanja vrednosti sa praznog steka rezultuje greskom i
56        vraca se 0 */
57     if (*adresa_vrha == NULL)
58         return 0;

59     /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
60        adresu kopira etiketa sa vrha steka */
61     if (etiketa != NULL)
62         strcpy(etiketa, (*adresa_vrha)->etiketa);

63     /* Element sa vrha steka se uklanja */
64     pomocni = *adresa_vrha;
65     *adresa_vrha = (*adresa_vrha)->sledeci;
66     free(pomocni);

67     /* Vraca se indikator uspesno izvršene radnje */
68     return 1;
69 }

71 char *vrh_steka(Cvor * vrh)
72 {
73     /* Prazan stek nema cvor koji je vrh i vraca se NULL */
74     if (vrh == NULL)
75         return NULL;

76     /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
77        koji je na vrhu steka. */
78     return vrh->etiketa;
79 }

81 void prikazi_stek(Cvor * vrh)
82 {
83 }
```

```

90  /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
    for (; vrh != NULL; vrh = vrh->sledeci)
        printf("<%s>\n", vrh->etiketa);
92 }

94 int uzmi_etiketu(FILE * f, char *etiketa)
{
96     int c;
    int i = 0;
98     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
    etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
100     nije zapoceto citanje. */
    /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
102     OTVORENA ili ZATVORENA. */
    int stanje = VAN_ETIKETE;
104     int tip;

106     /* HTML je neosetljiv na razliku izmedju malih i velikih slova,
    dok to u C-u ne vazi. Zato ce sve etikete biti prevedene u
108     zapis samo malim slovima. */
    while ((c = fgetc(f)) != EOF) {
110         switch (stanje) {
            case VAN_ETIKETE:
112                 if (c == '<')
                    stanje = PROCITANO_MANJE;
114                 break;
            case PROCITANO_MANJE:
116                 if (c == '/') {
                    /* Cita se zatvorena etiketa */
                    tip = ZATVORENA;
118                 } else {
                    if (isalpha(c)) {
120                         /* Cita se otvorena etiketa */
                        tip = OTVORENA;
122                         etiketa[i++] = tolower(c);
124                     }
                }
126                /* Od sada se cita etiketa i zato se menja stanje */
                stanje = U_ETIKETI;
128                break;
            case U_ETIKETI:
130                if (isalpha(c) && i < MAX - 1) {
                    /* Ako je procitani karakter slovo i nije prekoracena
132                     dozvoljena duzina etikete, procitani karakter se smanjuje
                    i smesta u etiketu */
                    etiketa[i++] = tolower(c);
134                } else {
136                    /* Inace, staje se sa citanjem etikete. Korektno se završava
                    niska koja sadrzi procitanu etiketu i vraća se njen tip */
138                    etiketa[i] = '\0';
                    return tip;
140                }
        }
    }
}

```

```

        break;
    }
}
/* Doslo se do kraja datoteke pre nego sto je procitana naredna
   etiketa i vraca se EOF. */
return EOF;
}

```

*main.c*

```

1  #include "stek.h"

3  /* Glavni program */
int main(int argc, char **argv)
5  {
    /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
       nije procitana. */
    Cvor *vrh = NULL;
    char etiketa[MAX];
    int tip;
    int uparene = 1;
    FILE *f = NULL;

13     /* Ime datoteke se preuzima iz komandne linije. */
    if (argc < 2) {
        fprintf(stderr, "Greska:");
        fprintf(stderr, "Program se poziva sa:\n %s ime_html_datoteke\n",
17             argv[0]);
        exit(EXIT_FAILURE);
    }

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

27     /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
    while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
        /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
           etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
           potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
           koje nemaju sadrzaj (npr link). Zbog jednostavnosti
           pretpostavlja se da njih nema u HTML dokumentu. */
        if (tip == OTVORENA) {
            if (strcmp(etiketa, "br") != 0
37                && strcmp(etiketa, "hr") != 0
                && strcmp(etiketa, "meta") != 0)
            if (potisni_na_stek(&vrh, etiketa) == 1) {
39                fprintf(stderr,
41

```

```

                                "Greska: Neuspesna alokacija memorije za nov cvor\n
");
43         oslobodi_stek(&vrh);
        exit(EXIT_FAILURE);
45     }
}
47     /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da
        je u pitanju zatvaranje etikete koja je poslednja otvorena, a
49     jos uvek nije zatvorena. Ona se mora nalaziti na vrhu steka.
        Ako je taj uslov ispunjen, skida se sa steka, jer je upravo
51     zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
        nisu pravilno uparene. */
53     else if (tip == ZATVORENA) {
        if (vrh_steka(vrh) != NULL
55         && strcmp(vrh_steka(vrh), etiketa) == 0)
            skini_sa_steka(&vrh, NULL);
57         else {
            printf("Etikete nisu pravilno uparene\n");
            printf("(nadjena je etiketa </%s>", etiketa);
59             if (vrh_steka(vrh) != NULL)
                printf(", a poslednja otvorena je </%s>)\n",
61                 vrh_steka(vrh));
            else
                printf(" koja nije otvorena)\n");
63             uparene = 0;
            break;
65         }
67     }
}
69 }
/* Zavrшено je citanje i datoteka se zatvara */
71 fclose(f);

73 /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi
    trebalo da bude prazan. Ukoliko nije, tada postoje etikete koje
75     su ostale otvorene */
if (uparene) {
77     if (vrh_steka(vrh) == NULL)
        printf("Etikete su pravilno uparene!\n");
79     else {
        printf("Etikete nisu pravilno uparene\n");
        printf("(etiketa </%s> nije zatvorena)\n", vrh_steka(vrh));
81         /* Oslobadja se memorija zauzeta stekom */
        oslobodi_stek(&vrh);
83     }
85 }

87 exit(EXIT_SUCCESS);
}
```

## Rešenje 4.10

*red.h*

```

1  #ifndef _RED_H_
2  #define _RED_H_
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define MAX 1000
8  #define JMBG_DUZINA 14
9
10 /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11    opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
18    korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25    poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
26    greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27    treba smestiti u novi cvor zbog smestanja manjeg podatka na
28    sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29    bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31
32 /* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
33 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
34
35 /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
36    greske pri alokaciji memorije za novi cvor, inace vraca 0. */
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
38                Zahtev * zahtev);
39
40 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
41    pokazivac razlicit od NULL, tada se u strukturu na koju on
42    pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
43    suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1
44    u suprotnom. */
45 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
46                  Zahtev * zahtev);
47

```

## 4 Dinamičke strukture podataka

---

```
/* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev
korisnika na pocetku reda. Ukoliko je red prazan funkcija vraca
NULL. */
51 Zahtev *pocetak_reda(Cvor * pocetak);

53 /* Funkcija prikazuje sadrzaj reda. */
void prikazi_red(Cvor * pocetak);
55
#endif
```

*red.c*

```
#include "red.h"

2
Cvor *napravi_cvor(Zahtev * zahtev)
4 {
    /* Alokacija memorije za novi cvor uz proveru uspesnost alokacije
    */
    6 Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        8 return NULL;

    10 /* Inicijalizacija polja strukture */
    novi->nalog = *zahtev;
    12 novi->sledeci = NULL;

    14 /* Vraca se adresa novog cvora */
    return novi;
    16 }

18 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
    20 Cvor *pomocni = NULL;

    22 /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
    while (*pocetak != NULL) {
        24 /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
        osloboditi cvor sa pocetka reda */
        26 pomocni = *pocetak;
        *pocetak = (*pocetak)->sledeci;
        28 free(pomocni);
    }

    30 /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
    treba postaviti na NULL. */
    32 *kraj = NULL;
}

34
36 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
    Zahtev * zahtev)
{
    38 /* Kreira se novi cvor i proverava se uspesnost kreiranja */
```

```

40  Cvor *novi = napravi_cvor(zahtev);
    if (novi == NULL)
        return 1;
42
    /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
44     kraj, to je podjednako efikasno kao dodavanje na pocetak liste
        */
    if (*adresa_kraja != NULL) {
46         (*adresa_kraja)->sledeci = novi;
        *adresa_kraja = novi;
48     } else {
        /* Ako je red bio ranije prazan */
50         *adresa_pocetka = novi;
        *adresa_kraja = novi;
52     }

54     /* Vraca se indikator uspesnog dodavanja */
    return 0;
56 }

58 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                   Zahtev * zahtev)
60 {
    Cvor *pomocni = NULL;
62
    /* Ako je red prazan */
64     if (*adresa_pocetka == NULL)
        return 0;
66
    /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
68     zahtev koji je na pocetku reda. */
    if (zahtev != NULL)
70         *zahtev = (*adresa_pocetka)->nalog;

72     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i
        azurira se pokazivac na adresi adresa_pocetka da pokazuje na
74     sledeci cvor u redu. */
    pomocni = *adresa_pocetka;
76     *adresa_pocetka = (*adresa_pocetka)->sledeci;
    free(pomocni);
78
    /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
80     potrebno je azurirati i vrednost pokazivaca na adresi
        adresa_kraja na NULL */
82     if (*adresa_pocetka == NULL)
        *adresa_kraja = NULL;
84
    return 1;
86 }

88 Zahtev *pocetak_reda(Cvor * pocetak)
    {

```

## 4 Dinamičke strukture podataka

```
90  /* U praznom redu nema zahteva */
91  if (pocetak == NULL)
92      return NULL;

94  /* Inace, vraća se pokazivac na zahtev sa pocetka reda */
95  return &(pocetak->nalog);
96  }

98  void prikazi_red(Cvor * pocetak)
99  {
100     /* Prikazuje se sadržaj reda od pocetka prema kraju */
101     for (; pocetak != NULL; pocetak = pocetak->sledeci)
102         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
104     printf("\n");
105 }
```

*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;
17
18     /* Sluzbenik evidentira korisnicke zahteve unosjenjem njihovog JMBG
19        broja i opisa potrebne usluge. */
20     printf("Sluzbenik evidentira korisnicke zahteve:\n");
21     while (1) {
22
23         /* Ucitava se JMBG broj */
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
29            JMBG broja procitao i da bi fgets nakon toga procitala
30            ispravan red sa opisom zahteva */
31         getchar();
```



```

33  /* Ucitava se opis problema */
printf("\tOpis problema: ");
35  fgets(nov_zahtev.opis, MAX - 1, stdin);
/* Ako je poslednji karakter nov red, eliminise se */
37  if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
    nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

39
/* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
41  if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
    fprintf(stderr,
43         "Greska: Neuspesna alokacija memorije za nov cvor\n");
    oslobodi_red(&pocetak, &kraj);
45     exit(EXIT_FAILURE);
}
47 }

49 /* Otvaranje datoteke za dopisivanje izvestaja */
FILE *izlaz = fopen("izvestaj.txt", "a");
51 if (izlaz == NULL) {
    fprintf(stderr,
53         "Greska: Neuspesno otvaranje datoteke izvestaj.txt\n");
    exit(EXIT_FAILURE);
55 }

57 /* Dokle god ima korisnika u redu, treba ih usluziti */
while (1) {
59     sledeci = pocetak_reda(pocetak);
/* Ako nema nikog vise u redu, prekida se petlja */
61     if (sledeci == NULL)
        break;

63     printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
65     printf("i zahtevom: %s\n", sledeci->opis);

67     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

69     broj_usluzenih++;

71     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
    scanf("%s", odgovor);

73
    if (strcmp(odgovor, "Da") == 0)
75         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
    else
77         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
            nov_zahtev.opis);

79
    if (broj_usluzenih == VREME_ZA_PAUZU) {
81         printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);

83
        if (strcmp(odgovor, "Da") == 0)

```

```

85         break;
86     else
87         broj_usluzenih = 0;
88     }
89 }

91 /******
92  Usluzivanje korisnika moze da se izvrsi i na sledeci nacin: */
93 /******

95 while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
96     printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
97         nov_zahtev.jmbg);
98     printf("sa zahtevom: %s\n", nov_zahtev.opis);
99     broj_usluzenih++;

101     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
102     scanf("%s", odgovor);
103     if (strcmp(odgovor, "Da") == 0)
104         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
105     else
106         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
107             nov_zahtev.jmbg, nov_zahtev.opis);

109     if (broj_usluzenih == VREME_ZA_PAUZU) {
110         printf("\nDa li je kraj smene? [Da/Ne] ");
111         scanf("%s", odgovor);
112         if (strcmp(odgovor, "Da") == 0)
113             break;
114         else
115             broj_usluzenih = 0;
116     }
117 }

119 /******
120  Datoteka vise nije potrebna i treba je zatvoriti. */
121 fclose(izlaz);

123 /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
124    neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
125    koju zauzima red sa neobradjenim zahtevima korisnika. */
126 oslobodi_red(&pocetak, &kraj);

127 exit(EXIT_SUCCESS);
128 }

```

### Rešenje 4.11

*dvostruko\_povezana\_lista.h*

```
#ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
```

```
2 #define _DVOSTRUKO_POVEZANA_LISTA_H_
4 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   vrednost i pokazivace na sledeci i prethodni cvor liste. */
6 typedef struct cvor {
   int vrednost;
8   struct cvor *sledeci;
   struct cvor *prethodni;
10 } Cvor;

12 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14 na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji
18 na adresi adresa_kraja. */
void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);

22 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
24 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
   adresa_kraja, int broj);

26 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
28 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
30 int broj);

32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   novi 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,
42 inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
44 broj);

46 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
48 NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

50
52 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
   neopadajuće sortirana. Vraca pokazivac na cvor liste koji sadrzi
```

```
54     trazeni broj ili NULL u slucaju da takav cvor ne postoji. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

56
/* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi
58     ciji pokazivac glava se nalazi na adresi adresa_glave. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
60         tekuci);

62 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj.
    Azurira pokazivac na glavu liste, koji može biti promenjen u
64     slucaju da se obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
66         broj);

68 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na cinjenicu da je prosledjena lista neopadajuće
70     sortirana. Azurira pokazivac na glavu liste, koji može biti
    promenjen ukoliko se obrise stara glava liste. */
72 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
    adresa_kraja, int broj);
74
/* Funkcija prikazuje vrednosti cvorova liste počev od glave ka
76     kraju liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);
78
/* Funkcija prikazuje vrednosti cvorova liste počevši od kraja ka
80     glavi liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu_unazad(Cvor * kraj);
82
#endif
```

*dvostruko\_povezana\_lista.c*

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"

5  Cvor *napravi_cvor(int broj)
   {
7      /* Alokacija memorije za novi cvor uz proveru uspesnosti */
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9      if (novi == NULL)
          return NULL;

11
      /* Inicijalizacija polja strukture */
13      novi->vrednost = broj;
      novi->sledeci = NULL;

15
      /* Vraca se adresa novog cvora */
17      return novi;
   }
```

```
19 void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
21 {
22     Cvor *pomocni = NULL;
23
24     /* Ako lista nije prazna, onda treba osloboditi memoriju */
25     while (*adresa_glave != NULL) {
26         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
27            osloboditi memoriju cvora koji predstavlja glavu liste */
28         pomocni = (*adresa_glave)->sledeci;
29         free(*adresa_glave);
30         /* Sledeci cvor je nova glava liste */
31         *adresa_glave = pomocni;
32     }
33     /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj
34        liste treba postaviti na NULL */
35     *adresa_kraja = NULL;
36 }
37
38 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
39                             adresa_kraja, int broj)
40 {
41     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
42     Cvor *novi = napravi_cvor(broj);
43     if (novi == NULL)
44         return 1;
45
46     /* Sledbenik novog cvora je glava stare liste */
47     novi->sledeci = *adresa_glave;
48
49     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
50        treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
51        krajnji */
52     if (*adresa_glave != NULL)
53         (*adresa_glave)->prethodni = novi;
54     else
55         *adresa_kraja = novi;
56
57     /* Novi cvor je nova glava liste */
58     *adresa_glave = novi;
59
60     /* Vraca se indikator uspesnog dodavanja */
61     return 0;
62 }
63
64 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
65                         int broj)
66 {
67     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
68     Cvor *novi = napravi_cvor(broj);
69     if (novi == NULL)
70         return 1;
```

```
71  /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
73     ujedno i cela lista. Azurira se vrednost na koju pokazuju
        adresa_glave i adresa_kraja */
75  if (*adresa_glave == NULL) {
76      *adresa_glave = novi;
77      *adresa_kraja = novi;
78  } else {
79      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
        sledbenik poslednjeg cvora i azurira se samo pokazivac na
81     kraj liste */
        (*adresa_kraja)->sledeci = novi;
83     novi->prethodni = (*adresa_kraja);
        *adresa_kraja = novi;
85  }

87  /* Vraca se indikator uspesnog dodavanja */
    return 0;
89  }

91  Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
    {
93      /* U praznoj listi nema takvog mesta i vraca se NULL */
        if (glava == NULL)
95            return NULL;

97      /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
        pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
99     vrednost vecu ili jednaku od vrednosti novog cvora.*/
        /* Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
101     provera da li se doslo do poslednjeg cvora liste pre nego sto
        se proveru vrednost u sledecem cvoru jer u slucaju poslednjeg,
103     sledeci ne postoji pa ni njegova vrednost. */
        while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
105            glava = glava->sledeci;

107      /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
        poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci
109     ima vrednost vecu od broj */
        return glava;
111  }

113  int dodaj_iza(Cvor * tekuci, int broj)
    {
115      /* Kreira se novi cvor i proverava se uspesnost kreiranja */
        Cvor *novi = napravi_cvor(broj);
117        if (novi == NULL)
            return 1;

119        novi->sledeci = tekuci->sledeci;
121        novi->prethodni = tekuci;
```

```

123  /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
125      a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
      na ispravne adrese */
127      if (tekuci->sledeci != NULL)
          tekuci->sledeci->prethodni = novi;
          tekuci->sledeci = novi;

129
      /* Vraca se indikator uspesnog dodavanja */
131      return 0;
  }

133  int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
135                      broj)
  {
137      /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor */
      if (*adresa_glave == NULL) {
139          /* Kreira se novi cvor i proverava se uspesnost kreiranja */
          Cvor *novi = napravi_cvor(broj);
141          if (novi == NULL)
              return 1;

143          /* Azuriraju se vrednosti pocetka i kraja liste */
          *adresa_glave = novi;
          *adresa_kraja = novi;

145          /* Vraca se indikator uspesnog dodavanja */
          return 0;
147      }

149      /* Ukoliko je vrednost glave liste veca ili jednaka od nove
151          vrednosti onda novi cvor treba staviti na pocetak liste */
      if ((*adresa_glave)->vrednost >= broj) {
153          return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
      }

155
      /* Pronazi se cvor iza koga treba uvezati novi cvor */
157      Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
      /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
159      if (dodaj_iza(pomocni, broj) == 1)
          return 1;

161      /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
          novi cvor poslednji u listi. */
163      if (pomocni == *adresa_kraja)
          *adresa_kraja = pomocni->sledeci;

165      return 0;
167  }

169  Cvor *pretrazi_listu(Cvor * glava, int broj)
  {
171      /* Obilaze se cvorovi liste */
      for (; glava != NULL; glava = glava->sledeci)

```

```
175     /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
176         se obustavlja */
177     if (glava->vrednost == broj)
178         return glava;
179
180     /* Nema trazenog broja u listi i vraca se NULL */
181     return NULL;
182 }
183
184 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
185 {
186     /* Obilaze se cvorovi liste */
187     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
188     for (; glava != NULL && glava->vrednost <= broj;
189         glava = glava->sledeci)
190     /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
191         se obustavlja */
192         if (glava->vrednost == broj)
193             return glava;
194
195     /* Nema trazenog broja u listi i bice vrateno NULL */
196     return NULL;
197 }
198
199 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
200     lako realizovati jer on sadrzi pokazivace na svog sledbenika i
201     prethodnika u listi. U funkciji se bise cvor zadat argumentom
202     tekuci */
203 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
204     tekuci)
205 {
206     /* Ako je tekuci NULL pokazivac, nema potrebe za brisanjem */
207     if (tekuci == NULL)
208         return;
209
210     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
211         njegov sledbenik bude sledbenik tekuceg cvora */
212     if (tekuci->prethodni != NULL)
213         tekuci->prethodni->sledeci = tekuci->sledeci;
214
215     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
216         treba da bude prethodnik tekuceg cvora */
217     if (tekuci->sledeci != NULL)
218         tekuci->sledeci->prethodni = tekuci->prethodni;
219
220     /* Ako je glava cvor koji se brise, nova glava liste ce biti
221         sledbenik stare glave */
222     if (tekuci == *adresa_glave)
223         *adresa_glave = tekuci->sledeci;
224
225     /* Ako je cvor koji se brise poslednji u listi, azurira se i
226         pokazivac na kraj liste */
```



```

227     if (tekuci == *adresa_kraja)
        *adresa_kraja = tekuci->prethodni;
229
        /* Oslobadja se dinamički alocirani prostor za cvor tekuci */
231     free(tekuci);
    }
233
    void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja,
235                    int broj)
    {
237         Cvor *tekuci = *adresa_glave;

239         /* Sve dok ima cvorova čija je vrednost jednaka zadatom broju,
            takvi cvorovi se brišu iz liste. */
241         while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
            obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
243     }

245     void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                       adresa_kraja, int broj)
247     {
        Cvor *tekuci = *adresa_glave;
249

        /* Sve dok ima cvorova čija je vrednost jednaka zadatom broju,
            takvi cvorovi se brišu iz liste. */
251         while ((tekuci =
253                 pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
            obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
255     }

257     void ispisi_listu(Cvor * glava)
    {
259         putchar('[');
        /* Unutar zagrade ispisuju se vrednosti u cvorovima liste od
261            početka prema kraju liste */
        for (; glava != NULL; glava = glava->sledeci) {
263             printf("%d", glava->vrednost);
            if (glava->sledeci != NULL)
265                 printf(", ");
        }

267         printf("]\n");
269     }

271     void ispisi_listu_unazad(Cvor * kraj)
    {
273         putchar('[');
        /* Unutar zagrade ispisuju se vrednosti u cvorovima liste od kraja
275            prema početku liste */
        for (; kraj != NULL; kraj = kraj->prethodni) {
277             printf("%d", kraj->vrednost);
            if (kraj->prethodni != NULL)

```

## 4 Dinamičke strukture podataka

```
279     printf(", ");
    }
281     printf("]\n");
}
```

*main\_a.c*

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

4
/* 1) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
10     da bi operacije poput dodavanja na kraj liste i ispisivanja
        liste unazad bile efikasne poput dodavanja na pocetak liste i
12     ispisivanja liste od pocetnog do poslednjeg cvora. */
    Cvor *glava = NULL;
14     Cvor *kraj = NULL;
    Cvor *trazeni = NULL;
16     int broj;

18     /* Testira se funkcija za dodavanje novog broja na pocetak liste */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
20     while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
22         memorije za novi cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa */
        if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
24             fprintf(stderr,
26                 "Greska: Neuspesna alokacija memorije za cvor.\n");
                oslobodi_listu(&glava, &kraj);
28                 exit(EXIT_FAILURE);
            }
30             printf("\tLista: ");
                ispisi_listu(glava);
32         }

34     /* Testira se funkcija za pretragu liste */
    printf("\nUnesite broj koji se trazi u listi: ");
36     scanf("%d", &broj);

38     /* Pokazivac trazeni dobija vrednost rezultata pretrage */
    trazeni = pretrazi_listu(glava, broj);
40     if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
42     else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
44 }
```

```

46  /* Ispisuje se lista unazad */
    printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);
48
    /* Oslobadja se memorija zauzeta za cvorove liste */
50  oslobodi_listu(&glava, &kraj);
52  exit(EXIT_SUCCESS);
}

```

main\_b.c

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"

5  /* 2) Glavni program */
   int main()
7  {
   /* Lista je prazna na pocetku. */
9   Cvor *glava = NULL;
   Cvor *kraj = NULL;
11  int broj;

13  /* Testira se funkcija za dodavanje novog broja na kraj liste */
   printf("Unesite brojeve (CTRL+D za kraj unosa): ");
15  while (scanf("%d", &broj) > 0) {
   /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17   memorije za novi cvor. Memoriju alociranu za cvorove liste
   treba osloboditi pre napustanja programa */
19   if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
       fprintf(stderr,
21         "Greska: Neuspesna alokacija memorije za cvor.\n");
       oslobodi_listu(&glava, &kraj);
23       exit(EXIT_FAILURE);
   }
25   printf("\tLista: ");
   ispisi_listu(glava);
27 }

29 /* Testira se funkcija za brisanje elemenata iz liste */
   printf("\nUnesite broj koji se brise iz liste: ");
31  scanf("%d", &broj);

33  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
   procitanom sa ulaza */
35  obrisi_cvor(&glava, &kraj, broj);

37  printf("Lista nakon brisanja: ");
   ispisi_listu(glava);
39

```

## 4 Dinamičke strukture podataka

---

```
/* Ispisuje se lista unazad */
41 printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);

43

/* Oslobadja se memorija zauzeta za cvorove liste */
45 oslobodi_listu(&glava, &kraj);

47 exit(EXIT_SUCCESS);
}
```

*main.c.c*

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

4

/* 3) Glavni program */
6 int main()
{
8     /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
    Cvor *kraj = NULL;
10    Cvor *trazeni = NULL;
12    int broj;

14    /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
       bude uredjena neopadajuće */
16    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
    while (scanf("%d", &broj) > 0) {
18        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za novi cvor. Memoriju alociranu za cvorove liste
           treba osloboditi pre napustanja programa */
20        if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
22            fprintf(stderr,
                "Greska: Neuspesna alokacija memorije za cvor.\n");
24            oslobodi_listu(&glava, &kraj);
            exit(EXIT_FAILURE);
26        }
        printf("\tLista: ");
28        ispisi_listu(glava);
    }

30

    /* Testira se funkcija za pretragu liste */
32    printf("\nUnesite broj koji se trazi u listi: ");
    scanf("%d", &broj);

34

    /* Pokazivac trazeni dobija vrednost rezultata pretrage */
36    trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
38        printf("Broj %d se ne nalazi u listi!\n", broj);
    else
```

```

40     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

42     /* Testira se funkcija za brisanje elemenata iz liste */
    printf("\nUnesite broj koji se brise iz liste: ");
44     scanf("%d", &broj);

46     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza */
48     obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

50     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);
52

54     /* Ispisuje se lista unazad */
    printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(kraj);
56

58     /* Oslobadja se memorija zauzeta za cvorove liste */
    oslobodi_listu(&glava, &kraj);

60     exit(EXIT_SUCCESS);
}

```

## Rešenje 4.14

*stabla.h*

```

1  #ifndef _STABLA_H_
2  #define _STABLA_H_ 1

4  /* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
    stabla */
6  typedef struct cvor {
    int broj;
8     struct cvor *levo;
    struct cvor *desno;
10 } Cvor;

12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
    inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
    funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je
18    doslo do greske */
    int dodaj_u_stablo(Cvor ** adresa_korena, int broj);
20

22 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
    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 najveću vrednost u
    stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
void obrisi_element(Cvor ** adresa_korena, int broj);

34

36 /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
    postablo - Koren - Desno podstablo ) */
void ispisi_stablo_infiksno(Cvor * koren);

38

40 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
    Levo podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren);

42

44 /* j) Funkcija koja ispisuje stablo u postfiksnoj notaciji ( Levo
    podstablo - Desno postablo - Koren) */
void ispisi_stablo_postfiksno(Cvor * koren);

46

48 /* k) Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena);

50 #endif
```

*stabla.c*

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

4

Cvor *napravi_cvor(int broj)
6 {
    /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    10 if (novi == NULL)
        return NULL;

    12

    /* Inicijalizuju se polja novog cvora. */
    14 novi->broj = broj;
    novi->levo = NULL;
    16 novi->desno = NULL;

    18

    /* Vraca se adresa novog cvora. */
    return novi;
    20 }
```

```
22 int dodaj_u_stablo(Cvor ** adresa_korena, int broj)
23 {
24     /* Ako je stablo prazno */
25     if (*adresa_korena == NULL) {
26
27         /* Kreira se novi cvor */
28         Cvor *novi_cvor = napravi_cvor(broj);
29
30         /* Proverava se uspesnost kreiranja */
31         if (novi_cvor == NULL) {
32
33             /* Ukoliko je doslo do greske, vraca se odgovarajuca vrednost */
34             return 1;
35         }
36         /* Inace ... */
37         /* Novi cvor se proglašava korenom stabla */
38         *adresa_korena = novi_cvor;
39
40         /* I vraca se indikator uspesnosti kreiranja */
41         return 0;
42     }
43
44     /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za
45        zadati broj */
46
47     /* Ako je zadata vrednost manja od vrednosti korena */
48     if (broj < (*adresa_korena)->broj)
49
50         /* Broj se dodaje u levo podstablo */
51         return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
52
53     else
54         /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
55            dodaje u desno podstablo */
56         return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
57 }
58
59 Cvor *pretrazi_stablo(Cvor * koren, int broj)
60 {
61
62     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
63     if (koren == NULL)
64         return NULL;
65
66     /* Ako je trazena vrednost sadrzana u korenu */
67     if (koren->broj == broj) {
68
69         /* Prekida se pretraga */
70         return koren;
71     }
72 }
```

```
74  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
    if (broj < koren->broj)

76      /* Pretraga se nastavlja u levom podstablu */
      return pretrazi_stablo(koren->levo, broj);

78  else
80      /* U suprotnom, pretraga se nastavlja u desnom podstablu */
      return pretrazi_stablo(koren->desno, broj);
82  }

84  Cvor *pronadji_najmanji(Cvor * koren)
  {
86
88      /* Ako je stablo prazno, prekida se pretraga */
      if (koren == NULL)
          return NULL;

90
92      /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
          levo od njega */

94      /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
          najmanju vrednost */
      if (koren->levo == NULL)
          return koren;

98
100     /* Inace, pretragu treba nastaviti u levom podstablu */
    return pronadji_najmanji(koren->levo);
102  }

104  Cvor *pronadji_najveci(Cvor * koren)
  {
106
108     /* Ako je stablo prazno, prekida se pretraga */
    if (koren == NULL)
        return NULL;

110
112     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
        desno od njega */

114     /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
        najveću vrednost */
    if (koren->desno == NULL)
        return koren;

116
118     /* Inace, pretragu treba nastaviti u desnom podstablu */
    return pronadji_najveci(koren->desno);
120  }

122  void obrisi_element(Cvor ** adresa_korena, int broj)
  {
124      Cvor *pomocni_cvor = NULL;
```



```
126  /* Ako je stablo prazno, brisanje nije primenljivo */
127  if (*adresa_korena == NULL)
128      return;
129
130  /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
131     stabla, ona se eventualno nalazi u levom podstablu, pa treba
132     rekursivno primeniti postupak na levo podstablo. Koren ovako
133     modifikovanog stabla je nepromenjen. */
134  if (broj < (*adresa_korena)->broj) {
135      obrisi_element(&(*adresa_korena)->levo, broj);
136      return;
137  }
138
139  /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
140     stabla, ona se eventualno nalazi u desnom podstablu pa treba
141     rekursivno primeniti postupak na desno podstablo. Koren ovako
142     modifikovanog stabla je nepromenjen. */
143  if ((*adresa_korena)->broj < broj) {
144      obrisi_element(&(*adresa_korena)->desno, broj);
145      return;
146  }
147
148  /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
149     jednaka broju koji se brise (tj. slucaj kada treba obrisati
150     koren) */
151
152  /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
153     prazno stablo (vraca se NULL) */
154  if ((*adresa_korena)->levo == NULL
155      && (*adresa_korena)->desno == NULL) {
156      free(*adresa_korena);
157      *adresa_korena = NULL;
158      return;
159  }
160
161  /* Ako koren ima samo levog sina, tada se brisanje vrši tako sto
162     se brise koren, a novi koren postaje levi sin */
163  if ((*adresa_korena)->levo != NULL
164      && (*adresa_korena)->desno == NULL) {
165      pomocni_cvor = (*adresa_korena)->levo;
166      free(*adresa_korena);
167      *adresa_korena = pomocni_cvor;
168      return;
169  }
170
171  /* Ako koren ima samo desnog sina, tada se brisanje vrši tako sto
172     se brise koren, a novi koren postaje desni sin */
173  if ((*adresa_korena)->desno != NULL
174      && (*adresa_korena)->levo == NULL) {
175      pomocni_cvor = (*adresa_korena)->desno;
176      free(*adresa_korena);
177      *adresa_korena = pomocni_cvor;
```

```

178     return;
179 }
180 /* Slučaj kada koren ima oba sina - najpre se potraži sledbenik
181 korena (u smislu poretka) u stablu. To je upravo po vrednosti
182 najmanji cvor u desnom podstablu. On se može pronaći npr.
183 funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
184 vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
185 broj koji se briše). Zatim se prosto rekurzivno pozove funkcija
186 za brisanje na desno podstablu. S obzirom da u njemu treba
187 obrisati najmanji element, a on zasigurno ima najviše jednog
188 potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
189 od jednostavnijih načina koji su gore opisani. */
190 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
191 (*adresa_korena)->broj = pomocni_cvor->broj;
192 pomocni_cvor->broj = broj;
193 obrisni_element(&(*adresa_korena)->desno, broj);
194 }
195
196 void ispisi_stablo_infiksno(Cvor * koren)
197 {
198     /* Ako stablo nije prazno */
199     if (koren != NULL) {
200
201         /* Prvo se ispisuju svi cvorovi levo od korena */
202         ispisi_stablo_infiksno(koren->levo);
203
204         /* Zatim se ispisuje vrednost u korenu */
205         printf("%d ", koren->broj);
206
207         /* Na kraju se ispisuju cvorovi desno od korena */
208         ispisi_stablo_infiksno(koren->desno);
209     }
210 }
211
212 void ispisi_stablo_prefiksno(Cvor * koren)
213 {
214     /* Ako stablo nije prazno */
215     if (koren != NULL) {
216
217         /* Prvo se ispisuje vrednost u korenu */
218         printf("%d ", koren->broj);
219
220         /* Zatim se ispisuju svi cvorovi levo od korena */
221         ispisi_stablo_prefiksno(koren->levo);
222
223         /* Na kraju se ispisuju svi cvorovi desno od korena */
224         ispisi_stablo_prefiksno(koren->desno);
225     }
226 }
227
228 void ispisi_stablo_postfiksno(Cvor * koren)

```

```

230 {
231     /* Ako stablo nije prazno */
232     if (koren != NULL) {
233
234         /* Prvo se ispisuju svi cvorovi levo od korena */
235         ispisi_stablo_postfiksno(koren->levo);
236
237         /* Zatim se ispisuju svi cvorovi desno od korena */
238         ispisi_stablo_postfiksno(koren->desno);
239
240         /* Na kraju se ispisuje vrednost u korenu */
241         printf("%d ", koren->broj);
242     }
243 }
244
245 void oslobodi_stablo(Cvor ** adresa_korena)
246 {
247     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
248     if (*adresa_korena == NULL)
249         return;
250
251     /* Inace ... */
252     /* Oslobadja se memorija zauzeta levim podstablom */
253     oslobodi_stablo(&(*adresa_korena)->levo);
254
255     /* Oslobadja se memorija zauzeta desnim podstablom */
256     oslobodi_stablo(&(*adresa_korena)->desno);
257
258     /* Oslobadja se memorija zauzeta korenom */
259     free(*adresa_korena);
260
261     /* Proglasava se stablo praznim */
262     *adresa_korena = NULL;
263 }

```

*main.c*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "stabla.h"
4
5  int main()
6  {
7      Cvor *koren;
8      int n;
9      Cvor *trazeni_cvor;
10
11     /* Proglasava se stablo praznim */
12     koren = NULL;
13

```

```

15  /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
    dodavanja */
printf("Unesite brojeve (CTRL+D za kraj unosa): ");
17  while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&koren, n) == 1) {
19      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
        oslobodi_stablo(&koren);
21      exit(EXIT_FAILURE);
    }
23  }

25  /* Generisu se trazeni ispisi: */
printf("\nInfiksni ispisi: ");
27  ispisi_stablo_infiksno(koren);
printf("\nPrefiksni ispisi: ");
29  ispisi_stablo_prefiksno(koren);
printf("\nPostfiksni ispisi: ");
31  ispisi_stablo_postfiksno(koren);

33  /* Demonstrira se rad funkcije za pretragu */
printf("\nTrazi se broj: ");
35  scanf("%d", &n);
trazeni_cvor = pretrazi_stablo(koren, n);
37  if (trazeni_cvor == NULL)
    printf("Broj se ne nalazi u stablu!\n");
39
    else
41      printf("Broj se nalazi u stablu!\n");

43  /* Demonstrira se rad funkcije za brisanje */
printf("Brise se broj: ");
45  scanf("%d", &n);
obrisi_element(&koren, n);
47  printf("Rezultujuce stablo: ");
ispisi_stablo_infiksno(koren);
49  printf("\n");

51  /* Oslobadja se memorija zauzeta stablom */
oslobodi_stablo(&koren);
53
    exit(EXIT_SUCCESS);
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 {
   char *rec;
12  int brojac;
   struct cvor *levo;
14  struct cvor *desno;
} Cvor;

16
/* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec)
{
20  /* Alocira se memorija za novi cvor i proverava se uspesnost
   alokacije. */
22  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
   if (novi_cvor == NULL)
24      return NULL;

26  /* Alocira se memorija za zadatu rec: potrebno je rezervisati
   memoriju za svaki karakter reci ukljucujuci i terminirajucu
   nulu */
28  novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
   if (novi_cvor->rec == NULL) {
30      free(novi_cvor);
32      return NULL;
   }

34
   /* Inicijalizuju se polja u novom cvoru */
36  strcpy(novi_cvor->rec, rec);
   novi_cvor->brojac = 1;
38  novi_cvor->levo = NULL;
   novi_cvor->desno = NULL;
40

   /* Vraca se adresa novog cvora */
42  return novi_cvor;
}

44
/* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
46  uspesno povratna vrednost je 0, u suprotnom povratna vrednost je
   1 */
48 int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
{
50  /* Ako je stablo prazno */
   if (*adresa_korena == NULL) {
52      /* Kreira se cvor koji sadrzi zadatu rec */
       Cvor *novi_cvor = napravi_cvor(rec);
54      /* Proverava se uspesnost kreiranja novog cvora */
       if (novi_cvor == NULL) {
56          /* I ukoliko je doslo do greske, vraca se odgovarajuca
           vrednost */
58          return 1;

```

```

    }
60    /* Inace... */
    /* Novi cvor se proglašava korenom stabla */
62    *adresa_korena = novi_cvor;

    /* I vraća se indikator uspešnog dodavanja */
    return 0;
66 }

68 /* Ako stablo nije prazno, traži se odgovarajuća pozicija za novu
    rec */

70 /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
    levo podstablo */
72 if (strcmp(rec, (*adresa_korena)->rec) < 0)
74     return dodaj_u_stablo(&(*adresa_korena)->levo, rec);

76 else
    /* Ako je rec leksikografski veća od reci u korenu ubacuje se u
    desno podstablo */
78 if (strcmp(rec, (*adresa_korena)->rec) > 0)
80     return dodaj_u_stablo(&(*adresa_korena)->desno, rec);

82 else {
    /* Ako je rec jednaka reci u korenu, uvećava se njen broj
    pojavljivanja */
84     (*adresa_korena)->brojac++;

86     /* I vraća se indikator uspešnog dodavanja */
88     return 0;
    }
90 }

92 /* Funkcija koja oslobađa memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
94 {
    /* Ako je stablo prazno, nepotrebno je oslobađati memoriju */
96     if (*adresa_korena == NULL)
        return;

98     /* Inace ... */
100    /* Oslobađa se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);

102    /* Oslobađa se memorija zauzeta desnim podstablom */
104    oslobodi_stablo(&(*adresa_korena)->desno);

106    /* Oslobađa se memorija zauzeta korenom */
    free((*adresa_korena)->rec);
108    free(*adresa_korena);

110    /* Stablo se proglašava praznim */

```

```

112     *adresa_korena = NULL;
113 }
114 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
115    sa najvećim brojem pojavljivanja) */
116 Cvor *nadjij_najfrekventniju_rec(Cvor * koren)
117 {
118     Cvor *max, *max_levo, *max_desno;
119
120     /* Ako je stablo prazno, prekida se sa pretragom */
121     if (koren == NULL)
122         return NULL;
123
124     /* Pronalazi se najfrekventnija rec u levom podstablu */
125     max_levo = nadjij_najfrekventniju_rec(koren->levo);
126
127     /* Pronalazi se najfrekventnija rec u desnom podstablu */
128     max_desno = nadjij_najfrekventniju_rec(koren->desno);
129
130     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
131        podstabla, korena i desnog podstabla */
132     max = koren;
133     if (max_levo != NULL && max_levo->brojac > max->brojac)
134         max = max_levo;
135     if (max_desno != NULL && max_desno->brojac > max->brojac)
136         max = max_desno;
137
138     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
139     return max;
140 }
141
142 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
143    pracen brojem pojavljivanja */
144 void prikazi_stablo(Cvor * koren)
145 {
146     /* Ako je stablo prazno, završava se sa ispisom */
147     if (koren == NULL)
148         return;
149
150     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz
151        levog podstabla */
152     prikazi_stablo(koren->levo);
153
154     /* Zatim rec iz korena */
155     printf("%s: %d\n", koren->rec, koren->brojac);
156
157     /* I nastavlja se sa ispisom reci iz desnog podstabla */
158     prikazi_stablo(koren->desno);
159 }
160
161 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
162    niz rec. Maksimalna dužina reci je odredjena argumentom max.

```

```

164     Funkcija vraća EOF ako u datoteci nema više reci ili 0 u
        suprotnom. Rec je niz malih ili velikih slova. */
166 int procitaj_rec(FILE * f, char rec[], int max)
167 {
168     /* Karakter koji se cita */
169     int c;
170
171     /* Indeks pozicije na koju se smesta procitani karakter */
172     int i = 0;
173
174     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
        nije stiglo do kraja datoteke... */
175     while (i < max - 1 && (c = fgetc(f)) != EOF) {
176         /* Proverava se da li je procitani karakter slovo */
177         if (isalpha(c))
178             /* Ako jeste, smesta se u niz - pritom se vrši konverzija u
                mala slova jer program treba da bude neosetljiv na razliku
                izmedju malih i velikih slova */
179             rec[i++] = tolower(c);
180
181         else
182             /* Ako nije, proverava se da li je procitano barem jedno slovo
                nove reci */
183             /* Ako jeste, prekida se sa citanjem */
184             if (i > 0)
185                 break;
186
187         /* U suprotnom se ide na sledecu iteraciju */
188     }
189
190     /* Dodaje se na rec terminirajuca nula */
191     rec[i] = '\0';
192
193     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
194     return i > 0 ? 0 : EOF;
195 }
196
197 int main(int argc, char **argv)
198 {
199     Cvor *koren = NULL, *max;
200     FILE *f;
201     char rec[MAX];
202
203     /* Provera da li je navedeno ime datoteke prilikom pokretanja
        programa */
204     if (argc < 2) {
205         fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
206         exit(EXIT_FAILURE);
207     }
208
209     /* Priprema datoteke za citanje */
210     if ((f = fopen(argv[1], "r")) == NULL) {

```



```

216     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
        argv[1]);
218     exit(EXIT_FAILURE);
}

220 /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
    pretrage uz proveru uspesnosti dodavanja */
222 while (procitaj_rec(f, rec, MAX) != EOF) {
    if (dodaj_u_stablo(&koren, rec) == 1) {
224         fprintf(stderr, "Greska: Neuspelo dodavanje reci %s.\n", rec);
        oslobodi_stablo(&koren);
226         exit(EXIT_FAILURE);
    }
228 }

230 /* Posto je citanje reci zavrшено, zatvara se datoteka */
fclose(f);

232
234 /* Prikazuju se sve reci iz teksta i brojevi njihovih
    pojavljivanja. */
prikazi_stablo(koren);

236
238 /* Pronalazi se najfrekventnija rec */
max = nadji_najfrekventniju_rec(koren);

240
242 /* Ako takve reci nema... */
if (max == NULL)

    /* Ispisuje se odgovarajuće obavestjenje */
    printf("U tekstu nema reci!\n");

244
246 else
    /* Inace, ispisuje se broj pojavljivanja reci */
    printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
        max->rec, max->brojac);

250
252 /* Oslobadja se dinamički alociran prostor za stablo */
oslobodi_stablo(&koren);

254 exit(EXIT_SUCCESS);
}

```

### Rešenje 4.16

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX_IME_DATOTEKE 50
7  #define MAX_CIFARA 13

```

```
9      #define MAX_IME_I_PREZIME 100
11
12     /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
13        broj telefona i redom pokazuje na levo i desno podstablo */
14     typedef struct cvor {
15         char ime_i_prezime[MAX_IME_I_PREZIME];
16         char telefon[MAX_CIFARA];
17         struct cvor *levo;
18         struct cvor *desno;
19     } Cvor;
20
21     /* Funkcija koja kreira novi cvor stabla */
22     Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
23     {
24         /* Alocira se memorija za novi cvor i proverava se uspesnost
25            alokacije. */
26         Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
27         if (novi_cvor == NULL)
28             return NULL;
29
30         /* Inicijalizuju se polja novog cvora */
31         strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
32         strcpy(novi_cvor->telefon, telefon);
33         novi_cvor->levo = NULL;
34         novi_cvor->desno = NULL;
35
36         /* Vraca se adresa novog cvora */
37         return novi_cvor;
38     }
39
40     /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
41        ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
42        povratna vrednost je 1 */
43     int
44     dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
45                   char *telefon)
46     {
47         /* Ako je stablo prazno */
48         if (*adresa_korena == NULL) {
49             /* Kreira se novi cvor */
50             Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
51             /* Proverava se uspesnost kreiranja novog cvora */
52             if (novi_cvor == NULL) {
53                 /* I ukoliko je doslo do greske, vraca se odgovarajuca
54                    vrednost */
55                 return 1;
56             }
57             /* Inace... */
58             /* Novi cvor se proglašava korenom stabla */
59             *adresa_korena = novi_cvor;
60
61             /* I vraca se indikator uspesnog dodavanja */
62         }
```

```

    return 0;
61 }

63 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
    unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
65 treba da bude pretrazivacko po ovom polju */

67 /* Ako je zadato ime i prezime leksikografski manje od imena i
    prezimena koje se nalazi u korenu, podaci se dodaju u levo
69 podstablo */
    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
71         < 0)
        return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
73                                telefon);

75 else
    /* Ako je zadato ime i prezime leksikografski vece od imena i
77    prezimena sadrzanog u korenu, podaci se dodaju u desno
    podstablo */
    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
79        return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
81                                telefon);

83 /* Pretostavka zadatka je da nema istih imena i prezimena u
    datoteci, pa se sledeca naredba nikada i neci izvršiti */
85 return 0;
}

87 /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
{
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
93        return;

95    /* Inace ... */
    /* Oslobadja se memorija zauzeta levim podstablom */
97    oslobodi_stablo(&(*adresa_korena)->levo);

    /* Oslobadja se memorija zauzeta desnim podstablom */
    oslobodi_stablo(&(*adresa_korena)->desno);
101

    /* Oslobadja se memorija zauzeta korenom */
103    free(*adresa_korena);

105    /* Stablo se proglašava praznim */
    *adresa_korena = NULL;
107 }

109 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
/* Napomena: ova funkcija nije trazena u zadatku ali se moze
111 koristiti za proveru da li je stablo lepo kreirano ili ne */

```

```

void prikazi_stablo(Cvor * koren)
113 {
    /* Ako je stablo prazno, završava se sa ispisom */
115     if (koren == NULL)
        return;
117
    /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
119     podstabla */
    prikazi_stablo(koren->levo);
121
    /* Zatim se ispisuju podaci iz korena */
123     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

    /* I nastavlja se sa ispisom podataka iz desnog podstabla */
    prikazi_stablo(koren->desno);
127 }

129 /* Funkcija učitava sledeći kontakt iz zadate datoteke i upisuje ime
    i prezime i broj telefona u odgovarajuće nizove. Maksimalna dužina
131 imena i prezimena određena je konstantom MAX_IME_PREZIME, a
    maksimalna dužina broja telefona konstantom MAX_CIFARA. Funkcija
133 vraća EOF ako nema više kontakata ili 0 u suprotnom. */
int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
135 {
    /* Karakter koji se čita */
137     int c;

    /* Indeks pozicije na koju se smesta procitani karakter */
139     int i = 0;

    /* Linije datoteke koje se obrađuju su formata Ime Prezime
141     BrojTelefona */
143
    /* Preskaku se eventualne praznine sa početka linije datoteke */
145     while ((c = fgetc(f)) != EOF && isspace(c));
147
    /* Prvo procitano slovo upisuje se u ime i prezime */
149     if (!feof(f))
        ime_i_prezime[i++] = c;
151
    /* Naznaka kraja citanja imena i prezimena će biti pojava prve
153 cifre tako da se citanje vrši sve dok se ne nađe na cifru.
    Pritom treba voditi računa da li ima dovoljno mesta za
155 smestanje procitanog karaktera i da se slučajno ne dodje do
    kraja datoteke */
157     while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
        if (!isdigit(c))
159             ime_i_prezime[i++] = c;

        else if (i > 0)
161             break;
163     }

```

```
165  /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
166     blanko karaktera */
167  ime_i_prezime[--i] = '\0';

169  /* I pocinje se sa citanjem broja telefona */
170  i = 0;

171
172  /* Upisuje se cifra koja je vec procitana */
173  telefon[i++] = c;

175  /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
176     karaktera cije prisustvo nije dozvoljeno u broju telefona */
177  while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
178      if (c == '/' || c == '-' || isdigit(c))
179          telefon[i++] = c;
180      else
181          break;

183  /* Upisuje se terminirajuca nula */
184  telefon[i] = '\0';

185
186  /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
187  return !feof(f) ? 0 : EOF;
188 }

189
190 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
191    prezimenom */
192 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
193 {
194     /* Ako je imenik prazan, zavrшава se sa pretragom */
195     if (koren == NULL)
196         return NULL;

197
198     /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
199        zavrшава sa pretragom */
200     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
201         return koren;

202
203     /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
204        korenu pretraga se nastavlja levo */
205     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
206         return pretrazi_imenik(koren->levo, ime_i_prezime);

207
208     else
209         /* U suprotnom, pretraga se nastavlja desno */
210         return pretrazi_imenik(koren->desno, ime_i_prezime);
211 }

212
213 int main(int argc, char **argv)
214 {
215     char ime_datoteke[MAX_IME_DATOTEKE];
```

```

217 Cvor *koren = NULL;
218 Cvor *trazeni;
219 FILE *f;
220 char ime_i_prezime[MAX_IME_I_PREZIME];
221 char telefon[MAX_CIFARA];
222 char c;
223 int i;

224 /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
225 printf("Unesite ime datoteke: ");
226 scanf("%s", ime_datoteke);
227 getchar();
228 if ((f = fopen(ime_datoteke, "r")) == NULL) {
229     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
230             ime_datoteke);
231     exit(EXIT_FAILURE);
232 }

233 /* Citaju se podaci iz datoteke i smestaju u binarno stablo
234    pretrage uz proveru uspesnosti dodavanja */
235 while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
236     if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
237         fprintf(stderr,
238                 "Greska: Neuspelo dodavanje podataka za osobu %s.\n",
239                 ime_i_prezime);
240         oslobodi_stablo(&koren);
241         exit(EXIT_FAILURE);
242     }

243 /* Zatvara se datoteka */
244 fclose(f);

245 /* Omogucava se pretraga imenika */
246 while (1) {
247     /* Ucitava se ime i prezime */
248     printf("Unesite ime i prezime: ");
249     i = 0;
250     while ((c = getchar()) != '\n')
251         ime_i_prezime[i++] = c;
252     ime_i_prezime[i] = '\0';

253     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
254        funkcionalnost */
255     if (strcmp(ime_i_prezime, "KRAJ") == 0)
256         break;

257     /* Inace se ispisuje rezultat pretrage */
258     trazeni = pretrazi_imenik(koren, ime_i_prezime);
259     if (trazeni == NULL)
260         printf("Broj nije u imeniku!\n");
261     else
262         printf("Broj je: %s \n", trazeni->telefon);

```

```

    }
269
    /* Oslobadja se memorija zauzeta imenikom */
271    oslobodi_stablo(&koren);
273
    exit(EXIT_SUCCESS);
}

```

### Rešenje 4.17

```

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

5  #define MAX 51

7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9  jezika i redom pokazivace na levo i desno podstablo */
   typedef struct cvor_stabla {
11     char ime[MAX];
       char prezime[MAX];
13     double uspeh;
       double matematika;
15     double jezik;
       struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
   } Cvor;
19

   /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
       double matematika, double jezik)
23 {
   /* Alocira se memorija za novi cvor i proverava se uspesnost
25     alokacije. */
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
       return NULL;
29

   /* Inicijalizuju se polja strukture */
31     strcpy(novi->ime, ime);
       strcpy(novi->prezime, prezime);
33     novi->uspeh = uspeh;
       novi->matematika = matematika;
35     novi->jezik = jezik;
       novi->levo = NULL;
37     novi->desno = NULL;

39     /* Vraca se adresa kreiranog cvora */
       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, vraća se odgovarajuća
57                vrednost */
58             return 1;
59         }
60         /* Inace... */
61         /* Novi cvor se proglašava korenom stabla */
62         *koren = novi_cvor;
63
64         /* I vraća se indikator uspešnog 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 }
78
79 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
80 void oslobodi_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     oslobodi_stablo(&(*koren)->levo);
89
90     /* Oslobadja se memorija zauzeta desnim podstablom */
91     oslobodi_stablo(&(*koren)->desno);
92 }
93

```



```
124      /* Oslobadja se memorija zauzeta korenom */
125      free(*koren);
126
127      /* Stablo se proglašava praznim */
128      *koren = NULL;
129  }
130
131  /* Funkcija ispisuje sadržaj stabla. Ukoliko je vrednost argumenta
132     položili jednaka 0 ispisuju se informacije o uenicima koji nisu
133     položili prijemni, a ako je vrednost argumenta razlicita od nule,
134     ispisuju se informacije o uenicima koji su položili prijemni */
135  void stampa(Cvor * koren, int položili)
136  {
137      /* Stablo je prazno - prekida se sa ispisom */
138      if (koren == NULL)
139          return;
140
141      /* Stampaju se informacije iz levog podstabla */
142      stampa(koren->levo, položili);
143
144      /* Stampaju se informacije iz korenog cvora */
145      if (položili && koren->matematika + koren->jezik >= 10)
146          printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
147                koren->prezime, koren->uspeh, koren->matematika,
148                koren->jezik,
149                koren->uspeh + koren->matematika + koren->jezik);
150      else if (!položili && koren->matematika + koren->jezik < 10)
151          printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
152                koren->prezime, koren->uspeh, koren->matematika,
153                koren->jezik,
154                koren->uspeh + koren->matematika + koren->jezik);
155
156      /* Stampaju se informacije iz desnog podstabla */
157      stampa(koren->desno, položili);
158  }
159
160  /* Funkcija koja određuje koliko studenata nije položilo prijemni
161     ispit */
162  int nisu_položili(Cvor * koren)
163  {
164      /* Ako je stablo prazno, broj onih koji nisu položili je 0 */
165      if (koren == NULL)
166          return 0;
167
168      /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov za
169         polaganje nije ispunjen za koreni cvor, broj studenata se
170         uvecava za 1 */
171      if (koren->matematika + koren->jezik < 10)
172          return 1 + nisu_položili(koren->levo) +
173                nisu_položili(koren->desno);
174  }
```

```

147     return nisu_položili(koren->levo) + nisu_položili(koren->desno);
148 }
149 int main(int argc, char **argv)
150 {
151     FILE *in;
152     Cvor *koren;
153     char ime[MAX], prezime[MAX];
154     double uspeh, matematika, jezik;
155
156     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
157     in = fopen("prijemni.txt", "r");
158     if (in == NULL) {
159         fprintf(stderr,
160             "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
161         exit(EXIT_FAILURE);
162     }
163
164     /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
165        dodavanja */
166     koren = NULL;
167     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
168                 &matematika, &jezik) != EOF) {
169         if (dodaj_u_stablo
170             (&koren, ime, prezime, uspeh, matematika, jezik)
171             == 1) {
172             fprintf(stderr,
173                 "Greska: Neuspelo dodavanje podataka za %s %s.\n", ime,
174                 prezime);
175             oslobodi_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 položili prijemni */
184     stampaj(koren, 1);
185
186     /* Linija se iscrtava samo ako postoje ucenici koji nisu položili
187        prijemni */
188     if (nisu_položili(koren) != 0)
189         printf("-----\n");
190
191     /* Stampaju se podaci o ucenicima koji nisu položili prijemni */
192     stampaj(koren, 0);
193
194     /* Oslobadja se memorija zauzeta stablom */
195     oslobodi_stablo(&koren);
196
197     exit(EXIT_SUCCESS);

```

```
}

```

### Rešenje 4.18

```

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

5  #define MAX_NISKA 51

7  /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
   osobe, dan i mesec rođenja i redom pokazivace na levo i desno
   podstablo */
9  typedef struct cvor_stabla {
11     char ime[MAX_NISKA];
   char prezime[MAX_NISKA];
13     int dan;
   int mesec;
15     struct cvor_stabla *levo;
   struct cvor_stabla *desno;
17 } Cvor;

19 /* Funkcija koja kreira novi cvor */
Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
   /* Alocira se memorija */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
25         return NULL;

27     /* Inicijalizuju se polja strukture */
   strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
   novi->dan = dan;
31     novi->mesec = mesec;
   novi->levo = NULL;
33     novi->desno = NULL;

35     /* Vraca se adresa novog cvora */
   return novi;
37 }

39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
   uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
   dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
   vrednost je 1 */
43 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
   int dan, int mesec)
45 {
   /* Ako je stablo prazno */
47     if (*koren == NULL) {

```

```

49      /* Kreira se novi cvor */
      Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51      /* Proverava se uspesnost kreiranja novog cvora */
      if (novi_cvor == NULL) {
53          /* I ukoliko je doslo do greske, vraca se odgovarajuca
              vrednost */
55          return 1;
      }
57      /* Inace... Novi cvor se proglašava korenom stabla */
      *koren = novi_cvor;

59
      /* I vraca se indikator uspesnog dodavanja */
61      return 0;
  }

63
  /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
     meseca */
65
  if (mesec < (*koren)->mesec)
67      return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
  else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
69      return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
  else
71      return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan,
                          mesec);
73 }

75 /* Funkcija vrši pretragu stabla i vraca cvor sa traženim datumom */
  Cvor *pretrazi(Cvor * koren, int dan, int mesec)
77 {
      /* Stablo je prazno, obustavlja se pretraga */
79      if (koren == NULL)
          return NULL;

81
      /* Ako je traženi datum u korenu */
83      if (koren->dan == dan && koren->mesec == mesec)
          return koren;

85
      /* Ako je mesec traženog datuma manji od meseca sadržanog u korenu
         ili ako su meseci isti ali je dan traženog datuma manji od
         aktuelnog datuma, pretražuje se levo podstablo - pre toga se
         svakako proverava da li leva grana postoji - ako ne postoji
         treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
         */
87
89      if (mesec < koren->mesec
          || (mesec == koren->mesec && dan < koren->dan)) {
91          if (koren->levo == NULL)
93              return koren;
          else
95              return pretrazi(koren->levo, dan, mesec);
97      }
  }

```

```

99  /* Inace se nastavlja pretraga u desnom delu */
    return pretrazi(koren->desno, dan, mesec);
101 }

103 /* Funkcija koja pronalazi najmanji datum u stablu */
    Cvor *pronadji_najmanji_datum(Cvor * koren)
105 {
    /* Stablo je prazno, obustavlja se pretraga */
107     if (koren == NULL)
        return NULL;

109     /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
        sadrzi najmanji datum */
111     if (koren->levo == NULL)
        return koren;
113     else
        /* Inace, trazimo manji datum u levom podstablu */
115         return pronadji_najmanji_datum(koren->levo);
117 }

119 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
    */
    void datum_u_nisku(int dan, int mesec, char datum[])
121 {
    if (dan < 10) {
123         datum[0] = '0';
        datum[1] = dan + '0';
125     } else {
        datum[0] = dan / 10 + '0';
127         datum[1] = dan % 10 + '0';
    }
129     datum[2] = '.';

131     if (mesec < 10) {
        datum[3] = '0';
133         datum[4] = mesec + '0';
    } else {
135         datum[3] = mesec / 10 + '0';
        datum[4] = mesec % 10 + '0';
137     }
    datum[5] = '.';
139     datum[6] = '\0';
    }

141 /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena)
143 {
    /* Stablo je prazno */
145     if (*adresa_korena == NULL)
        return;
147

149     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */

```

```
151     if ((*adresa_korena)->levo)
        oslobodi_stablo(&(*adresa_korena)->levo);

153     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
    if ((*adresa_korena)->desno)
155         oslobodi_stablo(&(*adresa_korena)->desno);

157     /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);

159     /* Proglasava se stablo praznim */
161     *adresa_korena = NULL;
}

163 int main(int argc, char **argv)
165 {
    FILE *in;
167     Cvor *koren;
    Cvor *slavljjenik;
169     char ime[MAX_NISKA], prezime[MAX_NISKA];
    int dan, mesec;
171     char datum[7];

173     /* Provera da li je zadato ime ulazne datoteke */
    if (argc < 2) {
175         /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
            programa */
177         fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
        exit(EXIT_FAILURE);
179     }

181     /* Inace, priprema se datoteka za citanje */
    in = fopen(argv[1], "r");
183     if (in == NULL) {
        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
185             argv[1]);
        exit(EXIT_FAILURE);
187     }

189     /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
        */
    koren = NULL;
191     while (fscanf
        (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
193         if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
            fprintf(stderr,
195                 "Greska: Neuspelo dodavanje podataka za %s %s.\n", ime,
                    prezime);
            oslobodi_stablo(&koren);
            exit(EXIT_FAILURE);
197         }
199 }
```

```
201  /* Datoteka se zatvara */
    fclose(in);
203
    /* Omogucuje se pretraga podataka */
205  while (1) {
207      /* Ucitava se novi datum */
      printf("Unesite datum: ");
209      if (scanf("%d.%d.", &dan, &mesec) == EOF)
          break;
211
      /* Pretrazuje se stablo */
213      slavljenik = pretrazi(koren, dan, mesec);
215
      /* Ispisuju se pronadjeni podaci */
217
      /* Ako slavljenik nije pronadjen, to moze znaci da: */
      /* 1. drvo je prazno */
219      if (slavljenik == NULL && koren == NULL) {
          printf("Nema podataka o ovom ni o sledecem rođjendanu.\n");
221          continue;
      }
223      /* 2. posle datuma koji je unesen, nema podataka u stablu - u
          ovom slucaju se pretraga vrši pocevši od naredne godine i
          ispisuje se najmanji datum */
225      if (slavljenik == NULL) {
227          slavljenik = pronadji_najmanji_datum(koren);
          datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
229          printf("Slavljenik: %s %s %s\n", slavljenik->ime,
                  slavljenik->prezime, datum);
231          continue;
      }
233
      /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
      /* 1. Pronadjeni su tacni podaci */
235      if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
237          printf("Slavljenik: %s %s\n", slavljenik->ime,
                  slavljenik->prezime);
239          continue;
      }
241
      /* 2. Pronadjeni su podaci o prvom sledecem rođjendanu */
243      datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
245          slavljenik->prezime, datum);
247  }
249
    /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
251
    exit(EXIT_SUCCESS);
}
```

### Rešenje 4.19

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```
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 proverava da li su dva stabla koja sadrže cele
8     brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
9     odnosno 0 ako nisu */
10 int identitet(Cvor * koren1, Cvor * koren2)
11 {
12     /* Ako su oba stabla prazna, jednaka su */
13     if (koren1 == NULL && koren2 == NULL)
14         return 1;
15
16     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
17     if (koren1 == NULL || koren2 == NULL)
18         return 0;
19
20     /* Ako su oba stabla neprazna i u korenu se nalaze različite
21        vrednosti, može se zaključiti da se razlikuju */
22     if (koren1->broj != koren2->broj)
23         return 0;
24
25     /* Inace, proverava se da li vazi jednakost i levih i desnih
26        podstabala */
27     return (identitet(koren1->levo, koren2->levo)
28             && identitet(koren1->desno, koren2->desno));
29 }
30
31 int main()
32 {
33     int broj;
34     Cvor *koren1, *koren2;
35
36     /* Učitavaju se elementi prvog stabla */
37     koren1 = NULL;
38     printf("Prvo stablo: ");
39     scanf("%d", &broj);
40     while (broj != 0) {
41         if (dodaj_u_stablo(&koren1, broj) == 1) {
42             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
43                     broj);
44             oslobodi_stablo(&koren1);
45             exit(EXIT_FAILURE);
46         }
47         scanf("%d", &broj);
48     }
```



```

48     }

50     /* Ucitavaju se elementi drugog stabla */
    koren2 = NULL;
52     printf("Drugo stablo: ");
    scanf("%d", &broj);
54     while (broj != 0) {
        if (dodaj_u_stablo(&koren2, broj) == 1) {
56             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
                broj);
58             oslobodi_stablo(&koren2);
            exit(EXIT_FAILURE);
60         }
        scanf("%d", &broj);
62     }

64     /* Poziva se funkcija koja ispituje identitet stabala i ispisuje
        se njen rezultat */
66     if (identitet(koren1, koren2))
        printf("Stabla jesu identicna.\n");
68     else
        printf("Stabla nisu identicna.\n");
70
72     /* Oslobadja se memorija zauzeta stablima */
    oslobodi_stablo(&koren1);
    oslobodi_stablo(&koren2);
74
76     exit(EXIT_SUCCESS);
}

```

### Rešenje 4.20

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6

8  /* Funkcija kreira novo stablo identicno stablu koje je dato
    korenom. Povratna vrednost funkcije je 0 ukoliko je kopiranje
    uspesno, odnosno 1 ukoliko je doslo do greske */
10 int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
    {
12         /* Izlaz iz rekurzije */
        if (koren == NULL) {
14             *duplikat = NULL;
            return 0;
16         }
    }

```

```
18  /* Duplira se koren stabla i postavlja da bude koren novog stabla
    */
    *duplikat = napravi_cvor(koren->broj);
20  if (*duplikat == NULL) {
        return 1;
22  }

24  /* Rekurzivno se dupliraju levo i desno podstablo i njihove adrese
    se cuvaju redom u pokazivacima na levo i desno podstablo korena
    duplikata */
26  int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
28  int kopija_desno =
        kopiraj_stablo(koren->desno, &(*duplikat)->desno);
30
    /* Ako je uspesno duplirano i levo i desno podstablo */
32  if (kopija_levo == 0 && kopija_desno == 0)
        /* Uspesno je duplirano i celo stablo */
34  return 0;
    /* Inace, prijavljuje se da je doslo do greske */
36  return 1;
}

38
40 /* Funkcija izracunava uniju dva skupa predstavljena stablima -
    rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
    Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
42  uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
44 {
    /* Ako drugo stablo nije prazno */
46  if (koren2 != NULL) {
        /* 1. Dodaje se njegov koren u prvo stablo */
48  if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
            return 1;
50  }

52  /* 2. Rekurzivno se racuna unija levog i desnog podstabla drugog
    stabla sa prvim stablom */
54  int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
    int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);
56
    /* Ako je unija podstabala uspesno kreirana */
58  if (unija_levo == 0 && unija_desno == 0)
        /* Uspesno je kreirana i unija stabala */
60  return 0;

62  /* U suprotnom se prijavljuje da je doslo do greske */
    return 1;
64  }

66  /* Ako je drugo stablo prazno, nista se ne preduzima */
    return 0;
```

```

68 }

70 /* Funkcija izracunava presek dva skupa predstavljana stablima -
71    rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
72    Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka
73    uspesno, odnosno 1 ukoliko je doslo do greske */
74 int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
75 {
76     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
77     if (*adresa_korena1 == NULL)
78         return 0;

80     /* Inace... Kreira se presek levog i desnog podstabla sa drugim
81        stablom, tj. iz levog i desnog podstabla prvog stabla brisu se
82        svi oni elementi koji ne postoje u drugom stablu */
83     int presek_levo = kreiraj_presek(&(*adresa_korena1)->levo, koren2);
84     int presek_desno =
85         kreiraj_presek(&(*adresa_korena1)->desno, koren2);
86     if (presek_levo == 0 && presek_desno == 0) {
87         /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se
88            on uklanja iz prvog stabla */
89         if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
90             obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

92         /* Presek stabala je uspesno kreiran */
93         return 0;
94     }
95     /* Inece, prijavljuje se da je doslo do greske */
96     return 1;
97 }

98 /* Funkcija izracunava razliku dva skupa predstavljana stablima -
99    rezultujući skup tj. stablo se dobija modifikacijom prvog stabla.
100    Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike
101    uspesno, odnosno 1 ukoliko je doslo do greske */
102 int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
103 {
104     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
105     if (*adresa_korena1 == NULL)
106         return 0;

108     /* Inace... */
109     /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
110        tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
111        elementi koji postoje i u drugom stablu */
112     int razlika_levo =
113         kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
114     int razlika_desno =
115         kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
116     if (razlika_levo == 0 && razlika_desno == 0) {
117         /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on
118            uklanja se iz prvog stabla */

```

```
120     if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
121         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
122
123     /* Razlika stabala je uspesno kreirana */
124     return 0;
125 }
126
127 /* Inace, prijavljuje se da je doslo do greske */
128 return 1;
129 }
130
131 int main()
132 {
133     Cvor *skup1;
134     Cvor *skup2;
135     Cvor *pomocni_skup = NULL;
136     int n;
137
138     /* Ucitavaju se elementi prvog skupa */
139     skup1 = NULL;
140     printf("Prvi skup: ");
141     while (scanf("%d", &n) != EOF) {
142         if (dodaj_u_stablo(&skup1, n) == 1) {
143             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
144             oslobodi_stablo(&skup1);
145             exit(EXIT_FAILURE);
146         }
147     }
148
149     /* Ucitavaju se elementi drugog skupa */
150     skup2 = NULL;
151     printf("Drugi skup: ");
152     while (scanf("%d", &n) != EOF) {
153         if (dodaj_u_stablo(&skup2, n) == 1) {
154             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
155             oslobodi_stablo(&skup2);
156             exit(EXIT_FAILURE);
157         }
158     }
159
160     /* Kreira se unija skupova: prvo se napravi kopija prvog skupa
161        kako bi se polazni skup mogao iskoristiti i za preostale
162        operacije */
163     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
164         oslobodi_stablo(&skup1);
165         oslobodi_stablo(&pomocni_skup);
166         exit(EXIT_FAILURE);
167     }
168     if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
169         oslobodi_stablo(&pomocni_skup);
170         oslobodi_stablo(&skup2);
171         exit(EXIT_FAILURE);
172     }
```

```

172 }
173 printf("Unija: ");
174 ispisi_stablo_infiksno(pomocni_skup);
175 putchar('\n');
176
177 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
178    operacije */
179 oslobodi_stablo(&pomocni_skup);
180
181 /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
182    kako bi se polazni skup mogao iskoristiti i za preostale
183    operacije */
184 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
185     oslobodi_stablo(&skup1);
186     oslobodi_stablo(&pomocni_skup);
187     exit(EXIT_FAILURE);
188 }
189 if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190     oslobodi_stablo(&pomocni_skup);
191     oslobodi_stablo(&skup2);
192     exit(EXIT_FAILURE);
193 }
194 printf("Presek: ");
195 ispisi_stablo_infiksno(pomocni_skup);
196 putchar('\n');
197
198 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
199    operacije */
200 oslobodi_stablo(&pomocni_skup);
201
202 /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
203    kako bi se polazni skup mogao iskoristiti i za preostale
204    operacije */
205 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
206     oslobodi_stablo(&skup1);
207     oslobodi_stablo(&pomocni_skup);
208     exit(EXIT_FAILURE);
209 }
210 if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
211     oslobodi_stablo(&pomocni_skup);
212     oslobodi_stablo(&skup2);
213     exit(EXIT_FAILURE);
214 }
215 printf("Razlika: ");
216 ispisi_stablo_infiksno(pomocni_skup);
217 putchar('\n');
218
219 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
220    operacije */
221 oslobodi_stablo(&pomocni_skup);
222
223 /* Oslobadja se memorija zauzeta polaznim skupovima */

```

## 4 Dinamičke strukture podataka

---

```
224   oslobodi_stablo(&skup1);
      oslobodi_stablo(&skup2);
226
      exit(EXIT_SUCCESS);
228 }
```

### Rešenje 4.21

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
7
   #define MAX 50
9
   /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
      cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11  su smestene u niz */
   int kreiraj_niz(Cvor * koren, int a[])
13 {
      int r, s;
15
      /* Stablo je prazno - u niz je smesteno 0 elemenata */
17   if (koren == NULL)
       return 0;
19
      /* Dodaju se u niz elementi iz levog podstabla */
21   r = kreiraj_niz(koren->levo, a);
23
      /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
         u niz i na osnovu nje se moze odrediti indeks novog elementa */
25
      /* Smesta se vrednost iz korena */
27   a[r] = koren->broj;
29
      /* Dodaju se elementi iz desnog podstabla */
      s = kreiraj_niz(koren->desno, a + r + 1);
31
      /* Racuna se indeks na koji treba smestiti naredni element */
33   return r + s + 1;
   }
35
   /* Funkcija sortira niz tako sto najpre elemente niza smesti u
37   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
      desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
39   kreiran i sortirao, a 1 ukoliko je doslo do greske.
```

```

41     Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu"
43     kao sto je to slucaj sa ostalim opisanim algoritmima sortiranja
44     jer se sortiranje vrši u pomocnoj dinamičkoj strukturi, a ne
45     razmenom elemenata niza. */
46 int sortiraj(int a[], int n)
47 {
48     int i;
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
60     niz a */
61     kreiraj_niz(koren, a);
62
63     /* Stablo vise nije potrebno pa se oslobadja zauzeta memorija */
64     oslobodi_stablo(&koren);
65
66     /* Vraca se indikator uspesnog sortiranja */
67     return 0;
68 }
69
70 int main()
71 {
72     int a[MAX];
73     int n, i;
74
75     /* Ucitavaju se dimenzija i elementi niza */
76     printf("n: ");
77     scanf("%d", &n);
78     if (n < 0 || n > MAX) {
79         printf("Greska: Pogresna dimenzija niza!\n");
80         exit(EXIT_FAILURE);
81     }
82
83     printf("a: ");
84     for (i = 0; i < n; i++)
85         scanf("%d", &a[i]);
86
87     /* Poziva se funkcija za sortiranje */
88     if (sortiraj(a, n) == 0) {
89         /* Ako je niz uspesno sortirao, ispisuje se rezultujući niz */
90         for (i = 0; i < n; i++)
91             printf("%d ", a[i]);
92         printf("\n");
93     } else {

```

```
93     /* Inace, obavestava se korisnik da je doslo do greske */
94     printf("Greska: Problem prilikom sortiranja niza!\n");
95 }
96
97 exit(EXIT_SUCCESS);
98 }
```

### Rešenje 4.22

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* a) Funkcija koja izracunava broj cvorova stabla */
int broj_cvorova(Cvor * koren)
{
    /* Ako je stablo prazno, broj cvorova je nula */
    if (koren == NULL)
        return 0;

    /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
       levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
       zato sto treba racunati i koren */
    return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
}

/* b) Funkcija koja izracunava broj listova stabla */
int broj_listova(Cvor * koren)
{
    /* Ako je stablo prazno, broj listova je nula */
    if (koren == NULL)
        return 0;

    /* Proverava se da li je tekuci cvor list */
    if (koren->levo == NULL && koren->desno == NULL)
        /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
           uvecati broj listova za 1 */
        return 1;

    /* U suprotnom se prebrojavaju listovi koje se nalaze u
       podstablima */
    return broj_listova(koren->levo) + broj_listova(koren->desno);
}

/* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
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 */
51     pozitivni_listovi(koren->levo);
52     pozitivni_listovi(koren->desno);
53 }
54
55 /* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
57 {
58     /* Ako je stablo prazno, zbir cvorova je 0 */
59     if (koren == NULL)
60         return 0;
61
62     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
63     elemenata u podstablima */
64     return koren->broj + zbir_svih_cvorova(koren->levo) +
65            zbir_svih_cvorova(koren->desno);
66 }
67
68 /* e) Funkcija koja izracunava najveći element stabla */
69 Cvor *najveci_element(Cvor * koren)
70 {
71     /* Ako je stablo prazno, obustavlja se pretraga */
72     if (koren == NULL)
73         return NULL;
74
75     /* Zbog prirode pretrazivackog stabla, vrednosti veće od korena se
76     nalaze u desnom podstablu */
77
78     /* Ako desnog podstabla nema */
79     if (koren->desno == NULL)
80         /* Najveća vrednost je koren */
81         return koren;
82
83     /* Inace, najveća vrednost se traži desno */
84     return najveci_element(koren->desno);
85 }
86
87 /* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
89 {
90     /* Dubina praznog stabla je 0 */
91     if (koren == NULL)

```

```
92     return 0;

94     /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

96     /* Izracunava se dubina desnog podstabla */
98     int dubina_desno = dubina_stabla(koren->desno);

100    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
        jer se racuna i koren */
102    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
104 }

106 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
    int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
108 {
    /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
        nivoa */

110    /* Ako nema vise cvorova, nema spustanja niz stablo */
    if (koren == NULL)
112        return 0;

114    /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije
        zbog rekurzivnih poziva uvecati broj cvorova za 1 */
116    if (i == 0)
        return 1;

118    /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
        */
120    return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
        + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
122 }

124

126 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
    void ispis_nivo(Cvor * koren, int i)
128 {
    /* Nema vise cvorova, nema spustanja kroz stablo */
130    if (koren == NULL)
        return;

132    /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
134    if (i == 0) {
        printf("%d ", koren->broj);
        return;
136    }

138    /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
        desnom podstablu */
140    ispis_nivo(koren->levo, i - 1);
    ispis_nivo(koren->desno, i - 1);
142 }
```

```
144 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
    stabla */
146 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
    {
148     /* Ako je stablo prazno, obustavlja se pretraga */
    if (koren == NULL)
150         return NULL;

152     /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
    if (i == 0)
154         return koren;

156     /* Pronalazi se maksimum na i-tom nivou levog podstabla */
    Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
158
    /* Pronalazi se maksimum na i-tom nivou desnog podstabla */
160     Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);

162     /* Trazi se i vraca maksimum izracunatih vrednosti */
    if (a == NULL && b == NULL)
164         return NULL;
    if (a == NULL)
166         return b;
    if (b == NULL)
168         return a;
    /* Ako su obe vrednosti razlicite od NULL, veca od vrednosti se
170     nalazi u b cvoru jer je stablo pretrazivacko */
    return b;
172 }

174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
    /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)
        return 0;

180
    /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
182     if (i == 0)
        return koren->broj;

184
    /* Inace, spustanje se nastavlja za jedan nivo nize i traže se
186     sume iz levog i desnog podstabla */
    return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
    }
190

192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
    manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
```

```

196  {
197      /* Ako je stablo prazno, zbir je nula */
198      if (koren == NULL)
199          return 0;
200
201      /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
202         prirode pretrazivackog stabla treba obici i levo i desno
203         podstablo */
204      if (koren->broj <= x)
205          return koren->broj + zbir_manjih_od_x(koren->levo, x) +
206              zbir_manjih_od_x(koren->desno, x);
207
208      /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
209         medju njima jedino moze biti onih koje zadovoljavaju uslov */
210      return zbir_manjih_od_x(koren->levo, x);
211  }
212
213  int main(int argc, char **argv)
214  {
215      /* Analiza argumenata komandne linije */
216      if (argc != 3) {
217          fprintf(stderr,
218              "Greska: Program se poziva sa: ./a.out nivo
219              broj_za_pretragu\n");
220          exit(EXIT_FAILURE);
221      }
222
223      int i = atoi(argv[1]);
224      int x = atoi(argv[2]);
225
226      /* Kreira se stablo uz proveru uspesnosti dodavanja novih
227         vrednosti */
228      Cvor *koren = NULL;
229      int broj;
230      while (scanf("%d", &broj) != EOF) {
231          if (dodaj_u_stablo(&koren, broj) == 1) {
232              fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
233                  broj);
234              oslobodi_stablo(&koren);
235              exit(EXIT_FAILURE);
236          }
237      }
238
239      /* Ispisuju se rezultati rada funkcija */
240      printf("Broj cvorova: %d\n", broj_cvorova(koren));
241      printf("Broj listova: %d\n", broj_listova(koren));
242      printf("Pozitivni listovi: ");
243      pozitivni_listovi(koren);
244      printf("\n");
245      printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
246      if (najveci_element(koren) == NULL)
247          printf("Najveci element: ne postoji\n");
248      else

```

```

246     printf("Najveci element: %d\n", najveci_element(koren)->broj);
248     printf("Dubina stabla: %d\n", dubina_stabla(koren));
250     printf("Broj cvorova na %d. nivou: %d\n", i,
            broj_cvorova_na_itom_nivou(koren, i));
252     printf("Elementi na %d. nivou: ", i);
    ispis_nivo(koren, i);
254     printf("\n");
    if (najveci_element_na_itom_nivou(koren, i) == NULL)
256         printf("Nema elemenata na %d. nivou!\n", i);
    else
258         printf("Maksimalni element na %d. nivou: %d\n", i,
                najveci_element_na_itom_nivou(koren, i)->broj);
260
    printf("Zbir elemenata na %d. nivou: %d\n", i,
262           zbir_cvorova_na_itom_nivou(koren, i));
    printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
264           zbir_manjih_od_x(koren, x));

266     /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
268
    exit(EXIT_SUCCESS);
270 }

```

### Rešenje 4.23

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6
/* Funkcija koja izracunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
{
10     /* Dubina praznog stabla je 0 */
    if (koren == NULL)
12         return 0;

14     /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);
16
    /* Izracunava se dubina desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);

20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje

```

```
        jer se racuna i koren */
22     return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }

26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
28 {
    /* Nema vise cvorova, nema spustanja niz stablo */
30     if (koren == NULL)
        return;
32
    /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
34     if (i == 0) {
        printf("%d ", koren->broj);
36         return;
    }
38     /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u
        desnom podstablu */
40     ispisi_nivo(koren->levo, i - 1);
    ispisi_nivo(koren->desno, i - 1);
42 }

44 /* Funkcija koja ispisuje stablo po nivoima */
void ispisi_stablo_po_nivoima(Cvor * koren)
46 {
    int i;
48
    /* Prvo se izracunava dubina stabla */
50     int dubina;
    dubina = dubina_stabla(koren);
52
    /* Ispisuje se nivo po nivo stabla */
54     for (i = 0; i < dubina; i++) {
        printf("%d. nivo: ", i);
56         ispisi_nivo(koren, i);
        printf("\n");
58     }
}

60
62 int main(int argc, char **argv)
{
    Cvor *koren;
64     int broj;

66     /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
        uspesnosti dodavanja */
68     koren = NULL;
    while (scanf("%d", &broj) != EOF) {
70         if (dodaj_u_stablo(&koren, broj) == 1) {
            fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
72                 broj);
        }
    }
}
```

```

    oslobodi_stablo(&koren);
74     exit(EXIT_FAILURE);
    }
76 }

78 /* Ispisuje se stablo po nivoima */
    ispisi_stablo_po_nivoima(koren);
80
    /* Oslobadja se memorija zauzeta stablom */
82     oslobodi_stablo(&koren);
84
    exit(EXIT_SUCCESS);
}

```

### Rešenje 4.25

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

```

1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  /* Funkcija koja izracunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
9  {
   /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
        return 0;
13
   /* Izracunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);

17     /* Izracunava se dubina desnog podstabla */
        int dubina_desno = dubina_stabla(koren->desno);
19

   /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
21     jer se racuna i koren */
        return dubina_levo >
23             dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
   }
25

   /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
27     stablo */
   int avl(Cvor * koren)
29 {
        int dubina_levo, dubina_desno;
31

   /* 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 podstablama */
50         return avl(koren->levo) + avl(koren->desno);
51     }
52 }
53
54 int main(int argc, char **argv)
55 {
56     Cvor *koren;
57     int broj;
58
59     /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
60        uspesnosti dodavanja */
61     koren = NULL;
62     while (scanf("%d", &broj) != EOF) {
63         if (dodaj_u_stablo(&koren, broj) == 1) {
64             fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
65                     broj);
66             oslobodi_stablo(&koren);
67             exit(EXIT_FAILURE);
68         }
69     }
70
71     /* Racuna se i ispisuje broj AVL cvorova */
72     printf("%d\n", avl(koren));
73
74     /* Oslobadja se memorija zauzeta stablom */
75     oslobodi_stablo(&koren);
76
77     exit(EXIT_SUCCESS);
78 }
```

### Rešenje 4.26

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.



```
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 kreira stablo prema zadatoj slici. Povratna
8     vrednost funkcije je 0 ako je stablo uspesno kreirano, odnosno 1
9     ukoliko je doslo do greske */
10 int kreiraj_hip(Cvor ** adresa_korena)
11 {
12     /* Stablo se proglašava praznim */
13     *adresa_korena = NULL;
14
15     /* Dodaje se cvor po cvor uz proveru uspesnosti dodavanja */
16     if (((*adresa_korena) = napravi_cvor(100)) == NULL)
17         return 1;
18     if (((*adresa_korena)->levo = napravi_cvor(19)) == NULL)
19         return 1;
20     if (((*adresa_korena)->levo->levo = napravi_cvor(17)) == NULL)
21         return 1;
22     if (((*adresa_korena)->levo->levo->levo = napravi_cvor(2)) == NULL)
23         return 1;
24     if (((*adresa_korena)->levo->levo->desno =
25         napravi_cvor(7)) == NULL)
26         return 1;
27     if (((*adresa_korena)->levo->desno = napravi_cvor(3)) == NULL)
28         return 1;
29     if (((*adresa_korena)->desno = napravi_cvor(36)) == NULL)
30         return 1;
31     if (((*adresa_korena)->desno->levo = napravi_cvor(25)) == NULL)
32         return 1;
33     if (((*adresa_korena)->desno->desno = napravi_cvor(1)) == NULL)
34         return 1;
35
36     /* Vraca se indikator uspesnog kreiranja */
37     return 0;
38 }
39
40 /* Funkcija proverava da li je zadato binarno stablo celih
41    pozitivnih brojeva hip. Ideja koja ce biti implementirana u
42    osnovi ima pronalazenje maksimalne vrednosti levog i maksimalne
43    vrednosti desnog podstabla - ako je vrednost u korenu veca od
44    izracunatih vrednosti, uoceni fragment stabla zadovoljava uslov
45    za hip. Zato ce funkcija vratiti maksimalne vrednosti iz uocenog
46    podstabala ili vrednost -1 ukoliko se zakljuci da stablo nije
47    hip. */
48 int hip(Cvor * koren)
49 {
50     int max_levo, max_desno;
```

```
52  /* Prazno stablo je hip - kao rezultat se vraca 0 kao najmanji
    pozitivan broj */
54  if (koren == NULL) {
    return 0;
56  }
    /* Ukoliko je stablo list... */
58  if (koren->levo == NULL && koren->desno == NULL) {
    /* Vraca se njegova vrednost */
60    return koren->broj;
    }

62
    /* Inace... Proverava se svojstvo za levo podstablo */
64    max_levo = hip(koren->levo);

66    /* Proverava se svojstvo za desno podstablo */
    max_desno = hip(koren->desno);

68
    /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
    ni celo stablo */
70    if (max_levo == -1 || max_desno == -1) {
72        return -1;
    }

74
    /* U suprotnom proverava se da li svojstvo vazi za uoceni cvor */
76    if (koren->broj > max_levo && koren->broj > max_desno) {
    /* Ako vazi, vraca se vrednost korena */
78        return koren->broj;
    }

80
    /* U suprotnom se zakljucuje da stablo nije hip */
82    return -1;
    }

84
86  int main(int argc, char **argv)
87  {
88      Cvor *koren;
    int hip_indikator;

90
    /* Kreira se stablo prema zadatoj slici */
    if (kreiraj_hip(&koren) == 1) {
92        fprintf(stderr, "Greska: Neuspesno kreiranje hipa.\n");
        oslobodi_stablo(&koren);
94        exit(EXIT_FAILURE);
    }

96
    /* Poziva se funkcija kojom se proverava da li je stablo hip */
98    hip_indikator = hip(koren);

100
    /* Ispisuje se rezultat */
    if (hip_indikator == -1) {
102        printf("Zadato stablo nije hip!\n");
    } else {
```

```
104     printf("Zadato stablo je hip!\n");
    }
106     /* Oslobadja se memorija zauzeta stablom */
108     oslobodi_stablo(&koren);
110     exit(EXIT_SUCCESS);
}
```



# 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:
```

### Test 3

```

POKRETANJE: ./a.out ulaz.txt

DATOTEKA ULAZ.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
-1

```

### Test 4

```

POKRETANJE: ./a.out

IZLAZ ZA GREŠKE:
-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 rad sa binarnim pretraživačkim stablima 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

```

**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 GREŠKE:
-1

```

## 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. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

### Test 1

```
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 GREŠKE:
-1
```

### Test 3

```
POKRETANJE: ./a.out ulaz.txt foo
DATOTEKA ULAZ.TXT NE POSTOJI
IZLAZ ZA GREŠKE:
-1
```

### Test 4

```
POKRETANJE: ./a.out ulaz.txt .
DATOTEKA ULAZ.TXT JE PRAZNA
IZLAZ:
0
```

**Zadatak A.5** Na početku datoteke `trouglovi.txt` nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac:  $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$ , gde je  $s$  poluobim trougla). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

### Test 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI
IZLAZ ZA GREŠKE:
-1
```

### Test 4

```
TROUGLOVI.TXT
0
IZLAZ:
```

<i>Test 1</i>	<i>Test 2</i>
TROUGLOVI.TXT	TROUGLOVI.TXT
4	3
0 0 0 1.2 1 0	1.2 3.2 1.1 4.3
0.3 0.3 0.5 0.5 0.9 1	
-2 0 0 0 0 1	
-2 0 0 0 0 1	
IZLAZ:	IZLAZ ZA GREŠKE:
	-1
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	

**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 rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

<p><i>Test 1</i></p> <pre> ULAZ:   1 5 3 6 1 4 7 9 IZLAZ:   1           </pre>	<p><i>Test 2</i></p> <pre> ULAZ:   2 5 3 6 1 0 4 7 9 IZLAZ:   2           </pre>	<p><i>Test 3</i></p> <pre> ULAZ:   0 4 2 5 IZLAZ:   0           </pre>
<p><i>Test 4</i></p> <pre> ULAZ:   3 IZLAZ:   0           </pre>	<p><i>Test 5</i></p> <pre> ULAZ:   -1 4 5 1 7 IZLAZ:   0           </pre>	

### 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 000000000000000000000000000001111, i ako je  $n = 1$  tada na standardni izlaz treba ispisati neočnaučen broj čiji je binarni zapis jednak





iz zadatka 2.19.

<p><i>Test 1</i></p> <pre> ULAZ: 4 1 2 3 4 2 1 4 3 3 4 2 1 4 3 1 2 IZLAZ: 10 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 3 1 1 1 1 1 1 1 1 1 IZLAZ: 3 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: 2 1 1 2 2 IZLAZ: - </pre>
<p><i>Test 4</i></p> <pre> ULAZ: 2 1 2 1 2 IZLAZ: - </pre>	<p><i>Test 5</i></p> <pre> ULAZ: 1 5 IZLAZ: 5 </pre>	<p><i>Test 6</i></p> <pre> ULAZ: 0 IZLAZ ZA GREŠKE: -1 </pre>

## 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 unet sa standardnog ulaza. U slučaju da je uneti broj negativan, na standardni izlaz za greške program ispisuje `-1`, a inače ispisuje na standardni izlaz broj dobijen primenom funkcije `zamena`.

<p><i>Test 1</i></p> <pre> ULAZ: 285278344 IZLAZ: 2281766929 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 1024 IZLAZ: 1024 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: 1 IZLAZ: 16777216 </pre>
<p><i>Test 4</i></p> <pre> ULAZ: 0 IZLAZ: 0 </pre>	<p><i>Test 5</i></p> <pre> ULAZ: -63 IZLAZ ZA GREŠKE: -1 </pre>	

**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 sa binarnim pretraživačkim 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

```

### Test 3

```

ULAZ:
  5 6
IZLAZ:
  1

```

### Test 4

```

ULAZ:
  7 5 8
IZLAZ:
  1

```

### Test 5

```

ULAZ:
  5 7 8
IZLAZ:
  2

```

**Zadatak A.12** Sa standardnog ulaza zadaje se ime datoteke u kojoj se nalazi matrica realnih brojeva jednostruke tačnosti i jedan realan broj. Napisati program koji iz datoteke učitava matricu realnih brojeva, a zatim pronalazi 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 vrstu. U slučaju greške ispisati  $-1$  na standardni izlaz za greške. Pretpostaviti da ime datoteke neće biti duže od 30 karaktera. **NAPOMENA:** *U zadatku treba koristiti dinamičku alokaciju memorije.*

### 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:
  brojevi.txt 12

DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ ZA GREŠKE:
  -1

```

## A.5 Rešenja

### Rešenje A.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #define MAKS 50
5
6  /* Funkcija vrši dinamičku alokaciju memorije potrebne n linija tj.
   n niski od kojih nijedna nije duža od MAKS karaktera. */
7
8  char **alociranje_memorije(int n)
9  {
10     char **linije = NULL;
11     int i, j;
12     /* Alocira se prostor za niz vrsti matrice */
13     linije = (char **) malloc(n * sizeof(char *));
14     /* U slučaju neuspješnog otvaranja ispisuje se -1 na stderr i
       program završava. */
15     if (linije == NULL)
16         return NULL;
17     /* Alocira se prostor za svaku vrstu matrice. Niska nije duža od
       MAKS karaktera, a 1 se dodaje zbog terminirajuće nule. */
18     for (i = 0; i < n; i++) {
19         linije[i] = malloc((MAKS + 1) * sizeof(char));
20         /* Ako alokacija nije prošla uspješno, oslobadjaju se svi
           prethodno alocirani resursi, i povratna vrednost je NULL */
21         if (linije[i] == NULL) {
22             for (j = 0; j < i; j++) {
23                 free(linije[j]);
24             }
25             free(linije);
26             return NULL;
27         }
28     }
29     return linije;
30 }
31
32 /* Funkcija oslobadjaja dinamički alociranu memoriju */
33 char **oslobadjanje_memorije(char **linije, int n)
34 {
35     int i;
36     /* Oslobadja se prostor rezervisan za svaku vrstu */
37     for (i = 0; i < n; i++) {
38         free(linije[i]);
39     }
40     /* Oslobadja se memorija za niz pokazivaca na vrste */
41     free(linije);
42
43     /* Matrica postaje prazna, tj. nealocirana */
44 }
```

```
    return NULL;
48 }

50 int main(int argc, char *argv[])
51 {
52     FILE *ulaz;
53     char **linije;
54     int i, n;

55     /* Proverava argumenata komandne linije. */
56     if (argc != 2) {
57         fprintf(stderr, "-1\n");
58         exit(EXIT_FAILURE);
59     }

60     /* Otvara se datoteka cije ime je navedeno kao argument komandne
61        linije neposredno nakon imena programa koji se poziva. U
62        slucaju neuspesnog otvaranja ispisuje se -1 na stderr i program
63        zavrшава izvršavanje. */
64     ulaz = fopen(argv[1], "r");
65     if (ulaz == NULL) {
66         fprintf(stderr, "-1\n");
67         exit(EXIT_FAILURE);
68     }

69     /* Ucitava se broj linija. */
70     fscanf(ulaz, "%d", &n);

71     /* Alociranje memorije na osnovu ucitanog broja linija. */
72     linije = alociranje_memorije(n);

73     /* U slucaju neuspesne alokacije ispisuje se -1 na stderr i
74        program završava. */
75     if (linije == NULL) {
76         fprintf(stderr, "-1\n");
77         exit(EXIT_FAILURE);
78     }

79     /* Iz datoteke se ucita svih n linija. */
80     for (i = 0; i < n; i++) {
81         fscanf(ulaz, "%s", linije[i]);
82     }

83     /* Ispisu se u odgovarajućem poretku ucitane linije koje
84        zadovoljavaju kriterijum. */
85     for (i = n - 1; i >= 0; i--) {
86         if (isupper(linije[i][0])) {
87             printf("%s\n", linije[i]);
88         }
89     }

90     /* Oslobadja se memorija koja je dinamički alocirana. */
91     linije = oslobadjanje_memorije(linije, n);
92
93
94
95
96
97
98
```

```
100  /* Zatvara se datoteka. */
    fclose(ulaz);
102  exit(EXIT_SUCCESS);
}
```

### Rešenje A.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

*main.c*

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

4
int sumiraj_n(Cvor * koren, int n)
6 {
    /* Ako je stablo prazno, suma je nula */
8     if (koren == NULL)
        return 0;
10    /* Inace ... */
    /* Ako je n jednako nula, vraca se broj iz korena */
12    if (n == 0)
        return koren->broj;
14    /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
        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
        n-tom nivou u pocetnom stablu */
18    return sumiraj_n(koren->levo, n - 1)
        + sumiraj_n(koren->desno, n - 1);
20 }

22 int main()
{
24     Cvor *koren = NULL;
    int n;
26     int nivo;

28     /* Ucitava se vrednost nivoa */
    scanf("%d", &nivo);
30     while (1) {
        scanf("%d", &n);
32         /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
        if (n == 0)
34             break;
        /* Ako nije, dodaje se procitani broj u stablo. */
36         if (dodaj_u_stablo(&koren, n) == 1) {
            fprintf(stderr, "-1\n");
        }
    }
}
```

```

38     oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
40     }
    }

42     /* Ispisuje se rezultat rada trazene funkcije */
44     printf("%d\n", sumiraj_n(koren, nivo));

46     /* Oslobadja se memorija */
        oslobodi_stablo(&koren);

48     exit(EXIT_SUCCESS);
50 }

```

### Rešenje A.3

```

#include <stdio.h>
2  #include <stdlib.h>
#define MAKS 50

4  /* Funkcija ucitava elemente matrice sa standardnog ulaza */
6  void ucitaj_matricu(int m[][MAKS], int v, int k)
{
8     int i, j;
    for (i = 0; i < v; i++) {
10        for (j = 0; j < k; j++) {
            scanf("%d", &m[i][j]);
12        }
    }
14 }

16 /* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
    koja ima v vrsta */
18 int broj_negativnih_u_koloni(int m[][MAKS], int v, int k)
{
20     int broj_negativnih = 0;
    int i;
22     for (i = 0; i < v; i++) {
        if (m[i][k] < 0)
24         broj_negativnih++;
    }
26     return broj_negativnih;
}

28 /* Funkcija vraca indeks kolone matrice m u kojoj se nalazi najvise
    negativnih elemenata */
30 int maks_indeks(int m[][MAKS], int v, int k)
32 {
    int j;
34     int broj_negativnih;
    /* Inicijalizacija na nulu indeksa kolone sa maksimalnim brojem

```

```
36     negativnih (maks_indeks_kolone), kao i maksimalnog broja
    negativnih elemenata u nekoj koloni (maks_broj_negativnih) */
38     int maks_indeks_kolone = 0;
    int maks_broj_negativnih = 0;

40
41     for (j = 0; j < k; j++) {
42         /* Racuna se broj negativnih u j-oj koloni */
        broj_negativnih = broj_negativnih_u_koloni(m, v, j);
44         /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
            maksimalnog, azurira se trenutni maksimalni broj negativnih
            kao i indeks kolone sa maksimalnim brojem negativnih */
46         if (maks_broj_negativnih < broj_negativnih) {
48             maks_indeks_kolone = j;
            maks_broj_negativnih = broj_negativnih;
50         }
    }
52     return maks_indeks_kolone;
}

54
55 int main()
56 {
57     int m[MAKS][MAKS];
58     int v, k;

60     /* Ucitava se broj vrsta matrice */
    scanf("%d", &v);

62
63     /* Proverava se validnost broja vrsta */
64     if (v < 0 || v > MAKS) {
        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
66     }

68
69     /* Ucitava se broj kolona matrice */
70     scanf("%d", &k);

72
73     /* Proverava se validnost broja kolona */
74     if (k < 0 || k > MAKS) {
        fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
76     }

77     /* Ucitaju se elementi matrice */
78     ucitaj_matricu(m, v, k);

80
81     /* Pronalazi se kolona koja sadrzi najveći broj negativnih
        elemenata i ispisuje se rezultat */
82     printf("%d\n", maks_indeks(m, v, k));

84     exit(EXIT_SUCCESS);
}
```



## 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    /* Otvara se datoteka 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    /* Cita se sadrzaj otvorene datoteke, liniju po liniju. */
24    while (fgets(linija, MAKS, f) != NULL) {
25        p = linija;
26        while (1) {
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    /* Zatvara se datoteka */
41    fclose(f);
42
43    /* Ispisuje se vrednost 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>
4
5 /* Struktura trougao */
6 typedef struct _trougao {
7     double xa, ya, xb, yb, xc, yc;
8 } trougao;
9
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 }
15
16 /* Funkcija racuna povrsinu trougla koristeći Heronov obrazac */
17 double povrsina(trougao t)
18 {
19     /* Racunaju se duzine 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     /* Racuna se poluobim trougla */
24     double s = (a + b + c) / 2;
25     /* Primenom Heronovog obrasca racuna se povrsina trougla */
26     return sqrt(s * (s - a) * (s - b) * (s - c));
27 }
28
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 opadajuće. */
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 {
50     FILE *f;
```

```
52     int n, i;
    trougao *niz;

54     /* Otvara se datoteka ciji je naziv trouglovi.txt */
    if ((f = fopen("trouglovi.txt", "r")) == NULL) {
56         fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
58     }

60     /* Ucitava se podatak o broju trouglova iz datoteke */
    if (fscanf(f, "%d", &n) != 1) {
62         fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
64     }

66     /* Dinamicka alokacija memorije: za niz trouglova duzine n
        rezervise se memorijski prostor */
    if ((niz = malloc(n * sizeof(trougao))) == NULL) {
68         fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
70     }

72     /* Ucitavaju se podaci u niz iz otvorene datoteke */
    for (i = 0; i < n; i++) {
74         if (fscanf(f, "%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
76             &niz[i].xb, &niz[i].yb, &niz[i].xc,
78             &niz[i].yc) != 6) {
            fprintf(stderr, "-1\n");
            exit(EXIT_FAILURE);
80         }
    }

82     /* Poziva se funkcija qsort da sortira niz na osnovu funkcije
    poredi */
    qsort(niz, n, sizeof(trougao), &poredi);

86     /* Ispisuje se sortirani niz na standardni izlaz */
    for (i = 0; i < n; i++)
88         printf("%g %g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
90             niz[i].yb, niz[i].xc, niz[i].yc);

92     /* Oslobadja se dinamicki alocirana memorija */
    free(niz);

94     /* Zatvara se datoteka */
    fclose(f);

96     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 vraća 1 u slučaju neuspješnog dodavanja elementa
7     u stablo, a inače 0. */
8  int ucitaj_stablo(Cvor ** koren)
9  {
10     *koren = NULL;
11     int x;
12     /* Sve dok ima brojeva na standardnom ulazu, učitani 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. Takođe, 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
29        jednog potomka funkcija vraća 1, inače 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
39        nivou levog podstabla */
40     return prebroj_n(koren->levo, n - 1)
41         + prebroj_n(koren->desno, n - 1);
42 }
43
44 int main()
45 {
```

```

46  Cvor *koren;
    int n;
48  scanf("%d", &n);

50  /* Ucitavaju se elementi u stablo. U slucaju neuspesne alokacije
    oslobodja se alocirana memorija i izlazi se iz programa. */
52  if (ucitaj_stablo(&koren) == 1) {
    fprintf(stderr, "-1\n");
54  oslobodi_stablo(&koren);
    exit(EXIT_FAILURE);
56  }

58  /* Ispisuje se rezultat */
    printf("%d\n", prebroj_n(koren, n));
60
    /* Oslobadja se dinamicki alocirana memorija za stablo */
62  oslobodi_stablo(&koren);

64  exit(EXIT_SUCCESS);
}

```

## Rešenje A.7

```

#include <stdio.h>

2
/* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
   zapis argumenta x rotira za n mesta udesno */
4  unsigned int rotiraj(unsigned int x, unsigned int n)
6  {
    int i;
    unsigned int maska = 1;
    /* Formira se maska sa n jedinica na kraju, npr za n=4 maska bi
       izgledala: 000...00001111 */
10   /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
       nastavku je drugi nacin. */
12   for (i = 1; i < n; i++)
14     maska = (maska << 1) | 1;
    /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n
       bitova binarne reprezentacije broja x ce "ispasti". Za rotaciju
       je potrebno da se tih n bitova postavi na pocetak broja.
       Kreirana maska omogucava da se tih n bitova izdvoji sa (maska &
       x), a zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta
       ulevo tih n bitova postavlja na pocetak. Primenom logicke
       disjunkcije dobija se rotirani broj. */
22   return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
   }
24
int main()
26 {
    unsigned int x, n;
28

```

```
30  /* Ucitaju se brojevi sa standardnog ulaza */
    scanf("%u%u", &x, &n);

32  /* Ispisuje se rezultat */
    printf("%u\n", rotiraj(x, n));

34  return 0;
}
```

### Rešenje A.8

*liste.h*

```
1  #ifndef _LISTE_H_
2  #define _LISTE_H_ 1

4  /* Struktura koja predstavlja cvor liste */
    typedef struct cvor {
6      double vrednost;
        struct cvor *sledeci;
8    } Cvor;

10 /* Pomocna funkcija koja kreira cvor. */
    Cvor *napravi_cvor(double broj);

12 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
14 void oslobodi_listu(Cvor ** adresa_glave);

16 /* Funkcija pronalazi i vraća pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
18 Cvor *pronadji_poslednji(Cvor * glava);

20 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
    greske pri alokaciji memorije, inace vraća 0. */
22 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

24 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
    liste. */
26 void ispisi_listu(Cvor * glava);

28 #endif
```

*liste.c*

```
1  #include <stdio.h>
    #include <stdlib.h>
3  #include "liste.h"
```

```
5  /* Pomocna funkcija koja kreira cvor. */
   Cvor *napravi_cvor(double broj)
7  {
   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9   if (novi == NULL)
       return NULL;
11  /* inicijalizacija polja u novom cvoru */
   novi->vrednost = broj;
13  novi->sledeci = NULL;
   return novi;
15 }

17 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
19 void oslobodi_listu(Cvor ** adresa_glave)
   {
21   Cvor *pomocni = NULL;
   while (*adresa_glave != NULL) {
23     pomocni = (*adresa_glave)->sledeci;
     free(*adresa_glave);
25     *adresa_glave = pomocni;
   }
27 }

29 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
31 Cvor *pronadji_poslednji(Cvor * glava)
   {
33   /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
   funkcija vraca NULL. */
   if (glava == NULL)
35     return NULL;
   while (glava->sledeci != NULL)
37     glava = glava->sledeci;
   return glava;
39 }

41 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
43 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
45 {
   Cvor *novi = napravi_cvor(broj);
47   if (novi == NULL)
       return 1;
   if (*adresa_glave == NULL) {
49     *adresa_glave = novi;
51     return 0;
   }
   Cvor *poslednji = pronadji_poslednji(*adresa_glave);
   poslednji->sledeci = novi;
53   return 0;
55 }
```

```
57  /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
59  void ispisi_listu(Cvor * glava)
    {
61      for (; glava != NULL; glava = glava->sledeci)
          printf("%.2lf ", glava->vrednost);
63      putchar('\n');
    }
```

*main.c*

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

4
  /* Funkcija umece novi cvor iza tekuceg u listi */
6  void dodaj_iza(Cvor * tekuci, Cvor * novi)
    {
8      /* Novi cvor se dodaje iza tekuceg cvora. */
      novi->sledeci = tekuci->sledeci;
10     tekuci->sledeci = novi;
    }

12
  /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
14     Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace
      vraca 0. */
16  int dopuni_listu(Cvor ** adresa_glave)
    {
18     Cvor *tekuci;
      Cvor *novi;
20     double aritmeticka_sredina;
      /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
22     if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
        return 1;
24     /* Promenljiva tekuci se inicijalizaciju da pokazuje na pocetni
        cvor */
26     tekuci = *adresa_glave;
      /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
28         vrednosti u susednim cvorovima i kreira cvor sa tom vrednoscu.
        U slucaju neuspele alokacije novog cvora, funkcija vraca 1.
30         Inace, novi cvor se umece izmedju dva cvora za koje je racunata
        aritmeticka sredina */
32     while (tekuci->sledeci != NULL) {
        aritmeticka_sredina =
34         ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
        novi = napravi_cvor(aritmeticka_sredina);
36         if (novi == NULL)
            return 1;
38         /* Poziva se funkcija koja umece novi cvor iza tekuceg cvora */
        dodaj_iza(tekuci, novi);
    }
```



```

40     /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
41        cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
42        polazne liste */
43     tekuci = tekuci->sledeci;
44     tekuci = tekuci->sledeci;
45 }
46 return 0;
47 }
48
49 int main()
50 {
51     Cvor *glava = NULL;
52     double broj;
53
54     /* Dok se ne stigne do kraja ulaza, ucitavaju se elementi i dodaju
55        se na kraj 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 osloboditi. */
60         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
61             fprintf(stderr, "Greska: Neuspela alokacija za cvor %lf.\n",
62                    broj);
63             oslobodi_listu(&glava);
64             exit(EXIT_FAILURE);
65         }
66     }
67
68     /* Poziva se funkcija da dopuni listu. Ako je funkcija vratila 1,
69        onda je bilo greske pri alokaciji memorije za nov cvor.
70        Memoriju alociranu za cvorove liste treba osloboditi. */
71     if (dopuni_listu(&glava) == 1) {
72         fprintf(stderr, "Greska: Neuspela alokacija za cvor %lf.\n",
73                broj);
74         oslobodi_listu(&glava);
75         exit(EXIT_FAILURE);
76     }
77
78     /* Ispisuju se elementi liste */
79     ispisi_listu(glava);
80
81     /* Oslobadja se memorija rezervisana za listu */
82     oslobodi_listu(&glava);
83
84     exit(EXIT_SUCCESS);
85 }

```

## Rešenje A.9

```

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

```

```
1  #include "matrica.h"
2
3  /* Funkcija racuna zbir elemenata v-te vrste */
4  int zbir_vrste(int **M, int n, int v)
5  {
6      int j, zbir = 0;
7      for (j = 0; j < n; j++)
8          zbir += M[v][j];
9      return zbir;
10 }
11
12 /* Funkcija racuna zbir elemenata k-te kolone */
13 int zbir_kolone(int **M, int n, int k)
14 {
15     int i, zbir = 0;
16     for (i = 0; i < n; i++)
17         zbir += M[i][k];
18     return zbir;
19 }
20
21 /* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
22 argument magican. Ukoliko jeste magican funkcija vraca 1, inace
23 0. Argument funkcije zbir ce sadrzati zbir elemenata neke vrste
24 ili kolone ukoliko je kvadrat magican. */
25 int magicni_kvadrat(int **M, int n, int *zbir_magicnog)
26 {
27     int i, j;
28     int zbir = 0, zbir_pom;
29     /* Promenljivu zbir inicijalizujemo na zbir 0-te vrese */
30     zbir = zbir_vrste(M, n, 0);
31
32     /* Racunaju se zbrovi u ostalim vrstama i ako neki razlikuje od
33 vrednosti promeljive zbir funkcija vraca 1 */
34     for (i = 1; i < n; i++) {
35         zbir_pom = zbir_vrste(M, n, i);
36         if (zbir_pom != zbir)
37             return 0;
38     }
39
40     /* Racunaju se zbrovi u svim kolonama i ako neki razlikuje od
41 vrednosti promeljive zbir funkcija vraca 1 */
42     for (j = 0; j < n; j++) {
43         zbir_pom = zbir_kolone(M, n, j);
44         if (zbir_pom != zbir)
45             return 0;
46     }
47
48     /* Inace su zbrovi svih vrsta i kolona jednaki, postavlja se
49 vrednost u zbir_magicnog i funkcija vraca 1 */
50     *zbir_magicnog = zbir;
51     return 1;
52 }
53
54 int main()
```

```

{
56     int n;
57     int **matrica = NULL;
58     int zbir_magicnog;
59     scanf("%d", &n);
60
61     /* Provera da li je n strogo pozitivan */
62     if (n <= 0) {
63         printf("-1\n");
64         exit(EXIT_FAILURE);
65     }
66
67     /* Dinamicka alokacija kvadratne matrice dimenzije n */
68     matrica = alociraj_matricu(n, n);
69     if (matrica == NULL) {
70         printf("-1\n");
71         exit(EXIT_FAILURE);
72     }
73
74     /* Ucitavaju se elementi matrice sa standardnog ulaza */
75     ucitaj_matricu(matrica, n, n);
76
77     /* Ispisuje se rezultat na osnovu fukcije magicni_kvadrat */
78     if (magicni_kvadrat(matrica, n, &zbir_magicnog)) {
79         printf("%d\n", zbir_magicnog);
80     } else
81         printf("-\n");
82
83     /* Oslobadja se dinamicki alocirana memorija */
84     matrica = deallociraj_matricu(matrica, n);
85
86     exit(EXIT_SUCCESS);
87 }

```

### Rešenje A.10

```

#include <stdio.h>
2 #include <stdlib.h>
3
4 #define BITOVA_U_BAJTU 8
5
6 /* Funkcija u datom broju x menja mesta prvom i četvrtom bajtu */
7 unsigned int zamena(unsigned int x)
8 {
9     /* Deklaracija promenljivih za odgovarajuće maske i pomocne
10      promenljive */
11     unsigned maska_prvi_bajt, maska_cetvrti_bajt;
12     unsigned maska_prvi_bajt_komplement, maska_cetvrti_bajt_komplement;
13     unsigned prvi_bajt, cetvrti_bajt;
14     unsigned i;

```

```

16  /* Maska prvi bajt odgovara broju cija je binarna reprezentacija
17     00000...00000111111111 (8 bitova najmanje tezine su jedinice, a
18     ostalo su nule) moze se dobiti i tako sto se maska_prvi_bajt
19     postavi na heksadekadnu vrednost FF. Drugi nacin za
20     inicijalizaciju maske maska_prvi_bajt je dodavanjem jedinica sa
21     desne strane: */
22  maska_prvi_bajt = 1;
23  for (i = 1; i < BITOVA_U_BAJTU; i++)
24      maska_prvi_bajt = maska_prvi_bajt << 1 | 1;

26  /* Maska cetvrti bajt odgovara broju cija je binarna
27     reprezentacija 1111111100000...00000 (8 bitova najvece tezine
28     su jedinice, a ostalo su nule) i moze se dobiti pomeranjem
29     bitova prethodno kreirane maske maska_prvi_bajt tako da
30     jedinice budu na poziciji bajta najvece tezine. */
31  maska_cetvrti_bajt =
32      maska_prvi_bajt << ((sizeof(unsigned) - 1) * BITOVA_U_BAJTU);

34  /* Primenom operatora ~ na maska_prvi_bajt dobija se broj cija je
35     binarna reprezentacija 11111...11111000000000 (8 bitova
36     najmanje tezine su nule, a ostalo su jedinice) */
37  maska_prvi_bajt_komplement = ~maska_prvi_bajt;
38  /* Primenom operatora ~ na maska_prvi_bajt dobija se broj cija je
39     binarna reprezentacija 0000000011111...11111 (8 bitova najvece
40     tezine su nule, a ostalo su jedinice) */
41  maska_cetvrti_bajt_komplement = ~maska_cetvrti_bajt;

42
43  /* U promenljivu prvi_bajt se smesta broj koji se dobija kada se
44     bitovi prvog bajta broja x pomere ulevo, tako da budu na
45     poziciji cetvrtog bajta */
46  prvi_bajt =
47      (maska_prvi_bajt & x) << ((sizeof(unsigned) - 1) *
48                                BITOVA_U_BAJTU);
49
50  /* U promenljivu cetvrti_bajt se smesta broj koji se dobija kada
51     se bitovi cetvrtog bajta broja x pomere udesno, tako da budu na
52     poziciji prvog bajta */
53  cetvrti_bajt =
54      (maska_cetvrti_bajt & x) >> ((sizeof(unsigned) - 1) *
55                                    BITOVA_U_BAJTU);

56  /* Na nule se postavlja 8 bitova najmanje tezine, a ostali bitovi
57     ostaju nepromenjeni */
58  x = x & maska_prvi_bajt_komplement;

59
60  /* Na nule se postavlja 8 bitova najvece tezine, a ostali bitovi
61     ostaju nepromenjeni */
62  x = x & maska_cetvrti_bajt_komplement;

63
64  /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi iz
65     prvog bajta */
66  x = x | prvi_bajt;

```

```

68  /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi iz
    prvog bajta */
70  x = x | cetvrti_bajt;

72  return x;
}

74
76  int main()
77  {
78      int x;

    /* Sa standardnog ulaza se ucitava ceo broj */
80  scanf("%d", &x);

82  /* Provera da li je uneti broj negativan */
    if (x < 0) {
84      fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
86  }

88  /* Ispisuje se rezultat primene funkcije zamena na uneti broj x */
    printf("%u\n", zamena(x));
90
    exit(EXIT_SUCCESS);
92  }

```

### Rešenje A.11

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

```

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

4
/* Funkcija racuna duzinu najduzeg puta od korena do nekog lista */
6  int najduzi_put(Cvor * koren)
{
8      /* Pomocne promenljive */
    int najduzi_u_levom, najduzi_u_desnom;

10
    /* Ako je stablo prazno, povratna vrednost je -1 */
12  if (koren == NULL)
    return -1;

14
    /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
16  najduzi_u_levom = najduzi_put(koren->levo);

    /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
18  najduzi_u_desnom = najduzi_put(koren->desno);

20

```

```

22  /* Veka od prethodno izracunatih vrednosti za podstabla se uvecava
    za 1 i vraca kao konacan rezultat */
    return 1 + (najduzi_u_levom >
24              najduzi_u_desnom ? najduzi_u_levom : najduzi_u_desnom);
}

26
27 int main()
28 {
    Cvor *stablo = NULL;
30     int x;

32     /* U svakoj iteraciji se procitani broj dodaje u stablo. */
    while (scanf("%d", &x) == 1)
34         if (dodaj_u_stablo(&stablo, x) == 1) {
            fprintf(stderr, "-1\n");
36             exit(EXIT_FAILURE);
        }

38
    /* Ispisuje se rezultat rada trazene funkcije */
40     printf("%d\n", najduzi_put(stablo));

42     /* Oslobadja se memorija */
    oslobodi_stablo(&stablo);

44
    exit(EXIT_SUCCESS);
46 }

```

### Rešenje A.12

```

1  #include <stdio.h>
    #include <stdlib.h>
3  /* Ime datoteke nije duze od 30 karaktera */
    #define MAX 31

5
6  /* Funkcija alokira memorijski prostor za matricu sa n vrsta i m
    kolona */
7  float **alociraj_matricu(int n, int m)
9  {
    float **matrica = NULL;
11     int i, j;

13     /* Alokira se prostor za niz vrsta matrice */
    matrica = (float **) malloc(n * sizeof(float *));
15     /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije
        ce biti NULL, sto mora biti provereno u main funkciji */
17     if (matrica == NULL)
        return NULL;

19
    /* Alokira se prostor za svaku vrstu matrice */
21     for (i = 0; i < n; i++) {
        matrica[i] = (float *) malloc(m * sizeof(float));
    }
}

```

```

23     /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
        prethodno alocirani resursi, i povratna vrednost je NULL */
25     if (matrica[i] == NULL) {
        for (j = 0; j < i; j++)
27         free(matrica[j]);
        free(matrica);
29         return NULL;
    }
31 }

33 return matrica;
}

35 /* Funkcija oslobadja alocirani memorijski prostor */
37 float **deallociraj_matricu(float **matrica, int n)
{
39     int i;
    /* Oslobadja se prostor rezervisan za svaku vrstu */
41     for (i = 0; i < n; i++)
        free(matrica[i]);

43     /* Oslobadja se memorija za niz pokazivaca na vrste */
45     free(matrica);

47     /* Matrica postaje prazna, tj. nealocirana */
    return NULL;
49 }

51 /* Funkcija prebrojava koliko se puta pojavljuje broj x u i-toj
    vrsti matrice A, gde je m broj elemenata u vrsti */
53 int prebroj_u_itoj_vrsti(float **A, int i, int m, int x)
{
55     int j;
    int broj = 0;
57     for (j = 0; j < m; j++) {
        if (A[i][j] == x)
59             broj++;
    }
61     return broj;
}

63 /* Funkcija vraca indeks vrste matrice A u kojoj se realan broj x
    pojavljuje najmanje puta */
65 int indeks_vrste(float x, float **A, int n, int m)
{
67     /* Indeks vrste sa minimalnim brojem pojavljivanja broja x */
69     int min;
    /* Broj pojavljivanja broja x u vrsti sa indeksom min */
71     int min_broj;
    /* Promenljiva u kojoj ce se racunati broj pojavljivanja broja x u
    tekucnoj vrsti */
73     int broj_u_vrsti;

```

```
75  /* Pomocne promenljive */
76  int i;
77
78  /* Promenljiva min se inicijalizuje na nulu, a min_broj na broj
79  pojavljivanja broja x u nultoj vrsti */
80  min = 0;
81  min_broj = prebroj_u_itoj_vrsti(A, 0, m, x);
82
83  /* Za svaku vrstu (osim nulte) se racuna broj pojavljivanja broja
84  x u njoj, pa ukoliko je taj broj manji od trenutno najmanjeg
85  azuriraju se promenljive min i min_broj */
86  for (i = 1; i < n; i++) {
87      broj_u_vrsti = prebroj_u_itoj_vrsti(A, i, m, x);
88      if (broj_u_vrsti < min_broj) {
89          min_broj = broj_u_vrsti;
90          min = i;
91      }
92  }
93
94  /* Funkcija vraca odgovarajuci indeks vrste */
95  return min;
96  }
97
98  int main()
99  {
100      FILE *in;
101      char datoteka[MAX];
102      float broj;
103      float **A = NULL;
104      int i, j, m, n;
105
106      /* Sa standardnog ulaza se ucitava ime datoteke i realan broj */
107      scanf("%s", datoteka);
108      scanf("%f", &broj);
109
110      /* Otvara se datoteka za citanje */
111      in = fopen(datoteka, "r");
112
113      /* Provera da li je datoteka uspesno otvorena */
114      if (in == NULL) {
115          fprintf(stderr, "-1\n");
116          exit(EXIT_FAILURE);
117      }
118
119      /* Dimenzije matrice se ucitavaju iz datoteke (prva dva cela broja
120      u datoteci). U slucaju neuspesnog ucitavanja, na standardni
121      izlaz za greske se ispisuje -1 i prekida se program. */
122      if (fscanf(in, "%d %d", &n, &m) == EOF) {
123          fprintf(stderr, "-1\n");
124          exit(EXIT_FAILURE);
125      }
126  }
```



```
127  /* Provera da li su ucitani brojevi m i n pozitivni */
129  if (n <= 0 || m <= 0) {
131      fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
  }

133  /* Alokacija matrice */
  A = alociraj_matricu(n, m);

135

137  /* Provera da li je alokacija uspela */
  if (A == NULL) {
139      fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
  }

141

143  /* Ucitavaju se elementi matrice iz datoteke */
  for (i = 0; i < n; i++) {
145      for (j = 0; j < m; j++)
          fscanf(in, "%f", &A[i][j]);
  }

147

149  /* Zatvara se datoteka */
  fclose(in);

151  /* Ispisuje se rezultat poziva funkcije */
  printf("%d\n", indeks_vrste(broj, A, n, m));

153

155  /* Oslobadja se memorija koju je zauzimala matrica */
  A = dealociraj_matricu(A, n);

157  exit(EXIT_SUCCESS);
}
```