PROGRAMIRANJE 2

Milena Vujošević Janičić, 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

CIP Каталогизација у публикацији
Народна библиотека Србије, Београд

ISBN 978-86-7589-107-9

Sadržaj

Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanju problema, ispravnom radu sa pokazivačima i dinamički alociranom memorijom, osnovnim algoritmima pretraživanja i sortiranja, kao i radu sa dinamičkim strukturama podataka, poput listi i stabala. Zadaci koji se nalaze u ovoj zbirci predstavljaju objedinjen skup zadataka sa vežbi i praktikuma ovog kursa, kao i primere zadataka sa održanih ispita. Elektronska verzija zbirke i propratna rešenja u elektronskom formatu, dostupni su besplatno u okviru strane kursa 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.

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 zadatke i rešenja zbirke su dobrodošli i osećajte se slobodno da ih pošaljete elektronskom poštom bilo kome od autora[1].

Autori

[1]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) Definisati strukturu `KompleksanBroj` koja opisuje kompleksan broj zadat njegovim realnim i imaginarnim delom.

(b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj `z` sa standardnog ulaza.

(c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` koja ispisuje kompleksan broj `z` na standardni izlaz u odgovarajućem formatu.

(d) Napisati funkciju `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja `z`.

(e) Napisati funkciju `float 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`.

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

Zadatak 1.4 Napisati biblioteku za rad sa razlomcima.

(a) Definisati strukturu `Razlomak` koja opisuje razlomak.

(b) Napisati funkciju `Razlomak ucitaj()` za učitavanje razlomka.

(c) Napisati funkciju `void ispisi(const Razlomak * r)` koja ispisuje razlomak `r`.

(d) Napisati funkciju `int brojilac(const Razlomak * r)` koja vraćaja brojilac razlomka `r`.

(e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraćaja imenilac razlomka `r`.

(f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.

(g) Napisati funkciju `double reciprocna_vrednost(const Razlomak * r)` koja vraća recipročnu vrednost razlomka `r`.

(h) Napisati funkciju `Razlomak skrati(const Razlomak * r)` koja vraća skraćenu vrednost datog razlomka `r`.

(i) Napisati funkciju `Razlomak saberi(const Razlomak * r1, const Razlomak * r2)` koja vraća zbir dva razlomka `r1` i `r2`.

(j) Napisati funkciju `Razlomak oduzmi(const Razlomak * r1, const Razlomak * r2)` koja vraća razliku dva razlomka `r1` i `r2`.

(k) Napisati funkciju `Razlomak pomnozi(const Razlomak * r1, const Razlomak * r2)` koja vraća proizvod dva razlomka `r1` i `r2`.

(l) Napisati funkciju `Razlomak podeli(const Razlomak * r1, const Razlomak * r2)` koja vraća količnik dva razlomka `r1` i `r2`.

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

Primer 1

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

1.2 Algoritmi za rad sa bitovima

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

Primer 1

```
Interakcija sa programom:
 Unesite broj tipa int:   0x4f4f4f4f
 Binarna reprezentacija: 01001111010011110100111101001111
 Unesite broj tipa short:   0x4f4f
 Binarna reprezentacija: 0100111101001111
 Unesite broj tipa char:   0x4f
 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:

(a) formiranjem odgovarajuće maske i njenim pomeranjem (funkcija `prebroj_bitove_1`)

(b) formiranjem odgovarajuće maske i pomeranjem promenljive x (funkcija `prebroj_bitove_2`).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati ceo broj u heksadekasnom formatu i redni broj funkcije koju treba primeniti (1 ili 2), a zatim na standardni izlaz ispisati broj jedinica u binarnom zapisu učitanog broja pozivom izabrane funkcije. Ukoliko korisnik ne unese ispravnu vrednost za 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: Neodgovarajuci 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:
  11111110000000000000000000000000
  Najmanji:
  00000000000000000000000001111111
```

Test 2

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

Test 3

```
Ulaz:
  0x00FF00FF
Izlaz:
  Najveci:
  11111111111111110000000000000000
  Najmanji:
  00000000000000001111111111111111
```

Test 4

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

Zadatak 1.8 Napisati funkcije za rad sa bitovima.

(a) Napisati funkciju `unsigned postavi_0(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se `n` bitova datog broja `x`, počevši od pozicije `p`, postave na 0.

(b) Napisati funkciju `unsigned postavi_1(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se `n` bitova datog broja `x`, počevši od pozicije `p`, postave na 1.

(c) Napisati funkciju `unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)` koja vraća broj u kome se `n` bitova najmanje težine poklapa sa n bitova broja `x` počevši od pozicije `p`, dok su mu ostali bitovi postavljeni na 0.

(d) Napisati funkciju `unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p, unsigned y)` koja vraća broj koji se dobija upisivanjem poslednjih `n` bitova najmanje težine broja `y` u broj `x`, počevši od pozicije `p`.

(e) Napisati funkciju `unsigned invertuj(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija invertovanjem `n` bitova broja `x` počevši

od pozicije p.

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve x, n, p, y koji se unose sa standardnog ulaza. Na standardni izlaz ispisati binarne reprezenatacije brojeva x i y, a zatim i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija. NAPOMENA: *Bit najmanje težine je krajnji desni bit i njegova pozicija se označava nultom dok se pozicije ostalih bitova uvećavaju za jedan, sa desna na levo.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite neoznacen ceo broj x: 235
 Unesite neoznacen ceo broj n: 9
 Unesite neoznacen ceo broj p: 24
 Unesite neoznacen ceo broj y: 127
 x =    235                                  = 00000000000000000000000011101011
 postavi_0(   235,     9,    24)             = 00000000000000000000000011101011

 x =    235                                  = 00000000000000000000000011101011
 postavi_1(   235,     9,    24)             = 00000001111111110000000011101011

 x =    235                                  = 00000000000000000000000011101011
 vrati_bitove(   235,     9,    24)          = 00000000000000000000000000000000

 x =    235                                  = 00000000000000000000000011101011
 y =    127                                  = 00000000000000000000000001111111
 postavi_1_n_bitove(  235,  9,  24,  127)    = 00000000111111110000000011101011

 x =    235                                  = 00000000000000000000000011101011
 invertuj(   235,     9,    24)              = 00000001111111110000000011101011
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite neoznacen ceo broj x: 2882398951
 Unesite neoznacen ceo broj n: 5
 Unesite neoznacen ceo broj p: 10
 Unesite neoznacen ceo broj y: 35156526
 x = 2882398951                              = 10101011110011011110101011100111
 postavi_0(2882398951,     5,    10)         = 10101011110011011110100000100111

 x = 2882398951                              = 10101011110011011110101011100111
 postavi_1(2882398951,     5,    10)         = 10101011110011011110111111100111

 x = 2882398951                              = 10101011110011011110101011100111
 vrati_bitove(2882398951,     5,    10)      = 00000000000000000000000000001011

 x = 2882398951                              = 10101011110011011110101011100111
 y =  35156526                               = 00000010000110000111001000101110
 postavi_1_n_bitove(2882398951, 5, 10, 35156526) = 10101011110011011110101110100111

 x = 2882398951                              = 10101011110011011110101011100111
 invertuj(2882398951,     5,    10)         = 10101011110011011110110100100111
```

Zadatak 1.9 Pod rotiranjem bitova ulevo podrazumeva se pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najveće težine pomera 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 heksadekasnom formatu, tatim ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem tri prethodno napisane funkcije sa argumentima `x` i `n`, a na kraju ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem funkcije `rotiraj_udesno_oznaceni` za argumente `-x` i `n`.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite neoznacen ceo broj x: ba11a7
 Unesite neoznacen ceo broj n: 5
 x = 00000000101110100001000110100111
 rotiraj_ulevo(ba11a7, 5)            = 00010111010000100011010011100000
 rotiraj_udesno(ba11a7, 5)           = 00111000000001011101000010001101
 rotiraj_udesno_oznaceni(ba11a7, 5)  = 00111000000001011101000010001101
 rotiraj_udesno_oznaceni(-ba11a7, 5) = 11000111111110100010111101110010
```

Zadatak 1.10 Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća ceo 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:
 00000000000000000000001001010101
 10101010010000000000000000000000
```

Test 2

```
ULAZ:
 -15
IZLAZ:
 11111111111111111111111111101011
 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 jednica nego nula, a inače vraća `0`. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1

```
ULAZ:
  10
IZLAZ:
  0
```

Test 2

```
ULAZ:
  2147377146
IZLAZ:
  1
```

Test 3

```
ULAZ:
  1111111115
IZLAZ:
  0
```

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

Test 1

```
ULAZ:
  11
IZLAZ:
  1
```

Test 2

```
ULAZ:
  1024
IZLAZ:
  0
```

Test 3

```
ULAZ:
  2147377146
IZLAZ:
  22
```

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

Primer 1

```
POKRETANJE: ./a.out 1 2

INTERAKCIJA SA PROGRAMOM:
ULAZ:
  11
IZLAZ:
  13
```

Primer 2

```
POKRETANJE: ./a.out 1 2

INTERAKCIJA SA PROGRAMOM:
ULAZ:
  1024
IZLAZ:
  1024
```

Primer 2

```
POKRETANJE: ./a.out 12 12

INTERAKCIJA SA PROGRAMOM:
ULAZ:
  12345
IZLAZ:
  12345
```

*** Zadatak 1.14** Napisati funkciju `void prevod(unsigned int x, char s[])` koja na osnovu neoznačenog broja x formira nisku s koja sadrži heksadekadni zapis broja x koristeći algoritam za brzo prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksade-

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

Test 1

ULAZ:
11
IZLAZ:
0000000B

Test 2

ULAZ:
1024
IZLAZ:
00000400

Test 3

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

ULAZ:
123 10
IZLAZ:
4294967285

Test 2

ULAZ:
3251 0
IZLAZ:
4294967295

Test 3

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

ULAZ:
123
IZLAZ:
0

Test 2

ULAZ:
3245
IZLAZ:
2

Test 3

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 standarni izlaz ispisati rezultat primene izabrane funkcije na unete brojeve. Ukoliko se na ulazu unese pogrešan redni broj funkcije, ispisati odgovarajuću poruku o grešci na standardni izlaz i prekinuti izvršavanje programa.

Primer 1

```
Interakcija sa programom:
 Unesite redni broj funkcije (1/2):
 1
 Unesite broj x:   2
 Unesite broj k:   10
 1024
```

Primer 2

```
Interakcija sa programom:
 Unesite redni broj funkcije (1/2):
 2
 Unesite broj x:   9
 Unesite broj k:   4
 6561
```

Zadatak 1.18 Koristeći uzajamnu (posrednu) rekurziju napisati:

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

(b) 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

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 pograma. 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.

Test 1

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

Test 2

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

Test 3

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

Test 4

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

Test 5

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

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 pograma.

Primer 1

```
Interakcija sa programom:
 Unesite redni broj funkcije (1 ili 2):
 1
 Unesite dimenziju niza:
 5
 Unesite elemente niza:
 1 2 3 4 5
 Suma elemenata je 15
```

Primer 2

```
Interakcija sa programom:
 Unesite redni broj funkcije (1 ili 2):
 2
 Unesite dimenziju niza:
 4
 Unesite elemente niza:
 -5 2 -3 6
 Suma elemenata je 0
```

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

učitati tri broja, a zatim elemente niza sve do kraja ulaza. Na standardni izlaz ispisati rezultat primene funkcije nad učitanim podacima. Pretpostaviti da neće biti uneto više od 256 brojeva.

Primer 1

```
Interakcija sa programom:
 Unesite tri cela broja:
 1 2 3
 Unesite elemente niza:
 4 1 2 3 4 5
 Uneti brojevi jesu uzastopni
 clanovi niza.
```

Primer 2

```
Interakcija sa programom:
 Unesite tri cela broja:
 1 2 3
 Unesite elemente niza:
 11 1 2 4 3 6
 Uneti brojevi jesu uzastopni
 clanovi niza.
```

Zadatak 1.28 Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja `x`. Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.

Test 1

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

Test 2

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

Test 3

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

Zadatak 1.29 Napisati rekurzivnu funkciju koja štampa bitovsku reprezentaciju neoznačenog celog broja, i program koji je testira za vrednost koja se zadaje sa standardnog ulaza.

Test 1

```
Ulaz:
 10
Izlaz:
 00000000000000000000000000001010
```

Test 2

```
Ulaz:
 0
Izlaz:
 00000000000000000000000000000000
```

Zadatak 1.30 Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: *Binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.*

Test 1

ULAZ:
5
IZLAZ:
5

Test 2

ULAZ:
125
IZLAZ:
7

Test 3

ULAZ:
8
IZLAZ:
1

Zadatak 1.31 Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. UPUTSTVO: *Binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.*

Test 1

ULAZ:
5
IZLAZ:
5

Test 2

ULAZ:
16
IZLAZ:
1

Test 3

ULAZ:
18
IZLAZ:
2

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

ULAZ:
a
IZLAZ:
da

Test 2

ULAZ:
aa
IZLAZ:
da

Test 3

ULAZ:
aba
IZLAZ:
da

Test 4

ULAZ:
programiranje
IZLAZ:
ne

Test 5

ULAZ:
anavolimilovana
IZLAZ:
da

*** Zadatak 1.33** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa $\{1, 2, ..., n\}$. 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.

(a) Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta $\binom{n}{k}$ koristeći osobine Paskalovog trougla.

(b) Napisati rekurzivnu funkciju koja izračunava d_n kao sumu elemenata n-te hipotenuze Paskalovog trougla.

Napisati program koji za unetu veličinu Paskalovog trougla i redni broj hipotenuze najpre iscrtava 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

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

/* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
   imaginaran deo kompleksnog broja */
typedef struct {
```

```
  float real;
  float imag;
} KompleksanBroj;

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

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

  /* 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 */

  if (z.real != 0) {
    printf("%.2f", z.real);

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

  /* Zavrsava se sa ispisom */
  printf(")");

}

/* Funkcija vraca vrednosti realnog dela kompleksnog broja */
```

```
60 float realan_deo(KompleksanBroj z)
   {
62   return z.real;
   }
64
   /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
66 float imaginaran_deo(KompleksanBroj z)
   {
68   return z.imag;
   }
70
   /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
72 float moduo(KompleksanBroj z)
   {
74   return sqrt(z.real * z.real + z.imag * z.imag);
   }
76
   /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
78    odgovara kompleksnom broju argumentu */
   KompleksanBroj konjugovan(KompleksanBroj z)
80 {
     /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
82      imaginarnom delu kompleksnog broja */

84   KompleksanBroj z1 = z;

86   z1.imag *= -1;

88   return z1;
   }
90
   /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
92    argumenata funkcije */
   KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
94 {
     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
96      broj ciji je realan deo zbir realnih delova kompleksnih brojeva
        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
98      brojeva z1 i z2 */

100  KompleksanBroj z = z1;

102  z.real += z2.real;
     z.imag += z2.imag;
104
     return z;
106 }

108 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
      argumenata funkcije */
110 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
    {
```

```
112   /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
         broj ciji je realan deo razlika realnih delova kompleksnih
114      brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
         kompleksnih brojeva z1 i z2 */
116
   KompleksanBroj z = z1;
118
   z.real -= z2.real;
120 z.imag -= z2.imag;

122 return z;
}
124
/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
126    argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
128 {
   /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
130      broj ciji se realan i imaginaran deo racunaju po formuli za
         mnozenje kompleksnih brojeva z1 i z2 */
132
   KompleksanBroj z;
134
   z.real = z1.real * z2.real - z1.imag * z2.imag;
136 z.imag = z1.real * z2.imag + z1.imag * z2.real;

138 return z;
}
140
/* Funkcija vraca argument zadatog kompleksnog broja */
142 float argument(KompleksanBroj z)
{
144 /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
      iz biblioteke math.h */
146
   return atan2(z.imag, z.real);
148 }

150 int main()
{
152 char c;

154 /* Deklaracija 3 promenljive tipa KompleksanBroj */
   KompleksanBroj z1, z2, z;
156
   /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
158      ispisivanje na standardni izlaz */
   ucitaj_kompleksan_broj(&z1);
160 ispisi_kompleksan_broj(z1);
   printf("\n");
162
   /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
```

```
164       na standardni izlaz */
    ucitaj_kompleksan_broj(&z2);
166   ispisi_kompleksan_broj(z2);
    printf("\n");
168
    /* Ucitavanje i provera znaka na osnovu koga korisnik bira
170       aritmeticku operaciju koja ce se izvrsiti nad kompleksnim
       brojevima */
172   getchar();
    printf("Unesite znak (+,-,*): ");
174   scanf("%c", &c);
    if (c != '+' && c != '-' && c != '*') {
176     printf("Greska: nedozvoljena vrednost operatora!\n");
      exit(EXIT_FAILURE);
178   }

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

192   /* Ispisuje se rezultat */
    ispisi_kompleksan_broj(z1);
194   printf(" %c ", c);
    ispisi_kompleksan_broj(z2);
196   printf("  =  ");
    ispisi_kompleksan_broj(z);
198
    /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
200       broja */
    printf("\nRealni_deo: %.f\nImaginarni_deo: %f\nModuo: %f\n",
202           realan_deo(z), imaginaran_deo(z), moduo(z));

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

210   /* Testira se funkcija koja racuna argument kompleksnog broja */
    printf("Argument kompleksnog broja: %f\n", argument(z));
212
    exit(EXIT_SUCCESS);
214 }
```

Rešenje 1.2

kompleksan_broj.h

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

/* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i
   onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
   posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
   narednoj pretrocesorskoj define direktivi. */
#ifndef  _KOMPLEKSAN_BROJ_H
#define _KOMPLEKSAN_BROJ_H

/* Zaglavlja standardne biblioteke koje sadrze deklaracije funkcija
   koje se koriste u definicijama funkcija navedenim u
    kompleksan_broj.c */
#include <stdio.h>
#include <math.h>

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

/* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
   definisane u kompleksan_broj.c */

/* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
   kompleksnog broja i smesta ih u strukturu cija je adresa argument
   funkcije */
void ucitaj_kompleksan_broj(KompleksanBroj * z);

/* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
   obliku (x + i y) */
void ispisi_kompleksan_broj(KompleksanBroj z);

/* Funkcija vraca vrednosti realnog dela kompleksnog broja */
float realan_deo(KompleksanBroj z);

/* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
float imaginaran_deo(KompleksanBroj z);

/* Funkcija vraca vrednost modula zadatog kompleksnog broja */
float moduo(KompleksanBroj z);

/* Funkcija vraca vrednost konjugovano kompleksnog broja koji
```

```
47    odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z);
49
/* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
51    argumenata funkcije */
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
53
/* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
55    argumenata funkcije */
KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
57
/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
59    argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
61
/* Funkcija vraca argument zadatog kompleksnog broja */
63 float argument(KompleksanBroj z);

65 /* Kraj zakljucanog dela */
#endif
```

kompleksan_broj.c

```
/* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima, jer je
2    neophodno da bude poznata definicija tipa KompleksanBroj. Takodje,
   time su ukljucena zaglavlja standardne biblioteke koja su navedena
4    u kompleksan_broj.h */
#include "kompleksan_broj.h"
6

8 void ucitaj_kompleksan_broj(KompleksanBroj * z)
{
10   /* Ucitavanje vrednosti sa standardnog ulaza */
   printf("Unesite realan i imaginaran deo kompleksnog broja: ");
12   scanf("%f", &z->real);
   scanf("%f", &z->imag);
14 }

16 void ispisi_kompleksan_broj(KompleksanBroj z)
{
18   /* Zapocinje se sa ispisom */
   printf("(");
20
   /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
22      razlicit od nule: tada se realni deo ispisuje na standardni
      izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
24      imaginarni deo pozitivan ili negativan, a potom i apsolutna
      vrednost imaginarnog dela kompleksnog broja 2) realni deo
26      kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
      s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
28      decimalnih mesta */
```

```
30    if (z.real != 0) {
        printf("%.2f", z.real);
32
        if (z.imag > 0)
34        printf(" + %.2f i", z.imag);
        else if (z.imag < 0)
36        printf(" - %.2f i", -z.imag);
    } else {
38      if (z.imag == 0)
          printf("0");
40      else
          printf("%.2f i", z.imag);
42  }

44  /* Zavrsava se sa ispisom */
    printf(")");
46  }

48  float realan_deo(KompleksanBroj z)
    {
50    /* Vraca se vrednost realnog dela kompleksnog broja */
      return z.real;
52  }

54  float imaginaran_deo(KompleksanBroj z)
    {
56    /* Vraca se vrednost imaginarnog dela kompleksnog broja */
      return z.imag;
58  }

60  float moduo(KompleksanBroj z)
    {
62    /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
      return sqrt(z.real * z.real + z.imag * z.imag);
64  }

66  KompleksanBroj konjugovan(KompleksanBroj z)
    {
68    /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
         se promeni znak imaginarnom delu kompleksnog broja */
70    KompleksanBroj z1 = z;
      z1.imag *= -1;
72    return z1;
    }
74
    KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76  {
      /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
78       broj ciji je realan deo zbir realnih delova kompleksnih brojeva
         z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
80       brojeva z1 i z2 */
```

```
  KompleksanBroj z = z1;

  z.real += z2.real;
  z.imag += z2.imag;

  return z;
}

KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
{
  /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
     broj ciji je realan deo razlika realnih delova kompleksnih
     brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
     kompleksnih brojeva z1 i z2 */
  KompleksanBroj z = z1;
  z.real -= z2.real;
  z.imag -= z2.imag;
  return z;
}

KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
{
  /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
     broj ciji se realan i imaginaran deo racunaju po formuli za
     mnozenje kompleksnih brojeva z1 i z2 */
  KompleksanBroj z;

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

  return z;
}

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

main.c

```
/*****************************************************************
Ovaj program koristi korektno definisanu biblioteku kompleksnih
brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
komplesnog broja i popis deklaracija podrzanih funkcija, a u
kompleksan_broj.c se nalaze njihove definicije.

Kompilacija programa se najjednostavnije postize naredbom
gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c

```

```
Kompilacija se moze uraditi i na sledeci nacin:
gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
gcc -Wall -c -o main.o main.c
gcc -lm -o kompleksan_broj kompleksan_broj.o main.o

Napomena: Prethodne komande se koriste kada se sva tri navedena
dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
kompleksan_broj.c kompleksan_broj.h) nalazi u direktorijumu sa imenom
header_dir prevodjenje se vrsi dodavanjem opcije opcije -I header_dir
gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c
     main.c
*********************************************************************/


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

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

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

  /* Ispisivanje polarnog oblika kompleksnog broja */
  printf("Polarni oblik kompleksnog broja je %.2f *  e^i * %.2f\n",
         moduo(z), argument(z));

  return 0;
}
```

Rešenje 1.3

polinom.h

```
#ifndef  _POLINOM_H
#define _POLINOM_H

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

/* Maksimalni stepen polinoma */
#define MAKS_STEPEN 20


/* Polinomi se predstavljaju strukturom koja cuva koeficijente
```

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

/* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
   obliku. Polinom se prenosi po adresi da bi se ustedela memorija:
   ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
   se nalazi polinom koji ispisujemo */
void ispisi(const Polinom * p);

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

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

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

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

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

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

polinom.c

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

void ispisi(const Polinom * p)
{
  int nulaPolinom = 1;
  int i;
  /* Ispisivanje polinoma pocinje od najviseg stepena ka najnizem da
     bi polinom bio ispisan na prirodan nacin. Ipisisuju se samo oni
     koeficijenti koji su razliciti od nule. Ispred pozitivnih
     koeficijenata je potrebno ispisati znak + (osim u slucaju
     koeficijenta uz najvisi stepen). */
  for (i = p->stepen; i >= 0; i--) {

    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)
      printf("0");
34  putchar('\n');
}
36
Polinom ucitaj()
38{
  int i;
40  Polinom p;

42  /* Ucitava se stepena polinoma */
  scanf("%d", &p.stepen);
44
  /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
46     dozvoljenog opsega */
  while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
48    printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
    scanf("%d", &p.stepen);
50  }

52  /* Unose se koeficijenti polinoma */
  for (i = p.stepen; i >= 0; i--)
54    scanf("%lf", &p.koef[i]);

56  /* Vraca se procitani polinom */
  return p;
58}

60double izracunaj(const Polinom * p, double x)
{
62  /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
       svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
64     vrednost odgovarajuceg koeficijenta */

66  /* Primer: Hornerov algoritam za polinom x^4+2x^3+3x^2+2x+1:
       x^4+2x^3+3x^2+2x+1 = (((x+2)*x + 3)*x + 2)*x + 1 */
68
  double rezultat = 0;
```

```
70    int i = p->stepen;
      for (; i >= 0; i--)
72      rezultat = rezultat * x + p->koef[i];
      return rezultat;
74  }

76  Polinom saberi(const Polinom * p, const Polinom * q)
    {
78    Polinom rez;
      int i;
80
      /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
82       stepenom */
      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
         sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
88       od stepena nekog od polaznih polinoma podrazumeva se da je
         koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
90       polinoma */
      for (i = 0; i <= rez.stepen; i++)
92      rez.koef[i] =
            (i > p->stepen ? 0 : p->koef[i]) +
94          (i > q->stepen ? 0 : q->koef[i]);

96    /* Vraca se dobijeni polinom */
      return rez;
98
    }
100
    Polinom pomnozi(const Polinom * p, const Polinom * q)
102 {
      int i, j;
104   Polinom r;

106   /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
        */
      r.stepen = p->stepen + q->stepen;
108   if (r.stepen > MAKS_STEPEN) {
        fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
110     exit(EXIT_FAILURE);
      }
112
      /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na nulu
        */
114   for (i = 0; i <= r.stepen; i++)
        r.koef[i] = 0;
116
      /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
118      za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
      for (i = 0; i <= p->stepen; i++)
```

```
120     for (j = 0; j <= q->stepen; j++)
          r.koef[i + j] += p->koef[i] * q->koef[j];
122
      /* Vraca se dobijeni polinom */
124   return r;
}
126
Polinom izvod(const Polinom * p)
128 {
      int i;
130   Polinom r;

132   /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
         polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
134      je rezultujuci polinom nula (izvod od konstante je nula). */
      if (p->stepen > 0) {
136     r.stepen = p->stepen - 1;

138     /* Racunanje koeficijenata rezultata na osnovu koeficijenata
           polaznog polinoma */
140     for (i = 0; i <= r.stepen; i++)
          r.koef[i] = (i + 1) * p->koef[i + 1];
142   } else
        r.koef[0] = r.stepen = 0;
144
      /* Vraca se dobijeni polinom */
146   return r;
}
148
Polinom n_izvod(const Polinom * p, int n)
150 {
      int i;
152   Polinom r;

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

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

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

170   /* Vraca se dobijeni polinom */
      return r;
```

```
}
```

main.c

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

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

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

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

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

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

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

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


  /* Racuna se n-ti izvoda polinoma i ispisuje se dobijeni polinoma
    */
  printf("Unesite izvod polinoma koji zelite:\n");
  scanf("%d", &n);
  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

```
#ifndef  _STAMPANJE_BITOVA_H
#define _STAMPANJE_BITOVA_H

#include <stdio.h>

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

/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja tipa 'short' u memoriji. */
void stampaj_bitove_short(short x);

/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   karaktera u memoriji. */
void stampaj_bitove_char(char x);

#endif
```

stampanje_bitova.c

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

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

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

  /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
     na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
     mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
     se jedini bit jedinica pomera udesno, kako bi se odredio naredni
     bit broja x koji je argument funkcije. Zatim se odgovarajuca
```

```
     cifra, ('0' ili '1'), ispisuje na standardnom izlazu. Neophodno
     je da promenljiva maska bude deklarisana kao neoznacen ceo broj
     kako bi se pomeranjem u desno vrsilo logicko pomeranje
     (popunjavanje nulama), a ne aritmeticko pomeranje (popunjavanje
     znakom broja). */
  for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
    putchar(x & maska ? '1' : '0');

  putchar('\n');
}

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

  /* Maska koja se koristi za "ocitavanje" bitova broja tipa short */
  unsigned short maska;

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

  putchar('\n');
}

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

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

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

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

main.c

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

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

  /* Ucitavanje broja tipa int */
```

```
  printf("Unesite broj tipa int: ");
  scanf("%x", &broj_int);

  /* Ispisivanje binarne reprezentacije unetog broja */
  printf("Binarna reprezentacija: ");
  stampaj_bitove(broj_int);

  /* Ucitavanje broja tipa short */
  printf("Unesite broj tipa short: ");
  scanf("%hx", &broj_short);

  /* Ispisivanje binarne reprezentacije unetog broja */
  printf("Binarna reprezentacija: ");
  stampaj_bitove_short(broj_short);

  /* Ucitavanje broja tipa char */
  printf("Unesite broj tipa char: ");
  scanf("%hhx", &broj_char);

  /* Ispisivanje binarne reprezentacije unetog broja */
  printf("Binarna reprezentacija: ");
  stampaj_bitove_char(broj_char);

  return 0;
}
```

Rešenje 1.6

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

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

  /* Formiranje se maska cija binarna reprezentacija izgleda
     100000...0000000, koja sluzi za ocitavanje bita najvece tezine.
     U svakoj iteraciji maska se pomera u desno za 1 mesto, i
     ocitavamo sledeci bit. Petlja se zavrsava kada vise nema
     jedinica tj. kada maska postane nula. */
  unsigned maska = 1 << broj_pomeranja;
  for (; maska != 0; maska >>= 1)
    x & maska ? br++ : 1;

  return br;
}

/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
```

```
   formiranjem odgovarajuce maske i pomeranjem promenljive x */
int prebroj_bitove_2(int x)
{
  int br = 0;
  unsigned broj_pomeranja = sizeof(int) * 8 - 1;

  /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1 bi
     vrsila aritmeticko pomeranje u desno, tj. popunjavanje bita
     najvece tezine bitom znaka. U tom slucaju nikad ne bi bio
     ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
     petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
     koja ocitava bit najvece tezine. */

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

  return br;
}

int main()
{
  int x, i;

  /* Ucitava se broj sa ulaza */
  printf("Unesite broj:\n");
  scanf("%x", &x);

  /* Dozvoljava se korisniku da bira na koji nacin ce biti izracunat
     broj jedinica u zapisu broja */
  printf("Unesite redni broj funkcije:\n");
  scanf("%d", &i);

  /* Ispisivanje rezultata */
  if (i == 1){
    printf("Poziva se funkcija prebroj_bitove_1\n");
    printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
  }else if (i == 2){
    printf("Poziva se funkcija prebroj_bitove_2\n");
    printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
  }else {
    fprintf(stderr, "Greska: Neodgovarajuci redni broj funkcije.\n");
    exit(EXIT_FAILURE);
  }

  exit(EXIT_SUCCESS);
}
```

Rešenje 1.7

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

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

/* Funkcija vraca najveci neoznaceni broj sastavljen od istih bitova
   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
unsigned najveci(unsigned x)
{
  unsigned velicina = sizeof(unsigned) * 8;

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

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

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

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

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

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

```
  }

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

int main()
{
  int broj;

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

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

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

  return 0;
}
```

Rešenje 1.8

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

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

/* Funckija postavlja na nulu n bitova pocev od pozicije p. */
unsigned postavi_0(unsigned x, unsigned n, unsigned p)
{
/*********************************************************************
  Formira se maska cija binarna reprezentacija ima n bitova
  postavljenih na 0 pocev od pozicije p, dok su svi ostali
  postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
  1111 1111 1111 1111 1111 1000 0011 1111
  To se postize na sledeci nacin:
  ~0                        1111 1111 1111 1111 1111 1111 1111 1111
  (~0 << n)                 1111 1111 1111 1111 1111 1111 1110 0000
  ~(~0 << n)                0000 0000 0000 0000 0000 0000 0001 1111
  (~(~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
*********************************************************************/
  unsigned maska = ~(~(~0 << n) << (p - n + 1));

  return x & maska;
}
```

```
/* Funckija postavlja na jedinicu n bitova pocev od pozicije p. */
unsigned postavi_1(unsigned x, unsigned n, unsigned p)
{

  /*********************************************************************
    Formira se maska kod koje je samo n bitova pocev od pocev od
    pozicije p jednako 1, a ostali su 0.
    Na primer, za n=5 i p=10 formira se maska oblika
    0000 0000 0000 0000 0000 0111 1100 0000
  *********************************************************************/
  unsigned maska = ~(~0 << n) << (p - n + 1);

  return x | maska;
}

/* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
   predstavlja n bitova pocev od pozicije p u binarnoj
   reprezentaciji broja x. */
unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
{

  /*********************************************************************
    Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
    Na primer, za n=5
    0000 0000 0000 0000 0000 0000 0001 1111
  *********************************************************************/
  unsigned maska = ~(~0 << n);

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

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

  /* 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 bitovi 1. */
  unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));

  /* 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 */
```

```
  unsigned x_postavi_0 = x & srednjih_n_0;

  /* U promenlijvu 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 bitovi
     osim najnizih n */
  unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);

  return x_postavi_0 ^ y_pomeri_srednje;
}

/* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
   njih n */
unsigned invertuj(unsigned x, unsigned n, unsigned p)
{
  /* Formira se maska sa n jedinica pocev od pozicije p. */
  unsigned maska = ~(~0 << n) << (p - n + 1);

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

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

  /* Ucitavaju se vrednosti sa standardnog ulaza */
  printf("Unesite neoznacen ceo broj x:\n");
  scanf("%u", &x);
  printf("Unesite neoznacen ceo broj n:\n");
  scanf("%u", &n);
  printf("Unesite neoznacen ceo broj p:\n");
  scanf("%u", &p);
  printf("Unesite neoznacen ceo broj y:\n");
  scanf("%u", &y);

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

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

```
  printf("\n");

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

  /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
     dobije kada se primeni funkcija postavi_1_n_bitova za x, n i p*/
  printf("x = %10u %36s = ", x, "");
  stampaj_bitove(x);
  printf("y = %10u %36s = ", y, "");
  stampaj_bitove(y);
  printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
  stampaj_bitove( postavi_1_n_bitova(x, n, p, y));
  printf("\n");

  /* Ispisuju se binarne reprezentacije broja x i broja koji se
     dobije kada se primeni funkcija invertuj za x, n i p*/
  printf("x = %10u %36s = ", x, "");
  stampaj_bitove(x);
  printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
  stampaj_bitove( invertuj(x, n, p));

  return 0;
}
```

Rešenje 1.9

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

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

/* Funkcija ceo broj x rotira u levo za n mesta. */
unsigned rotiraj_ulevo(int x, unsigned n)
{
  unsigned bit_najvece_tezine;

  /* Maska koja ima samo bit na poziciji najvece tezine postavljen na
     1 je neophodna da bi pre pomeranja u levo za 1 bit na poziciji
     najvece tezine bio sacuvan */
  unsigned bit_najvece_tezine_maska = 1 << (sizeof(unsigned) * 8 - 1)
    ;
  int i;

  /* n puta se vrsi rotaciju za jedan bit u levo. U svakoj iteraciji
     se odredi bit na poziciji najvece tezine, a potom se pomera
```

```
     binarna reprezentacija trenutne vrednosti promenljive x u levo
     za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
     na vrednost koju je imao bit na poziciji najvece tezine koji je
     istisnut pomeranjem */
  for (i = 0; i < n; i++) {
    bit_najvece_tezine = x & bit_najvece_tezine_maska;
    x = x << 1 | (bit_najvece_tezine ? 1 : 0);
  }

  /* Vraca se dobijena vrednost */
  return x;
}

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

  /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
     iteraciji se odredjuje bit na poziciji najmanje tezine broja x,
     zatim tako odredjeni bit se pomera u levo tako da bit na
     poziciji najmanje tezine dodje do pozicije najvece tezine.
     Zatim, nakon pomeranja binarne reprezentacije trenutne vrednosti
     promenljive x za 1 u desno, bit na poziciji najvece tezine se
     postaljva na vrednost vec zapamcenog bita koji je bio na
     poziciji najmanje tezine. */
  for (i = 0; i < n; i++) {
    bit_najmanje_tezine = x & 1;
    x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
  }

  /* Vraca se dobijena vrednost */
  return x;
}

/* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
   argument funkcije x oznaceni ceo broj */
int rotiraj_udesno_oznaceni(int x, unsigned n)
{
  unsigned bit_najmanje_tezine;
  int i;

  /* U svakoj iteraciji se odredjuje bit na poziciji najmanje tezine
     i smesta u promenljivu bit_najmanje_tezine. Kako je x oznacen
     ceo broj, tada se prilikom pomeranja u desno vrsi aritmeticko
     pomeranje i cuva se znak broja. Dakle, razlikuju se dva slucaja
     u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
     izvrsi pre petlje, s obzirom da rotiranjem u desno na poziciju
     nejvece tezine moze doci i 0 i 1, nezavisno od pocetnog znaka
     broja smestenog u promenljivu x. */
  for (i = 0; i < n; i++) {
```

```
    bit_najmanje_tezine = x & 1;

    if (x < 0)
  /*********************************************************************
    Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
    na poziciji najvece tezine. Na primer ako je x
    1010 1011 1100 1101 1110 0001 0010 001b
      (sa b je oznacen ili 1 ili 0 na poziciji najmanje tezine)
    Onda je sadrzaj promenljive bit_najmanje_tezine:
    0000 0000 0000 0000 0000 0000 0000 000b
    Nakon siftovanja sadrzaja promenljive x za 1 u desno
    1101 0101 1110 0110 1111 0000 1001 0001
    Kako bi umesto 1 na poziciji najvece tezine u trenutnoj binarnoj
    reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
    poziciju najvece tezine jer bi se time dobile 0, a u ovom slucaju
    su potrebne jedinice zbog bitovskog & zato se prvo vrsi
    komplementiranje, a zatim pomeranje
    ~bit_najmanje_tezine << (sizeof(int)*8 -1)
    B000 0000 0000 0000 0000 0000 0000 0000
    gde B oznacava ~b.
    Potom se ponovo vrsi komplementiranje kako bi se b nalazilo na
    poziciji najvece tezine i sve jedinice na ostalim pozicijama
    ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
    b111 1111 1111 1111 1111 1111 1111 1111
  *********************************************************************/
      x = (x >> 1) & ~(~bit_najmanje_tezine << (sizeof(int) * 8 - 1))
    ;
    else
      x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
  }

  /* Vraca se dobijena vrednost */
  return x;
}

int main()
{
  unsigned x, n;

  /* Ucitavanje vrednosti sa standardnog ulaza */
  printf("Unesite neoznacen ceo broj x:");
  scanf("%x", &x);
  printf("Unesite neoznacen ceo broj n:");
  scanf("%x", &n);

  /* Ispisivanje binarne reprezentacije broja x */
  printf("x\t\t\t\t= ");
  stampaj_bitove(x);

  /* Testiranje rada napisanih funkcija */
  printf("rotiraj_ulevo(%x,%u)\t\t= ", x, n);
  stampaj_bitove(rotiraj_ulevo(x, n));
```

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

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

130 return 0;
}
```

Rešenje 1.10

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

```
1  #include <stdio.h>
   #include "stampanje_bitova.h"
3
   /* Funkcija vraca vrednost cija je binarna reprezentacija slika u
5     ogledalu binarne reprezentacije broja x. */
   unsigned ogledalo(unsigned x)
7  {
     unsigned najnizi_bit;
9    unsigned rezultat = 0;

11   int i;
     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuce
13      vrednosti broja x se odredjuje i pamti u promenljivoj
        najnizi_bit, nakon cega se na promenljivu x primeni pomeranje u
15      desno */
     for (i = 0; i < sizeof(x) * 8; i++) {
17     najnizi_bit = x & 1;
       x >>= 1;
19     /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
          postavljeni bitovi dobijaju vecu poziciju. Novi bit se
21        postavlja na najnizu poziciju */
       rezultat <<= 1;
23     rezultat |= najnizi_bit;
     }
25
     /* Vraca se dobijena vrednost */
27   return rezultat;
   }
29
   int main()
31 {
     int broj;
33
     /* Ucitava se broj sa ulaza */
```

```
35    scanf("%x", &broj);

37    /* Ispisuje se njegova binarna reprezentacija */
      stampaj_bitove(broj);
39
      /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
41       funkcije ogledalo */
      stampaj_bitove(ogledalo(broj));
43
      return 0;
45  }
```

Rešenje 1.11

```
    #include <stdio.h>
2
    /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
4      jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
    int broj_01(unsigned int n)
6   {
      int broj_nula, broj_jedinica;
8     unsigned int maska;

10    broj_nula = 0;
      broj_jedinica = 0;
12
      /* Maska je inicijalizovana tako da moze da analizira bit najvece
14       tezine */
      maska = 1 << (sizeof(unsigned int) * 8 - 1);
16
      /* 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
         1 biti na svim pozicijama u binarnoj reprezentaciji maske */
20    while (maska != 0) {

22      /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
           1 i uveca se odgovarajuci brojac */
24      if (n & maska) {
          broj_jedinica++;
26      } else {
          broj_nula++;
28      }

30      /* Pomera se maska u desnu stranu */
        maska = maska >> 1;
32    }

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

```
38
int main()
40 {
   unsigned int n;
42
   /* Ucitavanje broja */
44 scanf("%u", &n);

46 /* Ispisivanje rezultata */
   printf("%d\n", broj_01(n));
48
   return 0;
50 }
```

Rešenje 1.12

```
#include <stdio.h>
2
/* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju u
4    binarnom zapisu celog čneoznaenog broja x */
int broj_parova(unsigned int x)
6 {
   int broj_parova;
8  unsigned int maska;

10 /* Vrednost promenljive koja predstavlja broj parova se
      inicijalizuje na 0 */
12 broj_parova = 0;

14 /* Postavlja se maska tako da moze da procitamo da li su dva
      najmanja bita u zapisu broja x 11 */
16 /* Binarna reprezentacija broja 3 je 000....00011 */
   maska = 3;
18
   while (x != 0) {
20   /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
     */
     if ((x & maska) == maska) {
22     broj_parova++;
     }
24
     /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
26      proveravao sledeci par bitova. Pomeranjem u desno bit najvece
        tezine se popunjava nulom jer je x neoznacen broj */
28   x = x >> 1;
   }
30
   /* Vraca se dobijena vrednost */
32 return broj_parova;
}
34
```

```
int main()
{
  unsigned int x;

  /* Ucitavanje broja */
  scanf("%u", &x);

  /* Ispisivanje rezultata */
  printf("%d\n", broj_parova(x));

  return 0;
}
```

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 <...>:
    00000000001101000100001111101<0101>
    u drugom koraku:
    00000000001101000100001111<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
       narednoj iteraciji posmatraju sledeca 4 bita */
    x = x >> 4;
  }

  /* Upisuje se terminirajuca nula */
  s[MAKS_DUZINA - 1] = '\0';
}

int main()
{
  unsigned int x;
  char s[MAKS_DUZINA];

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

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

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

  return 0;
}
```

Rešenje 1.17

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

/*****************************************************************
  Resenje linearne slozenosti:
  x^0 = 1
  x^k = x * x^(k-1)
*****************************************************************/
int stepen(int x, int k)
{
```

```
  if (k == 0)
    return 1;

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

/*****************************************************************
   Resenje logaritamske slozenosti:
   x^0 =1;
   x^k = x * (x^2 )^(k/2) , za neparno k
   x^k = (x^2)^(k/2) , za parno k
   Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
   se doslo do rezultata, i stoga je efikasnije.
*****************************************************************/
int stepen_2(int x, int k)
{
  if (k == 0)
    return 1;

  /* Ako je stepen paran */
  if ((k % 2) == 0)
    return stepen_2(x * x, k / 2);

  /* Inace (ukoliko je stepen neparan) */
  return x * stepen_2(x * x, k / 2);
}

int main()
{
  int x, k, ind;

  /* Ucitavanje rednog broja funkcije koja ce se primeniti */
  printf("Unesite redni broj funkcije (1/2):\n");
  scanf("%d", &ind);

  /* Ucitavanje vrednosti za x i k */
  printf("Unesite broj x:\n");
  scanf("%d", &x);
  printf("Unesite broj k:\n");
  scanf("%d", &k);

  /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
  if (x == 1)
    printf("%d\n", stepen(x, k));
  else if (x == 2)
    printf("%d\n", stepen_2(x, k));
  else {
    fprintf(stderr, "Greska: Neodgovarajuci redni broj funkcije.\n");
    exit(EXIT_FAILURE);
  }

```

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

Rešenje 1.18

```
#include <stdio.h>

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

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

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

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

int main()
{
  int n;

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

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

  return 0;
}
```

Rešenje 1.19

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

/* Pomocna funkcija koja izracunava n! * result. Koristi repnu
   rekurziju. Result je argument u kome se akumulira do tada
   izracunatu vrednost faktorijela. Kada dodje do izlaza iz rekurzije
   iz rekurzije potrebno je da vratimo result. */
int faktorijel_repna(int n, int result)
{
  if (n == 0)
    return result;

  return faktorijel_repna(n - 1, n * result);
}

/* U sledecim funkcijama je prikazan postupak oslobadjanja od
   repne rekurzije koja postoji u funkciji faktorijel_repna. */

/* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
   naredbama kojima se vrednost argumenta funkcije postavlja na
   vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
   goto naredbe za vracanje na pocetak tela funkcije. */
int faktorijel_repna_v1(int n, int result)
{
pocetak:
  if (n == 0)
    return result;

  result = n * result;
  n = n - 1;
  goto pocetak;
}

/* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
   praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
   iterativno resenje bez bezuslovnih skokova */
int faktorijel_repna_v2(int n, int result)
{
  while (n != 0) {
    result = n * result;
    n = n - 1;
  }

  return result;
}

/* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
   broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
   ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
   udobnosti korisnika, jer je sasvim prirodno da za faktorijel
```

```
   zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
   sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
   faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
   funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
   poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
   pozivom funkcije faktorijel_repna_v2. */
int faktorijel(int n)
{
  return faktorijel_repna(n, 1);
}

int main()
{
  int n;

  /* Ucitava se ceo broj */
  printf("Unesite n (<= 12): ");
  scanf("%d", &n);
  if (n > 12) {
    fprintf(stderr, "Greska: nedozvoljena vrednost za n!\n");
    exit(EXIT_FAILURE);
  }

  /* Ispisuje se rezultat */
  printf("%d! = %d\n", n, faktorijel(n));

  exit(EXIT_SUCCESS);
}
```

Rešenje 1.21

```
#include <stdio.h>

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

  if (n == 0)
    return 0;

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

  return f_1;
```

```
}

/* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
int f_rekurzivna(int n, int a, int b)
{
  /* Izlaz iz rekurzije */
  if (n == 0 || n == 1)
    return n;

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

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

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

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

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

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

  /* Rekurzivni pozivi */
  return f[n] =
      a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
```

```
}

int main()
{
  int n, a, b, ind;

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

  /* Ucitavaju se koeficijenti a i b */
  printf("Unesite koeficijente:\n");
  scanf("%d%d", &a, &b);

  /* Ucitava se broj n */
  printf("Unesite koji clan niza se racuna:\n");
  scanf("%d", &n);

  /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
     funkcije f_iterativna, f_rekurzivna ili f_napredna */
  if (ind == 0)
    printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
  else if (ind == 1)
    printf("F(%d) = %d\n", n, f_rekurzivna(n, a, b));
  else
    printf("F(%d) = %d\n", n, f_napredna(n, a, b));

  return 0;
}
```

Rešenje 1.22

```
#include <stdio.h>

/* Funkcija odredjuje zbir cifara zadatog broja x */
int zbir_cifara(unsigned int x)
{
  /* Izlazak iz rekurzije: ako je broj jednocifren */
  if (x < 10)
    return x;

  /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
     poslednje cifre + poslednja cifra tog broja */
  return zbir_cifara(x / 10) + x % 10;
}

int main()
{
```

```
  unsigned int x;

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

  /* Ispisivanje zbira cifara ucitanog broja */
  printf("%d\n", zbir_cifara(x));

  return 0;
}
```

Rešenje 1.23

```
#include <stdio.h>
#include <stdlib.h>
#define MAKS_DIM 100

/* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
   suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
   element niza. */
int suma_niza_1(int *a, int n)
{
  if (n <= 0)
    return 0;

  return suma_niza_1(a, n - 1) + a[n - 1];
}

/* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
   jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
   elementa niza i sume preostalih n-1 elementa. */
int suma_niza_2(int *a, int n)
{
  if (n <= 0)
    return 0;

  return a[0] + suma_niza_2(a + 1, n - 1);
}

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

  /* Ucitava se redni broj funkcije */
  printf("Unesite redni broj funkcije (1 ili 2):\n");
  scanf("%d", &ind);

  /* Ucitava se broj elemenata niza */
  printf("Unesite dimenziju niza:\n");
  scanf("%d", &n);
```

```
  /* Ucitava se n elemenata niza. */
  printf("Unesite elemente niza:\n");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
     funkcije suma_niza_1, ondosno suma_niza_2 */
  if (ind == 1)
    printf("Suma elemenata je %d\n", suma_niza_1(a, n));
  else if (ind == 2)
    printf("Suma elemenata je %d\n", suma_niza_2(a, n));
  else{
    fprintf(stderr, "Greska: Neodgovarajuci redni broj funkcije.\n");
    exit(EXIT_FAILURE);
  }

  exit(EXIT_SUCCESS);
}
```

Rešenje 1.24

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

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

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

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

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

  /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
     Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
     ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
     i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
```

```
30    int i = 0;
      while (scanf("%d", &brojevi[i]) != EOF) {
32      i++;
        if (i == MAKS_DIM)
34        break;
      }
36    n = i;

38    /* Stampa se maksimum unetog niza brojeva */
      printf("%d\n", maksimum_niza(brojevi, n));
40
      return 0;
42  }
```

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
13       dimenzije n primenom definicije skalarnog proizvoda
         a*b = a[0]*b[0] + a[1]*b[1] +...+  a[n-2]*a[n-2] + a[n-1]*a[n-1]
15       Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
         na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
17       polazna dva vektora otklanjanjem poslednjih elemenata, doda
         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      printf("Dimenzija mora biti prirodan broj <= %d!\n", MAKS_DIM);
        exit(EXIT_FAILURE);
35    }
```

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

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

  /* Ispisivanje rezultata skalarnog proizvoda dva ucitana niza */
  printf("Skalarni proizvod je %d\n", skalarno(a, b, n));

  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;

  /* Ispisuje se broj pojavljivanja */
  printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));

  return 0;
}
```

Rešenje 1.27

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

/* Funkcija koja proverava da li su tri zadata broja uzastopni
   clanovi niza */
int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
{
  /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i vraca
     se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi niza */
  if (n < 3)
    return 0;

  /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
     dovoljno je da su oni poslednja tri clana niza ili da se oni
     rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
  return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
      || tri_uzastopna_clana(x, y, z, a, n - 1);
}

int main()
{
  int x, y, z, a[MAKS_DIM];
  int n;

  /* Ucitavaju se tri cela broja za koje se ispituje da li su
     uzastopni clanovi niza */
  printf("Unesite tri cela broja:");
  scanf("%d%d%d", &x, &y, &z);

  /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
     Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
     ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
     i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
  printf("Unesite elemente niza:");
```

```
  int i = 0;
  while (scanf("%d", &a[i]) != EOF) {
    i++;
    if (i == MAKS_DIM)
      break;
  }
  n = i;

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

  return 0;
}
```

Rešenje 1.28

```
#include <stdio.h>

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

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

```
32
int main()
34 {
  int x;
36
  /* Ucitava se ceo broj */
38  scanf("%x", &x);

40  /* Ispisivanje rezultata */
  printf("%d\n", prebroj(x));
42
  return 0;
44 }
```

Rešenje 1.30

```
#include <stdio.h>
2
/* Rekurzivna funkcija za odredjivanje najvece oktalne cifre */
4 int maks_oktalna_cifra(unsigned x)
{
6   /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
       vrednost najvece oktalne cifre u broju 0 */
8   if (x == 0)
      return 0;
10
  /* Odredjivanje poslednje oktalne cifre u broju */
12  int poslednja_cifra = x & 7;

14  /* Odredjivanje maksimalne oktalne cifre 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 neoznacen ceo broj */
  scanf("%u", &x);
28
  /* Ispisuje se vrednost najvece oktalne cifre unetog broja */
30  printf("%d\n", maks_oktalna_cifra(x));

32  return 0;
}
```

Rešenje 1.31

```c
#include <stdio.h>

/* Rekurzivna funkcija za odredjivanje najvece heksadekadne cifre */
int maks_heksadekadna_cifra(unsigned x)
{
  /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
     vrednost najvece heksadekadne cifre u broju 0 */
  if (x == 0)
    return 0;

  /* Odredjivanje poslednje heksadekadne cifre u broju */
  int poslednja_cifra = x & 15;

  /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
     njega izbrise poslednja heksadekadna cifra */
  int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);

  return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
      : maks_bez_poslednje_cifre;
}

int main()
{
  unsigned x;

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

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

  return 0;
}
```

Rešenje 1.32

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

/* Niska moze imati najvise 31 karaktera + 1 za terminalnu nulu */
#define MAKS_DIM 32

/* Funkcija ispituje da li je zadata niska duzine n palindrom */
int palindrom(char s[], int n)
{
  /* Izlaz iz rekurzije - trivijalno, niska duzine 0 ili 1 je
     palindrom */
  if ((n == 1) || (n == 0))
```

```
    return 1;

  /* 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 */
  return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
}

int main()
{
  char s[MAKS_DIM];
  int n;

  /* Ucitavanje niske sa standardnog ulaza */
  scanf("%s", s);

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

  /* Ispisivanje da li je niska palindrom ili nije */
  if (palindrom(s, n))
    printf("da\n");
  else
    printf("ne\n");

  return 0;
}
```

Rešenje 1.33

```
#include <stdio.h>
#include <stdlib.h>
#define  MAKS_DUZINA_PERMUTACIJE 15

/* Funkcija koja ispisuje elemente niza a duzine n */
void ispisi_niz(int a[], int n)
{
  int i;

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

/* Funkcija koja vraca 1 ako se broj x nalazi u nizu a duzine n, a
   inace 0 */
int koriscen(int a[], int n, int x)
{
  int i;

  /* Obilaze se svi elementi niza */
```

```
22    for (i = 1; i <= n; i++)
        /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
24      if (a[i] == x)
          return 1;
26
    /* Zakljucuje se da broj nije pronadjen */
28    return 0;
  }
30
  /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija
32     kao argument niz a[] u koji se smesta permutacija, broj m oznacava
       da se u okviru tog poziva funkcije na m-tu poziciju u permutaciji
34     smesta jedan od preostalih celih brojeva, n je velicina skupa koji
       se permutuje. Funkciju se inicijalno poziva sa argumentom m = 1,
36     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 zavrsi, 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 zavrsava sa radom. Ukoliko takav broj postoji, onda se
         ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
58       ali sada sa drugacije postavljenim 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, trazeci broj koji
64         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 broja n i provera se da li je u odgovarajucem opsegu
        */
78    scanf("%d", &n);
      if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
80      fprintf(stderr,
                "Duzina permutacije mora biti broj iz intervala [0, %d]!\
        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.
8      Ukoliko je k strogo izmedju 0 i n, onda se koristi formula
       bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
10     koja se moze izvesti iz definicije binomnog koeficijenata */
    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 {
     return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
38 }

40 int main()
   {
42   int n, k, i, d, r;

44   /* Ucitavaju se brojevi d i r */
     scanf("%d %d", &d, &r);
46
     /* Ispisivanje Paskalovog trougla */
48   putchar('\n');
     for (n = 0; n <= d; n++) {
50     for (i = 0; i < d - n; i++)
         printf("  ");
52     for (k = 0; k <= n; k++)
         printf("%4d", binomni_koeficijent(n, k));
54     putchar('\n');
     }
56
     /* Provera da li je r nenegativan */
58   if (r < 0) {
       fprintf(stderr,
60             "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
       );
       exit(EXIT_FAILURE);
62   }

64   /* Ispisivanje sume elemenata hipotenuze */
     printf("%d\n", suma_elemenata_hipotenuze(r));
66
     exit(EXIT_SUCCESS);
68 }
```

2

Pokazivači

2.1 Pokazivačka aritmetika

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Primer 1

```
Interakcija sa programom:
 Unesite dimenziju niza:  3
 Unesite elemente niza:
 1 -2 3
 Nakon obrtanja elemenata, niz je:
 3 -2 1
 Nakon ponovnog obrtanja elemenata,
 niz je:
 3 -2 1
```

Primer 2

```
Interakcija sa programom:
 Unesite dimenziju niza:  0
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 učitanog niza.

Primer 1

```
Interakcija sa programom:
 Unesite dimenziju niza:  3
 Unesite elemente niza:
-1.1 2.2 3.3
 Zbir elemenata niza je 4.400.
 Proizvod elemenata niza je -7.986
 Minimalni element niza je -1.100
 Maksimalni element niza je 3.300
```

Primer 2

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

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: Neodgovarajuca dimenzija niza.
```

Primer 4

```
Interakcija sa programom:
 Unesite dimenziju niza:  101
Izlaz za greške:
 Greska: Neodgovarajuca dimenzija niza.
```

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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 zavrsavaju na ke

INTERAKCIJA SA PROGRAMOM:
 Broj reci koje se zavrsavaju na ke je 2.
```

Primer 2

```
POKRETANJE: ./a.out ulaz.txt sa -p

ULAZ.TXT
 Ovo je sadrzaj datoteke i u njoj ima
 reci koje pocinju sa sa

INTERAKCIJA SA PROGRAMOM:
 Broj reci koje pocinju na sa je 3.
```

Primer 3

```
POKRETANJE: ./a.out ulaz.txt sa -p

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
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 $n \times n$.

(a) Napisati funkciju koja izračunava trag matrice (sumu elemenata na glavnoj dijagonali).

(b) Napisati funkciju koja izračunava euklidsku normu matrice (koren sume kvadrata svih elemenata).

(c) Napisati funkciju koja izračunava gornju vandijagonalnu normu matrice (sumu apsolutnih vrednosti elemenata iznad glavne dijagonale).

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite broj vrsta matrice:  3
 Unesite elemente matrice 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: Neodgovarajuca dimenzija
 matrice.
```

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

(a) Napisati funkciju koja proverava da li su matrice jednake.

(b) Napisati funkciju koja izračunava zbir matrica.

(c) Napisati funkciju koja izračunava proizvod matrica.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati 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 $n \times n$.

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

(b) Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.

(c) Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.

(d) Napisati funkciju koja određuje broj negativnih elemenata matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice 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 dimenzija 3x3:
 1 2 3
 -4 -5 -6
 7 8 9
 Najveci element sporedne dijagonale je 7.
 Indeks kolone sa najmanjim elementom je 2.
 Indeks vrste sa najvecim elementom je 2.
 Broj negativnih elemenata matrice je 3.
```

Primer 2

```
POKRETANJE: ./a.out 4

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

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite broj vrsta matrice:  4
 Unesite elemente matrice 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 dimenzija $n \times m$.

(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 ($0 < n \leq 10$) matrice, a zatim i njene elemente. Na standardni izlaz spiralno ispisati elemente učitane matrice.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite broj vrsta i broj kolona
 matrice:
 3 3
 Unesite elemente matrice 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 alocira memoriju potrebnu za matricu dimenzija $n \times m$.

(b) Napisati funkciju `int **alociraj_kvadratnu_matricu(int n)` koja alocira memoriju za kvadratnu matricu dimenzije n.

(c) Napisati funkciju `int **dealociraj_matricu(int **A, int n)` koja dealocira memoriju matrice sa n vrsta. Povratna vrednost ove funkcije treba da bude "prazna" matrica.

(d) Napisati funkciju `void ucitaj_matricu(int **A, int n, int m)` koja učitava već alociranu matricu dimenzija $n \times m$ 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 dimenzija $n \times m$ 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 dimenzija $n \times m$ 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.

(j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzija $n \times m$ u već otvorenu datoteku `f`. U slučaju 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: Neodgovarajce dimenzije
 matrice.
```

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

Test 1

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

ULAZ.TXT
 4
 1 2 3 4
 5 6 7 8
 9 10 11 12
 13 14 15 16

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

Test 2

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

ULAZ.TXT
 dimenzija: 4
 1 2 3 4
 5 6 7 8
 9 10 11 12
 13 14 15 16

IZLAZ ZA GREŠKE:
 Greska: Neispravan pocetak
     datoteke.
```

Test 3

```
POKRETANJE: ./a.out

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

Zadatak 2.20 Data je celobrojna matrica dimenzija $n \times m$. 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 dimenzija $n \times m$ 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 $n \times n$.

(a) Napisati funkciju koja izračunava zbir apsolutnih vrednosti matrice ispod sporedne dijagonale.

(b) Napisati funkciju koja menja sadržaj matrice tako što polovi elemente iznad glavne dijagonale, duplira elemente ispod glavne dijagonale, dok elemente na glavnoj dijagonali ostavlja nepromenjene.

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

Primer 1

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

Primer 1

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

MATRICE.TXT
 3
 1.1 -2.2 3.3
 -4.4 5.5 -6.6
 7.7 -8.8 9.9
 -1.1 2.2 -3.3
 4.4 -5.5 6.6
 -7.7 8.8 -9.9
```

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

Zadatak 2.24 Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju 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 slicica.
Reprezentacija za koju je sakupio
najmanji broj slicica 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
cetvorougla.
Obim je 8.
Povrsina je 4.
```

Primer 2

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

INTERAKCIJA 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 argumentomo, odnosno izračunava i ispisuje vrednosti date funkcije u n ekvidistantnih tačaka na intervalu $[a, b]$. Re-

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):

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

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:

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

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

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

#define MAX 100

/* Funkcija obrce elemente niza koriscenjem indekse sintakse */
void obrni_niz_v1(int a[], int n)
{
  int i, j;

  for (i = 0, j = n - 1; i < j; i++, j--) {
    int t = a[i];
    a[i] = a[j];
    a[j] = t;
  }
```

```
16 }

18 /* Funkcija obrce elemente niza koriscenjem pokazivacke sintakse */
   void obrni_niz_v2(int *a, int n)
20 {
     /* Pokazivaci na elemente niza */
22   int *prvi, *poslednji;

24   /* Vrsi se obrtanje niza */
     for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
26     int t = *prvi;

28     /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
          vrednost koja se nalazi na adresi na koju pokazuje pokazivac
30        "poslednji". Nakon toga se pokazivac "prvi" uvecava za jedan
          sto za posledicu ima da "prvi" pokazuje na sledeci element u
32        nizu */
       *prvi++ = *poslednji;
34
       /* Vrednost promenljive "t" se postavlja na adresu na koju
36        pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
          umanjuje za jedan, cime pokazivac "poslednji" pokazuje
38        na element koji mu prethodi u nizu */
       *poslednji-- = t;
40   }

42   /*****************************************************************
       Drugi nacin za obrtanje niza
44
       for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
46                                             prvi++, poslednji--) {
         int t = *prvi;
48       *prvi = *poslednji;
         *poslednji = t;
50     }
     *****************************************************************
       */
52 }

54 int main()
   {
56   /* Deklarise se niz od najvise MAX elemenata */
     int a[MAX];
58
     /* Broj elemenata niza a */
60   int n;

62   /* Pokazivac na elemente niza */
     int *p;
64
     printf("Unesite dimenziju niza: ");
66   scanf("%d", &n);
```

```
68    /* Proverava se da li je doslo do prekoracenja ogranicenja
         dimenzije */
70    if (n <= 0 || n > MAX) {
        fprintf(stderr, "Greska: Neodgovarajuca dimenzija niza.\n");
72      exit(EXIT_FAILURE);
      }
74
      printf("Unesite elemente niza:\n");
76    for (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++)
84      printf("%d ", *p);
      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++)
92      printf("%d ", *p);
      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)
```

```
{
  double p = 1;

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

  return p;
}

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

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

  return min;
}

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

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

  return max;
}


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

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

  /* Proverava se da li je doslo do prekoracenja ogranicenja
     dimenzije */
  if (n <= 0 || n > MAX) {
```

```
    fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72    exit(EXIT_FAILURE);
  }
74
  printf("Unesite elemente niza:\n");
76  for (i = 0; i < n; i++)
    scanf("%lf", a + i);
78
  /* Vrsi se testiranje definisanih funkcija */
80  printf("Zbir elemenata niza je %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
8    ima neparan broj elemenata, srednji element ostaje nepromenjen */
void povecaj_smanji(int *a, int n)
10{
  int *prvi = a;
12  int *poslednji = a + n - 1;

14  while (prvi < poslednji) {

16    /* Uvecava se element na koji pokazuje pokazivac "prvi" */
    (*prvi)++;
18
    /* Pokazivac "prvi" se pomera na sledeci element */
20    prvi++;

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

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

30  /*****************************************************************
   Drugi nacin:
32   while (prvi < poslednji) {
```

```
      (*prvi++)++;
      (*poslednji--)--;
    }
  ****************************************************************/
}

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

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

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

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

  povecaj_smanji(a, n);

  printf("Transformisan niz je:\n");
  for (p = a; p - a < n; p++)
    printf("%d ", *p);
  printf("\n");

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.4

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;
  char tip_ispisa;

  printf("Broj argumenata komandne linije je %d.\n", argc);

  printf("Kako zelite da ispisete argumente, koriscenjem"
         " indeksne ili pokazivacke sintakse (I ili P)? ");
  scanf("%c", &tip_ispisa);
```

```
  printf("Argumenti komandne linije su:\n");
  if (tip_ispisa == 'I') {
    /* Ispisuju se argumenti komandne linije koriscenjem indeksne
       sintakse */
    for (i = 0; i < argc; i++)
      printf("%d %s\n", i, argv[i]);
  } else if (tip_ispisa == 'P') {
    /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
       sintakse */
    i = argc;
    for (; argc > 0; argc--)
      printf("%d %s\n", i - argc, *argv++);

    /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje na
       polje u memoriji koje se nalazi iza poslednjeg argumenta
       komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
       broja argumenta komandne linije to sada moze ponovo da se
       postavi "argv" da pokazuje na nulti argument komandne linije
    */
    argv = argv - i;
    argc = i;
  }

  printf("Pocetna slova argumenata komandne linije:\n");
  if (tip_ispisa == 'I') {
    /* koristeci indeksnu sintaksu */
    for (i = 0; i < argc; i++)
      printf("%c ", argv[i][0]);
    printf("\n");
  } else if (tip_ispisa == 'P') {
    /* koristeci pokazivacku sintaksu */
    for (i = 0; i < argc; i++)
      printf("%c ", **argv++);
    printf("\n");
  }

  return 0;
}
```

Rešenje 2.5

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

#define MAX 100

/* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
   cita spreda i odpozadi */
int palindrom(char *niska)
{
  int i, j;
```

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

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

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

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

  return 0;
}
```

Rešenje 2.6

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

#define MAX_KARAKTERA 100

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

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

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

```

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

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

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

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

  /* Zatvaranje datoteke */
  fclose(in);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.7

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

#define MAX_KARAKTERA 100

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

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

/* Implementacija funkcije strlen() iz standardne biblioteke */
```

```
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 funkcije
33    sufiks niske zadate prvi argumentom funkcije */
   int sufiks_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, sufiks) == 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 ucitavaju
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 zavrsavaju na %s je %d.\n", *(argv + 2)
        ,
91              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 pocinju 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
    return sqrt(norma);
27  }

29  /* Funkcija izracunava gornju vandijagonalnu normu matrice */
    int gornja_vandijagonalna_norma(int M[][MAX], int n)
31  {
    int norma = 0;
33  int i, j;

35  for (i = 0; i < n; i++) {
      for (j = i + 1; j < n; j++)
37      norma += abs(M[i][j]);
    }
39
    return norma;
41  }

43  int main()
    {
45  int A[MAX][MAX];
    int i, j, n;
47
    printf("Unesite broj vrsta matrice: ");
49  scanf("%d", &n);

51  /* Provera prekoracenja dimenzija matrice */
    if (n > MAX || n <= 0) {
53    fprintf(stderr, "Greska: Neodgovarajuca dimenzija matrice.\n");
      exit(EXIT_FAILURE);
55  }

57  printf("Unesite elemente matrice po vrstama:\n ");
    for (i = 0; i < n; i++)
59    for (j = 0; j < n; j++)
        scanf("%d", &A[i][j]);
61
    /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
63  for (i = 0; i < n; i++) {
      /* Ispis elemenata i-te vrste */
65    for (j = 0; j < n; j++)
        printf("%d ", A[i][j]);
67    printf("\n");
    }
69
    /*************************************************************
71   Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
     Kod ovako definisane matrice, elementi su uzastopno smesteni u
73   memoriju, kao na traci. To znaci da su svi elementi prve vrste
     redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
```

```
   prve vrste smesten je prvi element druge vrste za kojim slede
   svi elementi te vrste i tako dalje redom.

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

  /* Ispisivanje rezultata na standardni izlaz */
  int tr = trag(A, n);
  printf("Trag matrice je %d.\n", tr);

  printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
    ;
  printf("Vandijagonalna norma matrice je = %d.\n",
         gornja_vandijagonalna_norma(A, n));

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.9

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

#define MAX 100

/* 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");
  }
```

```
}

/* Funkcija proverava da li su zadate kvadratne matrice a i b
   dimenzije n jednake */
int jednake_matrice(int a[][MAX], int b[][MAX], int n)
{
  int i, j;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (a[i][j] != b[i][j])
        return 0;

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

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

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      c[i][j] = a[i][j] + b[i][j];
}

/* Funkcija izracunava proizvod dve kvadratne matice */
void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
{
  int i, j, k;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) {
      /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
      c[i][j] = 0;
      for (k = 0; k < n; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
}

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

  /* Matrice zbira i proizvoda */
  int zbir[MAX][MAX], proizvod[MAX][MAX];

  /* Dimenzija kvadratnih matrica */
  int n;
```

```
  printf("Unesite broj vrsta matrica:\n");
  scanf("%d", &n);

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

  printf("Unesite elemente prve matrice po vrstama:\n");
  ucitaj_matricu(a, n);
  printf("Unesite elemente druge matrice po vrstama:\n");
  ucitaj_matricu(b, n);

  /* Izracunava se zbir i proizvod matrica */
  saberi(a, b, zbir, n);
  pomnozi(a, b, proizvod, n);

  /* Ispisuje se rezultat */
  if (jednake_matrice(a, b, n) == 1)
    printf("Matrice su jednake.\n");
  else
    printf("Matrice nisu jednake.\n");

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.10

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

#define MAX 64

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

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

```
      return 0;
  }

  return 1;
}

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

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

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

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

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

  return 1;
}

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

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

```
      zatvorenje[i][j] = m[i][j];

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (zatvorenje[i][j] == 1)
        zatvorenje[j][i] = 1;
}


/* Funkcija proverava da li je relacija tranzitivna. Relacija je
   tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
   relaciji sa elementom "j" i element "j" u relaciji sa elementom
   "k", onda je i element "i" u relaciji sa elementom "k" */
int tranzitivnost(int m[][MAX], int n)
{
  int i, j, k;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      /* Ispituje se da li postoji element koji narusava *
         tranzitivnost */
      for (k = 0; k < n; k++)
        if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
          return 0;

  return 1;
}


/* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
   relacije koriscenjem Varsalovog algoritma */
void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
{
  int i, j, k;

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

  /* Odredjuje se reflektivno zatvorenje matrice */
  for (i = 0; i < n; i++)
    zatvorenje[i][i] = 1;

  /* Primenom Varsalovog algoritma odredjuje se tranzitivno
     zatvorenje matrice */
  for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
        if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
            && (zatvorenje[i][j] == 0))
          zatvorenje[i][j] = 1;
```

```
}

/* Funkcija ispisuje elemente matrice */
void pisi_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");
  }
}

int main(int argc, char *argv[])
{
  FILE *ulaz;
  int m[MAX][MAX];
  int pomocna[MAX][MAX];
  int n, i, j;

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

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

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

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

  /* Ucitava se element po element matrice */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      fscanf(ulaz, "%d", &m[i][j]);

```

```
  /* Ispisuje se rezultat */
  printf("Relacija %s refleksivna.\n",
         refleksivnost(m, n) == 1 ? "jeste" : "nije");

  printf("Relacija %s simetricna.\n",
         simetricnost(m, n) == 1 ? "jeste" : "nije");

  printf("Relacija %s tranzitivna.\n",
         tranzitivnost(m, n) == 1 ? "jeste" : "nije");

  printf("Refleksivno zatvorenje relacije:\n");
  ref_zatvorenje(m, n, pomocna);
  pisi_matricu(pomocna, n);

  printf("Simetricno zatvorenje relacije:\n");
  sim_zatvorenje(m, n, pomocna);
  pisi_matricu(pomocna, n);

  printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
  ref_tran_zatvorenje(m, n, pomocna);
  pisi_matricu(pomocna, n);

  /* Zatvaranje datoteke */
  fclose(ulaz);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.11

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

#define MAX 32

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

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

  return max_na_sporednoj_dijagonali;
}

/* Funkcija izracunava indeks kolone najmanjeg elementa */
```

```
22 int indeks_min(int m[][MAX], int n)
   {
24   int i, j;
     int min = m[0][0], indeks_kolone = 0;
26
     for (i = 0; i < n; i++)
28     for (j = 0; j < n; j++)
         if (m[i][j] < min) {
30         min = m[i][j];
           indeks_kolone = j;
32       }

34   return indeks_kolone;
   }
36
   /* Funkcija izracunava indeks vrste najveceg elementa */
38 int indeks_max(int m[][MAX], int n)
   {
40   int i, j;
     int max = m[0][0], indeks_vrste = 0;
42
     for (i = 0; i < n; i++)
44     for (j = 0; j < n; j++)
         if (m[i][j] > max) {
46         max = m[i][j];
           indeks_vrste = i;
48       }
     return indeks_vrste;
50 }

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

58   for (i = 0; i < n; i++)
       for (j = 0; j < n; j++)
60       if (m[i][j] < 0)
           broj_negativnih++;
62
     return broj_negativnih;
64 }

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

72   /* Proverava se broj argumenata komandne linije */
     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
      if (n > MAX || n <= 0) {
84      fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
        fprintf(stderr, "matrice.\n");
86      exit(EXIT_FAILURE);
      }
88
      /* Ucitava se matrica */
90    printf("Unesite elemente matrice dimenzija %dx%d:\n", n, n);
      for (i = 0; i < n; i++)
92      for (j = 0; j < n; j++)
          scanf("%d", &m[i][j]);
94
      printf("Najveci element sporedne dijagonale je %d.\n",
96           max_sporedna_dijagonala(m, n));

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

104   printf("Broj negativnih elemenata matrice je %d.\n",
           broj_negativnih(m, n));
106
      exit(EXIT_SUCCESS);
108 }
```

Rešenje 2.12

```
1   #include <stdio.h>
    #include <stdlib.h>
3
    #define MAX 32
5
    /* Funkcija ucitava elemente kvadratne matrice dimenzije n
7      sa standardnog ulaza */
    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]);
}

/* 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");
  }
}

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

  /* Ispituje se uslov normiranosti */
  for (i = 0; i < n; i++) {
    proizvod = 0;

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

    if (proizvod != 1)
      return 0;
  }

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

      proizvod = 0;

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

      if (proizvod != 0)
        return 0;
    }
  }

  /* Ako su oba uslova ispunjena, matrica je ortonormirana */
```

```
  return 1;
}

int main()
{
  int A[MAX][MAX];
  int n;

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

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++) {
```

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

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

40    /* Ispituje se uslov normiranosti */
      for (i = 0; i < n; i++) {
42      proizvod = 0;

44      for (j = 0; j < n; j++)
          proizvod += m[i][j] * m[i][j];
46
        if (proizvod != 1)
48        return 0;
      }
50
      /* Ispituje se uslov ortogonalnosti */
52    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
54
          proizvod = 0;
56
          for (k = 0; k < n; k++)
58          proizvod += m[i][k] * m[j][k];

60        if (proizvod != 0)
            return 0;
62      }
      }
64
      /* Ako su oba uslova ispunjena, matrica je ortonormirana */
66    return 1;
    }
68
    int main()
70  {
      int A[MAX][MAX];
72    int n;

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

```
76
   if (n > MAX || n <= 0) {
78   fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
     fprintf(stderr, "matrice.\n");
80   exit(EXIT_FAILURE);
   }
82
   printf("Unesite elemente matrice po vrstama:\n");
84 ucitaj_matricu(A, n);

86 printf("Matrica %s ortonormirana.\n",
          ortonormirana(A, n) ? "je" : "nije");
88
   exit(EXIT_SUCCESS);
90 }
```

Rešenje 2.15

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   int main()
5  {
     int *p = NULL;
7    int i, n;

9    printf("Unesite dimenziju niza: ");
     scanf("%d", &n);
11
     /* Alocira se prostor za n celih brojeva */
13   if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
       fprintf(stderr, "Greska: Neupela alokacija memorije.\n");
15     exit(EXIT_FAILURE);
     }
17
     printf("Unesite elemente niza: ");
19   for (i = 0; i < n; i++)
       scanf("%d", &p[i]);
21
     printf("Niz u obrnutom poretku je: ");
23   for (i = n - 1; i >= 0; i--)
       printf("%d ", p[i]);
25   printf("\n");

27   /* Oslobadja se prostor rezervisan funkcijom malloc() */
     free(p);
29
     exit(EXIT_SUCCESS);
31 }
```

Rešenje 2.16

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

#define KORAK 10

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

  /* Velicina alocirane memorije */
  int alocirano;

  /* Broj elemenata niza */
  int n;

  /* Broj koji se ucitava sa ulaza */
  int x;
  int i;
  int *b = NULL;
  char realokacija;

  /* Inicijalizacija */
  alocirano = n = 0;

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

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

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

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

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

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

```

```
        free(a);

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

        /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
           "a" bude inicijalizovano na NULL */
        a = (int *) realloc(a, alocirano * sizeof(int));
        if (a == NULL) {
          fprintf(stderr, "Greska: Neupesna realokacija memorije.\n")
    ;
          exit(EXIT_FAILURE);
        }
      }
    }

    a[n++] = x;

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

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

  /* Oslobadja se dinamicki alocirana memorija */
  free(a);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.17

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

#define MAX 1000

/* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
   nadovezivanja niski. Adresa kreiranog niza se vraca kao
   povratna vrednost. */
char *nadovezi(char *s, char *t)
{
  char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
                            * sizeof(char));

```

```
  /* Proverava se da li je memorija uspesno alocirana */
  if (p == NULL) {
    fprintf(stderr, "Greska: Neuspesna alokacija memorije.\n");
    exit(EXIT_FAILURE);
  }

  /* Kopiraju se i nadovezuju niske karaktera */
  strcpy(p, s);
  strcat(p, t);

  return p;
}

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

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.18

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

int main()
{
  int i, j;

  /* Pokazivac na niz vrsta matrice realnih brojeva */
  double **A = NULL;

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

  /* Trag matice */
```

```
  double trag = 0;

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

  /* čDinamiki se alocira prostor za niz vrsta matrice */
  A = (double **)malloc(sizeof(double *) * n);

  /* Provera se da li je doslo do greske pri alokaciji */
  if (A == NULL) {
    fprintf(stderr, "Greska: Neupesna alokacija memorije.\n");
    exit(EXIT_FAILURE);
  }

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

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

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

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

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.19

matrica.h

```
#ifndef _MATRICA_H_
#define _MATRICA_H_ 1

/* Funkcija dinamicki alocira memoriju za matricu dimenzije n x m */
int **alociraj_matricu(int n, int m);

/* Funkcija dinamicki alocira memoriju za kvadratnu matricu dimenzije
   n x n */
int **alociraj_kvadratnu_matricu(int n);

/* Funkcija oslobadja memoriju za matricu sa n vrsta */
int **dealociraj_matricu(int **matrica, int n);

/* Funkcija ucitava vec alociranu matricu dimenzije n x m sa
   standardnog ulaza */
void ucitaj_matricu(int **matrica, int n, int m);

/* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
   standardnog ulaza */
void ucitaj_kvadratnu_matricu(int **matrica, int n);

/* Funkcija ispisuje matricu dimenzije n x m na standardnom izlazu */
void ispisi_matricu(int **matrica, int n, int m);

/* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
   izlazu */
void ispisi_kvadratnu_matricu(int **matrica, int n);

/* Funkcija ucitava vec alociranu matricu dimenzije n x m iz datoteke
   f */
int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
    ;

/* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n x n
   iz datoteke f */
int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
                                         FILE * f);

/* Funkcija upisuje matricu dimenzije n x m u datoteku f */
int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);

/* Funkcija upisuje kvadratnu matricu dimenzije n x n u datoteku f */
int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
                                       FILE * f);

#endif
```

matrica.c

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

int **alociraj_matricu(int n, int m)
{
  int **matrica = NULL;
  int i, j;

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

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

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

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

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

```
51 void ucitaj_matricu(int **matrica, int n, int m)
   {
53   int i, j;
     /* Elementi matrice se ucitacaju po vrstama */
55   for (i = 0; i < n; i++)
       for (j = 0; j < m; j++)
57       scanf("%d", &matrica[i][j]);
   }
59
   void ucitaj_kvadratnu_matricu(int **matrica, int n)
61 {
     /* Ucitavanje matrice n x n */
63   ucitaj_matricu(matrica, n, n);
   }
65
   void ispisi_matricu(int **matrica, int n, int m)
67 {
     int i, j;
69   /* Ispis po vrstama */
     for (i = 0; i < n; i++) {
71     for (j = 0; j < m; j++)
         printf("%d ", matrica[i][j]);
73     printf("\n");
     }
75 }

77 void ispisi_kvadratnu_matricu(int **matrica, int n)
   {
79   /* Ispis matrice n x n */
     ispisi_matricu(matrica, n, n);
81 }

83 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
   {
85   int i, j;
     /* Elementi matrice se ucitacaju po vrstama */
87   for (i = 0; i < n; i++)
       for (j = 0; j < m; j++)
89       /* Ako je nemoguce ucitati sledeci element, povratna vrednost
            funkcije je 1, kao indikator neuspesnog ucitavanja */
91       if (fscanf(f, "%d", &matrica[i][j]) != 1)
           return 1;
93
     /* Uspesno ucitana matrica */
95   return 0;
   }
97
   int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
99                                            FILE * f)
   {
101  /* Ucitavanje matrice n x n iz datoteke */
```

```
  return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
}

int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
{
  int i, j;
  /* Ispis po vrstama */
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
      /* Ako je nemoguce ispisati sledeci element, povratna vrednost
         funkcije je 1, kao indikator neuspesnog ispisa */
      if (fprintf(f, "%d ", matrica[i][j]) <= 0)
        return 1;
    fprintf(f, "\n");
  }

  /* Uspesno upisana matrica */
  return 0;
}

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

main_a.c

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

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

  /* Ucitavanje dimenzije matrice */
  printf("Unesite broj vrsta matrice: ");
  scanf("%d", &n);
  printf("Unesite broj kolona matrice: ");
  scanf("%d", &m);

  /* Provera dimenzija matrice */
  if (n <= 0 || m <= 0) {
    fprintf(stderr, "Greska: Neodgovarajce dimenzije matrice.\n");
    exit(EXIT_FAILURE);
  }

```

```
  /* Alokacija matrice i provera alokacije */
  matrica = alociraj_matricu(n, m);
  if (matrica == NULL) {
    fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje matrice sa standardnog ulaza */
  printf("Unesite elemente matrice po vrstama:\n");
  ucitaj_matricu(matrica, n, m);

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

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

  /* Zatvaranje datoteke */
  fclose(f);

  /* Oslobadjanje memorije koju je zauzimala matrica */
  matrica = dealociraj_matricu(matrica, n);

  exit(EXIT_SUCCESS);
}
```

main_b.c

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

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

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

```
    exit(EXIT_FAILURE);
  }

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

  /* Ucitavanje dimenzije matrice */
  if (fscanf(f, "%d", &n) != 1) {
    fprintf(stderr, "Greska: Neispravan pocetak datoteke.\n");
    exit(EXIT_FAILURE);
  }

  /* Provera dimenzije matrice */
  if (n <= 0) {
    fprintf(stderr, "Greska: Neodgovarajca dimenzija matrice.\n");
    exit(EXIT_FAILURE);
  }

  /* Alokacija matrice i provera alokacije */
  matrica = alociraj_kvadratnu_matricu(n);
  if (matrica == NULL) {
    fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje matrice iz datoteke */
  if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
    fprintf(stderr, "Greska: Neuspesno ucitavanje matrice iz datoteke
    .\n");
    matrica = dealociraj_matricu(matrica, n);
    exit(EXIT_FAILURE);
  }

  /* Zatvaranje datoteke */
  fclose(f);

  /* Ispis matrice na standardnom izlazu */
  ispisi_kvadratnu_matricu(matrica, n);

  /* Oslobadjanje memorije koju je zauzimala matrica */
  matrica = dealociraj_matricu(matrica, n);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.20

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrica.h"

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

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

int main()
{
  int m, n;
  int **matrica = NULL;

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

  /* Alocira se matrica */
  matrica = alociraj_matricu(n, m);
  /* Provera alokacije */
  if (matrica == NULL) {
    fprintf(stderr, "Greska: Neuspesna alokacija matrice.\n");
    exit(EXIT_FAILURE);
  }

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

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

  /* Oslobadjanje memorije */
  matrica = dealociraj_matricu(matrica, n);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.22

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

```

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

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

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

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

  return zbir;
}

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

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

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

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

```
}

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

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

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

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

/* Funckija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
   n x n */
void oslobodi_memoriju(float **m, int n)
{
  int i;

  for (i = 0; i < n; i++)
    free(m[i]);
  free(m);
}

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

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

```
  }

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

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

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

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

  float zbir = zbir_ispod_sporedne_dijagonale(a, n);

  /* Poziva se funkcija za transformaciju matrice */
  izmeni(a, n);

  /* Ispisivanje rezultata */
  printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
  printf("je %.2f.\n", zbir);

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

  /* Oslobadja se memorija */
  oslobodi_memoriju(a, n);

  /* Zatvara se datoteka */
  fclose(ulaz);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.27

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

/* Funkcija tabela() prihvata granice intervala a i b, broj
   ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
   funkciju koja prihvata double argument, i vraca double vrednost.
   Za tako datu funkciju ispisuju se njene vrednosti u intervalu
```

```
   [a,b] u n ekvidistantnih tacaka intervala */
void tabela(double a, double b, int n, double (*fp) (double))
{
  int i;
  double x;

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

double sqr(double a)
{
  return a * a;
}

int main(int argc, char *argv[])
{
  double a, b;
  int n;

  char ime_funkcije[6];

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

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

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

  /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
  if (strcmp(ime_funkcije, "sin") == 0)
    fp = &sin;
  else if (strcmp(ime_funkcije, "cos") == 0)
    fp = &cos;
  else if (strcmp(ime_funkcije, "tan") == 0)
    fp = &tan;
  else if (strcmp(ime_funkcije, "atan") == 0)
```

```
    fp = &atan;
  else if (strcmp(ime_funkcije, "acos") == 0)
    fp = &acos;
  else if (strcmp(ime_funkcije, "asin") == 0)
    fp = &asin;
  else if (strcmp(ime_funkcije, "exp") == 0)
    fp = &exp;
  else if (strcmp(ime_funkcije, "log") == 0)
    fp = &log;
  else if (strcmp(ime_funkcije, "log10") == 0)
    fp = &log10;
  else if (strcmp(ime_funkcije, "sqrt") == 0)
    fp = &sqrt;
  else if (strcmp(ime_funkcije, "floor") == 0)
    fp = &floor;
  else if (strcmp(ime_funkcije, "ceil") == 0)
    fp = &ceil;
  else if (strcmp(ime_funkcije, "sqr") == 0)
    fp = &sqr;
  else {
    printf("Program jos uvek ne podrzava trazenu funkciju!\n");
    exit(EXIT_SUCCESS);
  }

  printf("Unesite krajeve intervala:\n");
  scanf("%lf %lf", &a, &b);

  printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
  printf("(ukljucujuci krajeve intervala)?\n");
  scanf("%d", &n);

  /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
     uneti broj veci od 2 */
  if (n < 2) {
    fprintf(stderr, "Greska: Broj tacaka mreze mora biti bar 2!\n");
    exit(EXIT_FAILURE);
  }

  /* Ispisivanje imena funkcije */
  printf("     x %10s(x)\n", ime_funkcije);

  /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
     komandne linije */
  tabela(a, b, n, fp);

  exit(EXIT_SUCCESS);
}
```

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`.

Test 1

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

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

VREMENA.TXT

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

Test 2

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

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

VREMENA.TXT

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

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

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 složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

Primer 1

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

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

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

Primer 2

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

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

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

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
```

Test 3

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

DAT.TXT
 12 53
 2.342 34.1
 -0.3 0.23
 -1 2.1
 123.5 756.12

IZLAZ:
 -0.3 0.23
```

Zadatak 3.6 Napisati funkciju koja određuje nulu funkcije `cos(x)` na intervalu $[0, 2]$ metodom polovljenja intervala. Algoritam se završava kada se vrednost kosinusne funkcije razlikuje za najviše 0.001 od nule. UPUTSTVO: *Korisiti algoritam analogan algoritmu binarne pretrage, metod polovljenja intervala.* NAPOMENA: *Ovaj metod se može primeniti na funkciju* `cos(x)` *na intervalu* $[0, 2]$ *zato što je ona na ovom intervalu neprekidna, i vrednosti funkcije na krajevima intervala su različitog znaka.*

Test 1

```
IZLAZ:
 1.57031
```

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 agrument 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.

Test 1

```
Ulaz:
 -151 -44 5 12 13 15

Izlaz:
 2
```

Test 2

```
Ulaz:
 -100 -15 -11 -8 -7 -5

Izlaz:
 -1
```

Test 3

```
Ulaz:
 -100 -15 0 13 55 124
 258 315 516 7000

Izlaz:
 3
```

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.

Test 1

```
Ulaz:
 151 44 5 -12 -13 -15

Izlaz:
 3
```

Test 2

```
Ulaz:
 100 55 15 0 -15 -124
 -155 -258 -315 -516

Izlaz:
 4
```

Test 3

```
Ulaz:
 100 15 11 8 7 5 4 3 2

Izlaz:
 -1
```

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

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

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

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

Test 1

```
Ulaz:
 4

Izlaz:
 2 2
```

Test 2

```
Ulaz:
 17

Izlaz:
 4 4
```

Test 3

```
Ulaz:
 1031

Izlaz:
 10 10
```

*** 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 $0 \leq \alpha \leq 90°$, na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom α jednak (neke duži bivaju presečene, a neke ne). Program prvo učitava broj N, a zatim i same koordinate temena duži. Uputstvo: *Vršiti binarnu pretragu intervala* $[0, 90°]$.

Primer 1

```
Interakcija sa programom:
 Unesite broj tacaka: 2
 Unesite koordinate tacaka:
 2 0 2 1
 1 2 2 2
 26.57
```

Primer 2

```
Interakcija sa programom:
 Unesite broj tacaka: 2
 Unesite koordinate tacaka:
 1 0 1 1
 0 1 1 1
 45
```

Primer 3

```
Interakcija sa programom:
 Unesite broj tacaka: 3
 Unesite koordinate tacaka:
 1 0 1 1
 2 0 2 1
 1 2 2 2
 26.57
```

3.2 Algoritmi sortiranja

Zadatak 3.12 Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. `selection sort`), sortiranja spajanjem (engl. `merge sort`), brzog sortiranja (engl. `quick sort`), mehurastog sortiranja (engl. `bubble sort`), sortiranja direktnim umetanjem (engl. `insertion sort`) i sortiranja umetanjem sa inkrementom (engl. `shell sort`). Upotrebiti biblioteku kako bi se napravilo 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.

Test 1

```
POKRETANJE: time ./a.out
      200000

IZLAZ:
 real 0m42.168s
 user 0m42.100s
 sys 0m0.000s
```

Test 2

```
POKRETANJE: time ./a.out
      400000

IZLAZ:
 real 2m48.395s
 user 2m48.128s
 sys 0m0.000s
```

Test 3

```
POKRETANJE: time ./a.out
      800000

IZLAZ:
 real 11m13.703s
 user 11m12.636s
 sys 0m0.000s
```

Test 4

```
POKRETANJE: time ./a.out
      800000 -r

IZLAZ:
 real 11m21.533s
 user 11m20.436s
 sys 0m0.020s
```

Test 5

```
POKRETANJE: time ./a.out
      800000 -q

IZLAZ:
 real 0m0.159s
 user 0m0.156s
 sys 0m0.000s
```

Test 6

```
POKRETANJE: time ./a.out
      800000 -m

IZLAZ:
 real 0m0.137s
 user 0m0.136s
 sys 0m0.000s
```

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

Primer 1

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

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite prvu nisku anagram
 Unesite drugu nisku anagrm
 nisu
```

Primer 3

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

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

Test 1

```
Ulaz:
 23 64 123 76 22 7

Izlaz:
 1
```

Test 2

```
Ulaz:
 21 654 65 123 65 12 61

Izlaz:
 0
```

Test 3

```
Ulaz:
 34 30

Izlaz:
 4
```

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

```
Ulaz:
 4 23 5 2 4 6 7 34 6 4 5

Izlaz:
 4
```

Test 2

```
Ulaz:
 2 4 6 2 6 7 99 1

Izlaz:
 2
```

Test 3

```
Ulaz:
 123

Izlaz:
 123
```

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

Primer 1

```
Interakcija sa programom:
 Unesite trazeni zbir: 34
 Unesite elemente niza:
 134 4 1 6 30 23
 da
```

Primer 2

```
Interakcija sa programom:
 Unesite trazeni zbir: 12
 Unesite elemente niza:
 53 1 43 3 56 13
 ne
```

Primer 3

```
Interakcija sa programom:
 Unesite trazeni zbir: 52
 Unesite elemente niza:
 52
 ne
```

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

manje od 256.

Primer 1

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

Primer 2

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

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
 biracki-spisak.txt.
```

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

Test 1

```
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, prozivođačima i cenama učitati iz datoteke `artikli.txt`. Pretraživanje niza artikala vršiti binarnom pretragom.

Primer 1

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

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

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

  UKUPNO: 270.00 dinara.

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

 232
  Greska: Ne postoji proizvod sa trazenim 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 u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili, opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na

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

Test 1

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

DAT1.TXT
```
Studenti sortirani po imenu
leksikografski rastuce:
Aleksandra Cvetic 4 6
Andrija Petrovic 2 5
Anja Ilic 3 1
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Lazar Micic 1 3
Marija Stankovic 1 3
Nenad Brankovic 2 4
Ognjen Peric 1 2
Uros Milic 2 5
```

DAT2.TXT
```
Studenti sortirani po broju zadataka
opadajuce, pa po duzini imena rastuce:
Aleksandra Cvetic 4 6
Uros Milic 2 5
Dragan Markovic 3 5
Andrija Petrovic 2 5
Nenad Brankovic 2 4
Lazar Micic 1 3
Bojan Golubovic 4 3
Marija Stankovic 1 3
Ognjen Peric 1 2
Anja Ilic 3 1
Ivana Stankovic 3 1
```

DAT3.TXT
```
Studenti sortirani po prisustvu
opadajuce, pa po broju zadataka,
pa po prezimenima leksikografski
opadajuce:
Aleksandra Cvetic 4 6
Bojan Golubovic 4 3
Dragan Markovic 3 5
Ivana Stankovic 3 1
Anja Ilic 3 1
Andrija Petrovic 2 5
Uros Milic 2 5
Nenad Brankovic 2 4
Marija Stankovic 1 3
Lazar Micic 1 3
Ognjen Peric 1 2
```

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

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

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

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

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

Test 1

```
POKRETANJE: ./a.out

PESME.TXT
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

IZLAZ:
  Jelena - Sunce, 92321
  Mika - Kisa, 5341
  Ana - Nebo, 2342
  Pera - Ptice, 327
  Laza - Oblaci, 29
```

Test 2

```
POKRETANJE: ./a.out -i

PESME.TXT
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

IZLAZ:
  Ana - Nebo, 2342
  Jelena - Sunce, 92321
  Laza - Oblaci, 29
  Mika - Kisa, 5341
  Pera - Ptice, 327
```

Test 3

```
POKRETANJE: ./a.out -n

PESME.TXT
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

IZLAZ:
  Mika - Kisa, 5341
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
```

*** 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. Pretaživanje 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

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

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 Definisati strukturu `Datum`. Napisati funkciju koja poredi dva

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

Primer 1

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

DATOTEKA.TXT
  1.1.2013.
  13.12.2016.
  11.11.2011.
  3.5.2015.
  5.2.2009.
```

```
INTERAKCIJA SA PROGRAMOM:
  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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 1000000

/* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
   -lrt zbog funkcije clock_gettime() */

/* Naredne tri funkcije koje vrse pretragu, ukoliko se trazeni
   element pronadje u nizu, vracaju indeks pozicije na kojoj je
   element pronadjen. Ovaj indeks je uvek nenegativan. Ako element
   nije pronadjen u nizu, funkcije vracaju negativnu vrednost -1, kao
   indikator neuspesne pretrage. */

/* Linearna pretraga: Funkcija pretrazuje niz a[] celih brojeva
   duzine n, trazeci u njemu prvo pojavljivanje elementa x. Pretraga
   se vrsi prostom iteracijom kroz niz. */
int linearna_pretraga(int a[], int n, int x)
{
  int i;
  for (i = 0; i < n; i++)
    if (a[i] == x)
      return i;
  return -1;
}
```

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

/* Interpolaciona pretraga: Funkcija trazi u sortiranom nizu a[]
   duzine n broj x. Pretraga koristi osobinu sortiranosti niza i
   zasniva se na linearnoj interpolaciji vrednosti koja se trazi
   vrednostima na krajevima prostora pretrage. */
int interpolaciona_pretraga(int a[], int n, int x)
{
  int levi = 0;
  int desni = n - 1;
  int srednji;
  /* Dokle god je indeks levi levo od indeksa desni...  */
  while (levi <= desni) {
    /* Ako je trazeni element manji od pocetnog ili veci od
       poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
       nije u tom delu niza. Ova provera je neophodna, da se ne bi
       dogodilo da se prilikom izracunavanja indeksa srednji izadje
       izvan opsega indeksa [levi,desni] */
    if (x < a[levi] || x > a[desni])
      return -1;
    /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
       a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
       jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
       desni). Ova provera je neophodna, jer bi se u suprotnom
       prilikom izracunavanja indeksa srednji pojavilo deljenje
```

```
79          nulom. */
      else if (a[levi] == a[desni])
81          return levi;
      /* Racunanje srednjeg indeksa */
83      srednji =
            levi +
85          (int) ((double) (x - a[levi]) / (a[desni] - a[levi]) *
                   (desni - levi));
87      /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
           verovatno biti blize trazenoj vrednosti nego da je prosto uvek
89         uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
           porediti sa pretragom recnika: ako neko trazi rec na slovo
91         'B', sigurno nece da otvori recnik na polovini, vec verovatno
           negde blize pocetku. */
93      /* Ako je element sa indeksom srednji veci od trazenog, tada se
           trazeni element mora nalaziti u levoj polovini niza */
95      if (x < a[srednji])
          desni = srednji - 1;
97      /* Ako je element sa indeksom srednji manji od trazenog, tada se
           trazeni element mora nalaziti u desnoj polovini niza */
99      else if (x > a[srednji])
          levi = srednji + 1;
101     else
          /* Ako je element sa indeksom srednji jednak trazenom, onda se
103          pretraga zavrsava na poziciji srednji */
          return srednji;
105   }
    /* U slucaju neuspesne pretrage vraca se -1 */
107   return -1;
  }
109
  int main(int argc, char **argv)
111 {
    int a[MAX];
113   int n, i, x;
    struct timespec vreme1, vreme2, vreme3, vreme4, vreme5, vreme6;
115   FILE *f;
    /* Provera argumenata komandne linije */
117   if (argc != 3) {
      fprintf(stderr,
119             "koriscenje programa: %s dim_niza broj\n", argv[0]);
      exit(EXIT_FAILURE);
121   }

123   /* Dimenzija niza */
    n = atoi(argv[1]);
125   if (n > MAX || n <= 0) {
      fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
127     exit(EXIT_FAILURE);
    }
129
    /* Broj koji se trazi */
```

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

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

```
183   fprintf(f, "\tBinarna: %19ld ns\n",
              (vreme4.tv_sec - vreme3.tv_sec) * 1000000000 +
185           vreme4.tv_nsec - vreme3.tv_nsec);
      fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
187           (vreme6.tv_sec - vreme5.tv_sec) * 1000000000 +
              vreme6.tv_nsec - vreme5.tv_nsec);
189   /* Zatvaranje datoteke */
      fclose(f);
191
      exit(EXIT_SUCCESS);
193 }
```

Rešenje 3.2

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

4 #define MAX 1024

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

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

32 int binarna_pretraga_r(int a[], int l, int d, int x)
  {
34  int srednji;
    if (l > d)
36    return -1;
    /* Sredisnja pozicija na kojoj se trazi vrednost x */
```

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

52  int interpolaciona_pretraga_r(int a[], int l, int d, int x)
    {
54    int p;
      if (x < a[l] || x > a[d])
56      return -1;
      if (a[d] == a[l])
58      return l;
      /* Pozicija na kojoj se trazi vrednost x */
60    p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
      if (a[p] == x)
62      return p;
      if (a[p] < x)
64      return interpolaciona_pretraga_r(a, p + 1, d, x);
      else
66      return interpolaciona_pretraga_r(a, l, p - 1, x);
    }
68

    int main()
70  {
      int a[MAX];
72    int x;
      int i, indeks;
74

      /* Ucitavanje trazenog broja */
76    printf("Unesite trazeni broj: ");
      scanf("%d", &x);
78

      /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
80       korisnik pritisne CTRL+D za naznaku kraja */
      i = 0;
82    printf("Unesite sortiran niz elemenata: ");
      while (i < MAX && scanf("%d", &a[i]) == 1) {
84      if (i > 0 && a[i] < a[i - 1]) {
          fprintf(stderr,
86                "Elementi moraju biti uneseni u neopadajucem poretku\n"
        );
          exit(EXIT_FAILURE);
88      }
```

```
    i++;
  }

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

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

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

  return 0;
}
```

Rešenje 3.3

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

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

/* Ucitan niz studenata ce biti sortiran rastuce prema indeksu, jer
   su studenti u datoteci vec sortirani. Iz tog razloga pretraga po
```

```
20    indeksu se vrsi 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
24    sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
      ako element nije pronadjen. */
26 int binarna_pretraga(Student a[], int n, long x)
{
28   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 stutenta na toj poziciji veci od trazenog, tada
36        se trazeni indeks mora nalaziti u levoj polovini niza */
       if (x < a[srednji].indeks)
38       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)
42       levi = srednji + 1;
       else
44       /* Ako je jednak trazenom indeksu x, tada je pronadjen student
            sa trazenom indeksom na poziciji srednji */
46       return srednji;
     }
48   /* Ako nije pronadjen, vraca se -1 */
     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++)
       /* Poredjenje prezimena i-tog studenta i poslatog x */
58     if (strcmp(a[i].prezime, x) == 0)
         return i;
60   return -1;
   }
62
/* Main funkcija mora imati argumente jer se ime datoteke i opcija
64    prosledjuju kao argumenti komandne linije */
   int main(int argc, char *argv[])
66 {
     Student dosije[MAX_STUDENATA];
68   FILE *fin = NULL;
     int i;
70   int br_studenata = 0;
     long trazen_indeks = 0;
```

```
72  char trazeno_prezime[MAX_DUZINA];
    int bin_pretraga;
74
    /* Provera da li je korisnik prilikom poziva programa prosledio ime
76     datoteke sa informacijama o studentima i opciju pretrage */
    if (argc != 3) {
78    fprintf(stderr,
              "Greska: Program se poziva sa %s ime_datoteke opcija\n",
80            argv[0]);
      exit(EXIT_FAILURE);
82  }

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

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

102 /* Cita se sve dok postoji red sa informacijama o studentu */
    i = 0;
104 while (1) {
      if (i == MAX_STUDENATA)
106     break;
      if (fscanf
108       (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
           dosije[i].prezime) != 3)
110     break;
      i++;
112 }
    br_studenata = i;
114
    /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
116 fclose(fin);

118 /* Pretraga po indeksu */
    if (bin_pretraga) {
120   /* Unos indeksa koji se binarno trazi u nizu */
      printf("Unesite indeks studenta cije informacije zelite: ");
122   scanf("%ld", &trazen_indeks);
      i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
```

```
    /* Rezultat binarne pretrage */
    if (i == -1)
      printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
      printf("Indeks: %ld, Ime i prezime: %s %s\n",
             dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
  }
  /* Pretraga po prezimenu */
  else {
    /* Unos prezimena koje se linearno trazi u nizu */
    printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
    i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    /* Rezultat linearne pretrage */
    if (i == -1)
      printf("Ne postoji student sa prezimenom %s\n",
             trazeno_prezime);
    else
      printf("Indeks: %ld, Ime i prezime: %s %s\n",
             dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
  }

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.4

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

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

int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
{
  /* Ako je pozicija elementa na levom kraju veca od pozicije
     elementa na desnom kraju dela niza koji se pretrazuje, onda se
     zapravo pretrazuje prazan deo niza. U praznom delu niza nema
     trazenog elementa pa se vraca -1 */
  if (levi > desni)
    return -1;
  /* Racunanje pozicije 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 trazenim prezimenom, pocinje se sa
45       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
    int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
53  {
      /* Ako je niz prazan, vraca se -1 */
55    if (n == 0)
        return -1;
57    /* Ako se trazi poslednji student sa trazenim prezimenom, pocinje
         se sa poslednjim studentom u nizu. */
59    if (strcmp(a[n - 1].prezime, x) == 0)
        return n - 1;
61    return linearna_pretraga_rekurzivna(a, n - 1, x);
    }
63
    /* Main funkcija mora imati argumente jer se ime datoteke i opcija
65     prosledjuju kao argumenti komandne linije */
    int main(int argc, char *argv[])
67  {
      Student dosije[MAX_STUDENATA];
69    FILE *fin = NULL;
      int i;
71    int br_studenata = 0;
      long trazen_indeks = 0;
73    char trazeno_prezime[MAX_DUZINA];
      int bin_pretraga;
75
      /* Provera da li je korisnik prilikom poziva programa prosledio ime
```

```
77      datoteke sa informacijama o studentima i opciju pretrage */
    if (argc != 3) {
79    fprintf(stderr,
              "Greska: Program se poziva sa %s ime_datoteke opcija\n",
81            argv[0]);
      exit(EXIT_FAILURE);
83  }

85  /* Provera prosledjene opcije */
    if (strcmp(argv[2], "-indeks") == 0)
87    bin_pretraga = 1;
    else if (strcmp(argv[2], "-prezime") == 0)
89    bin_pretraga = 0;
    else {
91    fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
      exit(EXIT_FAILURE);
93  }

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

103 /* Cita se sve dok postoji red sa informacijama o studentu */
    i = 0;
105 while (1) {
      if (i == MAX_STUDENATA)
107     break;
      if (fscanf
109       (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
           dosije[i].prezime) != 3)
111     break;
      i++;
113 }
    br_studenata = i;
115
    /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
117 fclose(fin);

119 /* Pretraga po indeksu */
    if (bin_pretraga) {
121   /* Unos indeksa koji se binarno trazi u nizu */
      printf("Unesite indeks studenta cije informacije zelite: ");
123   scanf("%ld", &trazen_indeks);
      i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
125                                     trazen_indeks);
      /* Rezultat binarne pretrage */
127   if (i == -1)
        printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
```

```
129     else
          printf("Indeks: %ld, Ime i prezime: %s %s\n",
131               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
    }
133   /* Pretraga po prezimenu */
    else {
135     /* Unos prezimena koje se linearno trazi u nizu */
      printf("Unesite prezime studenta cije informacije zelite: ");
137     scanf("%s", trazeno_prezime);
      i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
139                                         trazeno_prezime);
      /* Rezultat linearne pretrage */
141     if (i == -1)
        printf("Ne postoji student sa prezimenom %s\n",
143               trazeno_prezime);
      else
145       printf("Indeks: %ld, Ime i prezime: %s %s\n",
              dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
147   }

149   exit(EXIT_SUCCESS);
}
```

Rešenje 3.5

```
1 #include <stdio.h>
  #include <string.h>
3 #include <math.h>
  #include <stdlib.h>
5
  /* Struktura koja opisuje tacku u ravni */
7 typedef struct Tacka {
    float x;
9   float y;
  } Tacka;
11
  /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13    pocetka (0,0) */
  float rastojanje(Tacka A)
15 {
    return sqrt(A.x * A.x + A.y * A.y);
17 }

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

```
27      if (rastojanje(t[i]) < rastojanje(najbliza)) {
          najbliza = t[i];
29      }
    }
31  return najbliza;
}
33
/* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih tacaka
35     t dimenzije n */
Tacka najbliza_x_osi(Tacka t[], int n)
37  {

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

65  #define MAX 1024

67  int main(int argc, char *argv[])
    {
69    FILE *ulaz;
    Tacka tacke[MAX];
71  Tacka najbliza;
    int i, n;
73
    /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75     ime datoteke sa tackama */
    if (argc < 2) {
77    fprintf(stderr,
              "koriscenje programa: %s ime_datoteke\n", argv[0]);
```

```
79      exit(EXIT_FAILURE);
    }
81
    /* Otvaranje datoteke za citanje */
83  ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
85    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
              argv[1]);
87    exit(EXIT_FAILURE);
    }
89
    /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91     tackama; i predstavlja indeks tekuce tacke */
    i = 0;
93  while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
      i++;
95  }
    n = i;
97
    /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
99     dodatnih argumenata */
    if (argc == 2)
101   /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
      najbliza = najbliza_koordinatnom(tacke, n);
103 /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
       pitanju opcija -x */
105 else if (strcmp(argv[2], "-x") == 0)
      /* Racuna se rastojanje u odnosu na x osu */
107   najbliza = najbliza_x_osi(tacke, n);
    /* Ako je u pitanju opcija -y */
109 else if (strcmp(argv[2], "-y") == 0)
      /* Racuna se rastojanje u odnosu na y osu */
111   najbliza = najbliza_y_osi(tacke, n);
    else {
113   /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
         korisnika i prekida se izvrsavanje programa */
115   fprintf(stderr, "Pogresna opcija\n");
      exit(EXIT_FAILURE);
117 }

119 /* Stampanje koordinata trazene tacke */
    printf("%g %g\n", najbliza.x, najbliza.y);
121
    /* Zatvaranje datoteke */
123 fclose(ulaz);

125 exit(EXIT_SUCCESS);
}
```

Rešenje 3.6

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

/* Tacnost */
#define EPS 0.001

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

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

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

  /* Stampanje vrednosti trazene tacke */
  printf("%g\n", s);

  return 0;
}
```

Rešenje 3.7

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

int main(int argc, char **argv)
{
  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: ");
      fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
20    fprintf(stderr,
              "Program se poziva sa %s ime_funkcije iz math.h.\n",
22            argv[0]);
      exit(EXIT_FAILURE);
24  }

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

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

50  printf("Unesite krajeve intervala: ");
    scanf("%lf %lf", &l, &d);
52
    if ((*fp) (l) * (*fp) (d) >= 0) {
54    fprintf(stderr,
              "%s na intervalu [%g, %g] ne zadovoljava uslove\n",
56            ime_funkcije, l, d);
      exit(EXIT_FAILURE);
58  }

60  printf("Unesite preciznost: ");
```

```
  scanf("%lf", &epsilon);

  /* Sve dok se ne pronadje trazena vrednost argumenta */
  while (1) {
    /* Polovi se interval */
    s = (l + d) / 2;
    /* Ako je apsolutna vrednost trazene funkcije u ovoj tacki manja
       od zadate tacnosti, prekida se pretraga */
    if (fabs((*fp) (s)) < epsilon) {
      break;
    }
    /* Ako je nula u levom delu intervala, nastavlja se pretraga na
       [l, s] */
    if ((*fp) (l) * (*fp) (s) < 0)
      d = s;
    else
      /* Inace, na intervalu [s, d] */
      l = s;
  }

  /* Stampanje vrednosti trazene tacke */
  printf("%g\n", s);

  return 0;
}
```

Rešenje 3.8

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

#define MAX 256

int prvi_veci_od_nule(int niz[], int n)
{
  /* Granice pretrage */
  int l = 0, d = n - 1;
  int s;
  /* Sve dok je leva manja od desne granice */
  while (l <= d) {
    /* Racuna se sredisnja pozicija */
    s = (l + d) / 2;
    /* Ako je broj na toj poziciji veci od nule, a eventualni njegov
       prethodnik manji ili jednak nuli, pretraga je zavrsena */
    if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
      return s;
    /* U slucaju broja manjeg ili jednakog nuli, pretrazuje se desna
       polovina niza */
    if (niz[s] <= 0)
      l = s + 1;
    /* A inace, leva polovina */
```

```
24      else
          d = s - 1;
26    }
    return -1;
28  }

30  int main()
  {
32    int niz[MAX];
    int n = 0;
34
    /* Unos niza */
36    while (scanf("%d", &niz[n]) == 1)
      n++;
38
    /* Stampanje rezultata */
40    printf("%d\n", prvi_veci_od_nule(niz, n));

42    return 0;
  }
```

Rešenje 3.9

```
1   #include <stdio.h>
  #include <stdlib.h>
3
  #define MAX 256
5
  int prvi_manji_od_nule(int niz[], int n)
7   {
    /* Granice pretrage */
9     int l = 0, d = n - 1;
    int s;
11    /* Sve dok je leva manja od desne granice */
    while (l <= d) {
13      /* Racuna se sredisnja pozicija */
      s = (l + d) / 2;
15      /* Ako je broj na toj poziciji manji od nule, a eventualni njegov
           prethodnik veci ili jednak nuli, pretraga se zavrsava */
17      if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
          return s;
19      /* Ako je broj veci ili jednak nuli, 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   /* Stampanje rezultata */
  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
     najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
17     do 31 */
  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 */
23     s = (l + d) / 2;
    /* Proverava se da li je na toj poziciji trazena jedinica */
25     if ((1 << s) <= x && (1 << (s + 1)) > x)
      return s;
27     /* Pretraga desne polovine binarnog zapisa */
    if ((1 << s) > x)
29       l = s - 1;
    /* Pretraga leve polovine binarnog zapisa */
31     else
      d = s + 1;
33   }
```

```
  return s;
}

int main()
{
  unsigned int x;

  /* Unos podatka */
  scanf("%u", &x);

  /* Provera da li je uneti broj pozitivan */
  if (x == 0) {
    fprintf(stderr, "Logaritam od nule nije definisan\n");
    exit(EXIT_FAILURE);
  }

  /* Ispis povratnih vrednosti funkcija */
  printf("%u %u\n", logaritam_a(x), logaritam_b(x));

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.12

sort.h

```
#ifndef _SORT_H_
#define _SORT_H_ 1

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

/* Insertion sort: Funkcija sortira niz celih brojeva metodom
   sortiranja umetanjem.  Ideja algoritma je sledeca: neka je na
   pocetku i-te iteracije niz prvih i elemenata
   (a[0],a[1],...,a[i-1]) sortirano. U i-toj iteraciji treba element
   a[i] umetnuti na pravu poziciju medju prvih i elemenata tako da se
   dobije niz duzine i+1 koji je sortiran. Ovo se radi tako sto se
   i-ti element najpre uporedi sa njegovim prvim levim susedom
   (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i niz
   a[0],a[1],...,a[i] je sortiran, pa se moze preci na sledecu
   iteraciju. Ako je a[i-1] vece, tada se zamenjuju a[i] i a[i-1], a
   zatim se proverava da li je potrebno dalje potiskivanje elementa u
```

```
   levo, poredeci ga sa njegovim novim levim susedom. Ovim uzastopnim
26    premestanjem se a[i] umece na pravo mesto u nizu. */
   void insertion_sort(int a[], int n);
28
   /* Bubble sort: Funkcija sortira niz celih brojeva metodom mehurica.
30    Ideja algoritma je sledeca: prolazi se kroz niz redom poredeci
      susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
32    poretku.  Ovim se najveci element poput mehurica istiskuje na
      "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je potrebno
34    ovaj postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih
      n-1 elemenata niza bez poslednjeg koji je postavljen na pravu
36    poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
      kracim prefiksima niza, cime se jedan po jedan istiskuju
38    elemenenti na svoje prave pozicije. */
   void bubble_sort(int a[], int n);
40
   /* Selsort: Ovaj algoritam je jednostavno prosirenje sortiranja
42    umetanjem koje dopusta direktnu razmenu udaljenih elemenata.
      Prosirenje se sastoji u tome da se kroz algoritam umetanja prolazi
44    vise puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
      koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
46    tako se dobija h-sortiran niz, tj. niz u kome su elementi na
      rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
48    drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
      za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
50    kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
      h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
52    algoritma, ali bilo koja vrednost ce rezultovati ispravnim
      sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
54    vrednost 1. */
   void shell_sort(int a[], int n);
56
   /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
58    Sortiranje se vrsi od elementa na poziciji l do onog na poziciji
      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
      podeli na dve polovine, levu i desnu, koje zatim rekurzivno
62    sortira. Od ova dva sortirana podniza, sortiran niz se dobija
      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
66    originalni niz. */
   void merge_sort(int a[], int l, int d);
68
   /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
70    i d. Njena ideja sortiranja je izbor jednog elementa niza, koji se
      naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
72    svi elementi levo od njega bice manji, a svi desno bice veci od
      njega. Kako je pivot doveden na svoje mesto, da bi niz bio
74    kompletno sortiran, potrebno je sortirati elemente levo (manje) od
      njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
76    manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
```

```
   deo moze se uraditi rekurzivno. */
void quick_sort(int a[], int l, int d);

#endif
```

sort.c

```
#include "sort.h"

#define MAX 1000000

void selection_sort(int a[], int n)
{
  int i, j;
  int min;
  int pom;

  /* U svakoj iteraciji ove petlje pronalazi se najmanji element
     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
     poziciju i, dok se element na pozciji i premesta na poziciju
     min, na kojoj se nalazio najmanji od navedenih elemenata. */
  for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
       najmanji od elemenata a[i],...,a[n-1]. */
    min = i;
    for (j = i + 1; j < n; j++)
      if (a[j] < a[min])
        min = j;

    /* 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;
    }
  }
}

void insertion_sort(int a[], int n)
{
  int i, j;

  /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
     sortiran */
  for (i = 1; i < n; i++) {
    /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
       potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
       bude sortiran. Indeks j je trenutna pozicija na kojoj se
       element koji se umece nalazi. Petlja se zavrsava ili kada
       element dodje do levog kraja (j==0) ili kada se naidje na
```

```
        element a[j-1] koji je manji od a[j]. */
    int temp = a[i];
    for (j = i; j > 0 && temp < a[j - 1]; j--)
      a[j] = a[j - 1];
    a[j] = temp;
  }
}

void bubble_sort(int a[], int n)
{
  int i, j;
  int ind;

  for (i = n, ind = 1; i > 1 && ind; i--)
    /* Poput "mehurica" potiskuje se najveci element medju elementima
       od a[0] do a[i-1] na poziciju i-1 uporedjujuci susedne
       elemente niza i potiskujuci veci u desno */
    for (j = 0, ind = 0; j < i - 1; j++)
      if (a[j] > a[j + 1]) {
        int temp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = temp;
        /* Promenljiva ind registruje da je bilo premestanja. Samo u
           tom slucaju ima smisla ici na sledecu iteraciju, jer ako
           nije bilo premestanja, znaci da su svi elementi vec u
           dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
           niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
           ispravno, ali manje efikasano, jer bi se cesto nepotrebno
           vrsila mnoga uporedjivanja, kada je vec jasno da je
           sortiranje zavrseno. */
        ind = 1;
      }
}

void shell_sort(int a[], int n)
{
  int h = n / 2, i, j;
  while (h > 0) {
    /* Insertion sort sa korakom h */
    for (i = h; i < n; i++) {
      int temp = a[i];
      j = i;
      while (j >= h && a[j - h] > temp) {
        a[j] = a[j - h];
        j -= h;
      }
      a[j] = temp;
    }
    h = h / 2;
  }
}

```

```
void merge_sort(int a[], int l, int d)
{
  int s;
  static int b[MAX];            /* Pomocni niz */
  int i, j, k;

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

  /* Odredjivanje sredisnjeg indeksa */
  s = (l + d) / 2;

  /* Rekurzivni pozivi */
  merge_sort(a, l, s);
  merge_sort(a, s + 1, d);

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

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

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

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

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

/* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
{
```

```
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}

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

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

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

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

  /* Rekurzivno sortiranje elementa manjih od pivota */
  quick_sort(a, l, pivot_pozicija - 1);
  /* Rekurzivno sortiranje elementa vecih od pivota */
  quick_sort(a, pivot_pozicija + 1, d);
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sort.h"

/* Maksimalna duzina niza */
#define MAX 1000000

int main(int argc, char *argv[])
{
  /*****************************************************************
     tip_sortiranja == 0 => selectionsort, (podrazumevano)
     tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
     tip_sortiranja == 2 => bubblesort,    -b opcija komandne linije
     tip_sortiranja == 3 => shellsort,     -s opcija komandne linije
```

```
   tip_sortiranja == 4 => mergesort,     -m opcija komandne linije
   tip_sortiranja == 5 => quicksort,     -q opcija komandne linije
*****************************************************************/
int tip_sortiranja = 0;
/*****************************************************************
   tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
   tip_niza == 1 => rastuce sortirani nizovi,   -r opcija
   tip_niza == 2 => opadajuce soritrani nizovi, -o opcija
*****************************************************************/
int tip_niza = 0;

/* Dimenzija niza koji se sortira */
int dimenzija;
int i;
int niz[MAX];

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

/* Ocitavanje opcija i argumenata prilikom poziva programa */
for (i = 1; i < argc; i++) {
  /* Ako je u pitanju opcija... */
  if (argv[i][0] == '-') {
    switch (argv[i][1]) {
    case 'i':
      tip_sortiranja = 1;
      break;
    case 'b':
      tip_sortiranja = 2;
      break;
    case 's':
      tip_sortiranja = 3;
      break;
    case 'm':
      tip_sortiranja = 4;
      break;
    case 'q':
      tip_sortiranja = 5;
      break;
    case 'r':
      tip_niza = 1;
      break;
    case 'o':
      tip_niza = 2;
      break;
    default:
      printf("Pogresna opcija -%c\n", argv[i][1]);
      return 1;
```

```
        break;
      }
    }
    /* Ako je u pitanju argument, onda je to duzina niza koji treba
       da se sortira */
    else {
      dimenzija = atoi(argv[i]);
      if (dimenzija <= 0 || dimenzija > MAX) {
        fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
        exit(EXIT_FAILURE);
      }
    }
  }

  /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
     niza dobijenom iz komandne linije. srand() funkcija obezbedjuje
     novi seed za pozivanje rand funkcije, i kako generisani niz ne
     bi uvek bio isti seed je postavljen na tekuce vreme u sekundama
     od Nove godine 1970. rand()%100 daje brojeve izmedju 0 i 99 */
  srand(time(NULL));
  if (tip_niza == 0)
    for (i = 0; i < dimenzija; i++)
      niz[i] = rand();
  else if (tip_niza == 1)
    for (i = 0; i < dimenzija; i++)
      niz[i] = i == 0 ? rand() % 100 : niz[i - 1] + rand() % 100;
  else
    for (i = 0; i < dimenzija; i++)
      niz[i] = i == 0 ? rand() % 100 : niz[i - 1] - rand() % 100;

  /* Ispisivanje elemenata niza */
  /*****************************************************************
    Ovaj deo je iskomentarisan jer sledeci ispis ne treba da se nadje
    na standardnom izlazu. Njegova svrha je samo bila provera da li
    je niz generisan u skladu sa opcijama komandne linije.

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


  /* Sortiranje niza na odgovarajuci nacin */
  if (tip_sortiranja == 0)
    selection_sort(niz, dimenzija);
  else if (tip_sortiranja == 1)
    insertion_sort(niz, dimenzija);
  else if (tip_sortiranja == 2)
    bubble_sort(niz, dimenzija);
  else if (tip_sortiranja == 3)
    shell_sort(niz, dimenzija);
  else if (tip_sortiranja == 4)
```

```
    merge_sort(niz, 0, dimenzija - 1);
  else
    quick_sort(niz, 0, dimenzija - 1);

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.13

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

#define MAX_DIM 128

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

/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
{
  int i;
```

```
28
   /* Ako dve niske imaju razlicit broj karaktera onda one nisu
30     anagrami */
   if (strlen(s) != strlen(t))
32   return 0;

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

38 /* Dve sortirane niske su anagrami ako i samo ako su jednake */
   for (i = 0; s[i] != '\0'; i++)
40   if (s[i] != t[i])
       return 0;
42 return 1;
}
44
int main()
46 {
   char s[MAX_DIM], t[MAX_DIM];
48
   /* Ucitavanje niski sa ulaza */
50 printf("Unesite prvu nisku: ");
   scanf("%s", s);
52 printf("Unesite drugu nisku: ");
   scanf("%s", t);
54
   /* Poziv funkcije */
56 if (anagrami(s, t))
     printf("jesu\n");
58 else
     printf("nisu\n");
60
   return 0;
62 }
```

Rešenje 3.14

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

```
1 #include <stdio.h>
  #include "sort.h"
3
  #define MAX 256
5
  /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
7    sortiranom nizu celih brojeva */
  int najmanje_rastojanje(int a[], int n)
9 {
   int i, min;
11 min = a[1] - a[0];
```

```
  for (i = 2; i < n; i++)
    if (a[i] - a[i - 1] < min)
      min = a[i] - a[i - 1];
  return min;
}

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

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

  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
     se selection sort. */
  selection_sort(a, i);

  /* Ispis rezultata */
  printf("%d\n", najmanje_rastojanje(a, i));

  return 0;
}
```

Rešenje 3.15

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

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

#define MAX_DIM 256

/* Funkcija za odredjivanje onog elementa sortiranog niza koji se
   najvise puta pojavio u tom nizu */
int najvise_puta(int a[], int n)
{
  int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
  /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
  for (i = 0; i < n; i = j) {
    br_pojava = 1;
    for (j = i + 1; j < n && a[i] == a[j]; j++)
      br_pojava++;
    /* Ispitivanje da li se do tog trenutka i-ti element pojavio
       najvise puta u nizu */
    if (br_pojava > max_br_pojava) {
      max_br_pojava = br_pojava;
      i_max_pojava = i;
    }
```

```
  }
  /* Vraca se element koji se najvise puta pojavio u nizu */
  return a[i_max_pojava];
}

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

  /* Ucitavanje elemenata niza sve do kraja ulaza */
  i = 0;
  while (scanf("%d", &a[i]) != EOF)
    i++;

  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
     se merge sort. */
  merge_sort(a, 0, i - 1);

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

  return 0;
}
```

Rešenje 3.16

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

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

#define MAX_DIM 256

/* 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 */
int binarna_pretraga(int a[], int n, int x)
{
  int levi = 0, desni = n - 1, srednji;

  while (levi <= desni) {
    srednji = (levi + desni) / 2;
    if (a[srednji] == x)
      return 1;
    else if (a[srednji] > x)
      desni = srednji - 1;
    else if (a[srednji] < x)
      levi = srednji + 1;
  }
  return 0;
```

```
}

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

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

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

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

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

  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 prepisujemo u treci */
  while (i < dim1)
    niz3[k++] = niz1[i++];

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

int main()
{
  int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
  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)
      break;
    j++;
  }

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

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

  return 0;
}
```

Rešenje 3.18

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

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

  /* Ako nema dovoljno arguemenata komandne linije */
  if (argc < 3) {
    fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
    ;
    exit(EXIT_FAILURE);
  }

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

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

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

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

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

  /* Sve dok nije dostignut kraj neke datoteke */
```

```
  while (!kraj1 && !kraj2) {
    int tmp = strcmp(ime1, ime2);
    if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
      /* Ime i prezime iz prve datoteke je leksikografski ranije, i
         biva upisano u izlaznu datoteku */
      fprintf(fout, "%s %s\n", ime1, prezime1);
      /* Citanje narednog studenta iz prve datoteke */
      if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
        kraj1 = 1;
    } else {
      /* Ime i prezime iz druge datoteke je leksikografski ranije, i
         biva upisano u izlaznu datoteku */
      fprintf(fout, "%s %s\n", ime2, prezime2);
      /* Citanje narednog studenta iz druge datoteke */
      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;
    }
  }

  /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
     treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
   */
  while (!kraj1) {
    fprintf(fout, "%s %s\n", ime1, prezime1);
    if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
      kraj1 = 1;
  }

  /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
     datoteke, onda ima jos studenata u drugoj datoteci, koje treba
     prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
  while (!kraj2) {
    fprintf(fout, "%s %s\n", ime2, prezime2);
    if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
      kraj2 = 1;
  }

  /* Zatvaranje datoteka */
  fclose(fin1);
  fclose(fin2);
  fclose(fout);

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.19

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

```c
#include <stdlib.h>

#define MAX_BR_TACAKA 128

/* Struktura koja reprezentuje koordinate tacke */
typedef struct Tacka {
  int x;
  int y;
} Tacka;

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

/* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
   pocetka */
void sortiraj_po_rastojanju(Tacka t[], int n)
{
  int min, i, j;
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (rastojanje(t[j]) < rastojanje(t[min])) {
        min = j;
      }
    }
    if (min != i) {
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}

/* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
{
  int min, i, j;
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (abs(t[j].x) < abs(t[min].x)) {
        min = j;
      }
    }
    if (min != i) {
```

```
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}

/* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
void sortiraj_po_y(Tacka t[], int n)
{
  int min, i, j;
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (abs(t[j].y) < abs(t[min].y)) {
        min = j;
      }
    }
    if (min != i) {
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}

int main(int argc, char *argv[])
{
  FILE *ulaz;
  FILE *izlaz;
  Tacka tacke[MAX_BR_TACAKA];
  int i, n;

  /* Proveravanje broja argumenata komandne linije: ocekuje se ime
     izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
     datoteke, tj. 4 argumenta */
  if (argc != 4) {
    fprintf(stderr,
            "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
    return 0;
  }

  /* Otvaranje datoteke u kojoj su zadate tacke */
  ulaz = fopen(argv[2], "r");
  if (ulaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[2]);
    return 0;
  }

```

```
  /* Otvaranje datoteke u koju treba upisati rezultat */
  izlaz = fopen(argv[3], "w");
  if (izlaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[3]);
    return 0;
  }

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

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

  /* 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
     sortiranja */
  switch (argv[1][1]) {
  case 'x':
    /* Sortiranje po vrednosti x koordinate */
    sortiraj_po_x(tacke, n);
    break;
  case 'y':
    /* Sortiranje po vrednosti y koordinate */
    sortiraj_po_y(tacke, n);
    break;
  case 'o':
    /* Sortiranje po udaljenosti od koorinatnog pocetka */
    sortiraj_po_rastojanju(tacke, n);
    break;
  }

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

  /* Zatvaranje otvorenih datoteka */
  fclose(ulaz);
  fclose(izlaz);

  return 0;
}
```

Rešenje 3.20

```c
#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;
    }
  }
}

/* Funkcija sortira niz gradjana rastuce po prezimenima */
void sort_prezime(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].prezime,...,a[n-1].prezime. */
    min = i;
    for (j = i + 1; j < n; j++)
      if (strcmp(a[j].prezime, a[min].prezime) < 0)
```

```
51          min = j;
      /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
53       su (i) i min razliciti, inace je nepotrebno. */
      if (min != i) {
55      pom = a[i];
        a[i] = a[min];
57      a[min] = pom;
      }
59    }
  }
61
  /* Pretraga niza Gradjana */
63 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
  {
65    int i;
    for (i = 0; i < n; i++)
67      if (strcmp(a[i].ime, x->ime) == 0
          && strcmp(a[i].prezime, x->prezime) == 0)
69        return i;
    return -1;
71 }


73
  int main()
75 {
    Gradjanin spisak1[MAX], spisak2[MAX];
77  int isti_rbr = 0;
    int i, n;
79  FILE *fp = NULL;

81  /* Otvaranje datoteke */
    if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
83    fprintf(stderr,
              "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
85    exit(EXIT_FAILURE);
    }
87
    /* Citanje sadrzaja */
89  for (i = 0;
        fscanf(fp, "%s %s", spisak1[i].ime,
91             spisak1[i].prezime) != EOF; i++)
      spisak2[i] = spisak1[i];
93  n = i;

95  /* Zatvaranje datoteke */
    fclose(fp);
97
    sort_ime(spisak1, n);
99
    /*****************************************************************
101   Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
      sortiranih nizova. Koriscen je samo u fazi testiranja programa.
```

```
103
    printf("Biracki spisak [uredjen prema imenima]:\n");
105   for(i=0; i<n; i++)
        printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107 ****************************************************************/

109 sort_prezime(spisak2, n);

111 /****************************************************************
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
113   sortiranih nizova. Koriscen je samo u fazi testiranja programa.

115   printf("Biracki spisak [uredjen prema prezimenima]:\n");
    for(i=0; i<n; i++)
117   printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
  ****************************************************************/
119
  /* Linearno pretrazivanje nizova */
121 for (i = 0; i < n; i++)
    if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
123   isti_rbr++;

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

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

Rešenje 3.22

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

5 #define MAX_BR_RECI 128
  #define MAX_DUZINA_RECI 32
7
  /* Funkcija koja izracunava broj suglasnika u reci */
9 int broj_suglasnika(char s[])
  {
11   char c;
    int i;
13   int suglasnici = 0;
```

```
  /* Prolaz karakter po karakter kroz zadatu nisku */
  for (i = 0; s[i]; i++) {
    /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
       pokriven slucaj i malih i velikih suglasnika. */
    if (isalpha(s[i])) {
      c = toupper(s[i]);
      /* Ukoliko slovo nije samoglasnik uvecava se brojac. */
      if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
        suglasnici++;
    }
  }
  /* Vraca se izracunata vrednost */
  return suglasnici;
}

/* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
   duzini reci se mora proslediti zbog pravilnog upravljanja
   memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECI], int n)
{
  int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
      duzina_j, duzina_min;
  char tmp[MAX_DUZINA_RECI];
  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i; j < n; j++) {
      /* Prvo se uporedjuje broj suglasnika */
      broj_suglasnika_j = broj_suglasnika(reci[j]);
      broj_suglasnika_min = broj_suglasnika(reci[min]);
      if (broj_suglasnika_j < broj_suglasnika_min)
        min = j;
      else if (broj_suglasnika_j == broj_suglasnika_min) {
        /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
           se duzine */
        duzina_j = strlen(reci[j]);
        duzina_min = strlen(reci[min]);

        if (duzina_j < duzina_min)
          min = j;
        else {
          /* Ako reci imaju i isti broj suglasnika i iste duzine,
             uporedjuju se leksikografski */
          if (duzina_j == duzina_min
              && strcmp(reci[j], reci[min]) < 0)
            min = j;
        }
      }
    }
    if (min != i) {
      strcpy(tmp, reci[min]);
      strcpy(reci[min], reci[i]);
      strcpy(reci[i], tmp);
```

```
    }
  }
}

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

  /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
     a drugi maksimalnu duzinu pojedinacne reci */
  char reci[MAX_BR_RECI][MAX_DUZINA_RECI];

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

  /* 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 */
    if (i == MAX_BR_RECI)
      break;
    /* Priprema brojaca za narednu iteraciju */
    i++;
  }

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

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

  /* Zatvaranje datoteke */
  fclose(ulaz);

  return 0;
}
```

Rešenje 3.23

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

#define MAX_ARTIKALA 100000

/* Struktura koja predstavlja jedan artikal */
typedef struct art {
  long kod;
  char naziv[20];
  char proizvodjac[20];
  float cena;
} Artikal;

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

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

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

  for (i = 0; i < n - 1; i++) {
```

```
    min = i;
52    for (j = i + 1; j < n; j++)
      if (a[j].kod < a[min].kod)
54      min = j;
    if (min != i) {
56    pom = a[i];
      a[i] = a[min];
58    a[min] = pom;
    }
60  }
}
62
int main()
64{
  Artikal asortiman[MAX_ARTIKALA];
66  long kod;
  int i, n;
68  float racun;

70  FILE *fp = NULL;

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

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

88  n = i;

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

```
    efikasnija binarna pretraga. */
  selection_sort(asortiman, n);

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

  kod = 0;
  while (1) {
    printf("---------------------------\n");
    printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
    printf("- Za nov racun unesite kod artikla!\n\n");
    /* Unos bar koda provog artikla sledeceg kupca */
    if (scanf("%ld", &kod) == EOF)
      break;
    /* Trenutni racun novog kupca */
    racun = 0;
    /* Za sve artikle trenutnog kupca */
    while (1) {
      /* Vrsi se njihov pronalazak u nizu */
      if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
        printf("\tGreska: Ne postoji proizvod sa trazenim kodom!\n");
      } else {
        printf("\tTrazili ste:\t%s %s %12.2f\n",
               asortiman[i].naziv, asortiman[i].proizvodjac,
               asortiman[i].cena);
        /* I dodavanje na ukupan racun */
        racun += asortiman[i].cena;
      }
      /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
         nema vise artikla */
      printf("Unesite kod artikla [ili 0 za prekid]: \t");
      scanf("%ld", &kod);
      if (kod == 0)
        break;
    }
    /* Stampanje ukupnog racuna trenutnog kupca */
    printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
  }

  printf("Kraj rada kase!\n");

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.24

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

#define MAX 500

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

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

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

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

/* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
   zadataka opadajuce, a ukoliko neki studenti imaju isti broj
   uradjenih zadataka sortiraju se po duzini imena rastuce. */
void sort_zadatke_pa_imena(Student niz[], int n)
{
  int i, j;
  int max;
  Student pom;
  for (i = 0; i < n - 1; i++) {
    max = i;
    for (j = i + 1; j < n; j++)
      if (niz[j].zadaci > niz[max].zadaci)
        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 opadajuce. Ukoliko neki studenti imaju isti broj casova,
63      sortiraju se opadajuce po broju uradjenih zadataka, a ukoliko se
        i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65      prezimenu opadajuce. */
    void sort_prisustvo_pa_zadatke_pa_prezimena(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;
79        else if (niz[j].prisustvo == niz[max].prisustvo
                   && 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
      /* Otvaranje datoteke za citanje */
99    if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
        fprintf(stderr, "Neupesno otvaranje datoteke aktivnost.txt.\n");
101     exit(EXIT_FAILURE);
      }
```

```
103
    /* Ucitavanje sadrzaja */
105 for (i = 0;
         fscanf(fp, "%s%s%d%d", praktikum[i].ime,
107             praktikum[i].prezime, &praktikum[i].prisustvo,
                &praktikum[i].zadaci) != EOF; i++);
109 /* Zatvaranje datoteke */
    fclose(fp);
111 br_studenata = i;

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

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

147 /* Kreiranje treceg spiska studenata po trecem kriterijumu */
    sort_prisustvo_pa_zadatke_pa_prezimena(praktikum, br_studenata);
149 /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat3.txt", "w")) == NULL) {
151   fprintf(stderr, "Neupesno otvaranje datoteke dat3.txt.\n");
      exit(EXIT_FAILURE);
153 }
    /* Upis niza u datoteku */
```

```
  fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
  fprintf(fp, "pa po broju zadataka,\n");
  fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
  for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
  /* Zatvaranje datoteke */
  fclose(fp);

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.25

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

#define KORAK 10

/* Struktura koja opisuje jednu pesmu */
typedef struct {
  char *izvodjac;
  char *naslov;
  int broj_gledanja;
} Pesma;

/* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
   rad qsort funkcije) */
int uporedi_gledanost(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return p2->broj_gledanja - p1->broj_gledanja;
}

/* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
   funkcije) */
int uporedi_naslove(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return strcmp(p1->naslov, p2->naslov);
}

/* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
   qsort funkcije) */
int uporedi_izvodjace(const void *pp1, const void *pp2)
```

```
{
38    Pesma *p1 = (Pesma *) pp1;
      Pesma *p2 = (Pesma *) pp2;
40
      return strcmp(p1->izvodjac, p2->izvodjac);
42  }

44  int main(int argc, char *argv[])
    {
46    FILE *ulaz;
      Pesma *pesme;                 /* Pokazivac na deo memorije za
48                                     cuvanje pesama */
      int alocirano_za_pesme;       /* Broj mesta alociranih za pesme */
50    int i;                        /* Redni broj pesme cije se
                                       informacije citaju */
52    int n;                        /* Ukupan broj pesama */
      int j, k;
54    char c;
      int alocirano;                /* Broj mesta alociranih za propratne
56                                     informacije o pesmama */
      int broj_gledanja;
58
      /* Priprema datoteke za citanje */
60    ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
      if (ulaz == NULL) {
62      fprintf(stderr, "Greska pri otvaranju ulazne datoteke!\n");
        return 0;
64    }

66    /* Citanje informacija o pesmama */
      pesme = NULL;
68    alocirano_za_pesme = 0;
      i = 0;
70
      while (1) {
72
        /* Proverava se da li je dostignut kraj datoteke */
74      c = fgetc(ulaz);
        if (c == EOF) {
76        /* Nema vise sadrzaja za citanje */
          break;
78      } else {
          /* Inace, vracamo procitani karakter nazad */
80        ungetc(c, ulaz);
        }
82
        /* Provera da li postoji dovoljno vec alocirane memorije za
84         citanje nove pesme */
        if (alocirano_za_pesme == i) {
86        /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
             novih KORAK mesta */
88        alocirano_za_pesme += KORAK;
```

```
      pesme =
90        (Pesma *) realloc(pesme,
                            alocirano_za_pesme * sizeof(Pesma));
92
      /* Proverava se da li je nova memorija uspesno realocirana */
94    if (pesme == NULL) {
        /* Ako nije ispisuje se obavestenje */
96      fprintf(stderr, "Problem sa alokacijom memorije!\n");
        /* I oslobadja sva memorija zauzeta do ovog koraka */
98      for (k = 0; k < i; k++) {
          free(pesme[k].izvodjac);
100       free(pesme[k].naslov);
        }
102     free(pesme);
        return 0;
104   }
    }
106
    /* Ako jeste, nastavlja se sa citanjem pesama ... */
108 /* Cita se ime izvodjaca */
    j = 0;                        /* Pozicija na koju treba smestiti
110                                  procitani karakter */
    alocirano = 0;                /* Broj alociranih mesta */
112 pesme[i].izvodjac = NULL;     /* Memorija za smestanje procitanih
                                    karaktera */
114
    /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
116    izvodjaca) citaju se karakteri iz datoteke */
    while ((c = fgetc(ulaz)) != ' ') {
118   /* Provera da li postoji dovoljno memorije za smestanje
         procitanog karaktera */
120   if (j == alocirano) {

122     /* Ako ne, ako je potrosena sva alocirana memorija, alocira
           se novih KORAK mesta */
124     alocirano += KORAK;
        pesme[i].izvodjac =
126         (char *) realloc(pesme[i].izvodjac,
                             alocirano * sizeof(char));
128
        /* Provera da li je nova alokacija uspesna */
130     if (pesme[i].izvodjac == NULL) {
          /* Ako nije oslobadja se sva memorija zauzeta do ovog
132          koraka */
          for (k = 0; k < i; k++) {
134         free(pesme[k].izvodjac);
            free(pesme[k].naslov);
136       }
          free(pesme);
138       /* I prekida sa izvrsavanjem programa */
          return 0;
140     }
```

```
142       }
          /* Ako postoji dovoljno alocirane memorije, smesta se vec
144          procitani karakter */
          pesme[i].izvodjac[j] = c;
146       j++;
          /* I nastavlja se sa citanjem */
148     }

150     /* Upis terminirajuce nule na kraj reci */
        pesme[i].izvodjac[j] = '\0';
152
        /* Preskace se karakter - */
154     fgetc(ulaz);

156     /* Preskace se razmak */
        fgetc(ulaz);
158
        /* Cita se naslov pesme */
160     j = 0;                      /* Pozicija na koju treba smestiti
                                       procitani karakter */
162     alocirano = 0;              /* Broj alociranih mesta */
        pesme[i].naslov = NULL;     /* Memorija za smestanje procitanih
164                                    karaktera */

166     /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
           karakteri iz datoteke */
168     while ((c = fgetc(ulaz)) != ',') {
          /* Provera da li postoji dovoljno memorije za smestanje
170        procitanog karaktera */
          if (j == alocirano) {
172         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
               se novih KORAK mesta */
174         alocirano += KORAK;
            pesme[i].naslov =
176             (char *) realloc(pesme[i].naslov,
                                 alocirano * sizeof(char));
178
            /* Provera da li je nova alokacija uspesna */
180         if (pesme[i].naslov == NULL) {
              /* Ako nije, oslobadja se sva memorija zauzeta do ovog
182              koraka */
              for (k = 0; k < i; k++) {
184             free(pesme[k].izvodjac);
                free(pesme[k].naslov);
186           }
              free(pesme[i].izvodjac);
188           free(pesme);

190           /* I prekida izvrsavanje programa */
              return 0;
192         }
```

```
      }
      /* Ako postoji dovoljno memorije, smesta se procitani karakter
    */
      pesme[i].naslov[j] = c;
      j++;
      /* I nastavlja dalje sa citanjem */
    }
    /* Upisuje se terminirajuca nula na kraj reci */
    pesme[i].naslov[j] = '\0';

    /* Preskace se razmak */
    fgetc(ulaz);

    /* Cita se broj gledanja */
    broj_gledanja = 0;

    /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
       datoteke */
    while ((c = fgetc(ulaz)) != '\n') {
      broj_gledanja = broj_gledanja * 10 + (c - '0');
    }
    pesme[i].broj_gledanja = broj_gledanja;

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

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

  /* Analiza argumenta komandne linije */
  if (argc == 1) {
    /* Nema dodatnih opcija => sortiranje po broju gledanja */
    qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
  } else {
    if (argc == 2 && strcmp(argv[1], "-n") == 0) {
      /* Sortiranje po naslovu */
      qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
    } else {
      if (argc == 2 && strcmp(argv[1], "-i") == 0) {
        /* Sortirnje po izvodjacu */
        qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
      } else {
        fprintf(stderr, "Nedozvoljeni argumenti!\n");
        free(pesme);
        return 0;
      }
    }
  }
```

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

  /* Oslobadjanje memorije */
  for (i = 0; i < n; i++) {
    free(pesme[i].izvodjac);
    free(pesme[i].naslov);
  }
  free(pesme);

  return 0;
}
```

Rešenje 3.28

NAPOMENA: *Rešenje koristi biblioteku za rad sa celobrojnim matricama iz zadatka 2.19.*

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

/* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
int zbir_vrste(int **a, int v, int m)
{
  int i, zbir = 0;

  for (i = 0; i < m; i++) {
    zbir += a[v][i];
  }
  return zbir;
}

/* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
   osnovu zbira koriscenjem selection sort algoritma */
void sortiraj_vrste(int **a, int n, int m)
{
  int i, j, min;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
        min = j;
      }
    }
    if (min != i) {
```

```
      int *tmp;
      tmp = a[i];
      a[i] = a[min];
      a[min] = tmp;
    }
  }
}

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

  /* Unos dimenzija matrice */
  printf("Unesite dimenzije matrice: ");
  scanf("%d %d", &n, &m);

  /* Alokacija memorije */
  a = alociraj_matricu(n, m);
  if (a == NULL) {
    fprintf(stderr, "Neuspesna alokacija matrice\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje elementa matrice */
  printf("Unesite elemente matrice po vrstama:\n");
  ucitaj_matricu(a, n, m);

  /* Poziv funkcije koja sortira vrste matrice prema zbiru */
  sortiraj_vrste(a, n, m);

  /* Ispis rezultujuce matrice */
  printf("Sortirana matrica je:\n");
  ispisi_matricu(a, n, m);

  /* Oslobadjanje memorije */
  a = dealociraj_matricu(a, n);

  return 0;
}
```

Rešenje 3.31

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

#define MAX 100

/* Funkcija poredjenja dva cela broja (neopadajuci poredak) */
```

```
9  int poredi_int(const void *a, const void *b)
   {
11   /* Potrebno je konvertovati void pokazivace u int pokazivace koji
        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13   int b1 = *((int *) a);
     int b2 = *((int *) b);
15
     /* Zbog moguceg prekoracenja opsega celih brojeva, oduzimanje b1-b2
17      treba izbegavati */
     if (b1 > b2)
19     return 1;
     else if (b1 < b2)
21     return -1;
     else
23     return 0;
   }
25
   /* Funkcija poredjenja dva cela broja (nerastuci poredak) */
27 int poredi_int_nerastuce(const void *a, const void *b)
   {
29   /* Za obrnuti poredak treba samo promeniti znak vrednosti koju koju
        vraca prethodna funkcija */
31   return -poredi_int(a, b);
   }
33
   int main()
35 {
     size_t n;
37   int i, x;
     int a[MAX], *p = NULL;
39
     /* Unos dimenzije */
41   printf("Uneti dimenziju niza: ");
     scanf("%ld", &n);
43   if (n > MAX)
       n = MAX;
45
     /* Unos elementa niza */
47   printf("Uneti elemente niza:\n");
     for (i = 0; i < n; i++)
49     scanf("%d", &a[i]);

51   /* Sortiranje niza celih brojeva */
     qsort(a, n, sizeof(int), &poredi_int);
53
     /* Prikaz sortiranog niz */
55   printf("Sortirani niz u rastucem poretku:\n");
     for (i = 0; i < n; i++)
57     printf("%d ", a[i]);
     putchar('\n');
59
     /* Pretrazivanje niza */
```

```
  /* Vrednost koja ce biti trazena u nizu */
  printf("Uneti element koji se trazi u nizu: ");
  scanf("%d", &x);

  /* Binarna pretraga */
  printf("Binarna pretraga: \n");
  p = bsearch(&x, a, n, sizeof(int), &poredi_int);
  if (p == NULL)
    printf("Elementa nema u nizu!\n");
  else
    printf("Element je nadjen na poziciji %ld\n", p - a);

  /* Linearna pretraga */
  printf("Linearna pretraga (lfind): \n");
  p = lfind(&x, a, &n, sizeof(int), &poredi_int);
  if (p == NULL)
    printf("Elementa nema u nizu!\n");
  else
    printf("Element je nadjen na poziciji %ld\n", p - a);

  return 0;
}
```

Rešenje 3.32

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

#define MAX 100

/* Funkcija racuna broj delilaca broja x */
int broj_delilaca(int x)
{
  int i;
  int br;

  /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
  if (x < 0)
    x = -x;
  if (x == 0)
    return 0;
  if (x == 1)
    return 1;
  /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
  br = 2;
  for (i = 2; i < sqrt(x); i++)
    if (x % i == 0)
      /* Ako i deli x onda su delioci: i, x/i */
      br += 2;
```

```
27    /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
         promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29       jos jednog delioca */
      if (i * i == x)
31      br++;

33    return br;
}
35
/* Funkcija poredjenja dva cela broja po broju delilaca */
37 int poredi_po_broju_delilaca(const void *a, const void *b)
{
39    int ak = *(int *) a;
      int bk = *(int *) b;
41    int n_d_a = broj_delilaca(ak);
      int n_d_b = broj_delilaca(bk);
43
      return n_d_a - n_d_b;
45 }

47 int main()
{
49    size_t n;
      int i;
51    int a[MAX];

53    /* Unos dimenzije */
      printf("Uneti dimenziju niza: ");
55    scanf("%ld", &n);
      if (n > MAX)
57      n = MAX;

59    /* Unos elementa niza */
      printf("Uneti elemente niza:\n");
61    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
63
      /* Sortiranje niza celih brojeva prema broju delilaca */
65    qsort(a, n, sizeof(int), &poredi_po_broju_delilaca);

67    /* Prikaz sortiranog niza */
      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

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

#define MAX_NISKI 1000
#define MAX_DUZINA 31

/*****************************************************************
  Niz nizova karaktera ovog potpisa
  char niske[3][4];
  se moze graficki predstaviti ovako:
  ---------------------------------------------------
  | a | b | c |\0 || d | e | \0|   || f | g | h | \0||
  ---------------------------------------------------
  Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
  nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
  adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
  i ona je tipa char*.

  Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
  koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
  reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
  nad njima.
*****************************************************************/
int poredi_leksikografski(const void *a, const void *b)
{
  return strcmp((char *) a, (char *) b);
}

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

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

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

```
51    fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
      exit(EXIT_FAILURE);
53  }

55  /* Citanje sadrzaja datoteke */
    for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
57
    /* Zatvaranje datoteke */
59  fclose(fp);
    n = i;
61
    /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
63     prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
       niske po duzini */
65  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

67  printf("Leksikografski sortirane niske:\n");
    for (i = 0; i < n; i++)
69    printf("%s ", niske[i]);
    printf("\n");
71
    /* Unos trazene niske */
73  printf("Uneti trazenu nisku: ");
    scanf("%s", x);
75
    /* Binarna pretraga */
77  /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
       je niz vec sortiran leksikografski. */
79  p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
                &poredi_leksikografski);
81
    if (p != NULL)
83    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
            p, (p - (char *) niske) / MAX_DUZINA);
85  else
      printf("Niska nije pronadjena u nizu\n");
87
    /* Sortiranje po duzini */
89  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

91  printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
93    printf("%s ", niske[i]);
    printf("\n");
95
    /* Linearna pretraga */
97  p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
              &poredi_leksikografski);
99
    if (p != NULL)
101   printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
            p, (p - (char *) niske) / MAX_DUZINA);
```

```
103   else
        printf("Niska nije pronadjena u nizu\n");
105
      exit(EXIT_SUCCESS);
107 }
```

Rešenje 3.34

```
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 pokazivaca na karaktere ovog potpisa
11   char *niske[3];
     posle alokacije u main-u se moze graficki predstaviti ovako:
13   -----                    -----------------
     | X | --------------->   | a | b | c | \0|
15   -----                    =================
     | Y | --------------->   | d | e | \0|
17   -----                    =================
     | Z | --------------->   | f | g | h | \0|
19   -----                    -----------------
     Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
21   njegov element pokazivac koji pokazuje na alocirane karaktere i-te
     reci. Njegov tip je char*.
23
     Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
25   koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i
     kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
27   moraju i kastovati. Da bi se leksikografski uporedili elementi X i
     Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
29   u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
     kastovani na odgovarajuci tip.
31 *****************************************************************/
   int poredi_leksikografski(const void *a, const void *b)
33 {
     return strcmp(*(char **) a, *(char **) b);
35 }

37 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
      leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
   {
41   return strlen(*(char **) a) - strlen(*(char **) b);
   }
43
```

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

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

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

  /* Citanje sadrzaja datoteke */
  i = 0;
  while (fscanf(fp, "%s", x) != EOF) {
    /* Alociranje dovoljne memorije za i-tu nisku */
    if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
    {
      fprintf(stderr, "Greska pri alociranju niske\n");
      exit(EXIT_FAILURE);
    }
    /* Kopiranje procitane niske na svoje mesto */
    strcpy(niske[i], x);
    i++;
  }

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

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

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

```
95    printf("\n");

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

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

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

117   /* Sortiramo po duzini */
      qsort(niske, n, sizeof(char *), &poredi_duzine);
119
      printf("Niske sortirane po duzini:\n");
121   for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
123   printf("\n");

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

129   exit(EXIT_SUCCESS);
    }
```

Rešenje 3.35

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

6   #define MAX 500

8   /* Struktura sa svim informacijama o pojedinacnom studentu */
    typedef struct {
10    char ime[21];
      char prezime[21];
12    int bodovi;
```

```
} Student;

/* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
   istim brojem bodova se dodatno sortiraju leksikografski po
   prezimenu */
int poredi1(const void *a, const void *b)
{
  Student *prvi = (Student *) a;
  Student *drugi = (Student *) b;

  if (prvi->bodovi > drugi->bodovi)
    return -1;
  else if (prvi->bodovi < drugi->bodovi)
    return 1;
  else
    /* Ako su jednaki po broju bodova, treba ih uporediti po
       prezimenu */
    return strcmp(prvi->prezime, drugi->prezime);
}

/* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
   Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
   parametar je element niza ciji se bodovi porede. */
int poredi2(const void *a, const void *b)
{
  int bodovi = *(int *) a;
  Student *s = (Student *) b;
  return s->bodovi - bodovi;
}

/* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
   Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
   parametar je element niza cije se prezime poredi. */
int poredi3(const void *a, const void *b)
{
  char *prezime = (char *) a;
  Student *s = (Student *) b;
  return strcmp(prezime, s->prezime);
}

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

  /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
     informacija korisniku kako se program koristi i prekida se
```

```
    izvrsavanje. */
  if (argc < 2) {
    fprintf(stderr,
            "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

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

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

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

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

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

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

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

  if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
  else
    printf("Nema studenta sa unetim brojem bodova\n");

  /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
     sortiran po bodovima. */
```

```
  printf("Unesite prezime: ");
  scanf("%s", prezime);

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

  if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
  else
    printf("Nema studenta sa unetim prezimenom\n");

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.36

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

#define MAX 128

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

/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
int anagrami(char s[], char t[])
{
  /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
  if (strlen(s) != strlen(t))
    return 0;

  /* Sortiranje niski */
  qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
  qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

  /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
     suprotnom, nisu */
  return !strcmp(s, t);
}

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

```

```
  /* Unos niski */
  printf("Unesite prvu nisku: ");
  scanf("%s", s);
  printf("Unesite drugu nisku: ");
  scanf("%s", t);

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

  return 0;
}
```

Rešenje 3.37

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

#define MAX 10
#define MAX_DUZINA 32

/* Funkcija porenjenja */
int uporedi_niske(const void *pa, const void *pb)
{
  return strcmp((char *) pa, (char *) pb);
}

int main()
{
  int i, n;
  char S[MAX][MAX_DUZINA];

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

  /* Unos niza niski */
  printf("Unesite niske:\n");
  for (i = 0; i < n; i++)
    scanf("%s", S[i]);

  /* Sortiranje niza niski */
  qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

  /*****************************************************************
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih niski. Koriscen je samo u fazi testiranja programa.
```

```
    printf("Sortirane niske su:\n");
    for(i = 0; i < n; i++)
      printf("%s ", S[i]);
  ****************************************************************/

  /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
     niza biti jedna do druge */
  for (i = 0; i < n - 1; i++)
    if (strcmp(S[i], S[i + 1]) == 0) {
      printf("ima\n");
      return 0;
    }
  printf("nema\n");

  return 0;
}
```

Rešenje 3.38

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

#define MAX 21

/* Struktura koja predstavlja jednog studenta */
typedef struct student {
  char nalog[8];
  char ime[MAX];
  char prezime[MAX];
  int poeni;
} Student;

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

/* Funkcija poredi studente prvo prema godini, zatim prema smeru i na
   kraju prema indeksu */
int uporedi_nalog(const void *a, const void *b)
{
  Student s = *(Student *) a;
  Student t = *(Student *) b;
  /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
     broj indeksa */
  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'
        ;
      int indeks2 =
65        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
          s.nalog[6] - '0';
67    if (godina1 != godina2)
        return godina1 - godina2;
69    else if (smer1 != smer2)
        return smer1 - smer2;
71    else
        return indeks1 - indeks2;
73  }

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

83    /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
```

```
    korisnik nije ispravno pokrenuo program. */
85  if (argc != 2 && argc != 3) {
      fprintf(stderr,
87            "Program se poziva sa: ./a.out -opcija [nalog]\n");
      exit(EXIT_FAILURE);
89  }

91  /* Otvaranje datoteke za citanje */
    in = fopen("studenti.txt", "r");
93  if (in == NULL) {
      fprintf(stderr,
95            "Greska prilikom otvarnja datoteke studenti.txt!\n");
      exit(EXIT_FAILURE);
97  }

99  /* Otvaranje datoteke za pisanje */
    out = fopen("izlaz.txt", "w");
101 if (out == NULL) {
      fprintf(stderr,
103           "Greska prilikom otvaranja datoteke izlaz.txt!\n");
      exit(EXIT_FAILURE);
105 }

107 /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
    while (fscanf
109        (in, "%s %s %s %d", niz_studenata[i].nalog,
           niz_studenata[i].ime, niz_studenata[i].prezime,
111        &niz_studenata[i].poeni) != EOF)
      i++;
113
    br_studenata = i;
115
    /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
117 if (strcmp(argv[1], "-p") == 0)
      qsort(niz_studenata, br_studenata, sizeof(Student),
119         &uporedi_poeni);
    /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
121 else if (strcmp(argv[1], "-n") == 0)
      qsort(niz_studenata, br_studenata, sizeof(Student),
123         &uporedi_nalog);

125 /* Sortirani studenti se ispisuju u izlaznu datoteku */
    for (i = 0; i < br_studenata; i++)
127   fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
            niz_studenata[i].ime, niz_studenata[i].prezime,
129         niz_studenata[i].poeni);

131 /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
       studenta... */
133 if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
      strcpy(nalog_trazeni, argv[2]);
135
```

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

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

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

  exit(EXIT_SUCCESS);
}
```

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:
 Unosite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [3, 2]
 14
 Lista: [14, 3, 2]
 5
 Lista: [5, 14, 3, 2]
 3
 Lista: [3, 5, 14, 3, 2]
 17
 Lista: [17, 3, 5, 14, 3, 2]

 Unesite broj koji se trazi:  5
 Trazeni broj 5 je u listi!
```

Primer 2

```
Interakcija sa programom:
 Unosite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [35, 14, 23]

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

(2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

Primer 1

```
Interakcija sa programom:
 Unosite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 14, 3]
 3
 Lista: [2, 3, 14, 3, 3]
 17
 Lista: [2, 3, 14, 3, 3, 17]
 3
 Lista: [2, 3, 14, 3, 3, 17, 3]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 14, 17]
```

Primer 2

```
Interakcija sa programom:
 Unesite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [23, 14]
 35
 Lista: [23, 14, 35]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [23, 14, 35]
```

(3) U programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. Napomena: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 3, 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: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [14, 23, 35]

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

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [14, 23, 35]
```

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 rekurzivno. 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 smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
  unsigned broj_pojavljivanja;
  char etiketa[20];
  struct _Element *sledeci;
} Element;
```

Test 1

```
POKRETANJE:  ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKE:
 Greska prilikom otvaranja
 datoteke datoteka.html.
```

Test 2

```
POKRETANJE: ./a.out datoteka.html

DATOTEKA.HTML
 <html>
  <head><title>Primer</title></head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    A sutra ce biti jos lepsi.
    <a link='http://www.google.com'> Link 1</a>
    <a link='http://www.math.rs'> Link 2</a>
  </body>
 </html>
```

```
IZLAZ:
 a - 4
 br - 1
 h1 - 2
 body - 2
 title - 2
 head - 2
 html - 2
```

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.

(a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

(b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

Test 1

```
BROJEVI.TXT
 43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
 12 15 16
```

Test 2

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAZ ZA GREŠKE:
 Greska prilikom otvaranja
 datoteke brojevi.txt.
```

Test 3

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
 Rezultat.txt ce biti prazna.
```

*** Zadatak 4.6** Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo 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.

Test 1

```
POKRETANJE: ./a.out dat1.txt dat2.txt

DAT1.TXT
 2 4 6 10 15

DAT2.TXT
 5 6 11 12 14 16

IZLAZ:
 [2, 4, 5, 6, 6, 10, 11, 12, 14, 15, 16]
```

Test 2

```
POKRETANJE: ./a.out dat1.txt dat2.txt

DAT1.TXT
 2 4 6 10 15

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ ZA GREŠKE:
 Greska prilikom otvaranja datoteke
 dat2.txt.
```

Test 3

```
POKRETANJE: ./a.out dat1.txt dat2.txt

DATOTEKA DAT1.TXT JE PRAZNA

DAT2.TXT
 5 6 11 12 14 16

IZLAZ:
 [5, 6, 11, 12, 14, 16]
```

Test 4

```
POKRETANJE: ./a.out dat1.txt

IZLAZ ZA GREŠKE:
 Program se poziva sa:
 ./a.out dat1.txt dat2.txt!
```

*** Zadatak 4.7** Date su dve jednostruko povezane liste $L1$ i $L2$. Napisati funkciju koja od ovih listi formira novu listu L koja sadrži naizmenično raspoređene čvorove listi $L1$ i $L2$: prvi čvor iz $L1$, prvi čvor iz $L2$, drugi čvor $L1$, drugi čvor $L2$, itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *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 prilikom otvaranja
 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 prilikom otvaranja
 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 vracate na kraj reda?` i ukoliko on da odgovor *Da*, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva

odlaže. Ukoliko odgovor nije *Da*, službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke `izvestaj.txt`. Ova datoteka, za svaki obrađen zahtev, sadrži $JMBG$ i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red 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 bibilioteku novim funkcijama.

(a) Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor * tekuci)` koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave` i poslednji čvor liste na adresi `adresa_kraja`.

(b) Napisati funkciju `void ispisi_listu_unazad(Cvor * kraj)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Zbog efikasnog izvršavanja operacija dodavanja na kraj liste i ispisivanja liste unazad treba, pored pokazivača na glavu liste, čuvati i pokazivač na poslednji čvor liste. NAPOMENA: *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

```
ULAZ:
 5 3

IZLAZ:
 3 1 5 2 4
```

Test 2

```
ULAZ:
 8 4

IZLAZ:
 4 8 5 2 1 3 7 6
```

Test 3

```
ULAZ:
 3 8

IZLAZ ZA GREŠKE:
 n mora biti uvek vece
 od k, a 3 < 8!
```

*** Zadatak 4.13** Grupa od n plesača na kostimima ima brojeve od 1 do n. Plesači najpre formiraju krug tako da brojevi sa njihovih kostima rastu u smeru kazaljke na satu. Svaki plesač levu ruku stavlja na rame plesača sa sledećim većim

brojem, a desnu na rame plesača sa prvim manjim brojem. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na rame plesača sa brojem n. Plesač sa brojem n svoju desnu ruku spušta na rame plesača sa brojem $n-1$, a levu na rame plesača sa brojem 1 i tako zatvara krug. Plesači izvode svoju plesnu tačku tako što iz formiranog kruga najpre izlazi k-ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k-ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n, k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.*

Test 1

```
ULAZ:
 5 3

IZLAZ:
 3 5 4 2 1
```

Test 2

```
ULAZ:
 8 4

IZLAZ:
 4 8 5 7 6 3 2 1
```

Test 3

```
ULAZ:
 5 8

IZLAZ ZA GREŠKE:
 n mora biti uvek vece
 od k, a 5 < 8!
```

4.2 Stabla

Zadatak 4.14 Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

(a) 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.

(b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.

(c) 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.

(d) 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, npr. *Pera Peric* 064/123–4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči *KRAJ*, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Primer 1

```
IMENIK.TXT
 Pera Peric 011/3240-987
 Marko Maric 064/1234-987
 Mirko Maric 011/589-333
 Sanja Savkovic 063/321-098
 Zika Zikic 021/759-858

INTERAKCIJA 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 u formatu *DD.MM.*. Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

Primer 1

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

RODJENDANI.TXT
  Marko Markovic 12.12.
  Milan Jevremovic 04.06.
  Maja Agic 23.04.
  Nadica Zec 01.01.
  Jovana Milic 05.05.

INTERAKCIJA SA PROGRAMOM:
  Unesite datum: 23.04.
  Slavljenik: Maja Agic
  Unesite datum: 01.01.
  Slavljenik: Nadica Zec
  Unesite datum: 01.05.
  Slavljenik: Jovana Milic 05.05.
  Unesite datum: 20.12.
  Slavljenik: Nadica Zec 01.01.
  Unesite datum:
```

Primer 2

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

DATOTEKA RODJENDANI.TXT NE POSTOJI

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

```
POKRETANJE: ./a.out 2 15

ULAZ:
  10 5 15 3 2 4 30 12 14 13

IZLAZ:
  Broj cvorova: 10
  Broj listova: 4
  Pozitivni listovi: 2 4 13 30
  Zbir cvorova: 108
  Najveci element: 30
  Dubina stabla: 5
  Broj cvorova na 2. nivou: 3
  Elementi na 2. nivou: 3 12 30
  Maksimalni element na 2. nivou: 30
  Zbir elemenata na 2. nivou: 45
  Zbir elemenata manjih ili jednakih od 15:
  78
```

Test 2

```
POKRETANJE: ./a.out 3 31

ULAZ:
  24 53 61 9 7 55 20 16

IZLAZ:
  Broj cvorova: 8
  Broj listova: 3
  Pozitivni listovi: 7 16 55
  Zbir cvorova: 245
  Najveci element: 61
  Dubina stabla: 4
  Broj cvorova na 3. nivou: 2
  Elementi na 3. nivou: 16 55
  Maksimalni element na 3. nivou: 55
  Zbir elemenata na 3. nivou: 71
  Zbir elemenata manjih ili jednakih od 31:
  76
```

Zadatak 4.23 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1

```
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
```

Test 2

```
Ulaz:
 6 11 8 3 -2

Izlaz:
 0.nivo: 6
 1.nivo: 3 11
 2.nivo: -2 8
```

Test 3

```
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
```

*** Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1

```
Interakcija sa programom:
 Prvo stablo: 11 20 5 3 0
 Drugo stablo: 8 14 30 1 0
 Stabla su slicna kao u ogledalu.
```

Primer 2

```
Interakcija sa programom:
 Prvo stablo: 11 20 5 3 0
 Drugo stablo: 8 20 15 0
 Stabla nisu slicna kao u ogledalu.
```

Zadatak 4.25 *AVL*-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor * koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza. Napomena: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1

```
Ulaz:
 10 5 15 2 11 16 1 13

Izlaz:
 7
```

Test 2

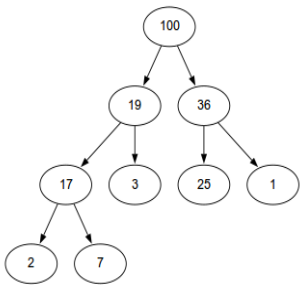
```
Ulaz:
 16 30 40 24 10 18 45 22

Izlaz:
 6
```

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 osta-

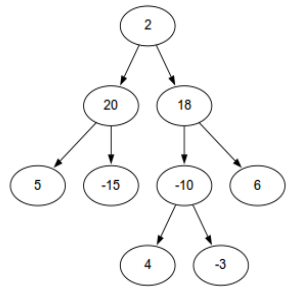
lih čvorova u njegovim podstablima. Napisati funkciju `int hip(Cvor * koren)` koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `hip` i ispisuje rezultat na standardni izlaz. NAPOMENA: *Za alokaciju i oslobađanje 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.

(a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

(b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.

(c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.

(d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

Test 1

```
IZLAZ:
 Vrednost u cvoru sa maksimalnim desnim zbirom: 18
 Vrednost u cvoru sa minimalnim levim zbirom: 18
 2 18 -10 4
 2 20 -15
```

4.3 Rešenja

Rešenje 4.1

lista.h

```
#ifndef _LISTA_H_
#define _LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

/* 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,
42    inace vraca 0. */
   int dodaj_sortirano(Cvor ** adresa_glave, int broj);
44
   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
46    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
      NULL u slucaju 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 cinjenicu da je lista koja se pretrazuje
52    neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
      sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
       */
54 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

56 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
      pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
58    obrise stara glava. */
   void obrisi_cvor(Cvor ** adresa_glave, int broj);
60
   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
62    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
      neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
64    promenjen ukoliko se obrise stara glava liste. */
   void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
66
   /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
68    liste, razdvojene zapetama 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   /* Alocira se memorija za novi cvor liste i proverava se uspesnost
       alokacije */
9   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
```

```
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  /* Ako lista nije prazna, onda treba osloboditi memoriju */
  while (*adresa_glave != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor koji predstavlja glavu liste */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);

    /* Sledeci cvor je nova glava liste */
    *adresa_glave = pomocni;
  }
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

Cvor *pronadji_poslednji(Cvor * glava)
{
  /* U praznoj listi nema cvorova pa se vraca NULL */
  if (glava == NULL)
    return NULL;

  /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
     ce pokazivati na cvor liste koji nema sledbenika, tj. na
     poslednji cvor liste pa se vraca vrednost pokazivaca glava.
```

```
     Pokazivac glava je argument funkcije i njegove promene nece se
63   odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
  while (glava->sledeci != NULL)
65   glava = glava->sledeci;

67 return glava;
}
69
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
71 {
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
73 Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
75   return 1;

77 /* Ako je lista prazna */
  if (*adresa_glave == NULL) {
79   /* Glava nove liste je upravo novi cvor */
    *adresa_glave = novi;
81 } else {
    /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
83     se dodaje na kraj liste kao sledbenik poslednjeg */
    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
85  poslednji->sledeci = novi;
  }
87
  /* Vraca se indikator uspesnog dodavanja */
89 return 0;
}
91
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
93 {
  /* U praznoj listi nema takvog mesta i vraca se NULL */
95 if (glava == NULL)
    return NULL;
97
  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
99   pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
     vecu ili jednaku vrednosti novog cvora.
101  Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
     provera da li se doslo do poslednjeg cvora liste pre nego sto se
103  proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
     sledeci ne postoji, pa ni njegova vrednost. */
105 while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
    glava = glava->sledeci;
107
  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
109   poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
     vrednost vecu od broj. */
111 return glava;
}
113
```

```
int dodaj_iza(Cvor * tekuci, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se dodaje iza tekuceg cvora. */
  novi->sledeci = tekuci->sledeci;
  tekuci->sledeci = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {
    /* Glava nove liste je novi cvor */
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
      return 1;

    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Inace, ako je broj manji ili jednak vrednosti u glavi liste,
     onda ga treba dodati na pocetak liste. */
  if ((*adresa_glave)->vrednost >= broj) {
    return dodaj_na_pocetak_liste(adresa_glave, broj);
  }

  /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  return dodaj_iza(pomocni, broj);
}

Cvor *pretrazi_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  for (; glava != NULL; glava = glava->sledeci)
    /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
       se obustavlja */
    if (glava->vrednost == broj)
      return glava;

```

```
  /* Nema trazenog broja u listi i vraca se NULL */
  return NULL;
}

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  /* U uslovu ostanka u petlji, bitan je redosled provera u
     konjunkciji. */
  while (glava != NULL && glava->vrednost < broj)
    glava = glava->sledeci;

  /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
     glava->vrednost veca od trazenog broja i tada treba vratiti
     NULL, jer trazen broj nije nadjen medju manjim brojevima pri
     pocetku sortirane liste, pa se moze zakljuciti da ga nema u
     listi. Drugi nacini, tako sto se doslo do kraja liste i glava je
     NULL ili tako sto je glava->vrednost == broj. U oba poslednja
     nacina treba vratiti pokazivac glava bilo da je NULL ili
     pokazivac na konkretan cvor. */
  if (glava->vrednost > broj)
    return NULL;
  else
    return glava;
}

void obrisi_cvor(Cvor ** adresa_glave, int broj)
{
  Cvor *tekuci = NULL;
  Cvor *pomocni = NULL;

  /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
     broju i azurira se pokazivac na glavu liste */
  while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
    /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
       adresi adresa_glave */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *adresa_glave = pomocni;
  }

  /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
     funkcije */
  if (*adresa_glave == NULL)
    return;

  /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
     pokazuje na cvor cija je vrednost razlicita od trazenog broja.
     Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
     sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
     ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
```

```
217       Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
        */
    tekuci = *adresa_glave;
219   while (tekuci->sledeci != NULL)
      if (tekuci->sledeci->vrednost == broj) {
221     /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
           prvo cuva u promenljivoj pomocni. */
223     pomocni = tekuci->sledeci;
        /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
225        njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
           sledeci od cvora koji se brise. */
227     tekuci->sledeci = pomocni->sledeci;
        /* Sada treba osloboditi cvor sa vrednoscu broj. */
229     free(pomocni);
      } else {
231     /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
           pomera na sledeci. */
233     tekuci = tekuci->sledeci;
      }
235   return;
  }
237
  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
239 {
    Cvor *tekuci = NULL;
241   Cvor *pomocni = NULL;

243   /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
        broju i azurira se pokazivac na glavu liste. */
245   while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
        {
      /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
247        adresi adresa_glave. */
      pomocni = (*adresa_glave)->sledeci;
249   free(*adresa_glave);
      *adresa_glave = pomocni;
251   }

253   /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
        prekida. Isto se radi i ukoliko glava liste sadrzi vrednost koja
255     je veca od broja, jer kako je lista sortirana rastuce nema
        potrebe broj traziti u repu liste. */
257   if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
      return;
259
    /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
261     na cvor cija vrednost je manja od trazenog broja, kao i svim
        cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
263     je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
        ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
265     cija vrednost je veca od trazenog broja. */
    tekuci = *adresa_glave;
```

```
267   while (tekuci->sledeci != NULL && 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_a.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 dodavanja novog broja na pocetak liste */
     printf("Unesite brojeve: (za kraj CTRL+D)\n");
15   while (scanf("%d", &broj) > 0) {
       /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17        memorije za nov cvor. Memoriju alociranu za cvorove liste
          treba osloboditi pre napustanja programa. */
19     if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
```

```
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testiranje funkcije 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);

  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

main_b.c

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

/* 2) Glavni program */
int main()
{
  /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
  int broj;

  /* Testira se funkcija za dodavanja novog broja na kraj liste */
  printf("Unesite brojeve: (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

```

```
  /* Testira se funkcije kojom se brise cvor liste */
  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
     procitanom sa ulaza */
  obrisi_cvor(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

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 neopadajuce */
  printf("Unosite brojeve (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_sortirano(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija 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);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.2

lista.h

```
#ifndef _LISTA_H_
#define _LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
```

```
24 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
      pri alokaciji memorije, inace vraca 0. */
26 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

28 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
      ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
30    memorije, inace vraca 0. */
   int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32
   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
34    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
      NULL u slucaju da takav cvor ne postoji u listi. */
36 Cvor *pretrazi_listu(Cvor * glava, int broj);

38 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
      U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
40    neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
      sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
       */
42 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

44 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
      pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
46    obrise stara glava liste. */
   void obrisi_cvor(Cvor ** adresa_glave, int broj);
48
   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
50    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
      neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
52    promenjen ukoliko se obrise stara glava liste. */
   void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
54
   /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
56    zapetama. */
   void ispisi_vrednosti(Cvor * glava);
58
   /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
60    liste, razdvojene zapetama i uokvirene zagradama. */
   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   /* Alocira se memorija za novi cvor liste i proverava se uspesnost
```

```
    alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
  /* Ako je lista vec prazna */
  if (*adresa_glave == NULL)
    return;

  /* Ako lista nije prazna, treba osloboditi memoriju. Treba
     osloboditi rep liste, pre oslobadjanja memorije za glavu liste
    */
  oslobodi_listu(&(*adresa_glave)->sledeci);
  /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
     glava u pozivajucoj funkciji tako da odgovara praznoj listi */
  free(*adresa_glave);
  *adresa_glave = NULL;
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {

    /* Novi cvor postaje glava liste */
    Cvor *novi = napravi_cvor(broj);
    /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
```

```
    */
    if (novi == NULL)
      return 1;

    /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
       azurira i pokazivacka promenljiva u pozivajucoj funkciji */
    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Ako lista nije prazna, broj se dodaje u rep liste. */
  /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
     novi broj se dodaje u rep liste u rekurzivnom pozivu.
     Informaciju o uspesnosti alokacije u rekurzivnom pozivu funkcija
     prosledjuje visem rekurzivnom pozivu koji tu informaciju vraca u
     rekurzivni poziv iznad, sve dok se ne vrati u main. Tek je iz
     main funkcije moguce pristupiti pravom pocetku liste i
     osloboditi je celu, ako ima potrebe. Ako je funkcija vratila 0,
     onda nije bilo greske. */
  return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
}

int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {

    /* Novi cvor postaje glava liste */
    Cvor *novi = napravi_cvor(broj);

    /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
    */
    if (novi == NULL)
      return 1;

    /* Azurira se glava liste */
    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
     glavi liste, onda se dodaje na pocetak liste */
  if ((*adresa_glave)->vrednost >= broj)
    return dodaj_na_pocetak_liste(adresa_glave, broj);

  /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
     bude sortirana lista. */
  return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
```

```
109 }

111 Cvor *pretrazi_listu(Cvor * glava, int broj)
    {
113   /* U praznoj listi nema vrednosti */
      if (glava == NULL)
115     return NULL;

117   /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
      if (glava->vrednost == broj)
119     return glava;

121   /* Inace, pretraga se nastavlja u repu liste */
      return pretrazi_listu(glava->sledeci, broj);
123 }

125 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
    {
127   /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
         vrednosti u glavi liste, jer je lista neopadajuce sortirana */
129   if (glava == NULL || glava->vrednost > broj)
        return NULL;
131
      /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
133   if (glava->vrednost == broj)
        return glava;
135
      /* Inace, pretraga se nastavlja u repu liste */
137   return pretrazi_listu(glava->sledeci, broj);
    }
139
    void obrisi_cvor(Cvor ** adresa_glave, int broj)
141 {
      /* U praznoj listi nema cvorova za brisanje. */
143   if (*adresa_glave == NULL)
        return;
145
      /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
147   obrisi_cvor(&(*adresa_glave)->sledeci, broj);

149   /* Preostaje provera da li glavu liste treba obrisati */
      if ((*adresa_glave)->vrednost == broj) {
151     /* Pomocni pokazuje na cvor koji treba da se obrise */
        Cvor *pomocni = *adresa_glave;
153     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
           brise se cvor koji je bio glava liste. */
155     *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
157   }
    }
159
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
```

```
{
  /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
     od broja, kako je lista sortirana rastuce nema potrebe broj
     traziti u repu liste i zato se funkcija prekida */
  if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
    return;

  /* Brisu se cvorovi iz repa koji imaju vrednost broj */
  obrisi_cvor(&(*adresa_glave)->sledeci, broj);

  /* Preostaje provera da li glavu liste treba obrisati */
  if ((*adresa_glave)->vrednost == broj) {
    /* Pomocni pokazuje na cvor koji treba da se obrise */
    Cvor *pomocni = *adresa_glave;
    /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
       brise se cvor koji je bio glava liste */
    *adresa_glave = (*adresa_glave)->sledeci;
    free(pomocni);
  }
}

void ispisi_vrednosti(Cvor * glava)
{
  /* Prazna lista */
  if (glava == NULL)
    return;

  /* Ispisuje se vrednost u glavi liste */
  printf("%d", glava->vrednost);

  /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
     se poziva ista funkcija za ispis ostalih. */
  if (glava->sledeci != NULL) {
    printf(", ");
    ispisi_vrednosti(glava->sledeci);
  }
}

void ispisi_listu(Cvor * glava)
{
  /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
     jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
     na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
     zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
     rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
     elemente razdvojene zapetom i razmakom. */
  putchar('[');
  ispisi_vrednosti(glava);
  printf("]\n");
}
```

Rešenje 4.3

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

#define MAX_DUZINA 20

/* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
typedef struct _Cvor {
  char etiketa[20];
  unsigned broj_pojavljivanja;
  struct _Cvor *sledeci;
} Cvor;

/* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
   ili NULL ako alokacija nije uspesno izvrsena */
Cvor *napravi_cvor(unsigned br, char *etiketa)
{
  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->broj_pojavljivanja = br;
  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

/* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
     azurira se vrednost glave liste */
  while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *adresa_glave = pomocni;
  }
  /* Pokazivac glava u main funkciji, na adresi adresa_glave, bice
     postavljen na NULL vrednost po izlasku iz petlje. */
}

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
```

```
   do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
                           char *etiketa)
{
  /* Kreira se novi cvor i proverava se uspesnost alokacije */
  Cvor *novi = napravi_cvor(br, etiketa);
  if (novi == NULL)
    return 1;

  /* Dodaje se novi cvor na pocetak liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
   NULL ako takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
{
  Cvor *tekuci;

  /* Obilazi se cvor po cvor liste */
  for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
    /* Ako tekuci cvor sadrzi trazenu etiketu, vraca se njegova
       vrednost */
    if (strcmp(tekuci->etiketa, etiketa) == 0)
      return tekuci;

  /* Cvor nije pronadjen */
  return NULL;
}

/* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Cvor * glava)
{
  /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
     broj njihovog pojavljivanja u HTML datoteci. */
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
}

/* Glavni program */
int main(int argc, char **argv)
{
  /* Provera se da li je program pozvan sa ispravnim brojem
     argumenata. */
  if (argc != 2) {
    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out datoteka.html!\n")
    ;
```

```
102       exit(EXIT_FAILURE);
    }
104
    /* Priprema datoteke za citanje */
106 FILE *in = NULL;
    in = fopen(argv[1], "r");
108 if (in == NULL) {
      fprintf(stderr,
110           "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
      exit(EXIT_FAILURE);
112 }

114 char c;
    int i = 0;
116 char procitana[MAX_DUZINA];
    Cvor *glava = NULL;
118 Cvor *trazeni = NULL;

120 /* Cita se karakter po karakter datoteke sve dok se ne procita cela
       datoteka */
122 while ((c = fgetc(in)) != EOF) {

124   /* Proverava se da li se pocinje sa citanjem nove etikete */
      if (c == '<') {
126     /* Proverava se da li se cita zatvarajuca etiketa */
        if ((c = fgetc(in)) == '/') {
128       i = 0;
          while ((c = fgetc(in)) != '>')
130         procitana[i++] = c;
        }
132     /* Cita se otvarajuca etiketa */
        else {
134       i = 0;
          procitana[i++] = c;
136       while ((c = fgetc(in)) != ' ' && c != '>')
            procitana[i++] = c;
138     }
        procitana[i] = '\0';
140
        /* Trazi se procitana etiketa medju postojecim cvorovima liste.
142          Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
    sa
             brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
144          etikete */
        trazeni = pretrazi_listu(glava, procitana);
146     if (trazeni == NULL) {
          if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
148         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
            oslobodi_listu(&glava);
150         exit(EXIT_FAILURE);
          }
152     } else
```

```
            trazeni->broj_pojavljivanja++;
154       }
        }
156
        /* Zatvaranje datoteke */
158     fclose(in);

160     /* Ispisuje se sadrzaj cvorova liste */
        ispisi_listu(glava);
162
        /* Oslobadja se memorija zauzeta listom */
164     oslobodi_listu(&glava);

166     exit(EXIT_SUCCESS);
    }
```

Rešenje 4.4

```
    #include <stdio.h>
2   #include <stdlib.h>
    #include <string.h>
4
    #define MAX_INDEKS 11
6   #define MAX_IME_PREZIME 21

8   /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
       studentu */
10  typedef struct _Cvor {
      char broj_indeksa[MAX_INDEKS];
12    char ime[MAX_IME_PREZIME];
      char prezime[MAX_IME_PREZIME];
14    struct _Cvor *sledeci;
    } Cvor;
16
    /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
18     novi cvor ili NULL ukoliko je doslo do greske */
    Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20  {
      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
22       alokacije */
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24    if (novi == NULL)
        return NULL;
26
      /* Inicijalizacija polja strukture */
28    strcpy(novi->broj_indeksa, broj_indeksa);
      strcpy(novi->ime, ime);
30    strcpy(novi->prezime, prezime);
      novi->sledeci = NULL;
32
      /* Vraca se adresa novog cvora */
```

```
34    return novi;
}
36
/* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void oslobodi_listu(Cvor ** adresa_glave)
{
40    /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
    if (*adresa_glave == NULL)
42      return;

44    /* Rekurzivnim pozivom se oslobadja rep liste */
    oslobodi_listu(&(*adresa_glave)->sledeci);
46
    /* Potom se oslobadja i glava liste */
48    free(*adresa_glave);

50    /* Proglasava se lista praznom */
    *adresa_glave = NULL;
52 }

54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
     do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
                            char *ime, char *prezime)
58 {
    /* Kreira se novi cvor i proverava se uspesnost alokacije */
60    Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
    if (novi == NULL)
62      return 1;

64    /* Dodaje se novi cvor na pocetak liste */
    novi->sledeci = *adresa_glave;
66    *adresa_glave = novi;

68    /* Vraca se indikator uspesnog dodavanja */
    return 0;
70 }

72 /* Funkcija ispisuje sadrzaj cvorova liste. */
void ispisi_listu(Cvor * glava)
74 {
    /* Pocevsi od glave liste */
76    for (; glava != NULL; glava = glava->sledeci)
      printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
78            glava->prezime);
}
80
/* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj indeksa.
82    U suprotnom funkcija vraca NULL */
Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
84 {
    /* Ako je lista prazna, ne postoji trazeni cvor */
```

```
86    if (glava == NULL)
        return NULL;
88
      /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste
        */
90    if (!strcmp(glava->broj_indeksa, broj_indeksa))
        return glava;
92
      /* Ukoliko u glavi liste nije trazeni indeks, pretraga se nastavlja
94       u repu liste */
      return pretrazi_listu(glava->sledeci, broj_indeksa);
96  }

98  /* Glavni program */
    int main(int argc, char **argv)
100 {
      /* Argumenti komandne linije su neophodni jer se iz komandne linije
102      dobija ime datoteke sa informacijama o studentima */
      if (argc != 2) {
104     fprintf(stderr,
                "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
106     exit(EXIT_FAILURE);
      }
108
      /* Priprema datoteke za citanje */
110   FILE *in = NULL;
      in = fopen(argv[1], "r");
112   if (in == NULL) {
        fprintf(stderr,
114             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
116   }

118   /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
        listu */
120   char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
      char broj_indeksa[MAX_INDEKS];
122   Cvor *glava = NULL;
      Cvor *trazeni = NULL;
124
      /* Ucitavanje vrednosti u listu */
126   while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
        if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
128       fprintf(stderr, "Neuspela alokacija za nov cvor\n");
          oslobodi_listu(&glava);
130       exit(EXIT_FAILURE);
        }
132
      /* Datoteka vise nije potrebna i zatvara se. */
134   fclose(in);

136   /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
```

```
  while (scanf("%s", broj_indeksa) != EOF) {
138   trazeni = pretrazi_listu(glava, broj_indeksa);
    if (trazeni == NULL)
140   printf("ne\n");
    else
142   printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
  }
144
  /* 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>
   #include <stdlib.h>
3  #include "lista.h"

5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
      na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7     pokazivaca postojecih cvorova listi */
   Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
     /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
11   Cvor *lista1 = *adresa_glave_1;
     Cvor *lista2 = *adresa_glave_2;
13
     /* Pokazivac na pocetni cvor rezultujuce liste */
15   Cvor *rezultujuca = NULL;
     Cvor *tekuci = NULL;
17
     /* Ako su obe liste prazne i rezultat je prazna lista */
19   if (lista1 == NULL && lista2 == NULL)
       return NULL;
21
     /* Ako je prva lista prazna, rezultat je druga lista */
23   if (lista1 == NULL)
       return lista2;
25
     /* Ako je druga lista prazna, rezultat je prva lista */
27   if (lista2 == NULL)
       return lista1;
29
     /* Odredjuje se prvi cvor rezultujuce liste - to je ili prvi cvor
31      liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
        koji sadrzi manju vrednost */
33   if (lista1->vrednost < lista2->vrednost) {
```

```
    rezultujuca = lista1;
    lista1 = lista1->sledeci;
  } else {
    rezultujuca = lista2;
    lista2 = lista2->sledeci;
  }
  /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste, ne
     sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
     koji sadrzi adresu trenutnog cvora rezultujuce liste */
  tekuci = rezultujuca;

  /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi cvor
     tako da bude uredjena neopadajuce. Pokazivac na listu iz koje se
     uzima cvor se azurira tako da pokazuje na sledeci cvor */
  while (lista1 != NULL && lista2 != NULL) {
    if (lista1->vrednost < lista2->vrednost) {
      tekuci->sledeci = lista1;
      lista1 = lista1->sledeci;
    } else {
      tekuci->sledeci = lista2;
      lista2 = lista2->sledeci;
    }
    tekuci = tekuci->sledeci;
  }

  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
     rezultujucu listu treba nadovezati ostatak druge liste */
  if (lista1 == NULL)
    tekuci->sledeci = lista2;
  else
    /* U suprotnom treba nadovezati ostatak prve liste */
    tekuci->sledeci = lista1;

  /* Preko adresa glava polaznih listi vrednosti pokazivaca u
     pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
     prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
     se moze doci prateci pokazivace iz glave rezultujuce liste, tako
     da stare pokazivace treba postaviti na NULL. */
  *adresa_glave_1 = NULL;
  *adresa_glave_2 = NULL;

  return rezultujuca;
}

int main(int argc, char **argv)
{
  /* Argumenti komandne linije su neophodni */
  if (argc != 3) {
    fprintf(stderr,
            "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
    exit(EXIT_FAILURE);
  }
```

```
87  /* Otvaranje datoteka u kojima se nalaze elementi listi */
    FILE *in1 = NULL;
89  in1 = fopen(argv[1], "r");
    if (in1 == NULL) {
91    fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
93  }
      exit(EXIT_FAILURE);
95
      FILE *in2 = NULL;
97    in2 = fopen(argv[2], "r");
      if (in2 == NULL) {
99      fprintf(stderr,
              "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
101     exit(EXIT_FAILURE);
      }
103
      /* Liste su na pocetku prazne */
105   int broj;
      Cvor *lista1 = NULL;
107   Cvor *lista2 = NULL;

109   /* Ucitavanje listi */
      while (fscanf(in1, "%d", &broj) != EOF)
111     dodaj_na_kraj_liste(&lista1, broj);

113   while (fscanf(in2, "%d", &broj) != EOF)
        dodaj_na_kraj_liste(&lista2, broj);
115
      /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
117   fclose(in1);
      fclose(in2);
119
      /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
121      objedinjavanjem listi */
      Cvor *rezultat = objedini(&lista1, &lista2);
123
      /* Ispis rezultujuce liste. */
125   ispisi_listu(rezultat);

127   /* Lista rezultat dobijena je prevezivanjem cvorova polaznih
         listi. Njenim oslobadjanjem bice oslobodjena sva zauzeta
129      memorija. */
      oslobodi_listu(&rezultat);
131
      exit(EXIT_SUCCESS);
133 }
```

Rešenje 4.8

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

/* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
   predstavlja vidjenu zagradu i pokazivac na sledeci cvor liste */
typedef struct cvor {
  char zagrada;
  struct cvor *sledeci;
} Cvor;

/* Funkcija koja oslobadja memoriju zauzetu stekom */
void oslobodi_stek(Cvor ** stek)
{
  Cvor *tekuci;
  Cvor *pomocni;

  /* Oslobadja se cvor po cvor steka */
  tekuci = *stek;
  while (tekuci != NULL) {
    pomocni = tekuci->sledeci;
    free(tekuci);
    tekuci = pomocni;
  }

  /* Stek se proglasava praznim */
  *stek = NULL;
}

/* Glavni program */
int main()
{
  /* Stek je na pocetku prazan */
  Cvor *stek = NULL;
  FILE *ulaz = NULL;
  char c;
  Cvor *pomocni = NULL;

  /* Otvaranje datotoke za citanje izraza */
  ulaz = fopen("izraz.txt", "r");
  if (ulaz == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke izraz.txt!\n");
    exit(EXIT_FAILURE);
  }

  /* Cita se karakter po karakter iz datoteke dok se ne dodje do
     kraja */
  while ((c = fgetc(ulaz)) != EOF) {
    /* Ako je ucitana otvorena zagrada, stavlja se na stek */
    if (c == '(' || c == '{' || c == '[') {
```

```
51        /* Alocira se memorija za novi cvor liste i proverava se
             uspesnost alokacije */
53        pomocni = (Cvor *) malloc(sizeof(Cvor));
          if (pomocni == NULL) {
55          fprintf(stderr, "Greska prilikom alokacije memorije!\n");
            /* Oslobadja se memorija zauzeta stekom */
57          oslobodi_stek(&stek);
            /* I prekida se sa izvrsavanjem programa */
59          exit(EXIT_FAILURE);
          }
61
          /* Inicijalizacija polja strukture */
63        pomocni->zagrada = c;

65        /* Promena vrha steka */
          pomocni->sledeci = stek;
67        stek = pomocni;
        }
69      /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
           prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
71         otvorena zagrada */
        else {
73        if (c == ')' || c == '}' || c == ']') {
            if (stek != NULL && ((stek->zagrada == '(' && c == ')')
75                               || (stek->zagrada == '{' && c == '}')
                                 || (stek->zagrada == '[' && c == ']')))
      {
77            /* Sa vrha steka se uklanja otvorena zagrada */
              pomocni = stek->sledeci;
79            free(stek);
              stek = pomocni;
81          } else {
              /* Dakle zagrade u izrazu nisu ispravno uparene */
83            break;
            }
85        }
        }
87    }

89    /* Procitana je cela datoteka i treba je zatvoriti */
      fclose(ulaz);
91
      /* Ako je stek prazan i procitana je cela datoteka, zagrade su
93       ispravno uparene */
      if (stek == NULL && c == EOF)
95      printf("Zagrade su ispravno uparene.\n");
      else {
97      /* U suprotnom se zakljucuje da zagrade nisu ispravno uparene */
        printf("Zagrade nisu ispravno uparene.\n");
99      /* Oslobadja se memorija koja je ostala zauzeta stekom */
        oslobodi_stek(&stek);
101   }
```

```
103   exit(EXIT_SUCCESS);
}
```

Rešenje 4.9

stek.h

```
#ifndef _STEK_H_
#define _STEK_H_

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

#define MAX 100

#define OTVORENA 1
#define ZATVORENA 2

#define VAN_ETIKETE 0
#define PROCITANO_MANJE 1
#define U_ETIKETI 2

/* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
   pokazivac na sledeci cvor */
typedef struct cvor {
  char etiketa[MAX];
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
   adresu ili NULL ako alokacija nije bila uspesna */
Cvor *napravi_cvor(char *etiketa);

/* Funkcija oslobadja memoriju zauzetu stekom */
void oslobodi_stek(Cvor ** adresa_vrha);

/* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske pri
   alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);

/* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
   pokazivac razlicit od NULL, tada u niz karaktera na koji on
   pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
   suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
   samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
   suprotnom */
int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);
```

```
/* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
   steka. Ukoliko je stek prazan, vraca NULL */
char *vrh_steka(Cvor * vrh);

/* Funkcija prikazuje stek od vrha prema dnu */
void prikazi_stek(Cvor * vrh);

/* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
   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
   procita etiketa. Vraca OTVORENA, ako je procitana otvorena
   etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
int uzmi_etiketu(FILE * f, char *etiketa);

#endif
```

stek.c

```
#include "stek.h"

Cvor *napravi_cvor(char *etiketa)
{
  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja u novom cvoru */
  if (strlen(etiketa) >= MAX) {
    fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
    etiketa[MAX - 1] = '\0';
  }
  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_stek(Cvor ** adresa_vrha)
{
  Cvor *pomocni;

  /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
  while (*adresa_vrha != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor koji predstavlja vrh steka */
    pomocni = *adresa_vrha;
    /* Sledeci cvor je novi vrh steka */
```

```
    *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);
  }

  /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
     adresa_vrha ce pokazivati na NULL. */
}

int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(etiketa);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
  novi->sledeci = *adresa_vrha;
  *adresa_vrha = novi;
  return 0;
}

int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
{
  Cvor *pomocni;

  /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
     vraca se 0 */
  if (*adresa_vrha == NULL)
    return 0;

  /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
     adresu kopira etiketa sa vrha steka */
  if (etiketa != NULL)
    strcpy(etiketa, (*adresa_vrha)->etiketa);

  /* Element sa vrha steka se uklanja */
  pomocni = *adresa_vrha;
  *adresa_vrha = (*adresa_vrha)->sledeci;
  free(pomocni);

  /* Vraca se indikator uspesno izvrsene radnje */
  return 1;
}

char *vrh_steka(Cvor * vrh)
{
  /* Prazan stek nema cvor koji je vrh i vraca se NULL */
  if (vrh == NULL)
    return NULL;

  /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
     koji je na vrhu steka. */
```

```
  return vrh->etiketa;
}

void prikazi_stek(Cvor * vrh)
{
  /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
  for (; vrh != NULL; vrh = vrh->sledeci)
    printf("<%s>\n", vrh->etiketa);
}

int uzmi_etiketu(FILE * f, char *etiketa)
{
  int c;
  int i = 0;
  /* Stanje predstavlja informaciju dokle se stalo sa citanjem
     etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
     nije zapoceto citanje. */
  /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
     OTVORENA ili ZATVORENA. */
  int stanje = VAN_ETIKETE;
  int tip;

  /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
     to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
     samo malim slovima. */
  while ((c = fgetc(f)) != EOF) {
    switch (stanje) {
    case VAN_ETIKETE:
      if (c == '<')
        stanje = PROCITANO_MANJE;
      break;
    case PROCITANO_MANJE:
      if (c == '/') {
        /* Cita se zatvorena etiketa */
        tip = ZATVORENA;
      } else {
        if (isalpha(c)) {
          /* Cita se otvorena etiketa */
          tip = OTVORENA;
          etiketa[i++] = tolower(c);
        }
      }
      /* Od sada se cita etiketa i zato se menja stanje */
      stanje = U_ETIKETI;
      break;
    case U_ETIKETI:
      if (isalpha(c) && i < MAX - 1) {
        /* Ako je procitani karakter slovo i nije prekoracena
           dozvoljena duzina etikete, procitani karakter se smanjuje
           i smesta u etiketu */
        etiketa[i++] = tolower(c);
      } else {
```

```
        /* Inace, staje se sa citanjem etikete. Korektno se zavrsava
           niska koja sadrzi procitanu etiketu i vraca se njen tip */
        etiketa[i] = '\0';
        return tip;
      }
      break;
    }
  }
  /* Doslo se do kraja datoteke pre nego sto je procitana naredna
     etiketa i vraca se EOF. */
  return EOF;
}
```

main.c

```
#include "stek.h"

/* Glavni program */
int main(int argc, char **argv)
{
  /* 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;

  /* Ime datoteke se preuzima iz komandne linije. */
  if (argc < 2) {
    fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  /* Datoteka se otvara za citanje */
  if ((f = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
            argv[1]);
    exit(EXIT_FAILURE);
  }

  /* 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
          && strcmp(etiketa, "hr") != 0
```

```
            && strcmp(etiketa, "meta") != 0)
38          if (potisni_na_stek(&vrh, etiketa) == 1) {
              fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40            oslobodi_stek(&vrh);
              exit(EXIT_FAILURE);
42          }
      }
44    /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
         u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
46       uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
         je taj uslov ispunjen, skida se sa steka, jer je upravo
48       zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
         nisu pravilno uparene. */
50    else if (tip == ZATVORENA) {
        if (vrh_steka(vrh) != NULL
52          && strcmp(vrh_steka(vrh), etiketa) == 0)
          skini_sa_steka(&vrh, NULL);
54      else {
          printf("Etikete nisu pravilno uparene\n");
56        printf("(nadjena je etiketa </%s>", etiketa);
          if (vrh_steka(vrh) != NULL)
58          printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh))
      ;
            else
60          printf(" koja nije otvorena)\n");
          uparene = 0;
62        break;
        }
64    }
  }
66 /* Zavrseno je citanje i datoteka se zatvara */
  fclose(f);
68
  /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
70    da bude prazan. Ukoliko nije, tada postoje etikete koje su
      ostale otvorene */
72 if (uparene) {
    if (vrh_steka(vrh) == NULL)
74    printf("Etikete su pravilno uparene!\n");
    else {
76    printf("Etikete nisu pravilno uparene\n");
      printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
78    /* Oslobadja se memorija zauzeta stekom */
      oslobodi_stek(&vrh);
80  }
  }
82
  exit(EXIT_SUCCESS);
84}
```

Rešenje 4.10

red.h

```
#ifndef _RED_H_
#define _RED_H_

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

#define MAX 1000
#define JMBG_DUZINA 14

/* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
   opis njegovog zahteva. */
typedef struct {
  char jmbg[JMBG_DUZINA];
  char opis[MAX];
} Zahtev;

/* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
   korisnika i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  Zahtev nalog;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
   poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
   greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
   treba smestiti u novi cvor zbog smestanja manjeg podatka na
   sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
   bajtovima(B) u odnosu na strukturu Zahtev. */
Cvor *napravi_cvor(Zahtev * zahtev);

/* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

/* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
   greske pri alokaciji memorije za novi cvor, inace vraca 0. */
int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                Zahtev * zahtev);

/* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
   pokazivac razlicit od NULL, tada se u strukturu na koju on
   pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
   suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
   suprotnom. */
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

```

```
/* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
   na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
Zahtev *pocetak_reda(Cvor * pocetak);

/* Funkcija prikazuje sadrzaj reda. */
void prikazi_red(Cvor * pocetak);

#endif
```

red.c

```
#include "red.h"

Cvor *napravi_cvor(Zahtev * zahtev)
{
  /* Alocira se memorija za novi cvor liste i proverava uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->nalog = *zahtev;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
  Cvor *pomocni = NULL;

  /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
  while (*pocetak != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor sa pocetka reda */
    pomocni = *pocetak;
    *pocetak = (*pocetak)->sledeci;
    free(pomocni);
  }
  /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
     treba postaviti na NULL. */
  *kraj = NULL;
}

int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                Zahtev * zahtev)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(zahtev);
```

```
41    if (novi == NULL)
        return 1;
43
      /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
45       kraj, to je podjednako efikasno kao dodavanje na pocetak liste
        */
47    if (*adresa_kraja != NULL) {
        (*adresa_kraja)->sledeci = novi;
        *adresa_kraja = novi;
49    } else {
        /* Ako je red bio ranije prazan */
51      *adresa_pocetka = novi;
        *adresa_kraja = novi;
53    }

55    /* Vraca se indikator uspesnog dodavanja */
      return 0;
57  }

59  int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                      Zahtev * zahtev)
61  {
      Cvor *pomocni = NULL;
63
      /* Ako je red prazan */
65    if (*adresa_pocetka == NULL)
        return 0;
67
      /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
69       zahtev koji je na pocetku reda.  */
      if (zahtev != NULL)
71      *zahtev = (*adresa_pocetka)->nalog;

73    /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
         na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
75       u redu. */
      pomocni = *adresa_pocetka;
77    *adresa_pocetka = (*adresa_pocetka)->sledeci;
      free(pomocni);
79
      /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
81       potrebno je azurirati i vrednost pokazivaca na adresi
         adresa_kraja na NULL */
83    if (*adresa_pocetka == NULL)
        *adresa_kraja = NULL;
85
      return 1;
87  }

89  Zahtev *pocetak_reda(Cvor * pocetak)
    {
91    /* U praznom redu nema zahteva */
```

```
  if (pocetak == NULL)
93  return NULL;

95  /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
  return &(pocetak->nalog);
97 }

99 void prikazi_red(Cvor * pocetak)
{
101  /* Prikazuje se sadrzaj reda od pocetka prema kraju */
  for (; pocetak != NULL; pocetak = pocetak->sledeci)
103  printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

105  printf("\n");
}
```

main.c

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

6 #define VREME_ZA_PAUZU 5

8 /* Glavni program */
int main(int argc, char **argv)
10 {
  /* Red je prazan. */
12  Cvor *pocetak = NULL, *kraj = NULL;
  Zahtev nov_zahtev;
14  Zahtev *sledeci = NULL;
  char odgovor[3];
16  int broj_usluzenih = 0;

18  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
     broja i opisa potrebne usluge. */
20  printf("Sluzbenik evidentira korisnicke zahteve:\n");
  while (1) {
22
    /* Ucitava se JMBG */
24    printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
    if (scanf("%s", nov_zahtev.jmbg) == EOF)
26      break;

28    /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
       JMBG broja procitao i da bi fgets nakon toga procitala
30      ispravan red sa opisom zahteva */
    getchar();
32
    /* Ucitava se opis problema */
```

```
34      printf("\tOpis problema: ");
        fgets(nov_zahtev.opis, MAX - 1, stdin);
36      /* Ako je poslednji karakter nov red, eliminise se */
        if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
38        nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

40      /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
        if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
42        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
          oslobodi_red(&pocetak, &kraj);
44        exit(EXIT_FAILURE);
        }
46    }

48    /* Otvaranje datoteke za dopisivanje izvestaja */
      FILE *izlaz = fopen("izvestaj.txt", "a");
50    if (izlaz == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
52      exit(EXIT_FAILURE);
      }
54
      /* Dokle god ima korisnika u redu, treba ih usluziti */
56    while (1) {
        sledeci = pocetak_reda(pocetak);
58      /* Ako nema nikog vise u redu, prekida se petlja */
        if (sledeci == NULL)
60        break;

62      printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
        printf("i zahtevom: %s\n", sledeci->opis);
64
        skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
66
        broj_usluzenih++;
68
        printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
70      scanf("%s", odgovor);

72      if (strcmp(odgovor, "Da") == 0)
          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
74      else
          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
76                nov_zahtev.opis);

78      if (broj_usluzenih == VREME_ZA_PAUZU) {
          printf("\nDa li je kraj smene? [Da/Ne] ");
80        scanf("%s", odgovor);

82        if (strcmp(odgovor, "Da") == 0)
            break;
84        else
            broj_usluzenih = 0;
```

```
86      }
    }
88
    /*****************************************************************
90    Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:

92    while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
        printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
94            nov_zahtev.jmbg);
        printf("sa zahtevom: %s\n", nov_zahtev.opis);
96      broj_usluzenih++;

98      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
        scanf("%s", odgovor);
100     if (strcmp(odgovor, "Da") == 0)
          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
102     else
          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
104               nov_zahtev.jmbg, nov_zahtev.opis);

106     if (broj_usluzenih == VREME_ZA_PAUZU) {
          printf("\nDa li je kraj smene? [Da/Ne] ");
108       scanf("%s", odgovor);
          if (strcmp(odgovor, "Da") == 0)
110         break;
          else
112         broj_usluzenih = 0;
        }
114   }
    *****************************************************************/
116 /* Datoteka vise nije potrebna i treba je zatvoriti. */
    fclose(izlaz);
118
    /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
120    neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
       koju zauzima red sa neobradjenim zahtevima korisnika. */
122 oslobodi_red(&pocetak, &kraj);

124 exit(EXIT_SUCCESS);
}
```

Rešenje 4.11

dvostruko_povezana_lista.h

```
#ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
#define _DVOSTRUKO_POVEZANA_LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   vrednost i pokazivace na sledeci i prethodni cvor liste. */
```

```
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
  struct cvor *prethodni;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
   adresi adresa_kraja. */
void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
                           adresa_kraja, int broj);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                        int broj);

/* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   novi cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

/* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
int dodaj_iza(Cvor * tekuci, int broj);

/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                    broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
   neopadajuce sortirana. Vraca pokazivac na cvor liste koji sadrzi
   trazeni broj ili NULL u slucaju da takav cvor ne postoji. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

/* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
```

```
   pokazivac glava se nalazi na adresi adresa_glave. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
                   tekuci);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                 broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
   sortirana. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                 adresa_kraja, int broj);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

/* Funkcija prikazuje vrednosti cvorova liste pocevsi od kraja ka
   glavi liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu_unazad(Cvor * kraj);

#endif
```

dvostruko_povezana_lista.c

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

Cvor *napravi_cvor(int broj)
{
  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
{
```

```
23    Cvor *pomocni = NULL;

25    /* Ako lista nije prazna, onda treba osloboditi memoriju */
      while (*adresa_glave != NULL) {
27      /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
           osloboditi memoriju cvora koji predstavlja glavu liste */
29      pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
31      /* Sledeci cvor je nova glava liste */
        *adresa_glave = pomocni;
33    }
      /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
35       treba postaviti na NULL */
      *adresa_kraja = NULL;
37  }

39  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
                               adresa_kraja, int broj)
41  {
      /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43    Cvor *novi = napravi_cvor(broj);
      if (novi == NULL)
45      return 1;

47    /* Sledbenik novog cvora je glava stare liste */
      novi->sledeci = *adresa_glave;
49
      /* Ako stara lista nije bila prazna, onda prethodni cvor glave
51       treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
         krajnji */
53    if (*adresa_glave != NULL)
        (*adresa_glave)->prethodni = novi;
55    else
        *adresa_kraja = novi;
57
      /* Novi cvor je nova glava liste */
59    *adresa_glave = novi;

61    /* Vraca se indikator uspesnog dodavanja */
      return 0;
63  }

65  int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                            int broj)
67  {
      /* Kreira se novi cvor i proverava se uspesnost kreiranja */
69    Cvor *novi = napravi_cvor(broj);
      if (novi == NULL)
71      return 1;

73    /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
         ujedno i cela lista. Azurira se vrednost na koju pokazuju
```

```
75      adresa_glave i adresa_kraja */
    if (*adresa_glave == NULL) {
77    *adresa_glave = novi;
      *adresa_kraja = novi;
79  } else {
      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
81       sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
         liste */
83    (*adresa_kraja)->sledeci = novi;
      novi->prethodni = (*adresa_kraja);
85    *adresa_kraja = novi;
    }
87
    /* Vraca se indikator uspesnog dodavanja */
89  return 0;
}
91
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
93{
    /* U praznoj listi nema takvog mesta i vraca se NULL */
95  if (glava == NULL)
      return NULL;
97
    /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
99     pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
       vrednost vecu ili jednaku od vrednosti novog cvora.
101
       Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
103    provera da li se doslo do poslednjeg cvora liste pre nego sto se
       proveri vrednost u sledecem cvoru jer u slucaju poslednjeg,
105    sledeci ne postoji pa ni njegova vrednost. */
    while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
107   glava = glava->sledeci;

109 /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
       poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
111    vrednost vecu od broj */
    return glava;
113}

115int dodaj_iza(Cvor * tekuci, int broj)
{
117 /* Kreira se novi cvor i provera se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
119 if (novi == NULL)
      return 1;
121
    novi->sledeci = tekuci->sledeci;
123 novi->prethodni = tekuci;

125 /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
       a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
```

```
127       na ispravne adrese */
    if (tekuci->sledeci != NULL)
129       tekuci->sledeci->prethodni = novi;
    tekuci->sledeci = novi;
131
    /* Vraca se indikator uspesnog dodavanja */
133   return 0;
}
135
int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
137                     broj)
{
139   /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste
        */
    if (*adresa_glave == NULL) {
141     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
      Cvor *novi = napravi_cvor(broj);
143     if (novi == NULL)
        return 1;
145
      /* Azuriraju se vrednosti pocetka i kraja liste */
147     *adresa_glave = novi;
      *adresa_kraja = novi;
149
      /* Vraca se indikator uspesnog dodavanja */
151     return 0;
    }
153
    /* Ukoliko je vrednost glave liste veca ili jednaka od nove
155       vrednosti onda novi cvor treba staviti na pocetak liste */
    if ((*adresa_glave)->vrednost >= broj) {
157     return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
    }
159
    /* Pronazi se cvor iza koga treba uvezati novi cvor */
161   Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
    /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
163   if (dodaj_iza(pomocni, broj) == 1)
      return 1;
165   /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
        novi cvor poslednji u listi. */
167   if (pomocni == *adresa_kraja)
      *adresa_kraja = pomocni->sledeci;
169
    return 0;
171 }

173 Cvor *pretrazi_listu(Cvor * glava, int broj)
{
175   /* Obilaze se cvorovi liste */
    for (; glava != NULL; glava = glava->sledeci)
177     /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
```

```
        se obustavlja */
179     if (glava->vrednost == broj)
          return glava;
181
    /* Nema trazenog broja u listi i vraca se NULL */
183   return NULL;
}
185
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
187 {
    /* Obilaze se cvorovi liste */
189   /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
    for (; glava != NULL && glava->vrednost <= broj;
191         glava = glava->sledeci)
      /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
193       se obustavlja */
      if (glava->vrednost == broj)
195       return glava;

197   /* Nema trazenog broja u listi i bice vraceno NULL */
    return NULL;
199 }

201 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
     lako realizovati jer on sadrzi pokazivace na svog sledbenika i
203    prethodnika u listi. U funkciji se bise cvor zadat argumentom
     tekuci */
205 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
                     tekuci)
207 {
    /* Ako je tekuci NULL pokazivac, nema sta da se brise */
209   if (tekuci == NULL)
      return;
211
    /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
213       njegov sledbenik bude sledbenik tekuceg cvora */
    if (tekuci->prethodni != NULL)
215     tekuci->prethodni->sledeci = tekuci->sledeci;

217   /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
       treba da bude prethodnik tekuceg cvora */
219   if (tekuci->sledeci != NULL)
      tekuci->sledeci->prethodni = tekuci->prethodni;
221
    /* Ako je glava cvor koji se brise, nova glava liste ce biti
223       sledbenik stare glave */
    if (tekuci == *adresa_glave)
225     *adresa_glave = tekuci->sledeci;

227   /* Ako je cvor koji se brise poslednji u listi, azurira se i
       pokazivac na kraj liste */
229   if (tekuci == *adresa_kraja)
```

```
    *adresa_kraja = tekuci->prethodni;

  /* Oslobadja se dinamicki alociran prostor za cvor tekuci */
  free(tekuci);
}

void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
    )
{
  Cvor *tekuci = *adresa_glave;

  /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
     cvorovi se brisu iz liste. */
  while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
    obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}

void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                 adresa_kraja, int broj)
{
  Cvor *tekuci = *adresa_glave;

  /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
     takvi cvorovi se brisu iz liste. */
  while ((tekuci =
          pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
    obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}

void ispisi_listu(Cvor * glava)
{
  putchar('[');
  /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
     pocetka prema kraju liste */
  for (; glava != NULL; glava = glava->sledeci) {
    printf("%d", glava->vrednost);
    if (glava->sledeci != NULL)
      printf(", ");
  }

  printf("]\n");
}

void ispisi_listu_unazad(Cvor * kraj)
{
  putchar('[');
  /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
     prema pocetku liste */
  for (; kraj != NULL; kraj = kraj->prethodni) {
    printf("%d", kraj->vrednost);
    if (kraj->prethodni != NULL)
      printf(", ");
```

```
281   }
      printf("]\n");
283 }
```

main_a.c

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"

5  /* 1) Glavni program */
   int main()
7  {
     /* Lista je prazna na pocetku */
9    /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
        da bi operacije poput dodavanja na kraj liste i ispisivanja
11      liste unazad bile efikasne poput dodavanja na pocetak liste i
        ispisivanja liste od pocetnog do poslednjeg cvora. */
13   Cvor *glava = NULL;
     Cvor *kraj = NULL;
15   Cvor *trazeni = NULL;
     int broj;
17
     /* Testira se funkcija za dodavanja novog broja na pocetak liste */
19   printf("Unesite brojeve: (za kraj CTRL+D)\n");
     while (scanf("%d", &broj) > 0) {
21     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
          memorije za novi cvor. Memoriju alociranu za cvorove liste
23        treba osloboditi pre napustanja programa */
       if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
25       fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
         oslobodi_listu(&glava, &kraj);
27       exit(EXIT_FAILURE);
       }
29     printf("\tLista: ");
       ispisi_listu(glava);
31   }

33   /* Testira se funkcija za pretragu liste */
     printf("\nUnesite broj koji se trazi u listi: ");
35   scanf("%d", &broj);

37   /* Pokazivac trazeni dobija vrednost rezultata pretrage */
     trazeni = pretrazi_listu(glava, broj);
39   if (trazeni == NULL)
       printf("Broj %d se ne nalazi u listi!\n", broj);
41   else
       printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
43
     /* Ispisuje se lista unazad */
45   printf("\nLista ispisana u nazad: ");
```

```
  ispisi_listu_unazad(kraj);

  /* Oslobadja se memorija zauzeta za cvorove liste */
  oslobodi_listu(&glava, &kraj);

  exit(EXIT_SUCCESS);
}
```

main_b.c

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

/* 2) Glavni program */
int main()
{
  /* Lista je prazna na pocetku. */
  Cvor *glava = NULL;
  Cvor *kraj = NULL;
  int broj;

  /* Testira se funkcija za dodavanja novog broja na kraj liste */
  printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* 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 */
    if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava, &kraj);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcija za brisanje elemenata iz liste */
  printf("\nUnesite broj koji se brise iz liste: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor(&glava, &kraj, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Ispisuje se lista unazad */
  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(kraj);
```

```
42
   /* Oslobadja se memorija zauzeta za cvorove liste */
44 oslobodi_listu(&glava, &kraj);

46 exit(EXIT_SUCCESS);
}
```

main_c.c

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"

5  /* 3) Glavni program */
   int main()
7  {
     /* Lista je prazna na pocetku */
9    Cvor *glava = NULL;
     Cvor *kraj = NULL;
11   Cvor *trazeni = NULL;
     int broj;
13
     /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
15      bude uredjena neopadajuce */
     printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
17   while (scanf("%d", &broj) > 0) {
       /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
19        memorije za novi cvor. Memoriju alociranu za cvorove liste
          treba osloboditi pre napustanja programa */
21     if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23       oslobodi_listu(&glava, &kraj);
         exit(EXIT_FAILURE);
25     }
       printf("\tLista: ");
27     ispisi_listu(glava);
     }
29
     /* Testira se funkcija za pretragu liste */
31   printf("\nUnesite broj koji se trazi u listi: ");
     scanf("%d", &broj);
33
     /* Pokazivac trazeni dobija vrednost rezultata pretrage */
35   trazeni = pretrazi_listu(glava, broj);
     if (trazeni == NULL)
37     printf("Broj %d se ne nalazi u listi!\n", broj);
     else
39     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

41   /* Testira se funkcija za brisanje elemenata iz liste */
     printf("\nUnesite broj koji se brise iz liste: ");
```

```
43    scanf("%d", &broj);

45    /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
         procitanom sa ulaza */
47    obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

49    printf("Lista nakon brisanja:  ");
      ispisi_listu(glava);
51
      /* Ispisuje se lista unazad */
53    printf("\nLista ispisana u nazad: ");
      ispisi_listu_unazad(kraj);
55
      /* Oslobadja se memorija zauzeta za cvorove liste */
57    oslobodi_listu(&glava, &kraj);

59    exit(EXIT_SUCCESS);
  }
```

Rešenje 4.14

stabla.h

```
  #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 alocira memoriju za novi cvor stabla,
     inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
     funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
18   do greske */
  int dodaj_u_stablo(Cvor ** adresa_korena, int broj);
20
  /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
22 Cvor *pretrazi_stablo(Cvor * koren, int broj);

24 /* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
     stablu */
26 Cvor *pronadji_najmanji(Cvor * koren);
```

```
28 /* f) Funkcija koja pronalazi cvor koji sadrzi najvecu vrednost u
      stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
   void obrisi_element(Cvor ** adresa_korena, int broj);
34
   /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
36    postablo - Koren - Desno podstablo ) */
   void ispisi_stablo_infiksno(Cvor * koren);
38
   /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
40    Levo podstablo - Desno podstablo ) */
   void ispisi_stablo_prefiksno(Cvor * koren);
42
   /* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
44    podstablo - Desno postablo - Koren) */
   void ispisi_stablo_postfiksno(Cvor * koren);
46
   /* k) Funkcija koja oslobadja memoriju zauzetu stablom.  */
48 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
8       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)
   {
24   /* Ako je stablo prazno */
     if (*adresa_korena == NULL) {
```

```
26
    /* Kreira se novi cvor */
28  Cvor *novi_cvor = napravi_cvor(broj);

30  /* Proverava se uspesnost kreiranja */
    if (novi_cvor == NULL) {
32
      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
34     */
      return 1;
36  }
    /* Inace ... */
38
    /* Novi cvor se proglasava korenom stabla */
40  *adresa_korena = novi_cvor;

42  /* I vraca se indikator uspesnosti kreiranja */
    return 0;
44 }

46 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati
     broj */
48
 /* Ako je zadata vrednost manja od vrednosti korena */
50 if (broj < (*adresa_korena)->broj)

52   /* Broj se dodaje u levo podstablo */
     return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
54
 else
56   /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
        dodaje u desno podstablo */
58   return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
}
60
Cvor *pretrazi_stablo(Cvor * koren, int broj)
62 {

64 /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
 if (koren == NULL)
66   return NULL;

68 /* Ako je trazena vrednost sadrzana u korenu */
 if (koren->broj == broj) {
70
   /* Prekidamo pretragu */
72 return koren;
 }
74
 /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
76 if (broj < koren->broj)
```

```
78      /* Pretraga se nastavlja u levom podstablu */
        return pretrazi_stablo(koren->levo, broj);
80
    else
82      /* U suprotnom, pretraga se nastavlja u desnom podstablu */
        return pretrazi_stablo(koren->desno, broj);
84  }

86  Cvor *pronadji_najmanji(Cvor * koren)
    {
88
    /* Ako je stablo prazno, prekida se pretraga */
90  if (koren == NULL)
      return NULL;
92
    /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
94     levo od njega */

96  /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
       najmanju vrednost */
98  if (koren->levo == NULL)
      return koren;
100
    /* Inace, pretragu treba nastaviti u levom podstablu */
102 return pronadji_najmanji(koren->levo);
    }
104
    Cvor *pronadji_najveci(Cvor * koren)
106 {
    /* Ako je stablo prazno, prekida se pretraga */
108 if (koren == NULL)
      return NULL;
110
    /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
112    desno od njega */

114 /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
       najvecu vrednost */
116 if (koren->desno == NULL)
      return koren;
118
    /* Inace, pretragu treba nastaviti u desnom podstablu */
120 return pronadji_najveci(koren->desno);
    }
122
    void obrisi_element(Cvor ** adresa_korena, int broj)
124 {
    Cvor *pomocni_cvor = NULL;
126
    /* Ako je stablo prazno, brisanje nije primenljivo */
128 if (*adresa_korena == NULL)
      return;
```

```
130
    /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132    stabla, ona se eventualno nalazi u levom podstablu, pa treba
       rekurzivno primeniti postupak na levo podstablo. Koren ovako
134    modifikovanog stabla je nepromenjen. */
    if (broj < (*adresa_korena)->broj) {
136   obrisi_element(&(*adresa_korena)->levo, broj);
      return;
138 }

140 /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
       stabla, ona se eventualno nalazi u desnom podstablu pa treba
142    rekurzivno primeniti postupak na desno podstablo. Koren ovako
       modifikovanog stabla je nepromenjen. */
144 if ((*adresa_korena)->broj < broj) {
      obrisi_element(&(*adresa_korena)->desno, broj);
146   return;
    }
148
    /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150    jednaka broju koji se brise (tj. slucaj kada treba obrisati
       koren) */
152
    /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154    prazno stablo (vraca se NULL) */
    if ((*adresa_korena)->levo == NULL
156     && (*adresa_korena)->desno == NULL) {
      free(*adresa_korena);
158   *adresa_korena = NULL;
      return;
160 }

162 /* Ako koren ima samo levog sina, tada se brisanje vrsi tako sto se
       brise koren, a novi koren postaje levi sin */
164 if ((*adresa_korena)->levo != NULL
       && (*adresa_korena)->desno == NULL) {
166   pomocni_cvor = (*adresa_korena)->levo;
      free(*adresa_korena);
168   *adresa_korena = pomocni_cvor;
      return;
170 }

172 /* Ako koren ima samo desnog sina, tada se brisanje vrsi tako sto
       se brise koren, a novi koren postaje desni sin */
174 if ((*adresa_korena)->desno != NULL
       && (*adresa_korena)->levo == NULL) {
176   pomocni_cvor = (*adresa_korena)->desno;
      free(*adresa_korena);
178   *adresa_korena = pomocni_cvor;
      return;
180 }
```

```
182   /* Slucaj kada koren ima oba sina - najpre se potrazi sledbenik
         korena (u smislu poretka) u stablu. To je upravo po vrednosti
184      najmanji cvor u desnom podstablu. On se moze pronaci npr.
         funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
186      vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
         broj koji se brise).  Zatim se prosto rekurzivno pozove funkcija
188      za brisanje na desno podstablo. S obzirom da u njemu treba
         obrisati najmanji element, a on zasigurno ima najvise jednog
190      potomka, jasno je da ce njegovo brisanje biti obavljeno na jedan
         od jednostavnijih nacina koji su gore opisani. */
192   pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
   (*adresa_korena)->broj = pomocni_cvor->broj;
194   pomocni_cvor->broj = broj;
   obrisi_element(&(*adresa_korena)->desno, broj);
196 }

198 void ispisi_stablo_infiksno(Cvor * koren)
{
200   /* Ako stablo nije prazno */
   if (koren != NULL) {
202
     /* Prvo se ispisuju svi cvorovi levo od korena */
204    ispisi_stablo_infiksno(koren->levo);

206    /* Zatim se ispisuje vrednost u korenu */
     printf("%d ", koren->broj);
208
     /* Na kraju se ispisuju cvorovi desno od korena */
210    ispisi_stablo_infiksno(koren->desno);
   }
212 }

214 void ispisi_stablo_prefiksno(Cvor * koren)
{
216   /* Ako stablo nije prazno */
   if (koren != NULL) {
218
     /* Prvo se ispisuje vrednost u korenu */
220    printf("%d ", koren->broj);

222    /* Zatim se ispisuju svi cvorovi levo od korena */
     ispisi_stablo_prefiksno(koren->levo);
224
     /* Na kraju se ispisuju svi cvorovi desno od korena */
226    ispisi_stablo_prefiksno(koren->desno);
   }
228 }

230 void ispisi_stablo_postfiksno(Cvor * koren)
{
232
   /* Ako stablo nije prazno */
```

```
234   if (koren != NULL) {

236     /* Prvo se ispisuju svi cvorovi levo od korena */
      ispisi_stablo_postfiksno(koren->levo);
238
      /* Zatim se ispisuju svi cvorovi desno od korena */
240   ispisi_stablo_postfiksno(koren->desno);

242     /* Na kraju se ispisuje vrednost u korenu */
      printf("%d ", koren->broj);
244   }
}
246
void oslobodi_stablo(Cvor ** adresa_korena)
248 {
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
250 if (*adresa_korena == NULL)
    return;
252
  /* Inace ... */
254 /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);
256
  /* Oslobadja se memorija zauzetu desnim podstablom */
258 oslobodi_stablo(&(*adresa_korena)->desno);

260 /* Oslobadja se memorija zauzeta korenom */
  free(*adresa_korena);
262
  /* Proglasava se stablo praznim */
264 *adresa_korena = NULL;
}
```

main.c

```
1 #include <stdio.h>
  #include <stdlib.h>
3 #include "stabla.h"

5 int main()
  {
7   Cvor *koren;
    int n;
9   Cvor *trazeni_cvor;

11  /* Proglasava se stablo praznim */
    koren = NULL;
13
    /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
15     dodavanja */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
```

```
  while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&koren, n) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Generisu se trazeni ispisi: */
  printf("\nInfiksni ispis: ");
  ispisi_stablo_infiksno(koren);
  printf("\nPrefiksni ispis: ");
  ispisi_stablo_prefiksno(koren);
  printf("\nPostfiksni ispis: ");
  ispisi_stablo_postfiksno(koren);

  /* Demonstrira se rad funkcije za pretragu */
  printf("\nTrazi se broj: ");
  scanf("%d", &n);
  trazeni_cvor = pretrazi_stablo(koren, n);
  if (trazeni_cvor == NULL)
    printf("Broj se ne nalazi u stablu!\n");

  else
    printf("Broj se nalazi u stablu!\n");

  /* Demonstrira se rad funkcije za brisanje */
  printf("Brise se broj: ");
  scanf("%d", &n);
  obrisi_element(&koren, n);
  printf("Rezultujuce stablo: ");
  ispisi_stablo_infiksno(koren);
  printf("\n");

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.15

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

#define MAX 50

/* 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));
30   if (novi_cvor->rec == NULL) {
       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 proglasava korenom stabla */
```

```
62      *adresa_korena = novi_cvor;

64      /* I vraca se indikator uspesnog dodavanja */
        return 0;
66    }

68    /* Ako stablo nije prazno, trazi odgovarajuca 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)
        return dodaj_u_stablo(&(*adresa_korena)->levo, rec);
74
      else
76      /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
           desno podstablo */
78    if (strcmp(rec, (*adresa_korena)->rec) > 0)
        return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
80
      else{
82      /* Ako je rec jednaka reci u korenu, uvecava se njen broj
           pojavljivanja */
84      (*adresa_korena)->brojac++;

86      /* I vraca se indikator uspesnog dodavanja */
        return 0;
88    }
  }
90
  /* Funkcija koja oslobadja memoriju zauzetu stablom */
92 void oslobodi_stablo(Cvor ** adresa_korena)
  {
94    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
      if (*adresa_korena == NULL)
96      return;

98    /* Inace ... */
      /* Oslobadja se memorija zauzeta levim podstablom */
100   oslobodi_stablo(&(*adresa_korena)->levo);

102   /* Oslobadja se memorija zauzeta desnim podstablom */
      oslobodi_stablo(&(*adresa_korena)->desno);
104
      /* Oslobadja se memorija zauzeta korenom */
106   free((*adresa_korena)->rec);
      free(*adresa_korena);
108
      /* Stablo se proglasava praznim */
110   *adresa_korena = NULL;
  }
112
```

```
/* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
   sa najvecim brojem pojavljivanja) */
Cvor *nadji_najfrekventniju_rec(Cvor * koren)
{
  Cvor *max, *max_levo, *max_desno;

  /* Ako je stablo prazno, prekida se sa pretragom */
  if (koren == NULL)
    return NULL;

  /* Pronalazi se najfrekventnija rec u levom podstablu */
  max_levo = nadji_najfrekventniju_rec(koren->levo);

  /* Pronalazi se najfrekventnija rec u desnom podstablu */
  max_desno = nadji_najfrekventniju_rec(koren->desno);

  /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
     podstabla, korena i desnog podstabla */
  max = koren;
  if (max_levo != NULL && max_levo->brojac > max->brojac)
    max = max_levo;
  if (max_desno != NULL && max_desno->brojac > max->brojac)
    max = max_desno;

  /* Vraca se adresa cvora sa najvecim brojem pojavljivanja */
  return max;
}

/* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
   pracene brojem pojavljivanja */
void prikazi_stablo(Cvor * koren)
{
  /* Ako je stablo prazno, zavrsava se sa ispisom */
  if (koren == NULL)
    return;

  /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
     podstabla */
  prikazi_stablo(koren->levo);

  /* Zatim rec iz korena */
  printf("%s: %d\n", koren->rec, koren->brojac);

  /* I nastavlja se sa ispisom reci iz desnog podstabla */
  prikazi_stablo(koren->desno);
}

/* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
   niz rec. Maksimalna duzina reci je odredjena argumentom max.
   Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
   suprotnom. Rec je niz malih ili velikih slova. */
int procitaj_rec(FILE * f, char rec[], int max)
```

```
{
166   /* Karakter koji se cita */
      int c;
168
      /* Indeks pozicije na koju se smesta procitani karakter */
170   int i = 0;

172   /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
         nije stiglo do kraja datoteke... */
174   while (i < max - 1 && (c = fgetc(f)) != EOF) {
        /* Proverava se da li je procitani karakter slovo */
176     if (isalpha(c))
          /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
178          mala slova jer program treba da bude neosetljiv na razliku
             izmedju malih i velikih slova */
180       rec[i++] = tolower(c);

182     else
          /* Ako nije, proverava se da li je procitano barem jedno slovo
184          nove reci */
          /* Ako jeste, prekida se sa citanjem */
186     if (i > 0)
          break;
188
        /* U suprotnom se ide na sledecu iteraciju */
190   }

192   /* Dodaje se na rec terminirajuca nula */
      rec[i] = '\0';
194
      /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
196   return i > 0 ? 0 : EOF;
}
198
int main(int argc, char **argv)
200 {
      Cvor *koren = NULL, *max;
202   FILE *f;
      char rec[MAX];
204
      /* Provera da li je navedeno ime datoteke prilikom pokretanja
206        programa */
      if (argc < 2) {
208     fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
        exit(EXIT_FAILURE);
210   }

212   /* Priprema datoteke za citanje */
      if ((f = fopen(argv[1], "r")) == NULL) {
214     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
                argv[1]);
216     exit(EXIT_FAILURE);
```

```
  }

  /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
     pretrage uz proveru uspesnosti dodavanja */
  while (procitaj_rec(f, rec, MAX) != EOF) {
    if (dodaj_u_stablo(&koren, rec) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje reci %s.\n", rec);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Posto je citanjem reci zavrseno, zatvara se datoteka */
  fclose(f);

  /* Prikazuju se sve reci iz teksta i brojevi njihovih
     pojavljivanja. */
  prikazi_stablo(koren);

  /* Pronalazi se najfrekventnija rec */
  max = nadji_najfrekventniju_rec(koren);

  /* Ako takve reci nema... */
  if (max == NULL)

    /* Ispisuje se odgovarajuce obavestenje */
    printf("U tekstu nema reci!\n");

  else
    /* Inace, ispisuje se broj pojavljivanja reci */
    printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
           max->rec, max->brojac);

  /* Oslobadja se dinamicki alociran prostor za stablo */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.16

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

#define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
#define MAX_IME_I_PREZIME 100

/* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime, broj
```

```
   telefona i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
  char ime_i_prezime[MAX_IME_I_PREZIME];
  char telefon[MAX_CIFARA];
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije. */
  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
  if (novi_cvor == NULL)
    return NULL;

  /* Inicijalizuju se polja novog cvora */
  strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
  strcpy(novi_cvor->telefon, telefon);
  novi_cvor->levo = NULL;
  novi_cvor->desno = NULL;

  /* Vraca se adresa novog cvora */
  return novi_cvor;
}

/* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
   ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
   povratna vrednost je 1 */
int
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
               char *telefon)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
    /* Kreira se novi cvor */
    Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
       */
      return 1;
    }
    /* Inace... */
    /* Novi cvor se proglasava korenom stabla */
    *adresa_korena = novi_cvor;

    /* I vraca se indikator uspesnog dodavanja */
    return 0;
  }
```

```
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 sadrzanog u korenu, podaci se dodaju u levo podstablo
       */
69    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
          < 0)
71      return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
                              telefon);
73
      else
75      /* Ako je zadato ime i prezime leksikografski vece od imena i
           prezimena sadrzanog u korenu, podaci se dodaju u desno
77         podstablo */
      if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
79      return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
                              telefon);
81  }

83  /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena)
85  {
      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
87    if (*adresa_korena == NULL)
        return;
89
      /* Inace ... */
91    /* Oslobadja se memorija zauzeta levim podstablom */
      oslobodi_stablo(&(*adresa_korena)->levo);
93
      /* Oslobadja se memorija zauzeta desnim podstablom */
95    oslobodi_stablo(&(*adresa_korena)->desno);

97    /* Oslobadja se memorija zauzeta korenom */
      free(*adresa_korena);
99
      /* Stablo se proglasava praznim */
101   *adresa_korena = NULL;
    }
103
    /* Funkcija koja ispisuje imenik u leksikografskom poretku */
105 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
       koristiti za proveru da li je stablo lepo kreirano ili ne */
107 void prikazi_stablo(Cvor * koren)
    {
109   /* Ako je stablo prazno, zavrsava se sa ispisom */
      if (koren == NULL)
111     return;

113   /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
```

```
     podstabla */
  prikazi_stablo(koren->levo);

  /* Zatim se ispisuju podaci iz korena */
  printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

  /* I nastavlja se sa ispisom podataka iz desnog podstabla */
  prikazi_stablo(koren->desno);
}

/* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje ime
   i prezime i broj telefona u odgovarajuce nizove. Maksimalna duzina
   imena i prezimena odredjena je konstantom MAX_IME_PREZIME, a
   maksimalna duzina broja telefona konstantom MAX_CIFARA. Funkcija
   vraca EOF ako nema vise kontakata ili 0 u suprotnom. */
int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
{
  /* Karakter koji se cita */
  int c;

  /* Indeks pozicije na koju se smesta procitani karakter */
  int i = 0;

  /* Linije datoteke koje se obradjuju su formata Ime Prezime
     BrojTelefona */

  /* Preskacu se eventualne praznine sa pocetka linije datoteke */
  while ((c = fgetc(f)) != EOF && isspace(c));

  /* Prvo procitano slovo upisuje se u ime i prezime */
  if (!feof(f))
    ime_i_prezime[i++] = c;

  /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
     cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
     Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
     procitanog karaktera i da se slucajno ne dodje do kraja datoteke
   */
  while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
    if (!isdigit(c))
      ime_i_prezime[i++] = c;

    else if (i > 0)
      break;
  }

  /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
     blanko karaktera */
  ime_i_prezime[--i] = '\0';

  /* I pocinje se sa citanjem broja telefona */
  i = 0;
```

```
167   /* Upisuje se cifra koja je vec procitana */
      telefon[i++] = c;
169
      /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
171        karaktera cije prisustvo nije dozvoljeno u broju telefona */
      while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
173     if (c == '/' || c == '-' || isdigit(c))
          telefon[i++] = c;
175     else
          break;
177
      /* Upisuje se terminirajuca nula */
179   telefon[i] = '\0';

181   /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
      return !feof(f) ? 0 : EOF;
183 }

185 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
     */
187 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
    {
189   /* Ako je imenik prazan, zavrsava se sa pretragom */
      if (koren == NULL)
191     return NULL;

193   /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
          zavrsava sa pretragom */
195   if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
        return koren;
197
      /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
199        korenu pretraga se nastavlja levo */
      if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
201     return pretrazi_imenik(koren->levo, ime_i_prezime);

203   else
        /* u suprotnom, pretraga se nastavlja desno */
205     return pretrazi_imenik(koren->desno, ime_i_prezime);
    }
207
    int main(int argc, char **argv)
209 {
      char ime_datoteke[MAX_IME_DATOTEKE];
211   Cvor *koren = NULL;
      Cvor *trazeni;
213   FILE *f;
      char ime_i_prezime[MAX_IME_I_PREZIME];
215   char telefon[MAX_CIFARA];
      char c;
217   int i;
```

```
219   /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
      printf("Unesite ime datoteke: ");
221   scanf("%s", ime_datoteke);
      getchar();
223   if ((f = fopen(ime_datoteke, "r")) == NULL) {
        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
225             ime_datoteke);
        exit(EXIT_FAILURE);
227   }

229   /* Citaju se podaci iz datoteke i smestanju u binarno stablo
         pretrage uz proveru uspesnosti dodavanja */
231   while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
        if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
233       fprintf(stderr, "Greska: Neuspelo dodavanje podataka za osobu %
      s.\n",
                  ime_i_prezime);
235       oslobodi_stablo(&koren);
          exit(EXIT_FAILURE);
237     }

239   /* Zatvara se datoteka */
      fclose(f);
241
      /* Omogucava se pretraga imenika */
243   while (1) {
        /* Ucitava se ime i prezime */
245     printf("Unesite ime i prezime: ");
        i = 0;
247     while ((c = getchar()) != '\n')
          ime_i_prezime[i++] = c;
249     ime_i_prezime[i] = '\0';

251     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
           funkcionalnost */
253     if (strcmp(ime_i_prezime, "KRAJ") == 0)
          break;
255
        /* Inace se ispisuje rezultat pretrage */
257     trazeni = pretrazi_imenik(koren, ime_i_prezime);
        if (trazeni == NULL)
259       printf("Broj nije u imeniku!\n");
        else
261       printf("Broj je: %s \n", trazeni->telefon);
      }
263
      /* Oslobadja se memorija zauzeta imenikom */
265   oslobodi_stablo(&koren);

267   exit(EXIT_SUCCESS);
    }
```

Rešenje 4.17

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

#define MAX 51

/* Struktura koja definise cvorove stabla: sadrzi ime i prezime
   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
   jezika i redom pokazivace na levo i desno podstablo */
typedef struct cvor_stabla {
  char ime[MAX];
  char prezime[MAX];
  double uspeh;
  double matematika;
  double jezik;
  struct cvor_stabla *levo;
  struct cvor_stabla *desno;
} Cvor;

/* Funkcija kojom se kreira cvor stabla */
Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                   double matematika, double jezik)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije. */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizuju se polja strukture */
  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
  novi->uspeh = uspeh;
  novi->matematika = matematika;
  novi->jezik = jezik;
  novi->levo = NULL;
  novi->desno = NULL;

  /* Vraca se adresa kreiranog cvora */
  return novi;
}

/* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
   ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
   povratna vrednost je 1 */
int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                   double uspeh, double matematika, double jezik)
{
  /* Ako je stablo prazno */
  if (*koren == NULL) {
```

```
51      /* Kreira se novi cvor */
        Cvor *novi_cvor =
53          napravi_cvor(ime, prezime, uspeh, matematika, jezik);
        /* Proverava se uspesnost kreiranja novog cvora */
55      if (novi_cvor == NULL) {
          /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57         */
          return 1;
59      }
        /* Inace... */
61      /* Novi cvor se proglasava korenom stabla */
        *koren = novi_cvor;
63
        /* I vraca se indikator uspesnog dodavanja */
65      return 0;
    }
67
    /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69     sortirano po ukupnom broju poena */
    if (uspeh + matematika + jezik >
71       (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
      return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
73                            matematika, jezik);
    else
75    return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
                            matematika, jezik);
77  }

79
    /* Funkcija kojom se oslobadja memorija zauzeta stablom */
81  void oslobodi_stablo(Cvor ** koren)
    {
83    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
      if (*koren == NULL)
85      return;

87    /* Inace ... */
      /* Oslobadja se memorija zauzeta levim podstablom */
89    oslobodi_stablo(&(*koren)->levo);

91    /* Oslobadja se memorija zauzeta desnim podstablom */
      oslobodi_stablo(&(*koren)->desno);
93
      /* Oslobadja se memorija zauzeta korenom */
95    free(*koren);

97    /* Stablo se proglasava praznim */
      *koren = NULL;
99  }

101

    /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
```

```
103    polozili jednaka 0 ispisuju se informacije o ucenicima koji nisu
       polozili prijemni, a ako je vrednost argumenta razlicita od nule,
105    ispisuju se informacije o ucenicima koji su polozili prijemni */
void stampaj(Cvor * koren, int polozili)
107 {
  /* Stablo je prazno - prekida se sa ispisom */
109   if (koren == NULL)
    return;
111
  /* Stampaju se informacije iz levog podstabla */
113   stampaj(koren->levo, polozili);

115   /* Stampaju se informacije iz korenog cvora */
  if (polozili && koren->matematika + koren->jezik >= 10)
117     printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
            koren->prezime, koren->uspeh, koren->matematika,
119         koren->jezik,
            koren->uspeh + koren->matematika + koren->jezik);
121   else if (!polozili && koren->matematika + koren->jezik < 10)
    printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
123         koren->prezime, koren->uspeh, koren->matematika,
            koren->jezik,
125         koren->uspeh + koren->matematika + koren->jezik);

127   /* Stampaju se informacije iz desnog podstabla */
  stampaj(koren->desno, polozili);
129 }

131 /* Funkcija koja odredjuje koliko studenata nije polozilo prijemni
    ispit */
133 int nisu_polozili(Cvor * koren)
{
135   /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
  if (koren == NULL)
137     return 0;

139   /* Pretraga se vrsi i u levom i u desnom podstablu - ako uslov za
       polaganje nije ispunjen za koreni cvor, broj studenata se
141    uvecava za 1 */
  if (koren->matematika + koren->jezik < 10)
143     return 1 + nisu_polozili(koren->levo) +
          nisu_polozili(koren->desno);
145
  return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
147 }

149 int main(int argc, char **argv)
{
151   FILE *in;
  Cvor *koren;
153   char ime[MAX], prezime[MAX];
  double uspeh, matematika, jezik;
```

```
155
    /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
157 in = fopen("prijemni.txt", "r");
    if (in == NULL) {
159   fprintf(stderr,
              "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
161   exit(EXIT_FAILURE);
    }
163
    /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
165    dodavanja */
    koren = NULL;
167 while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
                  &matematika, &jezik) != EOF) {
169   if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik
      )
          == 1) {
171     fprintf(stderr, "Greska: Neuspelo dodavanje podataka za %s %s.\
      n", ime,
                prezime);
173     oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
175   }
    }
177
    /* Zatvaranje datoteke */
179 fclose(in);

181 /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
    stampaj(koren, 1);
183
    /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
185    prijemni */
    if (nisu_polozili(koren) != 0)
187   printf("---------------------------------------\n");

189 /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
    stampaj(koren, 0);
191
    /* Oslobadja se memorija zauzeta stablom */
193 oslobodi_stablo(&koren);

195 exit(EXIT_SUCCESS);
}
```

Rešenje 4.18

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

```
#define MAX_NISKA 51

/* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
   osobe, dan i mesec rodjenja i redom pokazivace na levo i desno
   podstablo */
typedef struct cvor_stabla {
  char ime[MAX_NISKA];
  char prezime[MAX_NISKA];
  int dan;
  int mesec;
  struct cvor_stabla *levo;
  struct cvor_stabla *desno;
} Cvor;

/* Funkcija koja kreira novi cvor */
Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
{
  /* Alocira se memorija */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizuju se polja strukture */
  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
  novi->dan = dan;
  novi->mesec = mesec;
  novi->levo = NULL;
  novi->desno = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

/* 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 */
int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                   int dan, int mesec)
{
  /* Ako je stablo prazno */
  if (*koren == NULL) {

    /* Kreira se novi cvor */
    Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
       */
      return 1;
    }
```

```
57      /* Inace... */
        /* Novi cvor se proglasava korenom stabla */
59      *koren = novi_cvor;

61      /* I vraca se indikator uspesnog dodavanja */
        return 0;
63    }

65    /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
         meseca */
67    if (mesec < (*koren)->mesec)
        return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
69    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
        return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
71    else
        return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec)
         ;
73  }

75  /* Funkcija vrsi pretragu stabla i vraca cvor sa trazenim 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 trazeni datum u korenu */
83    if (koren->dan == dan && koren->mesec == mesec)
        return koren;
85
      /* Ako je mesec trazenog datuma manji od meseca sadrzanog u korenu
87       ili ako su meseci isti ali je dan trazenog datuma manji od
         aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
89       svakako proverava da li leva grana postoji - ako ne postoji
         treba vratiti prvi sledeci, a to je bas vrednost uocenog korena
        */
91    if (mesec < koren->mesec
          || (mesec == koren->mesec && dan < koren->dan)) {
93      if (koren->levo == NULL)
          return koren;
95      else
          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
111        sadrzi najmanji datum */
      if (koren->levo == NULL)
113     return koren;
      else
115     /* Inace, trazimo manji datum u levom podstablu */
        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 */
143 void oslobodi_stablo(Cvor ** adresa_korena)
    {
145   /* Stablo je prazno */
      if (*adresa_korena == NULL)
147     return;

149   /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
      if ((*adresa_korena)->levo)
151     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 *slavljenik;
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 izvrsavanjem
       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, "Greska: Neuspelo dodavanje podataka za %s %s.\
    n", ime,
195               prezime);
      oslobodi_stablo(&koren);
197      exit(EXIT_FAILURE);
    }
199
  /* Datoteka se zatvara */
201  fclose(in);

203  /* Omogucuje se pretraga podataka */
  while (1) {
205
    /* Ucitava se novi datum */
207    printf("Unesite datum: ");
```

```
    if (scanf("%d.%d.", &dan, &mesec) == EOF)
209   break;

211 /* Pretrazuje se stablo */
    slavljenik = pretrazi(koren, dan, mesec);
213
    /* Ispisuju se pronadjeni podaci */
215
    /* Ako slavljenik nije pronadjen, to moze znaci da: */
217 /* 1. drvo je prazno */
    if (slavljenik == NULL && koren == NULL) {
219   printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
      continue;
221 }
    /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223      ovom slucaju se pretraga vrsi pocevsi od naredne godine i
         ispisuje se najmanji datum */
225 if (slavljenik == NULL) {
      slavljenik = pronadji_najmanji_datum(koren);
227   datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229           slavljenik->prezime, datum);
      continue;
231 }

233 /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
    /* 1. Pronadjeni su tacni podaci */
235 if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
      printf("Slavljenik: %s %s\n", slavljenik->ime,
237           slavljenik->prezime);
      continue;
239 }

241 /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
    datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
243 printf("Slavljenik: %s %s %s\n", slavljenik->ime,
          slavljenik->prezime, datum);
245 }

247 /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);
249
  exit(EXIT_SUCCESS);
251 }
```

Rešenje 4.19

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

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

```c
#include <stdlib.h>

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja proverava da li su dva stabla koja sadrze cele
   brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
int identitet(Cvor * koren1, Cvor * koren2)
{
  /* Ako su oba stabla prazna, jednaka su */
  if (koren1 == NULL && koren2 == NULL)
    return 1;

  /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
  if (koren1 == NULL || koren2 == NULL)
    return 0;

  /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
     vrednosti, moze se zakljuciti da se razlikuju */
  if (koren1->broj != koren2->broj)
    return 0;

  /* Inace, proverava se da li vazi i jednakost levih i desnih
     podstabala */
  return (identitet(koren1->levo, koren2->levo)
          && identitet(koren1->desno, koren2->desno));
}

int main()
{
  int broj;
  Cvor *koren1, *koren2;

  /* Ucitavaju se elementi prvog stabla */
  koren1 = NULL;
  printf("Prvo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    if (dodaj_u_stablo(&koren1, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", broj)
    ;
      oslobodi_stablo(&koren1);
      exit(EXIT_FAILURE);
    }
    scanf("%d", &broj);
  }

  /* Ucitavaju se elementi drugog stabla */
  koren2 = NULL;
  printf("Drugo stablo: ");
  scanf("%d", &broj);
```

```
  while (broj != 0) {
    if (dodaj_u_stablo(&koren2, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", broj)
    ;
      oslobodi_stablo(&koren2);
      exit(EXIT_FAILURE);
    }
    scanf("%d", &broj);
  }

  /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
     njen rezultat. */
  if (identitet(koren1, koren2))
    printf("Stabla jesu identicna.\n");
  else
    printf("Stabla nisu identicna.\n");

  /* Oslobadja se memorija zauzeta stablima */
  oslobodi_stablo(&koren1);
  oslobodi_stablo(&koren2);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.20

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* 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 */
int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
{
  /* Izlaz iz rekurzije */
  if (koren == NULL) {
    *duplikat = NULL;
    return 0;
  }

  /* Duplira se koren stabla i postavlja da bude koren novog stabla
    */
  *duplikat = napravi_cvor(koren->broj);
  if (*duplikat == NULL) {
    return 1;
```

```
22    }

24    /* Rekurzivno se duplirju levo i desno podstablo i njihove adrese
         se cuvaju redom u pokazivacima na levo i desno podstablo korena
26       duplikata */
    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 -
       rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
42     Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
       uspesno, odnosno 1 ukoliko je doslo do greske */
44  int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
    {
46    /* Ako drugo stablo nije prazno */
      if (koren2 != NULL) {
48      /* 1. Dodaje se njegov koren u prvo stablo */
        if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
50        return 1;
        }
52
        /* 2. Rekurzivno se racuna unija levog i desnog podstabla drugog
54         stabla sa prvim stablom */
        int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
56      int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);

58      /* Ako je unija podstabala uspesno kreirana */
        if (unija_levo == 0 && unija_desno == 0)
60        /* Uspesno je kreirana i unija stabala */
          return 0;
62
        /* U suprotnom se prijavljuje da je doslo do greske */
64      return 1;
      }
66
      /* Ako je drugo stablo prazno, nista se ne preduzima */
68    return 0;
    }
70
    /* Funkcija izracunava presek dva skupa predstavljana stablima -
72     rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
       Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka
```

```
74    uspesno, odnosno 1 ukoliko je doslo do greske */
  int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
76 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
78  if (*adresa_korena1 == NULL)
      return 0;
80
    /* Inace... */
82  /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.
       iz levog i desnog podstabla prvog stabla brisu se svi oni
84     elementi koji ne postoje u drugom stablu */
    int presek_levo = kreiraj_presek(&(*adresa_korena1)->levo, koren2);
86  int presek_desno =
        kreiraj_presek(&(*adresa_korena1)->desno, koren2);
88  if (presek_levo == 0 && presek_desno == 0) {
      /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on
90       uklanja iz prvog stabla */
      if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
92      obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

94    /* Presek stabala je uspesno kreiran */
      return 0;
96  }
    /* Inece, prijavljuje se da je doslo do greske */
98  return 1;
  }
100
  /* Funkcija izracunava razliku dva skupa predstavljana stablima -
102    rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
       Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike
104    uspesno, odnosno 1 ukoliko je doslo do greske */
  int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
106 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
108 if (*adresa_korena1 == NULL)
      return 0;
110
    /* Inace... */
112 /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
       tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
114    elementi koji postoje i u drugom stablu */
    int razlika_levo =
116     kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
    int razlika_desno =
118     kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
    if (razlika_levo == 0 && razlika_desno == 0) {
120   /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on
       uklanja se iz prvog stabla */
122   if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
        obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
124
      /* Razlika stabala je uspesno kreirana */
```

```
126     return 0;
    }
128
    /* Inece, prijavljuje se da je doslo do greske */
130 return 1;
}
132
int main()
134 {
    Cvor *skup1;
136 Cvor *skup2;
    Cvor *pomocni_skup = NULL;
138 int n;

140 /* Ucitavaju se elementi prvog skupa */
    skup1 = NULL;
142 printf("Prvi skup: ");
    while (scanf("%d", &n) != EOF) {
144   if (dodaj_u_stablo(&skup1, n) == 1) {
        fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
146     oslobodi_stablo(&skup1);
        exit(EXIT_FAILURE);
148   }
    }
150
    /* Ucitavaju se elementi drugog skupa */
152 skup2 = NULL;
    printf("Drugi skup: ");
154 while (scanf("%d", &n) != EOF) {
      if (dodaj_u_stablo(&skup2, n) == 1) {
156     fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
        oslobodi_stablo(&skup2);
158     exit(EXIT_FAILURE);
      }
160 }

162 /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
       bi se isti mogao iskoristiti i za preostale operacije */
164 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
      oslobodi_stablo(&skup1);
166   oslobodi_stablo(&pomocni_skup);
      exit(EXIT_FAILURE);
168 }
    if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
170   oslobodi_stablo(&pomocni_skup);
      oslobodi_stablo(&skup2);
172   exit(EXIT_FAILURE);
    }
174 printf("Unija: ");
    ispisi_stablo_infiksno(pomocni_skup);
176 putchar('\n');
```

```
178   /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
         operacije */
180   oslobodi_stablo(&pomocni_skup);

182   /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
         kako bi se isti mogao iskoristiti i za preostale operacije */
184   if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
        oslobodi_stablo(&skup1);
186     oslobodi_stablo(&pomocni_skup);
        exit(EXIT_FAILURE);
188   }
      if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190     oslobodi_stablo(&pomocni_skup);
        oslobodi_stablo(&skup2);
192     exit(EXIT_FAILURE);
      }
194   printf("Presek: ");
      ispisi_stablo_infiksno(pomocni_skup);
196   putchar('\n');

198   /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
         operacije */
200   oslobodi_stablo(&pomocni_skup);

202   /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
         kako bi se isti mogao iskoristiti i za preostale operacije */
204   if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
        oslobodi_stablo(&skup1);
206     oslobodi_stablo(&pomocni_skup);
        exit(EXIT_FAILURE);
208   }
      if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
210     oslobodi_stablo(&pomocni_skup);
        oslobodi_stablo(&skup2);
212     exit(EXIT_FAILURE);
      }
214   printf("Razlika: ");
      ispisi_stablo_infiksno(pomocni_skup);
216   putchar('\n');

218   /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
         operacije */
220   oslobodi_stablo(&pomocni_skup);

222   /* Oslobadja se memorija zauzeta polaznim skupovima */
      oslobodi_stablo(&skup1);
224   oslobodi_stablo(&skup2);

226   exit(EXIT_SUCCESS);
}
```

Rešenje 4.21

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

#define MAX 50

/* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
   cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
   su smestene u niz. */
int kreiraj_niz(Cvor * koren, int a[])
{
  int r, s;

  /* Stablo je prazno - u niz je smesteno 0 elemenata */
  if (koren == NULL)
    return 0;

  /* Dodaju se u niz elementi iz levog podstabla */
  r = kreiraj_niz(koren->levo, a);

  /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
     u niz i na osnovu nje se moze odrediti indeks novog elementa */

  /* Smesta se vrednost iz korena */
  a[r] = koren->broj;

  /* Dodaju se elementi iz desnog podstabla */
  s = kreiraj_niz(koren->desno, a + r + 1);

  /* Racuna se indeks na koji treba smestiti naredni element */
  return r + s + 1;
}

/* Funkcija sortira niz tako sto najpre elemente niza smesti u
   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
   desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
   kreiran i sortiran, a 1 ukoliko je doslo do greske.

   Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
   sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
   sortiranje vrsi u pomocnoj dinamickoj strukturi, a ne razmenom
   elemenata niza. */
int sortiraj(int a[], int n)
{
  int i;
```

```
  Cvor *koren;

  /* Kreira se stablo smestanjem elemenata iz niza u stablo */
  koren = NULL;
  for (i = 0; i < n; i++) {
    if (dodaj_u_stablo(&koren, a[i]) == 1) {
      oslobodi_stablo(&koren);
      return 1;
    }
  }
  /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
     a */
  kreiraj_niz(koren, a);

  /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
     */
  oslobodi_stablo(&koren);

  /* Vraca se indikator uspesnog sortiranja */
  return 0;
}

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

  /* Ucitavaju se dimenzija i elementi niza */
  printf("n: ");
  scanf("%d", &n);
  if (n < 0 || n > MAX) {
    printf("Greska: Pogresna dimenzija niza!\n");
    exit(EXIT_FAILURE);
  }

  printf("a: ");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Poziva se funkcija za sortiranje */
  if (sortiraj(a, n) == 0) {
    /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
    for (i = 0; i < n; i++)
      printf("%d ", a[i]);
    printf("\n");
  } else {
    /* Inace, obavestava se korisnik da je doslo do greske */
    printf("Greska: Problem prilikom sortiranja niza!\n");
  }

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.22

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

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

/* Ukljucuje 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)
{
  /* Slucaj kada je stablo prazno */
  if (koren == NULL)
    return;

  /* Ako je cvor list i sadrzi pozitivnu vrednost */
  if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
    /* Stampa se */
```

```
48      printf("%d ", koren->broj);

50    /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
      pozitivni_listovi(koren->levo);
52    pozitivni_listovi(koren->desno);
}
54
/* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
{
58    /* Ako je stablo prazno, zbir cvorova je 0 */
      if (koren == NULL)
60      return 0;

62    /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
         elemenata u podstablima */
64    return koren->broj + zbir_svih_cvorova(koren->levo) +
          zbir_svih_cvorova(koren->desno);
66 }

68 /* e) Funkcija koja izracunava najveci element stabla */
   Cvor *najveci_element(Cvor * koren)
70 {
      /* Ako je stablo prazno, obustavlja se pretraga */
72    if (koren == NULL)
        return NULL;
74
      /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena se
76       nalaze u desnom podstablu */

78    /* Ako desnog podstabla nema */
      if (koren->desno == NULL)
80      /* Najveca vrednost je koren */
        return koren;
82
      /* Inace, najveca vrednost se trazi desno */
84    return najveci_element(koren->desno);
}
86
/* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
{
90    /* Dubina praznog stabla je 0 */
      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
110     nivoa */

112   /* Ako nema vise cvorova, nema spustanja niz stablo */
      if (koren == NULL)
114     return 0;

116   /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
          rekurzivnih poziva uvecati broj cvorova za 1 */
118   if (i == 0)
        return 1;
120
      /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
        */
122   return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
          + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
124 }

126 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
    void ispis_nivo(Cvor * koren, int i)
128 {
      /* Ideja je slicna ideji iz prethodne funkcije */
130
      /* Nema vise cvorova, nema spustanja kroz stablo */
132   if (koren == NULL)
        return;
134
      /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
136   if (i == 0) {
        printf("%d ", koren->broj);
138     return;
      }
140   /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
          desnom podstablu */
142   ispis_nivo(koren->levo, i - 1);
      ispis_nivo(koren->desno, i - 1);
144 }

146 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
       stabla */
148 Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
    {
150   /* Ako je stablo prazno, obustavlja se pretraga */
```

```
  if (koren == NULL)
152   return NULL;

154 /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
  if (i == 0)
156   return koren;

158 /* Pronalazi se maksimum na i-tom nivou levog podstabla */
  Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
160
  /* Pronalazi se maksimum na i-tom nivou desnog podstabla */
162 Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);

164 /* Trazi se i vraca maksimum izracunatih vrednosti */
  if (a == NULL && b == NULL)
166   return NULL;
  if (a == NULL)
168   return b;
  if (b == NULL)
170   return a;
  /* Ako su obe vrednosti razlicite od NULL, veca od vrednosti se
     nalazi u b
172 cvoru jer je stablo pretrazivacko */
  return b;
174 }

176 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
  int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
178 {
  /* Ako je stablo prazno, zbir je nula */
180 if (koren == NULL)
    return 0;
182
  /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
184 if (i == 0)
    return koren->broj;
186
  /* Inace, spustanje se nastavlja za jedan nivo nize i traze se sume
188    iz levog i desnog podstabla */
  return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
190     + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
  }
192

194 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
     manje ili jednake od date vrednosti x */
196 int zbir_manjih_od_x(Cvor * koren, int x)
  {
198 /* Ako je stablo prazno, zbir je nula */
  if (koren == NULL)
200   return 0;
```

```
202   /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
         prirode pretrazivackog stabla treba obici i levo i desno
204      podstablo */
      if (koren->broj <= x)
206     return koren->broj + zbir_manjih_od_x(koren->levo, x) +
            zbir_manjih_od_x(koren->desno, x);
208
      /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
210      medju njima jedino moze biti onih koje zadovoljavaju uslov */
      return zbir_manjih_od_x(koren->levo, x);
212 }

214 int main(int argc, char **argv)
    {
216   /* Analiza argumenata komandne linije */
      if (argc != 3) {
218     fprintf(stderr,
                "Greska: Program se poziva sa: ./a.out nivo
        broj_za_pretragu\n");
220     exit(EXIT_FAILURE);
      }
222   int i = atoi(argv[1]);
      int x = atoi(argv[2]);
224
      /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
226    */
      Cvor *koren = NULL;
228   int broj;
      while (scanf("%d", &broj) != EOF) {
230     if (dodaj_u_stablo(&koren, broj) == 1) {
          fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", broj)
        ;
232       oslobodi_stablo(&koren);
          exit(EXIT_FAILURE);
234     }
      }
236
      /* ispisuju se rezultati rada funkcija */
238   printf("Broj cvorova: %d\n", broj_cvorova(koren));
      printf("Broj listova: %d\n", broj_listova(koren));
240   printf("Pozitivni listovi: ");
      pozitivni_listovi(koren);
242   printf("\n");
      printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
244   if (najveci_element(koren) == NULL)
        printf("Najveci element: ne postoji\n");
246   else
        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,
```

```
252           broj_cvorova_na_itom_nivou(koren, i));
  printf("Elementi na %d. nivou: ", i);
254   ispis_nivo(koren, i);
  printf("\n");
256   if (najveci_element_na_itom_nivou(koren, i) == NULL)
    printf("Nema elemenata na %d. nivou!\n", i);
258   else
    printf("Maksimalni element na %d. nivou: %d\n", i,
260           najveci_element_na_itom_nivou(koren, i)->broj);

262   printf("Zbir elemenata na %d. nivou: %d\n", i,
          zbir_cvorova_na_itom_nivou(koren, i));
264   printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
          zbir_manjih_od_x(koren, x));
266
  /* Oslobadja se memorija zauzeta stablom */
268   oslobodi_stablo(&koren);

270   exit(EXIT_SUCCESS);
}
```

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 }
```

```c
/* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
{
  /* Ideja je slicna ideji iz prethodne funkcije */
  /* Nema vise cvorova, nema spustanja niz stablo */
  if (koren == NULL)
    return;

  /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
  if (i == 0) {
    printf("%d ", koren->broj);
    return;
  }
  /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
     podstablu */
  ispisi_nivo(koren->levo, i - 1);
  ispisi_nivo(koren->desno, i - 1);
}

/* Funkcija koja ispisuje stablo po nivoima */
void ispisi_stablo_po_nivoima(Cvor * koren)
{
  int i;

  /* Prvo se izracunava dubina stabla */
  int dubina;
  dubina = dubina_stabla(koren);

  /* Ispisuje se nivo po nivo stabla */
  for (i = 0; i < dubina; i++) {
    printf("%d. nivo: ", i);
    ispisi_nivo(koren, i);
    printf("\n");
  }
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int broj;

  /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
     uspesnosti dodavanja */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    if (dodaj_u_stablo(&koren, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", broj)
    ;
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }
```

```
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) {
       return 0;
35   }
```

```
37    /* Izracunava se dubina levog podstabla korena */
      dubina_levo = dubina_stabla(koren->levo);
39
      /* Izracunava se dubina desnog podstabla korena */
41    dubina_desno = dubina_stabla(koren->desno);

43    /* Ako je uslov za AVL stablo ispunjen */
      if (abs(dubina_desno - dubina_levo) <= 1) {
45      /* Racuna se broj AVL cvorova u levom i desnom podstablu i
           uvecava za jedan iz razloga sto koren ispunjava uslov */
47      return 1 + avl(koren->levo) + avl(koren->desno);
      } else {
49      /* Inace, racuna se samo broj AVL cvorova u podstablima */
        return avl(koren->levo) + avl(koren->desno);
51    }
  }
53
  int main(int argc, char **argv)
55{
    Cvor *koren;
57  int broj;

59  /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
       uspesnosti dodavanja */
61  koren = NULL;
    while (scanf("%d", &broj) != EOF) {
63    if (dodaj_u_stablo(&koren, broj) == 1) {
        fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", broj)
      ;
65      oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
67    }
    }
69
    /* Racuna se i ispisuje broj AVL cvorova */
71  printf("%d\n", avl(koren));

73  /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);
75
    exit(EXIT_SUCCESS);
77}
```

Rešenje 4.26

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

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

```c
/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
   brojeva hip. Ideja koja ce biti implementirana u osnovi ima
   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
   desnog podstabla - ako je vrednost u korenu veca od izracunatih
   vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
   ce funkcija vracati maksimalne vrednosti iz uocenog podstabala ili
   vrednost -1 ukoliko se zakljuci da stablo nije hip. */
int hip(Cvor * koren)
{
  int max_levo, max_desno;

  /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
     pozitivan broj */
  if (koren == NULL) {
    return 0;
  }
  /* Ukoliko je stablo list... */
  if (koren->levo == NULL && koren->desno == NULL) {
    /* Vraca se njegova vrednost */
    return koren->broj;
  }
  /* Inace... */

  /* Proverava se svojstvo za levo podstablo */
  max_levo = hip(koren->levo);

  /* Proverava se svojstvo za desno podstablo */
  max_desno = hip(koren->desno);

  /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
     ni celo stablo */
  if (max_levo == -1 || max_desno == -1) {
    return -1;
  }

  /* U suprotonom proverava se da li svojstvo vazi za uoceni cvor */
  if (koren->broj > max_levo && koren->broj > max_desno) {
    /* Ako vazi, vraca se vrednost korena */
    return koren->broj;
  }

  /* U suprotnom zakljucuje se da stablo nije hip */
  return -1;
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int hip_indikator;
```

```
56
   /* Kreira se stablo prema zadatoj slici */
58   koren = NULL;
   koren = napravi_cvor(100);
60   koren->levo = napravi_cvor(19);
   koren->levo->levo = napravi_cvor(17);
62   koren->levo->levo->levo = napravi_cvor(2);
   koren->levo->levo->desno = napravi_cvor(7);
64   koren->levo->desno = napravi_cvor(3);
   koren->desno = napravi_cvor(36);
66   koren->desno->levo = napravi_cvor(25);
   koren->desno->desno = napravi_cvor(1);
68
   /* Poziva se funkcija kojom se proverava da li je stablo hip */
70   hip_indikator = hip(koren);

72   /* Ispisuje se rezultat */
   if (hip_indikator == -1) {
74     printf("Zadato stablo nije hip!\n");
   } else {
76     printf("Zadato stablo je hip!\n");
   }
78
   /* Oslobadja se memorija zauzeta stablom */
80   oslobodi_stablo(&koren);

82   return 0;
}
```

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 arugment komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.
Potpis funkcije `strstr`:
`char *strstr(const char *haystack, const char *needle);`
Funkcija traži prvo pojavljivanje podniske needle u nisci haystack, i vraća pokazivač na početak podniske, ili $NULL$ ako podniska nije pronađena.

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 1

```
TROUGLOVI.TXT
 4
  0 0 0 1.2 1 0
  0.3 0.3 0.5 0.5 0.9 1
 -2 0 0 0 0 1
 -2 0 0 0 0 1

IZLAZ:
 2 0 2 2 -1 -1
 -2 0 0 0 0 1
 0 0 0 1.2 1 0
 0.3 0.3 0.5 0.5 0.9 1
```

Test 2

```
TROUGLOVI.TXT
 3
  1.2 3.2 1.1 4.3

IZLAZ ZA GREŠKE:
 -1
```

Test 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
 -1
```

Test 4

```
TROUGLOVI.TXT
 0

IZLAZ:
```

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.*

Test 1

```
ULAZ:
  1 5 3 6 1 4 7 9
IZLAZ:
 1
```

Test 2

```
ULAZ:
 2 5 3 6 1 0 4 7 9
IZLAZ:
 2
```

Test 3

```
ULAZ:
 0 4 2 5
IZLAZ:
 0
```

Test 4

```
ULAZ:
  3
IZLAZ:
 0
```

Test 5

```
ULAZ:
 -1 4 5 1 7
IZLAZ:
 0
```

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 `00000000000000000000000000001111`, i ako je $n = 1$ tada na standardni izlaz treba ispisati neočnačen broj čiji je binarni zapis jednak `10000000000000000000000000000111`).

Test 1

```
Ulaz:
 6 1
Izlaz:
 3
```

Test 2

```
Ulaz:
 15 3
Izlaz:
 3758096385
```

Test 3

```
Ulaz:
 31 100
Izlaz:
 4026531841
```

Test 4

```
Ulaz:
 4 0
Izlaz:
 4
```

Test 5

```
Ulaz:
 0 5
Izlaz:
 0
```

Zadatak A.8 Data je biblioteka za rad sa listama. Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

Test 1

```
Ulaz:
 1 2 3 4 5
Izlaz:
 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00
```

Test 2

```
Ulaz:
 12
Izlaz:
 12.00
```

Test 3

```
Ulaz:
 prazna lista
Izlaz:
```

Test 4

```
Ulaz:
 13.3 15.8
Izlaz:
 13.30 14.55
```

Zadatak A.9 Sa standardnog ulaza se učitava dimenzija n kvadratne celobrojne matrice A ($n > 0$), a zatim i elementi matrice A. Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje su sume brojeva u svim redovima i kolonama međusobno jednake). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati -". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1 na standardni izlaz za greške. NAPOMENA: *Koristiti biblioteku za rad sa celobrojnim matricama iz zadatka 2.19.*

Test 1

```
ULAZ:
 4
 1 2 3 4
 2 1 4 3
 3 4 2 1
 4 3 1 2
IZLAZ:
 10
```

Test 2

```
ULAZ:
 3
 1 1 1
 1 1 1
 1 1 1
IZLAZ:
 3
```

Test 3

```
ULAZ:
 2
 1 1
 2 2
IZLAZ:
 -
```

Test 4

```
ULAZ:
 2
 1 2
 1 2
IZLAZ:
 -
```

Test 5

```
ULAZ:
 1
 5
IZLAZ:
 5
```

Test 6

```
ULAZ:
 0
IZLAZ ZA GREŠKE:
 -1
```

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`.

Test 1

```
ULAZ:
 285278344
IZLAZ:
 2281766929
```

Test 2

```
ULAZ:
 1024
IZLAZ:
 1024
```

Test 3

```
ULAZ:
 1
IZLAZ:
 16777216
```

Test 4

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

Test 5

```
ULAZ:
 -63
IZLAZ ZA GREŠKE:
 -1
```

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:
  matrica.txt 7

MATRICA.TXT
  3 2
  1.1 -5.31
  -3.7 35.24
  1.4 2.09

IZLAZ:
  0
```

Test 4

```
ULAZ:
  brojevi.txt 12

DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ ZA GREŠKE:
  -1
```

A.5 Rešenja

Rešenje A.1

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAKS 50

/* Funkcija vrsi dinamicku alokaciju memorije potrebne n linija tj.
   n niski od kojih nijedna nije duza od MAKS karaktera. */
char **alociranje_memorije(int n)
{
  char **linije = NULL;
  int i, j;
  /* Alocira se prostor za niz vrsti matrice */
  linije = (char **) malloc(n * sizeof(char *));
  /* U slucaju neuspesnog otvaranja ispisuje se -1 na stderr i
     program zavrsava. */
  if (linije == NULL)
    return NULL;
```

```
18    /* Alocira se prostor za svaku vrstu matrice. Niska nije duza od
         MAKS karaktera, a 1 se dodaje zbog terminirajuce nule. */
20    for (i = 0; i < n; i++) {
        linije[i] = malloc((MAKS + 1) * sizeof(char));
22      /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
           prethodno alocirani resursi, i povratna vrednost je NULL */
24      if (linije[i] == NULL) {
          for (j = 0; j < i; j++) {
26          free(linije[j]);
          }
28        free(linije);
          return NULL;
30      }
    }
32  return linije;
}
34
/* Funkcija oslobadjaja dinamicki alociranu memoriju */
36 char **oslobadjanje_memorije(char **linije, int n)
{
38    int i;
    /* Oslobadja se prostor rezervisan za svaku vrstu */
40  for (i = 0; i < n; i++) {
      free(linije[i]);
42  }
    /* Oslobadja se memorija za niz pokazivaca na vrste */
44  free(linije);

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

50 int main(int argc, char *argv[])
{
52    FILE *ulaz;
    char **linije;
54  int i, n;

56  /* Proverava argumenata komandne linije. */
    if (argc != 2) {
58    fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
60  }

62  /* Otvaranje datoteke cije ime je navedeno kao argument komandne
       linije neposredno nakon imena programa koji se poziva. U slucaju
64     neuspesnog otvaranja ispisuje se -1 na stderr i program zavrsava
       izvrsavanje. */
66  ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
68    fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
```

```
  }
  /* Ucitavanje broja linija. */
  fscanf(ulaz, "%d", &n);

  /* Alociranje memorije na osnovu ucitanog broja linija. */
  linije = alociranje_memorije(n);

  /* U slucaju neuspesne alokacije ispisuje se -1 na stderr i program
     zavrsava. */
  if (linije == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje svih n linija iz datoteke. */
  for (i = 0; i < n; i++) {
    fscanf(ulaz, "%s", linije[i]);
  }

  /* Ispisivanje u odgovarajucem poretku ucitane linije koje
     zadovoljavaju kriterijum. */
  for (i = n - 1; i >= 0; i--) {
    if (isupper(linije[i][0])) {
      printf("%s\n", linije[i]);
    }
  }
  /* Oslobadjanje memorije koja je dinamicki alocirana. */
  linije = oslobadjanje_memorije(linije, n);

  /* Zatvaranje datoteke. */
  fclose(ulaz);

  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>
#include <stdlib.h>
#include "stabla.h"

int sumiraj_n(Cvor * koren, int n)
{
  /* Ako je stablo prazno, suma je nula */
  if (koren == NULL)
```

```
    return 0;
  /* Inace ... */
  /* Ako je n jednako nula, vraca se broj iz korena */
  if (n == 0)
    return koren->broj;
  /* 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
     te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
     n-tom nivou u pocetnom stablu */
  return sumiraj_n(koren->levo, n - 1) + sumiraj_n(koren->desno, n -
    1);
}

int main()
{
  Cvor *koren = NULL;
  int n;
  int nivo;

  /* Ucitava se vrednost nivoa */
  scanf("%d", &nivo);
  while (1) {
    scanf("%d", &n);
    /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
    if (n == 0)
      break;
    /* Ako nije, dodaje se procitani broj u stablo. */
    if (dodaj_u_stablo(&koren, n) == 1) {
      fprintf(stderr, "-1\n");
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Ispisuje se rezultat rada trazene funkcije */
  printf("%d\n", sumiraj_n(koren, nivo));

  /* Oslobadja se memorija */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.3

```
#include <stdio.h>
#include <stdlib.h>
#define MAKS 50

/* Funkcija ucitava elemenate matrice sa standardnog ulaza */
void ucitaj_matricu(int m[][MAKS], int v, int k)
```

```
{
  int i, j;
  for (i = 0; i < v; i++) {
    for (j = 0; j < k; j++) {
      scanf("%d", &m[i][j]);
    }
  }
}

/* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
   koja ima v vrsta */
int broj_negativnih_u_koloni(int m[][MAKS], int v, int k)
{
  int broj_negativnih = 0;
  int i;
  for (i = 0; i < v; i++) {
    if (m[i][k] < 0)
      broj_negativnih++;
  }
  return broj_negativnih;
}

/* Funkcija vraca indeks kolone matrice m u kojoj se nalazi najvise
   negativnih elemenata */
int maks_indeks(int m[][MAKS], int v, int k)
{
  int j;
  int broj_negativnih;
  /* Inicijalizacija na nulu indeksa kolone sa maksimalnim brojem
     negativnih (maks_indeks_kolone), kao i maksimalnog broja
     negativnih elemenata u nekoj koloni (maks_broj_negativnih) */
  int maks_indeks_kolone = 0;
  int maks_broj_negativnih = 0;

  for (j = 0; j < k; j++) {
    /* Racuna se broj negativnih u j-oj koloni */
    broj_negativnih = broj_negativnih_u_koloni(m, v, j);
    /* 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 */
    if (maks_broj_negativnih < broj_negativnih) {
      maks_indeks_kolone = j;
      maks_broj_negativnih = broj_negativnih;
    }
  }
  return maks_indeks_kolone;
}

int main()
{
  int m[MAKS][MAKS];
  int v, k;
```

```
60    /* Ucitavanje broja vrsta matrice */
      scanf("%d", &v);
62
      /* Provera validnosti broja vrsta */
64    if (v < 0 || v > MAKS) {
        fprintf(stderr, "-1\n");
66      exit(EXIT_FAILURE);
      }
68
      /* Ucitavanje broja kolona matrice */
70    scanf("%d", &k);

72    /* Provera validnosti broja kolona */
      if (k < 0 || k > MAKS) {
74      fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
76    }
      /* Ucitavanje elemenata matrice */
78    ucitaj_matricu(m, v, k);

80    /* Pronalazenje kolone koja sadrzi najveci broj negativnih
         elemenata i ispisivanje trazenog rezultata */
82    printf("%d\n", maks_indeks(m, v, k));

84    exit(EXIT_SUCCESS);
  }
```

Rešenje A.4

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include <string.h>
4 #define MAKS 128

6 int main(int argc, char **argv)
  {
8   FILE *f;
    int brojac = 0;
10  char linija[MAKS], *p;

12  /* Provera da li je broj argumenata komandne linije 3 */
    if (argc != 3) {
14    fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
16  }
    /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18     linije */
    if ((f = fopen(argv[1], "r")) == NULL) {
20    fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
```

```
  }
  /* Ucitavanje iz otvorene datoteke - liniju po liniju */
  while (fgets(linija, MAKS, f) != NULL) {
    p = linija;
    while (1) {
      p = strstr(p, argv[2]);

      /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
      if (p == NULL)
        break;
      /* Inace se uvecava brojac */
      brojac++;
      /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
         linije nakon uocene podniske */
      p = p + strlen(argv[2]);
    }
  }

  /* Zatvaranje datoteke */
  fclose(f);

  /* Ispisivanje vrednosti brojaca */
  printf("%d\n", brojac);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.5

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

/* Struktura trougao */
typedef struct _trougao {
  double xa, ya, xb, yb, xc, yc;
} trougao;

/* Funkcija racuna duzinu duzi */
double duzina(double x1, double y1, double x2, double y2)
{
  return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

/* Funkcija racuna povrsinu trougla koristeci Heronov obrazac */
double povrsina(trougao t)
{
  /* Racunanje duzina stranica trougla */
  double a = duzina(t.xb, t.yb, t.xc, t.yc);
  double b = duzina(t.xa, t.ya, t.xc, t.yc);
  double c = duzina(t.xa, t.ya, t.xb, t.yb);
```

```
  /* Poluobim */
  double s = (a + b + c) / 2;
  /* Primena Heronovog obrasca */
  return sqrt(s * (s - a) * (s - b) * (s - c));
}

/* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
   prvi argument funkcije manja od povrsine trougla koji je drugi
   element funkcije funcija vraca 1, ukoliko je veca -1, a ukoliko
   su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
   napisana tako da se moze proslediti funkciji qsort da se niz
   trouglova sortira po povrsini opadajuce. */
int poredi(const void *a, const void *b)
{
  trougao x = *(trougao *) a;
  trougao y = *(trougao *) b;
  double xp = povrsina(x);
  double yp = povrsina(y);
  if (xp < yp)
    return 1;
  if (xp > yp)
    return -1;
  return 0;
}

int main()
{
  FILE *f;
  int n, i;
  trougao *niz;

  /* Otvaranje datoteke ciji je naziv trouglovi.txt */
  if ((f = fopen("trouglovi.txt", "r")) == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje podtaka o broju trouglova iz datoteke */
  if (fscanf(f, "%d", &n) != 1) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  /* Dinamicka alokacija memotije za niz trouglova duzine n */
  if ((niz = malloc(n * sizeof(trougao))) == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje podataka u niz iz otvorene datoteke */
  for (i = 0; i < n; i++) {
    if (fscanf(f, "%lf%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
```

```
                &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
    {
      fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
    }
  }

  /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
     poredi */
  qsort(niz, n, sizeof(trougao), &poredi);

  /* Ispisivanje sortiranog niza na standardni izlaz */
  for (i = 0; i < n; i++)
    printf("%g %g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
           niz[i].yb, niz[i].xc, niz[i].yc);

  /* Oslobadjanje dinamicki alocirane memorije */
  free(niz);

  /* Zatvranje datoteke */
  fclose(f);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.6

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

main.c

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

/* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
   stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa u
   stablo, a inace 0. */
int ucitaj_stablo(Cvor ** koren)
{
  *koren = NULL;
  int x;
  /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
     dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
     je i povratna vrednost iz funkcije ucitaj_stablo 1. */
  while (scanf("%d", &x) == 1)
    if (dodaj_u_stablo(koren, x) == 1)
      return 1;
  return 0;
```

```
}

/* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
int prebroj_n(Cvor * koren, int n)
{
  /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
     negativan broj, na tom nivou nema cvorova (rezultat je nula). */
  if (koren == NULL || n < 0)
    return 0;
  /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima jednog
     potomka funkcija vraca 1, inace 0 */
  if (n == 0) {
    if (koren->levo == NULL && koren->desno != NULL)
      return 1;
    if (koren->levo != NULL && koren->desno == NULL)
      return 1;
    return 0;
  }
  /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
     (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
     levog podstabla */
  return prebroj_n(koren->levo, n - 1) + prebroj_n(koren->desno, n -
    1);
}

int main()
{
  Cvor *koren;
  int n;
  scanf("%d", &n);

  /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
     oslobodja se alocirana memorija i izlazi se iz programa. */
  if (ucitaj_stablo(&koren) == 1) {
    fprintf(stderr, "-1\n");
    oslobodi_stablo(&koren);
    exit(EXIT_FAILURE);
  }

  /* Ispisivanje rezultata */
  printf("%d\n", prebroj_n(koren, n));

  /* Oslobadjanje dinamicki alociranog stabla */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.7

```c
#include <stdio.h>

/* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
   zapis argumenta x rotira za n mesta udesno */
unsigned int rotiraj(unsigned int x, unsigned int n)
{
  int i;
  unsigned int maska = 1;
  /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
     izgledala: 000...00001111 */
  /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
     nastavku je drugi nacin. */
  for (i = 1; i < n; i++)
    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. */
  return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
}

int main()
{
  unsigned int x, n;

  /* Ucitavanje brojeva sa standardnog ulaza */
  scanf("%u%u", &x, &n);

  /* Ispisivanje rezultata */
  printf("%u\n", rotiraj(x, n));
  return 0;
}
```

Rešenje A.8

liste.h

```c
#ifndef _LISTE_H_
#define _LISTE_H_ 1

/* Struktura koja predstavlja cvor liste */
typedef struct cvor {
  double vrednost;
  struct cvor *sledeci;
```

```
8
} Cvor;
10
/* Pomocna funkcija koja kreira cvor. */
12 Cvor * napravi_cvor(double broj);

14 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
      liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
16 void oslobodi_listu(Cvor ** adresa_glave);

18 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
      ili NULL kao je lista prazna */
20 Cvor * pronadji_poslednji(Cvor * glava);

22 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
      greske pri alokaciji memorije,inace vraca 0. */
24 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

26 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
      liste. */
28 void ispisi_listu(Cvor * glava);

30 #endif
```

liste.c

```
#include <stdio.h>
2 #include <stdlib.h>
#include "liste.h"
4
/* Pomocna funkcija koja kreira cvor. */
6 Cvor *napravi_cvor(double broj)
{
8   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
10    return NULL;
  /* inicijalizacija polja u novom cvoru */
12  novi->vrednost = broj;
  novi->sledeci = NULL;
14  return novi;
}
16
/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
18    ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave)
20 {
  Cvor *pomocni = NULL;
22  while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
24    free(*adresa_glave);
    *adresa_glave = pomocni;
```

```
26    }
  }
28
  /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
30     ili NULL kao je lista prazna */
  Cvor *pronadji_poslednji(Cvor * glava)
32 {
    /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
34       funkcija vraca NULL. */
    if (glava == NULL)
36      return NULL;
    while (glava->sledeci != NULL)
38      glava = glava->sledeci;
    return glava;
40
  }
42
  /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
44     greske pri alokaciji memorije, inace vraca 0. */
  int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
46 {
    Cvor *novi = napravi_cvor(broj);
48  if (novi == NULL)
      return 1;
50  if (*adresa_glave == NULL) {
      *adresa_glave = novi;
52    return 0;
    }
54  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
56  return 0;
  }
58
  /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
60     */
  void ispisi_listu(Cvor * glava)
62 {
    for (; glava != NULL; glava = glava->sledeci)
      printf("%.2lf ", glava->vrednost);
64  putchar('\n');

66 }
```

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)
```

```
{
  /* Novi cvor se dodaje iza tekuceg cvora. */
  novi->sledeci = tekuci->sledeci;
  tekuci->sledeci = novi;
}


/* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
   Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
   0. */
int dopuni_listu(Cvor ** adresa_glave)
{
  Cvor *tekuci;
  Cvor *novi;
  double aritmeticka_sredina;
  /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
  if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
    return 1;
  /* Promenljiva tekuci se inicijalizacuje da pokazuje na pocetni
     cvor */
  tekuci = *adresa_glave;
  /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
     vrednosti u susednim cvorovims i kreira cvor sa tom vrednoscu. U
     slucaju neupele alokacije novog cvora, funkcija vraca 1. Inace,
     novi cvor se umece izmedju dva cvora za koje racunata
     aritmeticka sredina */
  while (tekuci->sledeci != NULL) {
    aritmeticka_sredina =
        ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
    novi = napravi_cvor(aritmeticka_sredina);
    if (novi == NULL)
      return 1;
    /* Poziva se funkcija koja umece novi cvor iza tekuceg cvora */
    dodaj_iza(tekuci, novi);
    /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
       cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
       polazne liste */
    tekuci = tekuci->sledeci;
    tekuci = tekuci->sledeci;
  }
  return 0;
}

int main()
{
  Cvor *glava = NULL;
  double broj;

  /* Ucitavanje se vrsi do kraja ulaza. Elementi se dodaju na kraj
     liste! */
  while (scanf("%lf", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
```

```
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %lf\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
  }

  /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
     onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
     alociranu za cvorove liste treba osloboditi. */
  if (dopuni_listu(&glava) == 1) {
    fprintf(stderr, "Neuspela alokacija za cvor %lf\n", broj);
    oslobodi_listu(&glava);
    exit(EXIT_FAILURE);
  }

  /* Ispisivanje liste */
  ispisi_listu(glava);

  /* Oslobadjanje liste */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.9

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

/* Funkcija racuna zbir elemenata v-te vrste */
int zbir_vrste(int **M, int n, int v)
{
  int j, zbir = 0;
  for (j = 0; j < n; j++)
    zbir += M[v][j];
  return zbir;
}

/* Funkcija racuna zbir elemenata k-te kolone */
int zbir_kolone(int **M, int n, int k)
{
  int i, zbir = 0;
  for (i = 0; i < n; i++)
    zbir += M[i][k];
  return zbir;
}

```

```
/* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
   argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
   Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
   kolone ukoliko je kvadrat magican. */
int magicni_kvadrat(int **M, int n, int *zbir_magicnog)
{
  int i, j;
  int zbir = 0, zbir_pom;
  /* Promenljivu zbir inicijalizujemo na zbir 0-te vreste */
  zbir = zbir_vrste(M, n, 0);

  /* Racunaju se zbirovi u ostalim vrstama i ako neki razlikuje od
     vrednosti promeljive zbir funkcija vraca 1 */
  for (i = 1; i < n; i++) {
    zbir_pom = zbir_vrste(M, n, i);
    if (zbir_pom != zbir)
      return 0;
  }
  /* Racunaju se zbirovi u svim kolonama i ako neki razlikuje od
     vrednosti promeljive zbir funkcija vraca 1 */
  for (j = 0; j < n; j++) {
    zbir_pom = zbir_kolone(M, n, j);
    if (zbir_pom != zbir)
      return 0;
  }
  /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
     vresnost u zbir_magicnog i funkcija vraca 1 */
  *zbir_magicnog = zbir;
  return 1;
}

int main()
{
  int n;
  int **matrica = NULL;
  int zbir_magicnog;
  scanf("%d", &n);

  /* Provera da li je n strogo pozitivan */
  if (n <= 0) {
    printf("-1\n");
    exit(EXIT_FAILURE);
  }

  /* Dinamicka alokacija kvadratne matrice dimenzije n */
  matrica = alociraj_matricu(n, n);
  if (matrica == NULL) {
    printf("-1\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje elemenata matrice sa standardnog ulaza */
```

```
  ucitaj_matricu(matrica, n, n);

  /* Ispisivanje rezultata na osnovu fukcije magicni_kvadrat */
  if (magicni_kvadrat(matrica, n, &zbir_magicnog)) {
    printf("%d\n", zbir_magicnog);
  } else
    printf("-\n");

  /* Oslobadjanje dinamicki alocirane memorije */
  matrica = dealociraj_matricu(matrica, n);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.10

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

#define BITOVA_U_BAJTU 8

/* Funkcija u datom broju x menja mesta prvom i četvrtom bajtu */
unsigned int zamena(unsigned int x){
  /* Deklaracija promenljivih za odgovarajuce maske i pomocne
     promenljive*/
  unsigned maska_prvi_bajt, maska_cetvrti_bajt;
  unsigned maska_prvi_bajt_komplement, maska_cetvrti_bajt_komplement;
  unsigned prvi_bajt, cetvrti_bajt;
  unsigned i;

  /* Maska prvi bajt odgovara broju cija je binarna reprezentacija
     00000....0000011111111 (8 bitova najmanje tezine su jedinice,
   a ostalo su nule) */
  maska_prvi_bajt = 1;
  for(i=1;i<BITOVA_U_BAJTU; i++)
    maska_prvi_bajt = maska_prvi_bajt<<1|1;

  /* Maska cetvrti bajt odgovara broju cija je binarna reprezentacija
     1111111100000....00000 (8 bitova najvece tezine su jedinice,
   a ostalo su nule) */
  maska_cetvrti_bajt = maska_prvi_bajt<<((sizeof(unsigned)-1)*
    BITOVA_U_BAJTU);

  /* Primenom operatora ~ na maska_prvi_bajt dobija se broj cija je
     binarna reprezentacija 11111....1111100000000 (8 bitova najmanje
   tezine su nule, a ostalo su jedinice) */
  maska_prvi_bajt_komplement =~ maska_prvi_bajt;
  /* Primenom operatora ~ na maska_prvi_bajt dobija se broj cija je
     binarna reprezentacija 0000000011111....11111 (8 bitova najvece
   tezine su nule, a ostalo su jedinice) */
  maska_cetvrti_bajt_komplement =~ maska_cetvrti_bajt;
```

```
36    /* U promenljivu prvi_bajt smestamo broj koji se dobija kada se
         bitovi prvog bajta broja x pomere ulevo, tako da budu na
38       poziciji cetvrtog bajta */
      prvi_bajt = (maska_prvi_bajt&x)<<((sizeof(unsigned)-1)*
        BITOVA_U_BAJTU);
40    /* U promenljivu cetvrti_bajt smestamo broj koji se dobija kada se
         bitovi cetvrtog bajta broja x pomere udesno, tako da budu na
42       poziciji prvog bajta */
      cetvrti_bajt = (maska_cetvrti_bajt&x)>>((sizeof(unsigned)-1)*
        BITOVA_U_BAJTU);
44
      /* Na nule se postavlja 8 bitova najmanje tezine, a ostali bitovi
46       ostaju nepromenjeni */
      x = x&maska_prvi_bajt_komplement;
48
      /* Na nule se postavlja 8 bitova najvece tezine, a ostali bitovi
50       ostaju nepromenjeni */
      x = x&maska_cetvrti_bajt_komplement;
52
      /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi
54       iz prvog bajta */
      x = x|prvi_bajt;
56
      /* Na bitove na poziciji cetvrtog bajta se postavljaju bitovi
58       iz prvog bajta */
      x = x|cetvrti_bajt;
60
      return x;
62  }

64  int main () {
      int x;
66
      /* Sa standardnog ulaza se ucitava ceo broj */
68    scanf("%d", &x);

70    /* Provera da li je uneti broj negativan */
      if(x<0){
72      fprintf(stderr, "-1\n");
        exit(EXIT_FAILURE);
74    }

76    /* Ispisivanje rezultata primene funkcije zamena na uneti broj x */
      printf("%u\n", zamena(x));
78
      exit(EXIT_SUCCESS);
80  }
```

Rešenje A.11

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

```c
#include <stdio.h>
#include <stdlib.h>
#include "stabla.h"

/* Funkcija racuna duzinu najduzeg puta od korena do nekog lista */
int najduzi_put(Cvor *koren) {
  /* Pomocne promenljive */
  int najduzi_u_levom,najduzi_u_desnom;

  /* Ako je stablo prazno, povratna vrednost je -1 */
  if(koren==NULL)
    return -1;

  /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
  najduzi_u_levom=najduzi_put(koren->levo);

  /* Rekurzivno se izracuna duzina najduzeg puta u levom podstablu */
  najduzi_u_desnom=najduzi_put(koren->desno);

  /* Veca od prethodno izracunatih vrednosti za podstabla se uvecava
     za 1 i vraca kao konacan rezultat */
  return 1 + (najduzi_u_levom > najduzi_u_desnom ? najduzi_u_levom :
    najduzi_u_desnom);
}

int main() {
  Cvor *stablo = NULL;
  int x;

  /* U svakoj iteraciji se procitani broj dodaje u stablo. */
  while (scanf("%d", &x) == 1)
    if (dodaj_u_stablo(&stablo, x) == 1) {
      fprintf(stderr, "-1\n");
      exit(EXIT_FAILURE);
    }

  /* Ispisuje se rezultat rada trazene funkcije */
  printf("%d\n", najduzi_put(stablo));

  /* Oslobadja se memorija */
  oslobodi_stablo(&stablo);

  exit(EXIT_SUCCESS);
}
```

Rešenje A.12

```c
#include <stdio.h>
#include <stdlib.h>
/* Ime datoteke nije duze od 30 karaktera */
#define MAX 31

/* Funkcija alocira memorijski prostor za matricu sa n vrsta i m
   kolona */
float **alociraj_matricu(int n, int m)
{
  float **matrica = NULL;
  int i, j;

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

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

/* Funkcija oslobadja alocirani memorijski prostor */
float **dealociraj_matricu(float **matrica, int n)
{
  int i;
  /* Oslobadja se prostor rezervisan za svaku vrstu */
  for (i = 0; i < n; i++)
    free(matrica[i]);

  /* Oslobadja se memorija za niz pokazivaca na vrste */
  free(matrica);

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

/* Funkcija prebrojava koliko se puta pojavljuje broj x u i-toj
```

```
   vrsti matrice A, gde je m broj elemenata u vrsti */
int prebroj_u_itoj_vrsti(float **A, int i, int m, int x){
  int j;
  int broj = 0;
  for(j = 0; j<m; j++){
    if(A[i][j] == x)
      broj++;
  }
  return broj;
}

/* Funkcija vraca indeks vrste matrice A u kojoj se realan broj x
   pojavljuje najmanje puta */
int indeks_vrste(float x, float **A, int n, int m){
  /* Indeks vrste sa minimalnim brojem pojavljivanja broja x */
  int min;
  /* Broj pojavljivanja broja x u vrsti sa indeksom min */
  int min_broj;
  /* Promenljiva u kojoj ce se racunati broj pojavljivanja broja x u
  tekucnoj vrsti */
  int broj_u_vrsti;
  /* Pomocne promenljive */
  int i;

  /* Promenljiva min se inicijalizuje na nulu, a min_broj na broj
  pojavljivanja broja x u nultoj vrsti */
  min=0;
  min_broj = prebroj_u_itoj_vrsti(A, 0, m, x);

  /* Za svaku vrstu (osim nulte) se racuna broj pojavljivanja broja
     x u njoj, pa ukoliko je taj broj manji od trenutno najmanjeg
     azuriraju se promenljive min i min_broj */
  for(i=1;i<n;i++){
  broj_u_vrsti = prebroj_u_itoj_vrsti(A, i, m, x);
    if(broj_u_vrsti<min_broj){
      min_broj=broj_u_vrsti;
      min=i;
    }
  }

  /* Funkcija vraca odgovarajuci indeks vrste */
  return min;
}

int main () {
  FILE *in;
  char datoteka[MAX];
  float broj;
  float **A=NULL;
  int i, j, m, n;

  /* Sa standardnog ulaza se ucitava ime datoteke i realan broj*/
```

```
103   scanf("%s",datoteka);
      scanf("%f",&broj);
105
      /* Otvaranje datoteke za citanje */
107   in=fopen(datoteka,"r");

109   /* Provera da li je datoteka uspesno otvorena */
      if(in==NULL){
111     fprintf(stderr,"-1\n");
        exit(EXIT_FAILURE);
113   }

115   /* Dimenzije matrice se ucitavaju iz datoteke (prva dva cela broja
         u datoteci). U slucaju neuspesnog ucitavanja, na standardni
117      izlaz za greske se ispisuje -1 i prekida se program. */
      if(fscanf(in,"%d %d",&n,&m)==EOF){
119     fprintf(stderr,"-1\n");
        exit(EXIT_FAILURE);
121   }

123   /* Provera da li su ucitani brojevi m i n pozitivni */
      if(n<=0 || m<=0){
125     fprintf(stderr,"-1\n");
        exit(EXIT_FAILURE);
127   }

129   /* Alokacija matrice */
      A = alociraj_matricu(n, m);
131
      /*Provera da li je alokacija uspela */
133   if(A == NULL){
        fprintf(stderr,"-1\n");
135     exit(EXIT_FAILURE);
      }
137
      /* Ucitavanje elemenata matrice iz datoteke */
139   for(i=0; i<n; i++){
        for(j=0; j<m; j++)
141       fscanf(in, "%f", &A[i][j]);
      }
143
      /* Zatvaranje datoteke */
145   fclose(in);

147   /* Ispisivanje rezultata poziva funkcije  */
      printf("%d\n", indeks_vrste(broj, A, n, m));
149
      /* Oslobadjanje memorije koju je zauzimala matrica */
151   A = dealociraj_matricu(A, n);

153   exit(EXIT_SUCCESS);
    }
```