Univerzitet u Beogradu
Matematički fakultet

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

Zbirka programa

Beograd, 2015.

Predgovor

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

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

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

Autori

Sadržaj

Glava 1

Uvodni zadaci

1.1 Podela koda po datotekama

Zadatak 1.1 Napisati program za rad sa kompleksnim brojevima.

(a) Definisati strukturu `KompleksanBroj` koja predstavlja kompleksan broj i sadrži realan i imaginaran deo kompleksnog broja.

(b) Napisati funkciju `ucitaj_kompleksan_broj` koja učitava kompleksan broj sa standardnog ulaza.

(c) Napisati funkciju `ispisi_kompleksan_broj` koja ispisuje kompleksan broj na standardni izlaz u odgovarajućem fomatu (npr. broj čiji je realan deo `2` a imaginarni `-3` ispisati kao $(2 - 3i)$ na standardni izlaz).

(d) Napisati funkciju `realan_deo` koja računa vrednosti realnog dela broja.

(e) Napisati funkciju `imaginaran_deo` koja računa vrednosti imaginarnog dela broja.

(f) Napisati funkciju `moduo` koja računa moduo kompleksnog broja.

(g) Napisati funkciju `konjugovan` koja računa konjugovano-kompleksni broj svog argumenta.

(h) Napisati funkciju `saberi` koja sabira dva kompleksna broja.

(i) Napisati funkciju `oduzmi` koja oduzima dva kompleksna broja.

(j) Napisati funkciju `mnozi` koja množi dva kompleksna broja.

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

Napisati program koji testira prethodno napisane funkcije za dva kompleksna broja $z1$ i $z2$ koja se unose sa standardnog ulaza i ispisuje:

(a) realni deo, imaginarni deo i moduo kompleksnog broja $z1$,

(b) konjugovano kompleksan broj i argument broja $z2$,

(c) zbir, razliku i proizvod brojeva $z1$ i $z2$.

Test Test 1

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

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

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

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

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

[Rešenje 1.1]

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

Test Test 1

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

[Rešenje 1.2]

Zadatak 1.3 Napisati malu biblioteku za rad sa polinomima.

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

(b) Napisati funkciju koja ispisuje polinom na standardni izlaz u što lepšem obliku.

(c) Napisati funkciju koja učitava polinom sa standardnog ulaza.

(d) Napisati funkciju za izračunavanje vrednosti polinoma u datoj tački koristeći Hornerov algoritam.

(e) Napisati funkciju koja sabira dva polinoma.

(f) Napisati funkciju koja množi dva polinoma.

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

Test Upotreba programa 1

```
Unesite polinom (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):
Unesite tacku u kojoj racunate vrednost polinoma
5
Vrednost polinoma u tacki je 194.00
Unesite drugi polinom (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):
2 1 0 1
Zbir polinoma je: 1.00x^3+3.00x^2+3.00x+5.00
Prozvod polinoma je: 1.00x^5+2.00x^4+4.00x^3+6.00x^2+3.00x+4.00
Unesite izvod polinoma koji zelite:
2
2. izvod prvog polinoma je: 6.00x+4.00
```

[Rešenje 1.3]

Zadatak 1.4 Napraviti biblioteku za rad sa razlomcima.

(a) Definisati strukturu za reprezentovanje razlomaka.

(b) Napisati funkcije za učitavanje i ispis razlomaka.

(c) Napisati funkcije koje vraćaju brojilac i imenilac.

(d) Napisati funkciju koja vraća vrednost razlomka kao `double` vrednost.

(e) Napisati funkciju koja izračunava recipročnu vrednost razlomka.

(f) Napisati funkciju koja skraćuje dati razlomak.

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

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

Test Test 1

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

1.2 Algoritmi za rad sa bitovima

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

Test 1

```
ULAZ:
 0x7F
IZLAZ:
 00000000000000000000000001111111
```

Test Test 2

```
Ulaz:   0x80
Izlaz:  00000000000000000000000010000000
```

Test Test 3

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

Test Test 4

```
Ulaz:   0xFFFFFFFF
Izlaz:  11111111111111111111111111111111
```

Test Test

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

[Rešenje 1.5]

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

(a) maskom,

(b) brojem x.

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

Test Test 4

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

[Rešenje 1.6]

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

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

Test Test 1

```
Ulaz:   0x7F
Izlaz:
 Najveci:
 11111110000000000000000000000000
 Najmanji:
 00000000000000000000000001111111
```

Test Test 2

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

Test Test 3

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

Test Test 4

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

Test Test 4

```
Ulaz:   0xABCDE123
Izlaz:
 Najveci:
 11111111111111111000000000000000
 Najmanji:
 00000000000000001111111111111111
```

[Rešenje 1.7]

Zadatak 1.8 Napisati program za rad sa bitovima.

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

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

(c) Napisati funkciju koja određuje broj koji se dobija od n bitova datog broja, počevši od pozicije p i vraća ih kao bitove najmanje težine rezultata.

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

(e) Napisati funkciju koja vraća broj koji se dobija invertovanjem n bitova broja x počevši od pozicije p.

(f) Napisati program koji testira prethodno napisane funkcije.

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

Test Test 1

```
Ulaz:   235 5 10 127
Izlaz:
  Broj   235                          = 00000000000000000000000011101011
  reset(  235,    5,   10)            = 00000000000000000000000000101011
  set(  235,    5,   10)              = 00000000000000000000011111101011
  get_bits(  235,    5,   10)         = 00000000000000000000000000000011
  y =                              127 = 00000000000000000000000001111111
  set_n_bits(  235,    5,   10,  127) = 00000000000000000000011111101011
  invert(  235,    5,   10)           = 00000000000000000000011100101011
```

[Rešenje 1.8]

Zadatak 1.9 Rotiranje ulevo podrazumeva pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najviše težine pomera na poziciju najmanje težine. Analogno, rotiranje udesno podrazumeva pomeranje svih bitova za jednu poziciju udesno, s tim što se bit sa pozicije najmanje težine pomera na poziciju najviše težine.

(a) Napisati funkciju `rotate_left` koja određuje broj koji se dobija rotiranjem `k` puta u levo datog celog broja `x`.

(b) Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem `k` puta u desno datog celog neoznačenog broja `x`.

(c) Napisati funkciju `rotate_right_signed` koja određuje broj koji se dobija rotiranjem `k` puta u desno datog celog broja `x`.

Napisati program koji testira prethodno napisane funkcije za broj `x` i broj `k` koji se sa standardnog ulaza unose u heksadekasnom formatu.

Test Test 1

```
Ulaz:   B10011A7 5
Izlaz:
  x                                    = 10110001000000000001000110100111
  rotate_left(2969571751,     5)       = 00100000000000100011010011110110
  rotate_right(2969571751,     5)      = 00111101100010000000000010001101
  rotate_right_signed(2969571751,     5) = 00111101100010000000000010001101
```

[Rešenje 1.9]

Zadatak 1.10 Napisati funkciju `mirror` koja određuje ceo broj čiji je binarni zapis slika u ogledalu binarnog zapisa argumenta funkcije. Napisati i program koji testira datu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu, tako što najpre ispisuje binarnu reprezentaciju unetog broja,

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

Test Test 1

```
Ulaz:  255
Izlaz:
  00000000000000000000001001010101
  10101010010000000000000000000000
```

[Rešenje 1.10]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

[Rešenje 1.11]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

[Rešenje 1.12]

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

Test Test 1

```
Poziv:  ./a.out 1 2
Ulaz:   11
Izlaz:  13
```

Test Test 2

```
Poziv: ./a.out 1 2
Ulaz:   1024
Izlaz:  1024
```

Test Test 3

```
Poziv: ./a.out 12 12
Ulaz:   12345
Izlaz:  12345
```

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

Test Test 1

```
Ulaz:   11
Izlaz:  0000000B
```

Test Test 2

```
Ulaz:  1024
Izlaz: 00000400
```

Test Test 3

```
Ulaz:  12345
Izlaz: 00003039
```

[Rešenje 1.14]

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

Test Test 1

```
Ulaz:   123 10
Izlaz:  4294967285
```

Test Test 2

```
Ulaz:  3251 0
Izlaz: 4294967295
```

Test Test 3

```
Ulaz:   12541 1024
Izlaz:  4294966271
```

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

Test Test 1

```
Ulaz:   123
Izlaz:  0
```

Test Test 2

```
Ulaz:   3245
Izlaz:  2
```

Test 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. Napisati program koji testira napisanu funkciju za vrednosti koje se unose sa standardnog ulaza.

Test Test 1

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

[Rešenje 1.17]

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

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

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

Test Test 1

```
Ulaz:   11
Izlaz: Uneti broj ima paran broj cifara
```

Test Test 2

```
Ulaz:   123
Izlaz: Uneti broj ima neparan
       broj cifara
```

[Rešenje 1.18]

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

Test Test 1

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

[Rešenje 1.19]

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

$$F(0) = 0$$

$$F(1) = 1$$

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

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

Test Test 1

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

[Rešenje 1.20]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

Test Test 5

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

[Rešenje 1.21]

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

Test Test 1

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

[Rešenje 1.22]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

[Rešenje 1.23]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

[Rešenje 1.24]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

[Rešenje 1.25]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

[Rešenje 1.26]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

[Rešenje 1.27]

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

Test Test 1

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

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

[Rešenje 1.29]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

[Rešenje 1.30]

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

Test Test 4

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

Test Test

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

[Rešenje 1.31]

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

Test Test 1

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

[Rešenje 1.32]

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

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

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

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

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

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

Test 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 Test 2

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

    32
```

[Rešenje 1.33]

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

Test Test 1

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

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

drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, a preostali štap se koristi kao pomoćni štap prilikom premeštanja.
Napisati program koji za proizvoljnu vrednost n, koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

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

1.4 Rešenja

Rešenje 1.1

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

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

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

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

```
  printf("(");
  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
    printf("%.2f i", z.imag);

  if (z.imag == 0 && z.real == 0)
    printf("0");

  printf(")");
}

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

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

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

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

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

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

```
  z.imag += z2.imag;

  return z;
}

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

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

  return z;
}

/* Funkcija vraca kompleksan broj cija vrednost je jednaka
   proizvodu argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj 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;
}

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


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

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

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

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

```
    printf("\nrealan_deo: %.f\nimaginaran_deo: %f\nmoduo %f\n",
132         realan_deo(z1), imaginaran_deo(z1), moduo(z1));
    printf("\n");
134
    /* Ispisujemo drugi kompleksan broj, a zatim i racunamo i
136     ispisujemo konjugovano kompleksan broj od z2 */
    ispisi_kompleksan_broj(z2);
138 printf("\nNjegov konjugovano kompleksan broj: ");
    ispisi_kompleksan_broj(konjugovan(z2));
140
    /* Testiramo funkciju koja racuna argument kompleksnih brojeva
142  */
    printf("\nArgument kompleksnog broja: %f\n", argument(z2));
144 printf("\n");

146 /* Testiramo sabiranje kompleksnih brojeva */
    ispisi_kompleksan_broj(z1);
148 printf(" + ");
    ispisi_kompleksan_broj(z2);
150 printf("  =  ");
    ispisi_kompleksan_broj(saberi(z1, z2));
152 printf("\n");

154 /* Testiramo oduzimanje kompleksnih brojeva */
    printf("\n");
156 ispisi_kompleksan_broj(z1);
    printf(" - ");
158 ispisi_kompleksan_broj(z2);
    printf("  =  ");
160 ispisi_kompleksan_broj(oduzmi(z1, z2));
    printf("\n");
162
    /* Testiramo mnozenje kompleksnih brojeva */
164 printf("\n");
    ispisi_kompleksan_broj(z1);
166 printf(" * ");
    ispisi_kompleksan_broj(z2);
168 printf("  =  ");
    ispisi_kompleksan_broj(mnozi(z1, z2));
170 printf("\n");
    /* Program se zavrsava uspesno, tj, bez greske */
172 return 0;
}
```

Rešenje 1.2

```
/* Ukljucujemo zaglavlje neophodno za rad sa kompleksnim brojevima
2 * Ovde je to neophodno jer nam je neophodno da bude poznata
     definicija tipa KompleksanBroj
  * i da budu ukljucena zaglavlja standardne biblioteke koja smo vec
     naveli u complex.h
```

```
*/
#include "complex.h"

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

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

   if(z.imag == 0 && z.real == 0 )
       printf("0");

   printf(")");
}

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

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

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

```

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

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

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

    return z;
}

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

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

    return z;
}

/* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
    argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj  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;
}

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

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

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

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

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

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

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

void ispisi_kompleksan_broj(KompleksanBroj z) ;

float realan_deo(KompleksanBroj z) ;

float imaginaran_deo(KompleksanBroj z);

float moduo(KompleksanBroj  z);

KompleksanBroj konjugovan(KompleksanBroj z) ;

KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj  z2 ) ;

KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj  z2 ) ;

KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj  z2 ) ;

float argument(KompleksanBroj z) ;

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

```
/*
 * Koristimo korektno definisanu biblioteku kompleksnih brojeva.
```

```
 * U zaglavlju complex.h nalazi se definicija komplesnog broja i
    popis deklaracija podrzanih funkcija
 * a u complex.c se nalaze njihove definicije.
 *
 * Ovde pisemo i main() funkciju drugaciju od prethodnog zadatka.
 *
 * I dalje kompilacija programa se najjednostavnije postize naredbom
 * gcc -Wall -lm -o izvrsni complex.c main.c

 Kompilaciju mozemo uraditi i na sledeci nacin:
gcc -Wall -c -o complex.o complex.c
gcc -Wall -c -o main.o main.c
gcc -lm -o complex complex.o main.o
*/


#include <stdio.h>
/* Ukljucujemo zaglavlje neophodno za rad sa kompleksnim brojevima */
#include "complex.h"

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

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

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

   return 0;
}
```

Rešenje 1.3

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


/* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
    obliku.
   Kako bi ustedeli kopiranje cele strukture, polinom prenosimo po
    adresi */
void ispisi(const Polinom * p)
{
    int i;
    for (i = p->stepen; i >= 0; i--) {
  if (p->koef[i]) {
      if (p->koef[i] >= 0 && i != p->stepen)
```

```
    putchar('+');
15      if (i > 1)
    printf("%.2fx^%d", p->koef[i], i);
17      else if (i == 1)
    printf("%.2fx", p->koef[i]);
19      else
    printf("%.2f", p->koef[i]);
21  }
    }
23  putchar('\n');
}
25
/* Funkcija koja ucitava polinom sa tastature */
27 Polinom ucitaj()
{
29  int i;
    Polinom p;
31
    /* Ucitavamo stepen polinoma */
33  scanf("%d", &p.stepen);

35  /* Ponavljamo ucitavanje stepena sve dok ne unesemo stepen iz
    dozvoljenog opsega */
    while (p.stepen > MAX_STEPEN || p.stepen < 0) {
37  printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
  scanf("%d", &p.stepen);
39  }

41  /* Unosimo koeficijente polinoma */
    for (i = p.stepen; i >= 0; i--)
43  scanf("%lf", &p.koef[i]);
    return p;
45 }

47 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
    algoritmom */
/* x^4+2x^3+3x^2+2x+1 = ( ( (x+2)*x + 3)*x + 2)*x + 1 */
49 double izracunaj(const Polinom * p, double x)
{
51  double rezultat = 0;
    int i = p->stepen;
53  for (; i >= 0; i--)
  rezultat = rezultat * x + p->koef[i];
55  return rezultat;
}
57
/* Funkcija koja sabira dva polinoma */
59 Polinom saberi(const Polinom * p, const Polinom * q)
{
61  Polinom rez;
    int i;
63
```

```
    rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

    for (i = 0; i <= rez.stepen; i++)
  rez.koef[i] =
      (i > p->stepen ? 0 : p->koef[i]) + (i >
            q->stepen ? 0 : q->
            koef[i]);
    return rez;

}

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

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

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

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

    return r;
}

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

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

  for (i = 0; i <= r.stepen; i++)
      r.koef[i] = (i + 1) * p->koef[i + 1];
    } else
  r.koef[0] = r.stepen = 0;

    return r;
}

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

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

125     if (n == 0)
      return *p;
127
        r = izvod(p);
129     for (i = 1; i < n; i++)
      r = izvod(&r);
131
        return r;
133 }
```

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

    printf("Unesite tacku u kojoj racunate vrednost polinoma\n");
    scanf("%lf", &x);

    /* Ispisujemo vrednost polinoma u toj tacki */
    printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&p, x));
```

```
34      /* Unesimo drugi polinom */
        printf
36    ("Unesite drugi polinom (prvo stepen, pa zatim koeficijente od
        najveceg stepena do nultog):\n");
        q = ucitaj();
38
      /* Sabiramno polinome i ispisujemo zbir ta dva polinoma */
40      r = saberi(&p, &q);
        printf("Zbir polinoma je: ");
42      ispisi(&r);

44      /* Mnozimo polinome i ispisujemo prozivod ta dva polinoma */
        r = pomnozi(&p, &q);
46      printf("Prozvod polinoma je: ");
        ispisi(&r);
48
      /* Izvod polinoma */
50      printf("Unesite izvod polinoma koji zelite:\n");
        scanf("%d", &n);
52      r = nIzvod(&p, n);
        printf("%d. izvod prvog polinoma je: ", n);
54      ispisi(&r);

56      /* Uspesno zavrsavamo program */
        return 0;
58  }
```

Rešenje 1.5

```
    #include <stdio.h>
2
    /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4      celog broja u memoriji. Bitove u zapisu broja
       ispisujemo sa leva na desno, tj. od bita najvece tezine ka
6      bitu najmanje tezine.  */
    void print_bits(unsigned x)
8   {

10    /* broj bitova celog broja */
      unsigned velicina = sizeof(unsigned) * 8;
12    /* maska koja se koristi za "ocitavanje" bitova */
      unsigned maska;
14
      /* Pocetna vrednost maske se postavlja na broj ciji binarni
16     zapis na mestu bita najvece tezine sadrzi jedinicu, a na
       svim ostalim mestima sadrzi nulu. U svakoj iteraciji ova
18     jedinica se pomera u desno, kako bi se ocitao naredni bit.
        Odgovarajuci
       karakter, ('0' ili '1'), ispisuje se na standardnom izlazu. Zbog
20     siftovanja maske u desno, koja na pocetku ima najvisi bit
       postavljen na 1, neophodno je da maska bude neoznacen ceo
```

```
22    broj kako bi se siftovanjem u desno vrsilo logicko sifotvanje
      (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
24    znakom broja). */
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
26    putchar(x & maska ? '1' : '0');

28  putchar('\n');
}
30

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

38  return 0;
}
```

Rešenje 1.6

```
1  #include <stdio.h>

3  /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
      celog broja u memoriji */
5  void print_bits(int x)
   {
7    unsigned velicina = sizeof(int) * 8;
     unsigned maska;
9
     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
11     putchar(x & maska ? '1' : '0');

13   putchar('\n');
   }
15
   /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
17    x pomeranjem broja x */
   int count_bits(int x)
19 {
     int br = 0;
21   unsigned wl = sizeof(unsigned) * 8 - 1;

23   /* Formiramo masku 100000...0000000, koja sluzi za ocitavanje
        bita najvece tezine. U svakoj iteraciji maska se pomera u
25      desno za 1 mesto, i ocitavamo sledeci bit. Petlja se
        zavrsava kada vise nema jedinica tj. kada maska postane
27      nula. */
     unsigned maska = 1 << wl;
29   for (; maska != 0; maska >>= 1)
       x & maska ? br++ : 1;
```

```
31
  return br;
33 }

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

```
1  #include <stdio.h>

3  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
     celog broja u memoriji */
5  void print_bits(int x)
   {
7    unsigned velicina = sizeof(int) * 8;
     unsigned maska;
9
     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
11     putchar(x & maska ? '1' : '0');

13   putchar('\n');
   }
15
   /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja
17    x pomeranjem broja x */
   int count_bits1(int x)
19 {
     int br = 0;
21   unsigned wl = sizeof(int) * 8 - 1;

23   /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1
       vrsila
        bi aritmeticko pomeranje u desno, tj. popunjavanje bita najvece
25      tezine bitom znaka. U tom slucaju nikad ne bi bio ispunjen
        uslov x!=0 i program bi bio zarobljen u beskonacnoj petlji.
27      Zbog toga se koristi pomeranj broja x ulevo i maska koja
        ocitava bit najvece tezine. */
29
     unsigned maska = 1 << wl;
31   for (; x != 0; x <<= 1)
       x & maska ? br++ : 1;
33
     return br;
35 }

37
```

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

  return 0;
}
```

Rešenje 1.7

```
#include <stdio.h>

/* Funkcija vraca najveci neoznaceni broj sastavljen iz istih
   bitova kao i x */
unsigned najveci(unsigned x)
{
  unsigned velicina = sizeof(unsigned) * 8;

  /* Formiramo masku 100000...0000000 */
  unsigned maska = 1 << (velicina - 1);

  /* Inicijalizujemo rezultat na 0 */
  unsigned rezultat = 0;

  /* Dokle god postoje jedinice u binarnoj reprezentaciji broja
     x (tj. dokle god je x razlicit od nule) pomeramo ga ulevo. */
  for (; x != 0; x <<= 1) {
    /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
       leva u rezultat */
    if (x & maska) {
      rezultat >>= 1;
      rezultat |= maska;
    }
  }

  return rezultat;
}

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

  /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
  for (; x != 0; x >>= 1) {
    /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
       desna u rezultat */
    if (x & 1) {
```

```
      rezultat <<= 1;
      rezultat |= 1;
    }
  }

  return rezultat;
}

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

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

  putchar('\n');
}

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

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

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

  return 0;
}
```

Rešenje 1.8

```
#include <stdio.h>

/* Funckija postavlja na nulu n bitova pocev od pozicije p.
   Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
   se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
   1010 1110 0111 1010 1011 1100 1101 1110 1000 0010 0111 */
unsigned reset(unsigned x, unsigned n, unsigned p)
{
  /* Cilj nam je da samo zeljene bitove anuliramo, a da ostali
     ostanu nepromenjeni. Formiramo masku koja 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 formiramo masku oblika 1111 1111
```

```
15      1111 1111 1111 1000 0011 1111 To postizemo na sledeci
        nacin: ~0 1111 1111 1111 1111 1111 1111 1111 1111 (~0 << n)
17      1111 1111 1111 1111 1111 1111 1110 0000 ~(~0 << n) 0000
        0000 0000 0000 0000 0000 0001 1111 (~(~0 << n) << ( p-n+1))
19      0000 0000 0000 0000 0000 0111 1100 0000 ~(~(~0 << n) << (
        p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111 */
21    unsigned maska = ~(~(~0 << n) << (p - n + 1));

23    return x & maska;
  }
25
  /* Funckija postavlja na 1 n bitova pocev od pozicije p.
27     Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
     se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
29     1010 1110 0111 1010 1011 1100 1101 1110 1111 1110 0111 */
  unsigned set(unsigned x, unsigned n, unsigned p)
31  {
    /* Kako zelimo da samo odredjenih n bitova postavimo na 1, dok
33       ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
       formiramo masku oblika 0000 0000 0000 0000 0000 0111 1100
35       0000 prateci vrlo slican postupak kao za prethodnu funkciju
     */
37    unsigned maska = ~(~0 << n) << (p - n + 1);

39    return x | maska;
  }
41
  /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
43     predstavlja n bitova pocev od pozicije p u binarnoj
     reprezentaciji broja x, pri cemu se pozicija broji sa desna
45     ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
     i broj 1010 1011 1100 1101 1110 1010 1110 0111 0000 0000 0000
47     0000 0000 0000 0000 1011 */
  unsigned get_bits(unsigned x, unsigned n, unsigned p)
49  {
    /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a
51       ostali su 0. Na primer za n=5 0000 0000 0000 0000 0000 0000
       0001 1111 */
53    unsigned maska = ~(~0 << n);

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


61
  /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
63     postavljeni na vrednosti n bitova najnize tezine binarne
     reprezentacije broja y */
65  unsigned set_n_bits(unsigned x, unsigned n, unsigned p,
                        unsigned y)
```

```
{
  /* Kreiramo masku kod 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 last_n_1 = ~(~0 << n);

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

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

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

  return x_reset ^ y_shift_middle;
}


/* Funkcija invertuje bitove u zapisu broja x pocevsi od
   pozicije p njih n */
unsigned invert(unsigned x, unsigned n, unsigned p)
{
  /* Formiramo masku sa n jedinica pocev od pozicije p Na primer
     za n=5 i p=10 0000 0000 0000 0000 0000 0111 1100 0000 */
  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;
}


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

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

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

```
119
120
121 int main()
    {
123   unsigned broj, p, n, y;
      scanf("%u%u%u%u", &broj, &n, &p, &y);
125   printf("Broj %5u %25s= ", broj, "");
      print_bits(broj);
127
128
129   printf("reset(%5u,%5u,%5u)%11s = ", broj, n, p, "");
      print_bits(reset(broj, n, p));
131
      printf("set(%5u,%5u,%5u)%13s = ", broj, n, p, "");
133   print_bits(set(broj, n, p));

135   printf("get_bits(%5u,%5u,%5u)%8s = ", broj, n, p, "");
      print_bits(get_bits(broj, n, p));
137
      printf("y = %31u = ", y);
139   print_bits(y);
      printf("set_n_bits(%5u,%5u,%5u,%5u) = ", broj, n, p, y);
141   print_bits(set_n_bits(broj, n, p, y));

143   printf("invert(%5u,%5u,%5u)%10s = ", broj, n, p, "");
      print_bits(invert(broj, n, p));
145
      return 0;
147 }
```

Rešenje 1.9

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

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

```
20      x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
    }
22  return x;
}
24
/* Funkcija neoznacen broj x rotira u desno za n Na primer za n
26    =5 i x cija je interna reprezentacija 1010 1011 1100 1101
     1110 0001 0010 0011 0001 1101 0101 1110 0110 1111 0000 1001 */
28 unsigned rotate_right(unsigned x, unsigned n)
{
30  unsigned last_bit;
    int i;
32
    /* n puta ponavljamo rotaciju u desno za jedan bit. U svakoj
34     iteraciji odredjujemo bit najmanje tezine broja x, zatm
       tako odredjeni bit siftujemo u levo tako da najnizi bit
36     dode do pozicije najviseg bita i nakon siftovanja x za 1 u
       desno postavljamo x-ov najvisi bit na vrednost najnizeg
38     bita. */
    for (i = 0; i < n; i++) {
40    last_bit = x & 1;
      x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
42  }

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


54
    /* U svakoj iteraciji odredjujemo bit najmanje tezine tj.
56     last_bit. Kako je x oznacen ceo broj, tada se prilikom
       siftovanja u desno vrsi aritmeticki sift i cuva se znak
58     broja. Iza tog razloga imamo dva slucaja u zavisnosti od
       znaka od x. Nije dovoljno da se ova provera izvrsi pre
60     petlje, jer rotiranjem u desno na mesto najviseg bita moze
       doci i 0 i 1, nezavisno od pocetnog znaka x. */
62  for (i = 0; i < n; i++) {
      last_bit = x & 1;
64
      if (x < 0)
66      /* Siftovanjem u desno broja koji je negativan dobijamo 1
           na najvisoj poziciji.  Na primer ako je x 1010 1011
68         1100 1101 1110 0001 0010 001b (sa b oznacavamo u
           primeru 1 ili 0 na najnizoj poziciji) last_bit je 0000
70         0000 0000 0000 0000 0000 0000 000b nakon Siftovanja za
           1 u desno 1101 0101 1110 0110 1111 0000 1001 0001 da
```

```
72          bismo najvisu 1 u x postavili na b nije dovoljno da ga
            siftujemo na najvisu poziciju jer bi se time dobile 0,
74          a nama su potrebne 1 zbog bitovskog & zato prvo
            komplementiramo, pa tek onda siftujemo ~last_bit <<
76          (sizeof(int)*8 -1) B000 0000 0000 0000 0000 0000 0000
            0000 (B oznacava ~b ) i ponovo komplementiramo da bismo
78          imali b na najvisoj poziciji i sve 1 na ostalim
            pozicijama ~(~last_bit << (sizeof(int)*8 -1)) b111 1111
80          1111 1111 1111 1111 1111 1111 1111 */
        x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
82    else
        x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
84  }

86  return x;
}
88

90 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
     celog broja u memoriji */
92 void print_bits(int x)
{
94  unsigned velicina = sizeof(int) * 8;
  unsigned maska;
96  for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
    putchar(x & maska ? '1' : '0');
98
  putchar('\n');
100 }

102 int main()
{
104  unsigned x, k;
  scanf("%x%x", &x, &k);
106  printf("x %36s = ", "");
  print_bits(x);
108  printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
  print_bits(rotate_left(x, k));
110
  printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
112  print_bits(rotate_right(x, k));

114  printf("rotate_right_signed(%7u,%6u) = ", x, k);
  print_bits(rotate_right_signed(x, k));
116
  return 0;
118 }
```

Rešenje 1.10

```c
#include <stdio.h>

/* Funkcija vraca vrednost cija je binarna reprezentacija slika
   u ogledalu binarne reprezentacije broja x. Na primer za x
   cija binarna reprezentacija izgleda ovako
   10101011110011011110010010010011 funkcija treba da vrati
   broj cija binarna reprezentacija izgleda:
   11000100100001111011001111010101 */
unsigned mirror(unsigned x)
{
  unsigned najnizi_bit;
  unsigned rezultat = 0;

  int i;
  /* Krecemo od najnizeg bita u zapisu broja x i dodajemo ga u
     rezultat */
  for (i = 0; i < sizeof(x) * 8; i++) {
    najnizi_bit = x & 1;
    x >>= 1;
    /* Potiskujemo trenutni rezultat ka levom kraju. Tako svi
       prethodno postavljeni bitovi dobijaju vecu poziciju. Novi
       bit postavljamo na najnizu poziciju */
    rezultat <<= 1;
    rezultat |= najnizi_bit;
  }
  return rezultat;
}


/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
   celog broja u memoriji */
void print_bits(int x)
{
  unsigned velicina = sizeof(int) * 8;
  unsigned maska;
  for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
    putchar(x & maska ? '1' : '0');

  putchar('\n');
}

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

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

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

```
54    return 0;
}
```

Rešenje 1.11

```
#include <stdio.h>

/* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n
   broj jedinica veci od broja nula. U suprotnom funkcija vraca
   0 */
int Broj01(unsigned int n)
{

  int broj_nula, broj_jedinica;
  unsigned int maska;

  broj_nula = 0;
  broj_jedinica = 0;

  /* Postavljamo masku tako da pocinjemo sa analiziranjem bita
     najvece tezine */
  maska = 1 << (sizeof(unsigned int) * 4 - 1);

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

    /* Proveravamo da li se na poziciji koju odredjuje maska
       nalazi 0 ili 1 i uvecavamo odgovarajuci brojac */
    if (n & maska) {
      broj_jedinica++;
    } else {
      broj_nula++;
    }

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

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

}

int main()
{
  unsigned int n;

  /* Ucitavamo broj */
```

```
46    scanf("%u", &n);

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

Rešenje 1.12

```
      #include <stdio.h>
2
      int broj_parova(unsigned int x)
4     {

6       int broj_parova;
        unsigned int maska;
8
        /* Postavljamo broj parova na 0 */
10      broj_parova = 0;

12      /* Postavljamo masku tako da mozemo da procitamo da li su dva
           najmanja bita u zapisu broja x 11 */
14      /* broj 3 je binarno 000....00011 */
        maska = 3;
16
        /* Dok ne obidjemo sve parove bitova u zapisu broja x */
18      while (x != 0) {

20        /* Proveravamo da li se na najmanjim pozicijama broj x
             nalazi 11 par */
22        if ((x & maska) == maska) {
            broj_parova++;
24        }

26        /* Pomeramo broj u desnu stranu tako da mozemo da ocitamo
             vrednost sledeceg para bitova. Pomeranjem u desno
28           bit najvece tezine se popunjava nulom jer je x
             neoznacen broj. */
30        x = x >> 1;
        }
32
        return broj_parova;
34

      }
36
      int main()
38    {
        unsigned int x;
40
        /* Ucitavamo broj */
```

```
42    scanf("%u", &x);

44    /* Ispsujemo vrednost funkcije */
      printf("%d\n", broj_parova(x));
46
      return 0;
48  }
```

Rešenje 1.13

Rešenje 1.14

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

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

10   Na primer, ako je duzina unsigned int 4 bajta onda je
     MAX_DUZINA 9 */
12
#define MAX_DUZINA sizeof(unsigned int)*2 +1
14

16  void prevod(unsigned int x, char s[])
    {
18
      int i;
20    unsigned int maska;
      int vrednost;
22
      /* Heksadekadni zapis broja 15 je 000...0001111 - ovo nam
24       odgovara ako hocemo da citamo 4 uzastopne cifre */
      maska = 15;
26
      /*
28       Broj cemo citati od pozicije najmanje tezine ka poziciji
         najvece tezine; npr. za broj
30       00000000001101000100001111010101 u prvom koraku cemo
         procitati bitove: 00000000001101000010000111101<0101>
32       (bitove izdvojene sa <...>) u drugom koraku cemo procitati:
         00000000001101000100001l<1101>0101 u trecem koraku cemo
34       procitati: 0000000000110100010<0011>11010101 i tako redom

36       indeks i oznacava poziciju na koju smestamo vrednost

38     */
```

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

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

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

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

int main()
{

  unsigned int x;
  char s[MAX_DUZINA];

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

  /* Pozivamo funkciju */
  prevod(x, s);

  /* Ispsujemo dobijenu nisku */
  printf("%s\n", s);

  return 0;
}
```

Rešenje 1.17

```
#include <stdio.h>

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

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

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

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

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


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

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

  /* Ako je stepen paran */
  if ((k % 2) == 0)
    return stepen2(x * x, k / 2);
  /* Inace (ukoliko je stepen neparan) */
  return x * stepen2(x * x, k / 2);
}

int main()
{
  int x, k;
  scanf("%d%d", &x, &k);

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

Rešenje 1.18

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

#define MAX 100

/* 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);

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

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

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

Rešenje 1.19

```c
#include <stdio.h>
    /* Repno-rekurzivna (eng. tail recursive) je ona funkcija
       Cije se telo zavrsava rekurzivnim pozivom, pri cemu taj
       rekurzivni poziv ne ucestvuje u nekom izrazu.
```

```
6         Kod ovih funkcija se po zavrsetku za tekuci rekurzivni
          poziv umesto skoka na adresu povratka skace na adresu
8         povratka za prethodni poziv, odnosno za poziv na manjoj
          dubini. Time se stedi i prostor i vreme.
10
          Ovakve funkcije se mogu lako zameniti odgovarajucom
12        iterativnom funkcijom, cime se smanjuje prostorna
          slozenost algoritma. */
14    /* PomoCna funkcija koja izracunava n! * result. Koristi
          repnu rekurziju. */
16    /* Result je argument u kom cemo akumulirati do tada
          izracunatu vrednost faktorijela. Kada zavrsimo, tj. kada
18        dodjemo do izlaza iz rekurzije potrebno je da vratimo
          result. */
20int faktorijelRepna(int n, int result)
{
22  if (n == 0)
      return result;
24
  return faktorijelRepna(n - 1, n * result);
26}

28/* Sada zelimo da se oslobodimo repne rekurzije koja postoji u
   funkciji faktorijelRepna, koristeci algoritam sa predavanja.
30
   Najpre cemo vrednost argumenta funkcije postaviti na vrednost
32 koja bi se prosledjivala rekurzivnom pozivu i pomocu goto
   naredbe vratiti se na pocetak tela funkcije. */
34
int faktorijelRepna_v1(int n, int result)
36{
pocetak:
38  if (n == 0)
      return result;
40
  result = n * result;
42  n = n - 1;
  goto pocetak;
44}

46/* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
   programerska praksa. Iskoristicemo prethodni medjukorak da
48 bismo dobili iterativno resenje bez bezuslovnih skokova. */
int faktorijelRepna_v2(int n, int result)
50{
  while (n != 0) {
52    result = n * result;
      n = n - 1;
54  }

56  return result;
```

```
}

/* Nasim gore navedenim funkcijama pored 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 faktorijelRepna sa pozivom
   faktorijelRepna_v1, a zatim sa pozivom funkcije
   faktorijelRepna_v2. */
int faktorijel(int n)
{
  return faktorijelRepna(n, 1);
}

/* Test program */
int main()
{
  int n;

  printf("Unesite n (<= 12): ");
  scanf("%d", &n);
  printf("%d! = %d\n", n, faktorijel(n));

  return 0;
}
```

Rešenje 1.20

Rešenje 1.21

Rešenje 1.22

```
#include <stdio.h>
#define MAX_DIM 1000

/*
   Ako je n==0, onda je suma(a,0) = 0 Ako je n>0, onda je
   suma(a,n) = a[n-1] + suma(a,n-1) Suma celog niza je jednaka
   sumi prvih n-1 elementa uvecenoj za poslednji element celog
   niza. */
int sumaNiza(int *a, int n)
{
  /* Ne stavljamo strogu jednakost n==0, za slucaj da korisnik
     prilikom prvog poziva, posalje negativan broj za velicinu
     niza. */
```

```
14    if (n <= 0)
        return 0;
16
    return a[n - 1] + sumaNiza(a, n - 1);
18  }

20  /*
       n==0, suma(a,0) = 0 n >0, suma(a,n) = a[0]+suma(a+1,n-1) Suma
22     celog niza je jednaka zbiru prvog elementa niza i sume
       preostalih n-1 elementa. */
24  int sumaNiza2(int *a, int n)
    {
26    if (n <= 0)
        return 0;
28
    return a[0] + sumaNiza2(a + 1, n - 1);
30  }

32  int main()
    {
34    int a[MAX_DIM];
      int n, i = 0;
36
      /* Ucitavamo broj elemenata niza */
38    scanf("%d", &n);

40    /* Ucitavamo n elemenata niza. */
      for (i = 0; i < n; i++)
42      scanf("%d", &a[i]);

44    printf("Suma elemenata je %d\n", sumaNiza(a, n));
      // printf("Suma elemenata je %d\n",sumaNiza2(a, n));
46
      return 0;
48  }
```

Rešenje 1.23

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

4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza
     niz dimenzije n */
6 int maksimum_niza(int niz[], int n)
  {
8   /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveci
       je ujedno i jedini element niza */
10  if (n == 1)
      return niz[0];
12
    /* ReSavamo problem manje dimenzije */
```

```
14   int max = maksimum_niza(niz, n - 1);

16   /* Ako nam je poznato resenje problema dimenzije n-1, resavamo
        problem dimenzije n */
18   return niz[n - 1] > max ? niz[n - 1] : max;
}
20
int main()
22 {
   int brojevi[MAX_DIM];
24   int n;

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

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

Rešenje 1.24

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

4  int skalarno(int a[], int b[], int n)
   {
6    /* Izlazak iz rekurzije */
     if (n == 0)
8      return 0;

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

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

20   /* Unosimo dimenziju nizova, */
     scanf("%d", &n);
22
     /* a zatim i same nizove. */
24   for (i = 0; i < n; i++)
```

```
    scanf("%d", &a[i]);

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

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

  return 0;
}
```

Rešenje 1.25

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

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

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

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

  /* UCitavamo broj koji se trazi */
  scanf("%d", &x);

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

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

Rešenje 1.26

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

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

  else
    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[MAX_DIM];
  int n;

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

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

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

  return 0;
}
```

Rešenje 1.27

```c
#include <stdio.h>

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

   Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
   siftovanjem sa desne strane uvek dopisuju 0, argument x ce u
```

```
   nekom rekurzivnom pozivu biti bas 0 i izacicemo iz rekurzije. */

int count(int x)
{
  /* izlaz iz rekurzije */
  if (x == 0)
    return 0;

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

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

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

  return 0;
}
```

Rešenje 1.29

```
#include<stdio.h>

/* Rekurzivna funkcija za odredjivanje najvece heksadekadne
   cifre u broju */
int max_oktalna_cifra(unsigned x)
{
  /* izlazak iz rekurzije */
  if (x == 0)
    return 0;
  /* Odredjivanje poslednje heksadekadne cifre u broju */
```

```
  int poslednja_cifra = x & 7;
  /* Odredjivanje maksimalne oktalne cifre u broju kada se iz
     njega izbrise poslednja oktalna cifra */
  int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
  return poslednja_cifra >
      max_bez_poslednje_cifre ? poslednja_cifra :
      max_bez_poslednje_cifre;
}

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

Rešenje 1.30

```
#include<stdio.h>


/* Rekurzivna funkcija za odredjivanje najvece oktalne cifre u
   broju */
int max_heksadekadna_cifra(unsigned x)
{
  /* Izlazak iz rekurzije */
  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 max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);
  return poslednja_cifra >
      max_bez_poslednje_cifre ? poslednja_cifra :
      max_bez_poslednje_cifre;
}

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

Rešenje 1.31

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

int palindrom(char s[], int n)
{
  if ((n == 1) || (n == 0))
    return 1;
  return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
}

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

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

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

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

  return 0;
}
```

Rešenje 1.32

```c
#include <stdio.h>
#include <stdlib.h>
#define  MAX_DUZINA_NIZA 50

void ispisiNiz(int a[], int n)
{
  int i;

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

/* Funkcija proverava da li se x vec nalazi u permutaciji na
   prethodnih 1...n mesta */
```

```
int koriscen(int a[], int n, int x)
{
  int i;
  for (i = 1; i <= n; i++)
    if (a[i] == x)
      return 1;

  return 0;
}

/* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[]
   je niz u koji smesta permutacije m - oznacava da se na m-tu
   poziciju u permutaciji smesta jedan od preostalih celih
   brojeva n- je velicina skupa koji se permutuje Funkciju
   pozivamo sa argumentom m=1 jer krecemo da formiramo
   permutaciju od 1. pozicije i nikada ne koristimo a[0]. */
void permutacija(int a[], int m, int n)
{
  int i;

  /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
     broj premasila velicinu skupa, onda se svi brojevi vec
     nalaze u permutaciji i ispisujemo permutaciju. */
  if (m > n) {
    ispisiNiz(a, n);
    return;
  }

  /* Ideja: pronalazimo prvi broj koji mozemo da postavimo na
     m-to mesto u nizu (broj koji se do sada nije pojavio u
     permutaciji). Zatim, rekurzivno pronalazimo one permutacije
     koje odgovaraju ovako postavljenom pocetku permutacije.
     Kada to zavrsimo, proveravamo 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, funkcija je zavrSila sa radom.
     Ukoliko takav broj postoji, onda ponovo pozivamo rekurzivno
     pronalazenje odgovarajucih permutacija, ali sada sa
     drugacije postavljenim prefiksom. */


  for (i = 1; i <= n; i++) {
    /* Ako se broj i nije do sada pojavio u permutaciji od 1 do
       m-1 pozicije, onda ga stavljamo na poziciju m i pozivamo
       funkciju da napravi permutaciju za jedan vece duzine, tj.
       m+1. Inace, nastavljamo dalje, trazeci broj koji se nije
       pojavio do sada u permutaciji. */
    if (!koriscen(a, m - 1, i)) {
      a[m] = i;
      /* Pozivamo ponovo funkciju da dopuni ostatak permutacije
         posle upisivanja i na poziciju m. */
      permutacija(a, m + 1, n);
    }
```

```
  }
}

int main(void)
{
  int n;
  int a[MAX_DUZINA_NIZA];

  printf("Unesite duzinu permutacije: ");
  scanf("%d", &n);
  if (n < 0 || n >= MAX_DUZINA_NIZA) {
    fprintf(stderr,
            "Duzina permutacije mora biti broj veci od 0 i manji od %
    d!\n",
            MAX_DUZINA_NIZA);
    exit(EXIT_FAILURE);
  }

  permutacija(a, 1, n);

  exit(EXIT_SUCCESS);
}
```

Rešenje 1.33

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

/* Rekurzivna funkcija za racunanje binomnog koeficijenta.  */
/* ako je k=0 ili k=n, onda je binomni koeficijent 0 ako je k
   izmedju 0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k) */
int binomniKoeficijent(int n, int k)
{
  return (0 < k
          && k < n) ? binomniKoeficijent(n - 1,
                                         k - 1) +
      binomniKoeficijent(n - 1, k) : 1;
}

/* Iterativno izracunavanje datog binomnog koeficijenta.

   int binomniKoeficijent (int n, int k) { int i, j, b; for
   (b=i=1, j=n; i<=k; b=b*j--/i++); return b; }

 */

/* Prostim opaZanjem se uocava da se svaki element n-te
   hipotenuze (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1
   hipotenuze. Uz pomenute dve nove ivicne jedinice lako se
   zakljucuje da ce suma elementa n-te hipotenuze biti tacno 2
   puta veca. */
```

```
int sumaElemenataHipotenuze(int n)
{
  return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
}

int main()
{
  int n, k, i, d;


  scanf("%d %d", &d, &n);

  /* Ispisivanje Paskalovog trougla */
  putchar('\n');
  for (n = 0; n <= d; n++) {
    for (i = 0; i < d - n; i++)
      printf("  ");
    for (k = 0; k <= n; k++)
      printf("%4d", binomniKoeficijent(n, k));
    putchar('\n');
  }

  if (n < 0) {
    fprintf(stderr,
            "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
    );
    exit(EXIT_FAILURE);
  }
  printf("%d\n", sumaElemenataHipotenuze(n));

  exit(EXIT_SUCCESS);
}
```

Glava 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 programa:
 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 programa:
 Unesite dimenziju niza:  0
 Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.1]

Zadatak 2.2 Dat je niz realnih brojeva dimenzije n.

(a) Napisati funkciju `zbir` koja izračunava zbir elemenata niza.

(b) Napisati funkciju `proizvod` koja izračunava proizvod elemenata niza.

(c) Napisati funkciju `min_element` koja izračunava najmanji elemenat niza.

(d) Napisati funkciju `max_element` koja izračunava najveći elemenat niza.

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

Primer 1

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

[Rešenje 2.2]

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

Primer 1

```
Interakcija programa:
 Unesite dimenziju niza:  5
 Unesite elemente niza:
 1 2 3 4 5
 Transformisan niz je:
 2 3 3 3 4
```

Primer 2

```
Interakcija programa:
 Unesite dimenziju niza:  4
 Unesite elemente niza:
 4 -3 2 -1
 Transformisan niz je:
 5 -2 1 -2
```

Primer 3

```
Interakcija programa:
 Unesite dimenziju niza:  0
 Greska: neodgovarajuca dimenzija niza.
```

Primer 4

```
Interakcija programa:
 Unesite dimenziju niza:  101
 Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Primer 1

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

Interakcija programa:
 Broj prihvacenih 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 su:
 . p 2 t -
```

Primer 2

```
Poziv: ./a.out

Interakcija programa:
 Broj prihvacenih 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 su:
 .
```

[Rešenje 2.4]

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

Milena: nedostaje poziv u prvom primeru

Primer 1

```
Interakcija programa:
 Broj argumenata komandne linije
 koji su palindromi je 4.
```

Primer 2

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

Interakcija programa:
 Broj argumenata komandne linije koji
 koji su palindromi je 4.
```

Primer 3

```
Poziv: ./a.out

Interakcija programa:
 Broj argumenata komandne linije koji
 koji su palindromi je 0.
```

[Rešenje 2.5]

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

Primer 1

```
Poziv: ./a.out ulaz.txt 1

Ulazna datoteka (ulaz.txt)
 Ovo je sadrzaj datoteke i u njoj ima
 reci koje imaju 1 karakter

Interakcija programa:
 Broj reci ciji je broj karaktera 1 je 3.
```

Primer 2

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

Ulazna datoteka (ulaz.txt)
 Ovo je sadrzaj datoteke i u njoj ima
 reci koje imaju 1 karakter

Interakcija programa:
 Greska: Nedovoljan broj argumenata
 komandne linije.
 Program se poziva sa
 ./a.out ime_dat br_karaktera.
```

Primer 3

```
Poziv: ./a.out ulaz.txt 2

Datoteka ulaz.txt ne postoji

Interakcija programa:
 Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

[Rešenje 2.6]

Zadatak 2.7 Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima zadati sufiks

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

Primer 1

```
Poziv: ./a.out ulaz.txt ke -s

Ulazna datoteka (ulaz.txt)
 Ovo je sadrzaj datoteke i u njoj ima reci
 koje se zavrsavaju na ke

Interakcija programa:
 Broj reci koje se zavrsavaju na ke je 2.
```

Primer 2

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

Ulazna datoteka (ulaz.txt)
 Ovo je sadrzaj datoteke i u njoj ima reci
 koje pocinju sa sa

Interakcija programa:
 Broj reci koje pocinju na sa je 3.
```

Primer 3

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

Datoteka ulaz.txt ne postoji

Interakcija programa:
 Greska: Neuspesno otvaranje
 datoteke ulaz.txt.
```

Primer 4

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

Ulazna datoteka (ulaz.txt)
 Ne postoji

Interakcija programa:
 Greska: Nedovoljan broj argumenata
 komandne linije.
 Program se poziva sa
 ./a.out ime_dat suf/pref -s/-p.
```

[Rešenje 2.7]

2.2 Višedimenzioni nizovi

Zadatak 2.8 Data je kvadratna matrica dimenzije n.

(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 dimanziju 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.

Test Test 1

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

Test Test 2

```
Ulaz:  0
Izlaz: Greska: neodgovarajuca dimenzija matrice.
```

Zadatak 2.9 Date su dve kvadratne matrice istih dimenzija n.

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

(b) Napisati funkciju koja izračunava zbir matrica.

(c) Napisati funkciju koja izračunava proizvod matrica.

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

Test Test 1

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

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

(a) Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.

(b) Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.

(c) Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.

(d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).

(e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).

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

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

Test Test 1

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

Zadatak 2.11 Data je kvadratna matrica dimenzije n.

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

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

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

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

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

Milena: Izbegavala bih komentare koji ulaze u kod i na taj nacin narusavaju citljivost koda, kao sto je to npr u funkciji indeks_min i indeks_max. Resenje 2.15 - izbacila bih napomenu iz komentara. Slicno mi se cini i za zadatak 2.17. Zadatak 2.17 - cini mi se da je resenje bez koriscnja biblioteckih funckija visak? Zadatak 2.19 — izvuci komentare za ucitaj i ispisi ispred funkcija, umesto sto su unutar funkcija. Zadatak 2.21 - cini mi se da komentari unutar funkcije izmeni narusavaju citljivost koda. Resenje 2.26 — izbaciti nasa slova iz komentara, izbaciti mozda napomenu sa pocetka jer je suvisna

Test Test 1

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

Test Test 2

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

Test Test 3

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

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

Test Test 1

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

Test Test 2

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

Test Test 3

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

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

(a) Napsiati funkciju koja učitava elemente matrice sa standardnog ulaza

(b) Napsiati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice.

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

Test Test 1

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

Test Test 2

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

Test Test 3

```
Ulaz:  11 4
Izlaz: Greska: neodgovarajuce dimenzije matrice.
```

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

Test Test 1

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

2.3 Dinamička alokacija memorije

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

Test Test 1

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

Test Test 2

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

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

(a) realokaciju memorije niza vršiti korišćenjem `malloc()` funkcije,

(b) realokaciju memorije niza vršiti korišćenjem `realloc()` funkcije.

Test Test 1

```
Ulaz:  1 -2 3 -4 0
Izlaz: -4 3 -2 1
```

Test Test 2

```
Ulaz:  0
Izlaz:
```

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

Test Test 1

```
Ulaz:  Jedan Dva
Izlaz: JedanDva
```

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

Test Test 1

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

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

(a) Napisati funkciju koja vrši učitavanje matrice sa standardnog ulaza.

(b) Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu).

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

Test Test 1

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

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

Test Test 1

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

Zadatak 2.21 Data je kvadratna realna matrica dimenzije n.

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

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

Test Test 1

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

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

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

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

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

2.4 Pokazivači na funkcije

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

Test Test 1

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

Test Test 2

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

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

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

Test Test 1

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

Test Test 2

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

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

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

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

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

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

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

2.5 Rešenja

Rešenje 2.1

```
#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;
  }
}

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

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

    /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
       vrednost koja se nalazi na adresi na koju pokazuje
       pokazivac "poslednji". Nakon toga se pokazivac "prvi"
       uvecava za jedan sto za posledicu ima da "prvi" pokazuje
       na sledeci element u nizu */
    *prvi++ = *poslednji;

    /* Vrednost promenljive "t" se postavlja na adresu na koju
       pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
       umanjuje za jedan, sto za posledicu ima da pokazivac
       "poslednji" sada pokazuje na element koji mu prethodi u
       nizu */
```

```
    *poslednji-- = t;
  }

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

int main()
{
  /* Deklarise se niz od najvise MAX elemenata */
  int a[MAX];

  /* Broj elemenata niza a */
  int n;

  /* Pokazivac na elemente niza */
  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);

  obrni_niz_v1(a, n);

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

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

  obrni_niz_v2(a, n);

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

  for (p = a; p - a < n; p++)
```

```
    printf("%d ", *p);
  printf("\n");

  return 0;
}
```

Rešenje 2.2

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

#define MAX 100

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

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

  return s;
}

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

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

  return p;
}

/* Funkcija izracunava minimalni element niza */
double min(double *a, int n)
{
  /* Na pocetku, minimalni element je prvi element */
  double min = a[0];
  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;
}
```

```c
/* 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");
    exit(EXIT_FAILURE);
  }

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

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

  return 0;
}
```

Rešenje 2.3

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

/* Funkcija povecava za jedan sve elemente u prvoj polovini niza
```

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

  while (prvi < poslednji) {

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

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

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

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

  /* Drugi nacin */
  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");

  return 0;
}
```

Rešenje 2.4

```
#include <stdio.h>

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

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

  printf("Kako zelite da ispisete argumente, ");
  printf("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 su:\n");
  if(tip_ispisa=='I') {
```

```
38      /* koristeci indeksnu sintaksu */
        for (i = 0; i < argc; i++)
40        printf("%c ", argv[i][0]);
        printf("\n");
42    } else if(tip_ispisa=='P'){
        /* koristeci pokazivacku sintaksu */
44      for (i = 0; i < argc; i++)
          printf("%c ", **argv++);
46      printf("\n");
    }
48
    return 0;
50  }
```

Rešenje 2.5

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

5   /* Funkcija ispituje da li je niska palindrom */
    int palindrom(char *niska)
7   {
      int i, j;
9     for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
        if (*(niska + i) != *(niska + j))
11        return 0;
      return 1;
13  }

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

19    /* Nulti argument komandne linije je ime izvrsnog programa */
      for (i = 1; i < argc; i++)
21      if (palindrom(*(argv + i)))
          n++;
23
      printf("Broj argumenata komandne linije koji su palindromi je %d.\n
        ", n);
25    return 0;
    }
```

Rešenje 2.6

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

```c
#define MAX_KARAKTERA 100

/* Implementacija funkcija 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];
  int br = 0, n;
  FILE *in;

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

  /* Otvara se datoteka 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);

  /* Zatvara se datoteka */
  fclose(in);
  return 0;
}
```

Rešenje 2.7

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

#define MAX_KARAKTERA 100

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

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

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

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

/* Funkcija ispituje da li je niska zadata drugim argumentom
   funkcije prefiks niske zadate prvi argumentom funkcije */
int prefiks_niske(char *niska, char *prefiks)
{
  int i;
  if (duzina_niske(prefiks) <= duzina_niske(niska)) {
    for (i = 0; i < duzina_niske(prefiks); i++)
      if (*(prefiks + i) != *(niska + i))
```

```
        return 0;
52    return 1;
  } else
54    return 0;
}
56
int main(int argc, char **argv)
58{
  /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
60    greska */
  if (argc < 4) {
62    printf("Greska: ");
    printf("Nedovoljan broj argumenata komandne linije.\n");
64    printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
            argv[0]);
66    exit(EXIT_FAILURE);
  }
68
  FILE *in;
70  int br = 0;
  char rec[MAX_KARAKTERA];
72
  in = fopen(*(argv + 1), "r");
74  if (in == NULL) {
    fprintf(stderr, "Greska: ");
76    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
            argv[1]);
78    exit(EXIT_FAILURE);
  }
80
  /* Provera se opcija kojom je pozvan program a zatim se
82    ucitavaju reci iz datoteke i broji se koliko njih zadovoljava
    trazeni uslov */
84  if (!(poredjenje_niski(*(argv + 3), "-s"))) {
    while (fscanf(in, "%s", rec) != EOF)
86    br += sufiks_niske(rec, *(argv + 2));
  printf("Broj reci koje se zavrsavaju na %s je %d.\n", *(argv + 2),
    br);
88  } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
    while (fscanf(in, "%s", rec) != EOF)
90    br += prefiks_niske(rec, *(argv + 2));
    printf("Broj reci koje pocinju na %s je %d.\n", *(argv + 2), br);
92  }

94  fclose(in);
  return 0;
96}
```

Rešenje 2.8

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

#define MAX 100

/* Deklarisemo funkcije koje cemo kasnije da definisemo */
double euklidska_norma(int M[][MAX], int n);
int trag(int M[][MAX], int n);
int gornja_vandijagonalna_norma(int M[][MAX], int n);

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

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

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

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

  /* Ispis elemenata matrice koriscenjem indeksne sintakse.
     Ispis vrsimo vrstu po vrstu */
  for (i = 0; i < n; i++) {
    /* Ispisujemo elemente i-te vrste */
    for (j = 0; j < n; j++)
      printf("%d ", A[i][j]);
    printf("\n");
  }

  /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
     Kod ovako definisane matrice, elementi su uzastopno
     smesteni u memoriju, kao na traci. To znaci da su svi
     elementi prve vrste redom smesteni jedan iza drugog. Odmah
     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"); } */

  int tr = trag(A, n);
```

```
  printf("trag = %d\n", tr);

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

  return 0;
}

/* Definisemo funkcije koju smo ranije deklarisali */

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

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

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

  return sqrt(norma);
}

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

  for (i = 0; i < n; i++) {
    for (j = i + 1; j < n; j++)
      norma += abs(M[i][j]);
  }

  return norma;
}
```

Rešenje 2.9

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

```c
#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++)
      /* Nasli smo elemente na istim pozicijama u matricama koji
         se razlikuju */
      if (a[i][j] != b[i][j])
        return 0;

  /* Prosla je provera jednakosti za sve parove elemenata koji
     su na istim pozicijama sto 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++) {
      /* Mnozimo i-tu vrstu 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], c[MAX][MAX];

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

  /* Dimenzija matrica */
  int n;
  int i, j;

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

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

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

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

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

```
    printf("ne\n");

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

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

  return 0;
}
```

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 sam sa sobom,
   odnosno ako se u matrici relacije na glavnoj dijagonali nalaze
   jedinice */
int refleksivnost(int m[][MAX], int n)
{
  int i;

  /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
     dijagonali vazi da je indeks vrste jednak indeksu kolone */
  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;

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

  /* Postavljamo na glavnoj dijagonali 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;

  /* Obilazimo elemente ispod glavne dijagonale matrice i
     uporedjujemo ih 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;

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

  /* Odredjujemo simetricno zatvorenje matrice */
  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++)
      /* Pokusavamo da pronadjemo 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 tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
{
  int i, j, k;

  /* Kopiramo pocetnu matricu u matricu rezultata */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      zatvorenje[i][j] = m[i][j];

  /* Primenom Varsalovog algoritma odredjujemo
     refleksivno-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, k;

  /* Ako korisnik nije uneo trazene argumente, prijavljujemo
```

```
     gresku */
  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);
  }

  /* Otvaramo datoteku 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);
  }

  /* Ucitavamo dimenziju matrice */
  fscanf(ulaz, "%d", &n);

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

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

  /* Ispisujemo trazene vrednosti */
  printf("Refleksivnost: %s\n",
         refleksivnost(m, n) == 1 ? "da" : "ne");

  printf("Simetricnost: %s\n",
         simetricnost(m, n) == 1 ? "da" : "ne");

  printf("Tranzitivnost: %s\n",
         tranzitivnost(m, n) == 1 ? "da" : "ne");

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

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

  printf("Refleksivno-tranzitivno zatvorenje:\n");
  tran_zatvorenje(m, n, pomocna);
```

```
  pisi_matricu(pomocna, n);

  /* Zatvaramo datoteku */
  fclose(ulaz);

  return 0;
}
```

Rešenje 2.11

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

#define MAX 32

int max_sporedna_dijagonala(int m[][MAX], int n)
{
  int i, j;
  /* Trazimo najveci element na sporednoj dijagonali. Za
     elemente sporedne dijagonale vazi da je zbir indeksa vrste
     i indeksa kolone jednak n-1. Za pocetnu vrednost maksimuma
     uzimamo element u gornjem desnom uglu */
  int max_na_sporednoj_dijagonali = m[0][n - 1];
  for (i = 1; i < n; i++)
    if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)
      max_na_sporednoj_dijagonali = m[i][n - 1 - i];

  return max_na_sporednoj_dijagonali;
}

/* Funkcija izracunava indeks kolone najmanjeg elementa */
int indeks_min(int m[][MAX], int n)
{
  int i, j;
  /* Za pocetnu vrednost minimuma uzimamo element u gornjem
     levom uglu */
  int min = m[0][0], indeks_kolone = 0;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      /* Ako je tekuci element manji od minimalnog */
      if (m[i][j] < min) {
        /* cuvamo njegovu vrednost */
        min = m[i][j];
        /* i cuvamo indeks kolone u kojoj se nalazi */
        indeks_kolone = j;
      }
  return indeks_kolone;
}

/* Funkcija izracunava indeks vrste najveceg elementa */
```

```
int indeks_max(int m[][MAX], int n)
{
  int i, j;
  /* Za maksimalni element uzimamo gornji levi ugao */
  int max = m[0][0], indeks_vrste = 0;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      /* Ako je tekuci element manji od minimalnog */
      if (m[i][j] > max) {
        /* cuvamo njegovu vrednost */
        max = m[i][j];
        /* i cuvamo indeks vrste u kojoj se nalazi */
        indeks_vrste = i;
      }
  return indeks_vrste;
}

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

  int broj_negativnih = 0;

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

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

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

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

  if (n > MAX || n <= 0) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
```

```
    fprintf(stderr, "matrice.\n");
    exit(EXIT_FAILURE);
  }

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

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

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

  /* Prekidamo izvrsavanje programa */
  return 0;
}
```

Rešenje 2.12

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

#define MAX 32

/* Funkcija ucitava elemente kvadratne matrice 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 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");
  }
}
```

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

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

40     /* Izracunavamo skalarni proizvod vrste sa samom sobom */
       proizvod = 0;
42
       for (j = 0; j < n; j++)
44       proizvod += m[i][j] * m[i][j];

46     /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
          zakljucujemo da matrica nije normirana */
48     if (proizvod != 1)
         return 0;
50   }

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

60       for (k = 0; k < n; k++)
           proizvod += m[i][k] * m[j][k];
62
         /* Ako proizvod dve bilo koje razlicite vrste nije jednak
64          nuli, odmah zakljucujemo da matrica nije ortogonalna */
         if (proizvod != 0)
66         return 0;
       }
68   }

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

74 int main()
   {
76   int A[MAX][MAX];
     int n;
78
     /* Ucitavamo vrednost dimenzije i proveravamo njenu korektnost
80    */
```

```
  scanf("%d", &n);

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

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

  /* Ispisujemo rezultat rada funkcije */
  if (ortonormirana(A, n))
    printf("da\n");
  else
    printf("ne\n");

  return 0;
}
```

Rešenje 2.13

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

#define MAX_V 10
#define MAX_K 10

/* Funkcija proverava da li su ispisani svi elementi iz matrice,
   odnosno da li se narusio prirodan poredak medju granicama */
int krajIspisa(int top, int bottom, int left, int right)
{
  return !(top <= bottom && left <= right);
}

/* Funkcija spiralno ispisuje elemente matrice */
void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
{
  int i, j, top, bottom, left, right;

  top = left = 0;
  bottom = n - 1;
  right = m - 1;

  while (!krajIspisa(top, bottom, left, right)) {
    /* Ispisuje se prvi red */
    for (j = left; j <= right; j++)
      printf("%d ", a[top][j]);

    /* Spustamo prvi red */
    top++;
```

```
30
      if (krajIspisa(top, bottom, left, right))
32      break;

34    for (i = top; i <= bottom; i++)
        printf("%d ", a[i][right]);
36
      /* Pomeramo desnu kolonu za naredni krug ispisa blize levom
38       kraju */
      right--;
40
      if (krajIspisa(top, bottom, left, right))
42      break;

44    /* Ispisujemo donju vrstu */
      for (j = right; j >= left; j--)
46      printf("%d ", a[bottom][j]);

48    /* Podizemo donju vrstu za naredni krug ispisa */
      bottom--;
50
      if (krajIspisa(top, bottom, left, right))
52      break;

54    /* Ispisujemo prvu kolonu */
      for (i = bottom; i >= top; i--)
56      printf("%d ", a[i][left]);

58    /* Pripremamo levu kolonu za naredni krug ispisa */
      left++;
60  }
  putchar('\n');
62}

64void ucitaj_matricu(int a[][MAX_K], int n, int m)
{
66  int i, j;

68  for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
70      scanf("%d", &a[i][j]);
}
72
int main()
74{
  int a[MAX_V][MAX_K];
76  int m, n;

78  /* Ucitaj broj vrsta i broj kolona matrice */
  scanf("%d", &n);
80  scanf("%d", &m);
```

```
82    if (n > MAX_V || n <= 0 || m > MAX_K || m <= 0) {
        fprintf(stderr, "Greska: neodgovarajuce dimenzije ");
84      fprintf(stderr, "matrice.\n");
        exit(EXIT_FAILURE);
86    }

88    ucitaj_matricu(a, n, m);
      ispisi_matricu_spiralno(a, n, m);
90
      return 0;
92  }
```

Rešenje 2.14

Rešenje 2.15

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

4 /* NAPOMENA: Primer demonstrira dinamicku alokaciju niza od n
     elemenata. Dovoljno je alocirati n * sizeof(T) bajtova, gde
6    je T tip elemenata niza. Povratnu adresu malloc()-a treba
     pretvoriti iz void * u T *, kako bismo dobili pokazivac koji
8    pokazuje na prvi element niza tipa T. Na dalje se elementima
     moze pristupati na isti nacin kao da nam je dato ime niza
10   (koje se tako i ponasa - kao pokazivac na element tipa T koji
     je prvi u nizu) */
12 int main()
  {
14   int *p = NULL;
     int i, n;
16
     /* Unosimo dimenziju niza. Ova vrednost nije ogranicena bilo
18      kakvom konstantom, kao sto je to ranije bio slucaj kod
        staticke alokacije gde je dimenzija niza bila unapred
20      ogranicena definisanim prostorom. */
     scanf("%d", &n);
22
     /* Alociramo prostor za n celih brojeva */
24   if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
       fprintf(stderr, "malloc(): ");
26     fprintf(stderr, "greska pri alokaciji memorije.\n");
       exit(EXIT_FAILURE);
28   }

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

34   /* Unosimo elemente niza */
```

```
  for (i = 0; i < n; i++)
    scanf("%d", &p[i]);

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

  /* Oslobadjamo prostor */
  free(p);

  return 0;
}
```

Rešenje 2.16

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

int main(void)
{
  /* 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;

  /* Inicijalizacija */
  alocirano = n = 0;

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

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

    /* Ako broj ucitanih elemenata niza odgovara broju
       alociranih mesta, za smestanje novog elementa treba
       obezbediti dodatni prostor. Da se ne bi za svaki sledeci
       element pojedinacno alocirala memorija, prilikom
       alokacije se vrsi rezervacija za jos KORAK dodatnih mesta
       za buduce elemente */
```

```
36      if (n == alocirano) {
          /* Povecava se broj alociranih mesta */
38        alocirano = alocirano + KORAK;

40        /* Vrsi se realokacija memorije sa novom velicinom */
          /*******************************************************/
42        /* Resenje sa funkcijom malloc() */
          /*******************************************************/
44        /* Vrsi se alokacija memorije sa novom velicinom, a adresa
             pocetka novog memorijskog bloka se cuva u promenljivoj
46           b */
          b = (int *) malloc(alocirano * sizeof(int));
48
          /* Ako prilikom alokacije dodje do neke greske */
50        if (b == NULL) {
            /* poruku ispisujemo na izlaz za greske */
52          fprintf(stderr, "malloc(): ");
            fprintf(stderr, "greska pri alokaciji memorije.\n");
54
            /* Pre kraja programa moramo svu dinamicki alociranu
56             memoriju da oslobodimo. U ovom slucaju samo memoriju
               na adresi a */
58          free(a);

60          /* Zavrsavamo program */
            exit(EXIT_FAILURE);
62        }

64        /* Svih n elemenata koji pocinju na adresi a prepisujemo
             na novu aderesu b */
66        for (i = 0; i < n; i++)
            b[i] = a[i];
68
          /* Posle prepisivanja oslobadjamo blok memorije sa
70           pocetnom adresom u a */
          free(a);
72
          /* Promenljivoj a dodeljujemo adresu pocetka novog, veceg
74           bloka koji je prilikom alokacije zapamcen u
             promenljivoj b */
76        a = b;

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

86           if(a == NULL) { fprintf(stderr, "realloc(): ");
             fprintf(stderr, "greska pri alokaciji memorije.\n");
```

```
88            exit(EXIT_FAILURE); } */
      }
90
      /* Smestamo element u niz */
92    a[n++] = x;

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

98  /* Ispisujemo brojeve u obrnutom poretku */
    for (n--; n >= 0; n--)
100   printf("%d ", a[n]);
    printf("\n");
102
    /* Oslobadjamo dinamicki alociranu memoriju */
104 free(a);

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

Rešenje 2.17

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 1000
6
/* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8    Ovako nesto se moze improvizovati tako sto se u funkciji
   dinamicki kreira niz potrebne velicine, popuni se potrebnim
10   informacijama, a zatim se vrati njegova adresa. Imajuci u
   vidu cinjenicu da dinamicki kreiran objekat ne nestaje kada
12   se izadje iz funkcije koja ga je kreirala, vraceni pokazivac
   se kasnije u pozivajucoj funkciji moze koristiti za pristup
14   "vracenom" nizu. Medjutim, pozivajuca funkcija ima
   odgovornost i da se brine o dealokaciji istog prostora */
16
/* Funkcija dinamicki kreira niz karaktera u koji smesta
18   rezultat nadovezivanja niski. Adresa niza se vraca kao
   povratna vrednost. */
20 char *nadovezi(char *s, char *t)
{
22   /* Dinamicki kreiramo prostor dovoljne velicine */
   char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
24                             * sizeof(char));

26   /* Proveravamo uspeh alokacije */
   if (p == NULL) {
```

```
28      fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
30      exit(EXIT_FAILURE);
    }
32
    /* Kopiramo i nadovezujemo stringove */
34
    /* Resenje bez koriscenja biblioteckih funkcija */
36  /*
       int i,j; for(i=j=0; s[j]!='\0'; i++, j++) p[i]=s[j];
38
       for(j=0; t[j]!='\0'; i++, j++) p[i]=t[j];
40
       p[i]='\0'; */
42
    /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
44     string.h */
    strcpy(p, s);
46  strcat(p, t);

48  /* Vracamo pokazivac p */
    return p;
50  }

52  int main()
    {
54    char *s = NULL;
      char s1[MAX], s2[MAX];
56
      /* Ucitavamo dve niske koje cemo da nadovezemo */
58    scanf("%s", s1);
      scanf("%s", s2);
60
      /* Pozivamo funkciju da nadoveze stringove */
62    s = nadovezi(s1, s2);

64    /* Prikazujemo rezultat */
      printf("%s\n", s);
66
      /* Oslobadjamo memoriju alociranu u funkciji nadovezi() */
68    free(s);

70    return 0;
    }
```

Rešenje 2.18

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

```c
int main()
{
  int i, j;

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

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

  /* Trag matice */
  double trag = 0;

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

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

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

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

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

      exit(EXIT_FAILURE);
    }
  }

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

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

```
58    for (i = 0; i < n; i++)
        trag += A[i][i];
60
      printf("%.2f\n", trag);
62
      /* Oslobadjamo prostor rezervisan za svaku vrstu */
64    for (j = 0; j < n; j++)
        free(A[j]);
66
      /* Oslobadjamo memoriju za niz pokazivaca na vrste */
68    free(A);

70    return 0;
  }
```

Rešenje 2.19

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

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

9    /* Popunjavamo matricu vrstu po vrstu */
     for (i = 0; i < n; i++)
11     /* Popunjavamo i-tu vrstu matrice */
       for (j = 0; j < m; j++)
13       scanf("%d", &M[i][j]);
   }
15
   void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
17 {
     int i, j;
19
     for (i = 0; i < n; i++) {
21     /* Ispisujemo elemente ispod glavne dijagonale matrice */
       for (j = 0; j <= i; j++)
23       printf("%d ", M[i][j]);
       printf("\n");
25   }
   }
27
   int main()
29 {
     int m, n, i, j;
31   int **matrica = NULL;

33   /* Unosimo dimenzije matrice */
```

```
  scanf("%d %d", &n, &m);

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

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

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

  ucitaj_matricu(matrica, n, m);

  ispisi_elemente_ispod_dijagonale(matrica, n, m);

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

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

  return 0;
}
```

Rešenje 2.20

```
#include<stdio.h>

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

Rešenje 2.21

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

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

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

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

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

```
52 }

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

60   for (i = 0; i < n; i++) {
       for (j = 0; j < n; j++)
62       printf("%.2f ", m[i][j]);
       printf("\n");
64   }
   }
66
   /* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
68 float **alociraj_memoriju(int n)
   {
70   int i, j;
     float **m;
72
     m = (float **) malloc(n * sizeof(float *));
74   if (m == NULL) {
       fprintf(stderr, "malloc(): Neuspela alokacija\n");
76     exit(EXIT_FAILURE);
     }
78
     /* Za svaku vrstu matrice */
80   for (i = 0; i < n; i++) {
       /* Alociramo memoriju */
82     m[i] = (float *) malloc(n * sizeof(float));

84     /* Proveravamo da li je doslo do greske pri alokaciji */
       if (m[i] == NULL) {
86       /* Ako jeste, ispisujemo poruku */
         printf("malloc(): neuspela alokacija memorije!\n");
88
         /* Oslobadjamo memoriju zauzetu do ovog koraka */
90       for (j = 0; j < i; j++)
           free(m[i]);
92       free(m);
         exit(EXIT_FAILURE);
94     }
     }
96   return m;
   }
98
   /* Funckija oslobadja memoriju zauzetu kvadratnom matricom
100    dimenzije n */
   void oslobodi_memoriju(float **m, int n)
102 {
     int i;
```

```
104
    for (i = 0; i < n; i++)
106   free(m[i]);
    free(m);
108 }

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

116   /* Ako korisnik nije uneo trazene argumente, prijavljujemo
        gresku */
118   if (argc < 2) {
        printf("Greska: ");
120     printf("Nedovoljan broj argumenata komandne linije.\n");
        printf("Program se poziva sa %s ime_dat.\n", argv[0]);
122     exit(EXIT_FAILURE);
    }
124
    /* Otvaramo datoteku za citanje */
126 ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
128   fprintf(stderr, "Greska: ");
      fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
130             argv[1]);
      exit(EXIT_FAILURE);
132 }

134 /* citamo dimenziju matrice */
    fscanf(ulaz, "%d", &n);
136
    /* Alociramo memoriju */
138 a = alociraj_memoriju(n);

140 /* Ucitavamo elemente matrice */
    ucitaj_matricu(ulaz, a, n);
142
    float zbir = zbir_ispod_sporedne_dijagonale(a, n);
144
    /* Pozivamo funkciju za modifikovanje elemenata */
146 izmeni(a, n);

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

152 printf("Transformisana matrica je:\n");
    ispisi_matricu(a, n);
154
    /* Oslobadjamo memoriju */
```

```
156   oslobodi_memoriju(a, n);

158   /* Zatvaramo datoteku */
      fclose(ulaz);
160
      /* i prekidamo sa izvrsavanjem programa */
162   return 0;
}
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

```
2  #include <stdio.h>
   #include <stdlib.h>
4  #include <math.h>
   #include <string.h>
6
   /* NAPOMENA: Zaglavlje math.h sadrzi deklaracije raznih
8     matematickih funkcija. đIzmeu ostalog, to su ćsledee
      funkcije: double sin(double x); double cos(double x); double
10    tan(double x); double asin(double x); double acos(double x);
      double atan(double x); double atan2(double y, double x);
12    double sinh(double x); double cosh(double x); double
      tanh(double x); double exp(double x); double log(double x);
14    double log10(double x); double pow(double x, double y);
      double sqrt(double x); double ceil(double x); double
16    floor(double x); double fabs(double x); */

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

28   printf("------------------------\n");
```

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

36/* Umesto da koristimo stepenu funkciju iz zaglavlja math.h ->
   pow(a,2) ćpozivaemo čkorisniku sqr(a) */
38double sqr(double a)
{
40  return a * a;
}
42
int main(int argc, char *argv[])
44{
  double a, b;
46  int n;
  /* Imena funkicja koja ćemo navoditi su ćkraa ili čtano
48     duga 5 karaktera */
  char ime_fje[6];
50  /* Pokazivac na funkciju koja ima jedan argument tipa double i
     povratnu vrednost istog tipa */
52  double (*fp) (double);

54  /* Ako korisnik nije uneo žtraene argumente, prijavljujemo
     šgreku */
56  if (argc < 2) {
      printf("Greska: ");
58    printf("Nedovoljan broj argumenata komandne linije.\n");
      printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
60           argv[0]);
      exit(EXIT_FAILURE);
62  }

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

68  /* Inicijalizujemo čpokaziva na funkciju koja treba da se
     tabelira */
70  if (strcmp(ime_fje, "sin") == 0)
      fp = &sin;
72  else if (strcmp(ime_fje, "cos") == 0)
      fp = &cos;
74  else if (strcmp(ime_fje, "tan") == 0)
      fp = &tan;
76  else if (strcmp(ime_fje, "atan") == 0)
      fp = &atan;
78  else if (strcmp(ime_fje, "acos") == 0)
      fp = &acos;
80  else if (strcmp(ime_fje, "asin") == 0)
```

```
    fp = &asin;
  else if (strcmp(ime_fje, "exp") == 0)
    fp = &exp;
  else if (strcmp(ime_fje, "log") == 0)
    fp = &log;
  else if (strcmp(ime_fje, "log10") == 0)
    fp = &log10;
  else if (strcmp(ime_fje, "sqrt") == 0)
    fp = &sqrt;
  else if (strcmp(ime_fje, "floor") == 0)
    fp = &floor;
  else if (strcmp(ime_fje, "ceil") == 0)
    fp = &ceil;
  else if (strcmp(ime_fje, "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 čukljuuje bar krajeve intervala, tako da se
     mora uneti broj veci od 2 */
  if (n < 2) {
    fprintf(stderr, "Broj čtaaka žmree mora biti bar 2!\n");
    exit(EXIT_FAILURE);
  }

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.27

Rešenje 2.28

Rešenje 2.29

Glava 3

Algoritmi pretrage i sortiranja

3.1 Pretraživanje

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

(a) Napisati funkciju koja vrši linearnu pretragu niza celih brojeva `a`, dužine `n`, tražeći u njemu broj `x`.

(b) Napisati funkciju koja vrši binarnu pretragu sortiranog niza `a`, dužine `n`, tražeći u njemu broj `x`.

(c) Napisati funkciju koja vrši interpoacionu pretragu sortiranog niza `a`, dužine `n`, tražeći u njemu broj `x`.

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

Test 1

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

Izlaz:
 Linearna pretraga
 Element nije u nizu
 ---------------------------------
 Binarna pretraga
 Element nije u nizu
 ---------------------------------
 Interpolaciona pretraga
 Element nije u nizu
 ---------------------------------
```

Test Test 2

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

Izlaz:
 Linearna pretraga
 Element nije u nizu
 ---------------------------------
 Binarna pretraga
 Element nije u nizu
 ---------------------------------
 Interpolaciona pretraga
 Element nije u nizu
 ---------------------------------
```

[Rešenje 3.1]

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

Primer 1

```
Interakcija programa:
 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 programa:
 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.
```

[Rešenje 3.2]

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

Primer 1

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

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 programa:
  Unesite indeks studenta cije informacije zelite: 20140076
  Indeks: 20140076, Ime i prezime: Sonja Stevanovic
  Unesite prezime studenta cije informacije zelite: Popovic
  Indeks: 20140032, Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

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

[Rešenje 3.4]

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

Test 1

```
Poziv: ./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

```
Poziv: ./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

```
Poziv: ./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
```

[Rešenje 3.5]

Zadatak 3.6 Napisati funkciju koja određuje nulu funkcije `cos(x)` na intervalu `[0,2]` metodom polovljenja intervala. Algoritam se završava kada se

vrednost kosinusne funkcije razlikuje za najviše 0.001 od nule. UPUTSTVO: *Koristiti algoritam analogan algoritmu binarne pretrage.*

Test 1

```
IZLAZ:
 1.57031
```

[Rešenje 3.6]

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

Test 1

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

IZLAZ:
 2
```

Test 2

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

IZLAZ:
 -1
```

Test 3

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

IZLAZ:
 3
```

[Rešenje 3.7]

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

Test 1

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

IZLAZ:
 3
```

Test 2

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

IZLAZ:
 4
```

Test 3

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

IZLAZ:
 -1
```

[Rešenje 3.8]

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

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

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

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

Test 1

```
Ulaz:
 4

Izlaz:
 2 2
```

Test 2

```
Ulaz:
 17

Izlaz:
 4 4
```

Test 3

```
Ulaz:
 1031

Izlaz:
 10 10
```

[Rešenje 3.9]

**** Zadatak 3.10** U prvom kvadrantu dato je $1 \leq \texttt{N} \leq 10000$ 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 programa:
 Unesi broj tacaka: 2
 Unesi koordinate tacaka:
 2 0 2 1
 1 2 2 2
 26.57
```

Primer 2

```
Interakcija programa:
 Unesi broj tacaka: 2
 Unesi koordinate tacaka:
 1 0 1 1
 0 1 1 1
 45
```

Primer 3

```
Interakcija programa:
 Unesi broj tacaka: 3
 Unesi koordinate tacaka:
 1 0 1 1
 2 0 2 1
 1 2 2 2
 26.57
```

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

```
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:
 A

Izlaz:
 Mika Antic
```

3.2 Sortiranje

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

Test 1

```
Ulaz:
 23 64 123 76 22 7

Izlaz:
 1
```

Test 2

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

Izlaz:
 0
```

Test 3

```
Ulaz:
 34 30

Izlaz:
 4
```

[Rešenje 3.12]

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

Primer 1

```
Interakcija programa:
 Unesite prvu nisku  anagram
 Unesite drugu nisku  ramgana
 jesu
```

Primer 2

```
Interakcija programa:
 Unesite prvu nisku  anagram
 Unesite drugu nisku  anagrm
 nisu
```

Primer 3

```
Interakcija programa:
 Unesite prvu nisku  test
 Unesite drugu nisku  tset
 jesu
```

[Rešenje 3.13]

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

Test 1

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

IZLAZ:
 4
```

Test 2

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

IZLAZ:
 2
```

Test 3

```
ULAZ:
 123

IZLAZ:
 123
```

[Rešenje 3.14]

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

Primer 1

```
INTERAKCIJA PROGRAMA:
 Unesite trazeni zbir: 34
 Unesite elemente niza:
 134 4 1 6 30 23
 da
```

Primer 2

```
INTERAKCIJA PROGRAMA:
 Unesite trazeni zbir: 12
 Unesite elemente niza:
 53 1 43 3 56 13
 ne
```

Primer 3

```
INTERAKCIJA PROGRAMA:
 Unesite trazeni zbir: 52
 Unesite elemente niza:
 52
 ne
```

[Rešenje 3.15]

Zadatak 3.16 Napraviti biblioteku `sort.h` i `sort.c` koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži `selection`, `merge`, `quick`, `bubble`, `insertion` i `shell sort`. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije `n`.

Test 1

```
Poziv: time a.out 100000 -i -o

Izlaz:
 real 0m17.631s
 user 0m17.604s
 sys 0m0.000s
```

Test 2

```
Poziv: time a.out 100000 -b -r

Izlaz:
 real 0m0.005s
 user 0m0.004s
 sys 0m0.000s
```

[Rešenje 3.16]

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

Primer 1

```
Interakcija programa:
 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 programa:
 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
```

[Rešenje 3.17]

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

Test 1

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

PRVI-DEO.TXT
  Andrija Petrovic
  Anja Ilic
  Ivana Markovic
  Lazar Micic
  Nenad Brankovic
  Sofija Filipovic
  Vladimir Savic
  Uros Milic

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 Stankovic
  Ivana Markovic
  Lazar Micic
  Marija Stankovic
  Nenad Brankovic
  Ognjen Peric
  Sofija Filipovic
  Uros Milic
  Vladimir Savic
```

[Rešenje 3.18]

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

(a) njihovog rastojanja od koordinatnog početka,

(b) `x` koordinata tačaka,

(c) `y` koordinata tačaka.

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

Test 1

```
Poziv: ./a.out -x in.txt out.txt

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

```
Poziv: ./a.out -o in.txt out.txt

IN.TXT
 3 4
 11 6
 7 3
 2 82
 -1 6

OUT.TXT
 3 4
 -1 6
 7 3
 11 6
 2 82
```

[Rešenje 3.19]

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

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

```
BIRACKI-SPISAK.TXT
 [datoteka ne postoji]

Izlaz:
 Problem pri otvaranju
 datoteke.
```

[Rešenje 3.20]

Zadatak 3.21 Definisana je struktura podataka

```
typedef struct dete
{
      char ime[MAX_IME];
      char prezime[MAX_IME];
```

```
    unsigned godiste;
} Dete;
```

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

Test 1

```
POZIV: ./a.out in.txt out.txt

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

```
POZIV: ./a.out in.txt out.txt

IN.OUT
 Milijana Maric 2009

OUT.TXT
 Milijana Maric 2009
```

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

Test 1

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

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

[Rešenje 3.22]

Zadatak 3.23 Napisati program koji simulira rad kase u prodavnici. Kupci prilaze kasi, a prodavac unošenjem bar-koda kupljenog proizvoda dodaje njegovu cenu na ukupan račun. Na kraju, program ispisuje ukupnu vrednost svih proi-

zvoda. 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 PROGRAMA:
 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!
```

[Rešenje 3.23]

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

kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po 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
```

[Rešenje 3.24]

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

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

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

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

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

Test 1

```
Poziv: ./a.out

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

```
Poziv: ./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

```
Poziv: ./a.out -n

pesme.txt
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

Izlaz:
  Mika - Kisa, 5341
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
```

[Rešenje 3.25]

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

Primer 1

```
Interakcija programa:
  Unesi niz operacija: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
  0 2 8 2 6
```

**** Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog

okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze 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
```

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.28 Napisati program koji ilustruje upotrebu bibilioteckih 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 programa:
 Uneti dimenziju niza: 10
 Uneti elemente niza:
 5 3 1 6 8 90 34 5 3 432
 Sortirani niz u rastucem poretku:
 1 3 3 5 5 6 8 34 90 432
 Uneti element koji se trazi u nizu: 34
 Binarna pretraga:
 Element je nadjen na poziciji 7
 Linearna pretraga (lfind):
 Element je nadjen na poziciji 7
```

Primer 2

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

[Rešenje 3.28]

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

Primer 1

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

[Rešenje 3.29]

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

(a) leksikografski,

(b) po dužini.

Niske se učitavaju iz fajla `niske.txt`, neće ih biti više od 1000 i svaka će biti dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (`bsearch`) zarad traženja niske unete sa standardnog ulaza, a potom linearnu pretragu koristeći funkciju `lfind`. Na kraju, niske bivaju sortirane po dužini. Rezultate svih sortiranja i pretraga ispisati na standardnom izlazu.

Primer 1

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

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

[Rešenje 3.30]

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

[Rešenje 3.31]

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

Primer 1

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

Ulazna datoteka (kolokvijum.txt):
 Aleksandra Cvetic 15
 Bojan Golubovic 30
 Dragan Markovic 25
 Filip Dukic 20
 Ivana Stankovic 25
 Marija Stankovic 15
 Ognjen Peric 20
 Uros Milic 10
 Andrija Petrovic 0
 Anja Ilic 5
 Ivana Markovic 5
 Lazar Micic 20
 Nenad Brankovic 15
```

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

[Rešenje 3.32]

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

[Rešenje 3.33]

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

Primer 1

```
Interakcija programa:
 Unesite broj niski: 4
 Unesite niske:
 prog search sort search
 ima
```

Primer 2

```
Interakcija programa:
 Unesite broj niski: 3
 Unesite niske:
 test kol ispit
 nema
```

Primer 3

```
Interakcija programa:
 Unesite broj niski: 5
 Unesite niske:
 a ab abc abcd abcde
 nema
```

[Rešenje 3.34]

Zadatak 3.35 Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr97125`, `mm09001`), ime i prezime i broj poena. I ime i prezime neće biti duže od 20 karaktera. Napisati

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

Test 1

```
Poziv: ./a.out -n mm13321

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

```
Poziv: /a.out -p

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

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

[Rešenje 3.35]

Zadatak 3.36 Definisana je struktura:

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

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

Primer 1

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

DATOTEKA.TXT
 1.1.2013.
 13.12.2016.
 11.11.2011.
 3.5.2015.
 5.2.2009.
```

```
Interakcija programa:
 Unesi sledeci datum: 13.12.2016.
 postoji
 Unesi sledeci datum: 10.5.2015.
 ne postoji
 Unesi sledeci datum: 5.2.2009.
 postoji
```

Zadatak 3.37 Za zadatu celobrojnu matricu dimenzije $n \times m$ napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrsti. Napisati 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.

Test 1

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

Test 2

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

[Rešenje 3.37]

Zadatak 3.38 Za 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.

Primer 1

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

Primer 2

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

3.4 Rešenja

Rešenje 3.1

```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() */

/* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
   njemu element x. Pretraga se vrsi prostom iteracijom kroz niz. Ako
   se element pronadje funkcija vraca indeks pozicije na kojoj je
   pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
   pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
   pretrage. */
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;
}

/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
   pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
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 levoj polovini niza */
    if (x < a[srednji])
      desni = srednji - 1;
    /* Ako je element sa sredisnjim indeksom manji od x, tada se x
       mora nalaziti u desnoj polovini 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;
}
```

```
52 /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
      pozicije nadjenog elementa ili -1, ako element nije pronadjen */
54 int interpolaciona_pretraga(int a[], int n, int x)
   {
56   int levi = 0;
     int desni = n - 1;
58   int srednji;
     /* Dokle god je indeks levi levo od indeksa desni...  */
60   while (levi <= desni) {
       /* Ako je trazeni element manji od pocetnog ili veci od
62        poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
          nije u tom delu niza. Ova provera je neophodna, da se ne bi
64        dogodilo da se prilikom izracunavanja indeksa srednji izadje
          izvan opsega indeksa [levi,desni] */
66     if (x < a[levi] || x > a[desni])
         return -1;
68     /* 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
70        jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
          desni). Ova provera je neophodna, jer bi se u suprotnom
72        prilikom izracunavanja indeksa srednji pojavilo deljenje
          nulom. */
74     else if (a[levi] == a[desni])
         return levi;
76     /* Racunanje srednjeg indeksa */
       srednji =
78         levi +
           ((double) (x - a[levi]) / (a[desni] - a[levi])) *
80         (desni - levi);
       /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
82        verovatno biti blize trazenoj vrednosti nego da je prosto uvek
          uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
84        porediti sa pretragom recnika: ako neko trazi rec na slovo 'B
     ',
          sigurno nece da otvori recnik na polovini, vec verovatno negde
86        blize pocetku. */
       /* Ako je element sa indeksom srednji veci od trazenog, tada se
88        trazeni element mora nalaziti u levoj polovini niza */
       if (x < a[srednji])
90       desni = srednji - 1;
       /* Ako je element sa indeksom srednji manji od trazenog, tada se
92        trazeni element mora nalaziti u desnoj polovini niza */
       else if (x > a[srednji])
94       levi = srednji + 1;
       else
96       /* Ako je element sa indeksom srednji jednak trazenom, onda se
            pretraga zavrsava na poziciji srednji */
98       return srednji;
   }
100  /* U slucaju neuspesne pretrage vraca se -1 */
     return -1;
102 }
```

```
104 /* Funkcija main */
    int main(int argc, char **argv)
106 {
      int a[MAX];
108   int n, i, x;
      struct timespec time1, time2, time3, time4, time5, time6;
110   FILE *f;

112   /* Provera argumenata komandne linije */
      if (argc != 3) {
114     fprintf(stderr,
                "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
        ;
116     exit(EXIT_FAILURE);
      }
118
      /* Dimenzija niza */
120   n = atoi(argv[1]);
      if (n > MAX || n <= 0) {
122     fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
        exit(EXIT_FAILURE);
124   }

126   /* Broj koji se trazi */
      x = atoi(argv[2]);
128
      /* Elementi niza se generisu slucajno, tako da je svaki sledeci
130       veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
          pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
132       izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
          godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
134   srandom(time(NULL));
      for (i = 0; i < n; i++)
136     a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;

138   /* Lineara pretraga */
      printf("Linearna pretraga\n");
140   /* Vreme proteklo od Nove godine 1970 */
      clock_gettime(CLOCK_REALTIME, &time1);
142   i = linearna_pretraga(a, n, x);
      /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
144       linearnu pretragu */
      clock_gettime(CLOCK_REALTIME, &time2);
146   if (i == -1)
        printf("Element nije u nizu\n");
148   else
        printf("Element je u nizu na poziciji %d\n", i);
150   printf("---------------------------------\n");

152   /* Binarna pretraga */
      printf("Binarna pretraga\n");
```

```
  clock_gettime(CLOCK_REALTIME, &time3);
  i = binarna_pretraga(a, n, x);
  clock_gettime(CLOCK_REALTIME, &time4);
  if (i == -1)
    printf("Element nije u nizu\n");
  else
    printf("Element je u nizu na poziciji %d\n", i);
  printf("---------------------------------\n");

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

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

  fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
  fprintf(f, "\tLinearna pretraga:%10ld ns\n",
          (time2.tv_sec - time1.tv_sec) * 1000000000 +
          time2.tv_nsec - time1.tv_nsec);
  fprintf(f, "\tBinarna: %19ld ns\n",
          (time4.tv_sec - time3.tv_sec) * 1000000000 +
          time4.tv_nsec - time3.tv_nsec);
  fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
          (time6.tv_sec - time5.tv_sec) * 1000000000 +
          time6.tv_nsec - time5.tv_nsec);

  /* Zatvaranje datoteke */
  fclose(f);

  return 0;
}
```

Rešenje 3.2

```
#include <stdio.h>

int lin_pretgraga_rek_sufiks(int a[], int n, int x)
{
  int tmp;
  /* Izlaz iz rekurzije */
```

```
  if (n <= 0)
    return -1;
  /* Ako je prvi element trazeni */
  if (a[0] == x)
    return 0;
  /* Pretraga ostatka niza */
  tmp = lin_pretgraga_rek_sufiks(a + 1, n - 1, x);
  return tmp < 0 ? tmp : tmp + 1;
}

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

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


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

```
  if (a[p] == x)
    return p;
  if (a[p] < x)
    return interp_pretgraga_rek(a, p + 1, d, x);
  else
    return interp_pretgraga_rek(a, l, p - 1, x);
}

#define MAX 1024

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

  /* Ucitavanje trazenog broja */
  printf("Unesite trazeni broj: ");
  scanf("%d", &x);

  /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
     korisnik pritisne CTRL+D za naznaku kraja */
  i = 0;
  printf("Unesite sortiran niz elemenata: ");
  while (scanf("%d", &a[i]) == 1) {
    i++;
  }

  /* Linearna pretraga */
  printf("Linearna pretraga\n");
  indeks = lin_pretgraga_rek_sufiks(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 = bin_pretgraga_rek(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 = interp_pretgraga_rek(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
   indeksu se vrsi binarno, dok pretraga po prezimenu mora linearno,
   jer nema garancije da postoji uredjenje po prezimenu. */

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

```
47    }
      /* Ako nije pronadjen, vraca se -1 */
49    return -1;
}
51
/* Linearnom pretragom niza studenata trazi se prezime x */
53 int linearna_pretraga(Student a[], int n, char x[])
{
55    int i;
      for (i = 0; i < n; i++)
57      /* Poredjenje prezimena i-tog studenta i poslatog x */
        if (strcmp(a[i].prezime, x) == 0)
59        return i;
      return -1;
61 }

63 /* Main funkcija mora imati argumente jer se ime datoteke prosledjuje
       kao argument komandne linije */
65 int main(int argc, char *argv[])
{
67    Student dosije[MAX_STUDENATA];
      FILE *fin = NULL;
69    int i;
      int br_studenata = 0;
71    long trazen_indeks = 0;
      char trazeno_prezime[MAX_DUZINA];
73
      /* Provera da li je korisnik prilikom poziva programa prosledio ime
75       datoteke sa informacijama o studentima */
      if (argc != 2) {
77      fprintf(stderr,
                "Greska: Program se poziva sa %s ime_datoteke\n",
79              argv[0]);
        exit(EXIT_FAILURE);
81    }

83    /* Otvaranje datoteke */
      fin = fopen(argv[1], "r");
85    if (fin == NULL) {
        fprintf(stderr,
87              "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
        exit(EXIT_FAILURE);
89    }

91    /* Citanje se vrsi sve dok postoji red sa informacijama o studentu
        */
      i = 0;
93    while (1) {
        if (i == MAX_STUDENATA)
95        break;
        if (fscanf
97          (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
```

```
          dosije[i].prezime) != 3)
99        break;
      i++;
101   }
    br_studenata = i;
103
    /* Nakon citanja, datoteka vise nije neophodna i odmah se zatvara
       */
105 fclose(fin);

107 /* Unos indeksa koji se binarno trazi u nizu */
    printf("Unesite indeks studenta cije informacije zelite: ");
109 scanf("%ld", &trazen_indeks);
    i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
111 /* Rezultat binarne pretrage */
    if (i == -1)
113   printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
115   printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
117
    /* Unos prezimena koje se linearno trazi u nizu */
119 printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
121 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    /* Rezultat linearne pretrage */
123 if (i == -1)
      printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
125 else
      printf("Indeks: %ld, Ime i prezime: %s %s\n",
127           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

129 return 0;
}
```

Rešenje 3.4

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

8 typedef struct {
    long indeks;
10  char ime[MAX_DUZINA];
    char prezime[MAX_DUZINA];
12 } Student;

14 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;
  /* Da li je srednji bas onaj trazeni */
  if (a[srednji].indeks == x) {
    return srednji;
  }
  /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
     poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
     je poznato da je niz sortiran po indeksu u rastucem poretku. */
  if (x < a[srednji].indeks)
    return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
  /* Inace ga treba traziti u desnoj polovini */
  else
    return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
}

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

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

/* Main funkcija mora imate argumente jer se ime datoteke prosledjuje
   kao argument komandne linije */
int main(int argc, char *argv[])
```

```
{
  Student dosije[MAX_STUDENATA];
  FILE *fin = NULL;
  int i;
  int br_studenata = 0;
  long trazen_indeks = 0;
  char trazeno_prezime[MAX_DUZINA];

  /* Provera da li je korisnik prilikom poziva prosledio ime datoteke
     sa informacijama o studentima */
  if (argc != 2) {
    fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

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

  /* Citanje se vrsi sve dok postoji sledeci red sa informacijama o
     studentu */
  i = 0;
  while (1) {
    if (i == MAX_STUDENATA)
      break;
    if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
         dosije[i].prezime) != 3)
      break;
    i++;
  }
  br_studenata = i;

  /* Nakon citanja datoteka nam vise nije neophodna i odmah se
     zatvara */
  fclose(fin);

  /* Unos indeksa koji se binarno trazi u nizu */
  printf("Unesite indeks studenta cije informacije zelite: ");
  scanf("%ld", &trazen_indeks);
  i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
                                  trazen_indeks);
  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);

  /* Unos prezimena koje se linearno trazi u nizu */
  printf("Unesite prezime studenta cije informacije zelite: ");
  scanf("%s", trazeno_prezime);
  i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
                                      trazeno_prezime);
  if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
  else
    printf
        ("Prvi takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

  return 0;
}
```

Rešenje 3.5

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

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

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

/* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u nizu
   zadatih tacaka t dimenzije n */
Tacka najbliza_koordinatnom(Tacka t[], int n)
{
  Tacka najbliza;
  int i;
  najbliza = t[0];
  for (i = 1; i < n; i++) {
    if (rastojanje(t[i]) < rastojanje(najbliza)) {
      najbliza = t[i];
    }
  }
  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     return 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      return 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");
      return EXIT_FAILURE;
117 }

119 /* Stampanje koordinata trazene tacke */
    printf("%g %g\n", najbliza.x, najbliza.y);
121
    /* Zatvaranje datoteke */
123 fclose(ulaz);

125 return 0;
}
```

Rešenje 3.6

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

4 /* Tacnost */
  #define EPS 0.001
6
```

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

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

  /* Sve dok se ne pronadje trazena vrednost argumenta */
  while (1) {
    /* Polovi se interval */
    s = (l + d) / 2;
    /* Ako je 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 vrednost trazene tacke */
  printf("%g\n", s);

  return 0;
}
```

Rešenje 3.7

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

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 */
    else
      d = s - 1;
  }
  return -1;
}

#define MAX 256

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

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

  /* Stampanje rezultata */
  printf("%d\n", prvi_veci_od_nule(niz, n));

  return 0;
}
```

Rešenje 3.8

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

int prvi_manji_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 manji od nule, a eventualni njegov
       prethodnik veci ili jednak nuli, pretraga se zavrsava */
    if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
      return s;
    /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
       niza */
    if (niz[s] >= 0)
      l = s + 1;
    /* A inace leva */
```

```
    else
      d = s - 1;
  }
  return -1;
}

#define MAX 256

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

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

  /* Stampanje rezultata */
  printf("%d\n", prvi_manji_od_nule(niz, n));

  return 0;
}
```

Rešenje 3.9

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

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

unsigned int logaritam_b(unsigned int x)
{
  /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
     najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
     do 31 */
  int d = 0, l = sizeof(unsigned int) * 8 - 1;
  int s;
  /* Sve dok je desna granica pretrage desnije od leve */
  while (d <= l) {
    /* Racuna se sredisnja pozicija */
    s = (l + d) / 2;
    /* Proverava se da li je na toj poziciji trazena jedinica */
    if ((1 << s) <= x && (1 << (s + 1)) > x)
      return s;
```

```
    /* Pretraga desne polovine binarnog zapisa */
    if ((1 << s) > x)
      l = s - 1;
    /* Pretraga leve polovine binarnog zapisa */
    else
      d = s + 1;
  }
  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));

  return 0;
}
```

Rešenje 3.12

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

/* Iterativna verzija funkcije koja sortira niz celih brojeva,
   primenom algoritma Selection Sort */
void selectionSort(int a[], int n)
{
  int i, j;
  int min;
  int pom;

  /* U svakoj iteraciji ove petlje se pronalazi najmanji element
     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
     poziciju i, dok se element na poziciji i premesta na poziciju
     min, na kojoj se nalazio najmanji od gore 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;
    }
  }
}

/* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
   sortiranom nizu celih brojeva */
int najmanje_rastojanje(int a[], int n)
{
  int i, min;
  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++;

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

  /* Ispis rezultata */
  printf("%d\n", najmanje_rastojanje(a, i));

  return 0;
}
```

Rešenje 3.13

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

#define MAX_DIM 128
```

```c
/* 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;

  /* Ako dve niske imaju razlicit broj karaktera onda one nisu
     anagrami */
  if (strlen(s) != strlen(t))
    return 0;

  /* Sortiramo niske */
  selectionSort(s);
  selectionSort(t);

  /* Dve sortirane niske su anagrami ako i samo ako su jednake */
  for (i = 0; s[i] != '\0'; i++)
    if (s[i] != t[i])
      return 0;
  return 1;
}

int main()
{
  char s[MAX_DIM], t[MAX_DIM];

  /* Ucitavanje niski sa ulaza */
  printf("Unesite prvu nisku: ");
  scanf("%s", s);
  printf("Unesite drugu nisku: ");
  scanf("%s", t);

  /* Poziv funkcije */
  if (anagrami(s, t))
    printf("jesu\n");
```

```
58    else
        printf("nisu\n");
60
    return 0;
62  }
```

Rešenje 3.14

```
1   #include<stdio.h>
    #define MAX_DIM 256
3
    /* Funkcija za sortiranje niza */
5   void selectionSort(int s[], int n)
    {
7     int i, j, min;
      char pom;
9     for (i = 0; i < n; i++) {
        min = i;
11      for (j = i + 1; j < n; j++)
          if (s[j] < s[min])
13          min = j;
        if (min != i) {
15        pom = s[i];
          s[i] = s[min];
17        s[min] = pom;
        }
19    }
    }
21
    /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
23     najvise puta pojavio u tom nizu */
    int najvise_puta(int a[], int n)
25  {
      int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
27    /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
      for (i = 0; i < n; i = j) {
29      br_pojava = 1;
        for (j = i + 1; j < n && a[i] == a[j]; j++)
31        br_pojava++;
        /* Ispitivanje da li se do tog trenutka i-ti element pojavio
33         najvise puta u nizu */
        if (br_pojava > max_br_pojava) {
35        max_br_pojava = br_pojava;
          i_max_pojava = i;
37      }
      }
39    /* Vraca se element koji se najvise puta pojavio u nizu */
      return a[i_max_pojava];
41  }

43  int main()
```

```
{
  int a[MAX_DIM], i;

  /* Ucitavanje elemenata niza sve do kraja ulaza */
  i = 0;
  while (scanf("%d", &a[i]) != EOF)
    i++;

  /* Niz se sortira */
  selectionSort(a, i);

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

  return 0;
}
```

Rešenje 3.15

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

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

/* 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;

  /* Sortira se niz */
  selectionSort(a, n);

  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 3.16

```
/* Datoteka sort.h */
#ifndef __SORT_H__
#define __SORT_H__ 1

/* Selection sort */
void selectionsort(int a[], int n);
/* Insertion sort */
void insertionsort(int a[], int n);
/* Bubble sort */
void bubblesort(int a[], int n);
/* Shell sort */
```

```
12 void shellsort(int a[], int n);
   /* Merge sort */
14 void mergesort(int a[], int l, int r);
   /* Quick sort */
16 void quicksort(int a[], int l, int r);

18 #endif
```

```
   /* Datoteka sort.c */
2
   #include "sort.h"
4
   /* Funkcija sortira niz celih brojeva metodom sortiranja izborom.
6     Ideja algoritma je sledeca: U svakoj iteraciji pronalazi se
      najmanji element i premesta se na pocetak niza. Dakle, u prvoj
8     iteraciji, pronalazi se najmanji element, i dovodi na nulto mesto
      u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
10    pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
      dovodi na i+1 poziciju. */
12 void selectionsort(int a[], int n)
   {
14   int i, j;
     int min;
16   int pom;

18   /* U svakoj iteraciji ove petlje pronalazi se najmanji element
        medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
20      poziciju i, dok se element na pozciji i premesta na poziciju
        min, na kojoj se nalazio najmanji od gore navedenih elemenata.
        */
22   for (i = 0; i < n - 1; i++) {
       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
24        najmanji od elemenata a[i],...,a[n-1]. */
       min = i;
26     for (j = i + 1; j < n; j++)
         if (a[j] < a[min])
28         min = j;

30     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
          su (i) i min razliciti, inace je nepotrebno. */
32     if (min != i) {
         pom = a[i];
34       a[i] = a[min];
         a[min] = pom;
36     }
     }
38 }

40 /* Funkcija sortira niz celih brojeva metodom sortiranja umetanjem.
      Ideja algoritma je sledeca: neka je na pocetku i-te iteracije niz
42    prvih i elemenata (a[0],a[1],...,a[i-1]) sortirano. U i-toj
      iteraciji treba element a[i] umetnuti na pravu poziciju medju
```

```
44    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
46    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
48    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
50    dalje potiskivanje elementa u levo, poredeci ga sa njegovim novim
      levim susedom. Ovim uzastopnim premestanjem se a[i] umece na pravo
52    mesto u nizu. */
  void insertionsort(int a[], int n)
54  {
      int i, j;
56
      /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
58       sortiran */
      for (i = 1; i < n; i++) {
60
        /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
62         potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
           bude sortiran. Indeks j je trenutna pozicija na kojoj se
64         element koji se umece nalazi. Petlja se zavrsava ili kada
           element dodje do levog kraja (j==0) ili kada se naidje na
66         element a[j-1] koji je manji od a[j]. */
        for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
68        int temp = a[j];
          a[j] = a[j - 1];
70        a[j - 1] = temp;
        }
72    }
  }
74
  /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
76     algoritma je sledeca: prolazi se kroz niz redom poredeci susedne
     elemente, i pri tom ih zamenjujuci ako su u pogresnom poretku.
78     Ovim se najveci element poput mehurica istiskuje na "povrsinu",
     tj. na krajnju desnu poziciju. Nakon toga je potrebno ovaj
80     postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih n-1
     elemenata niza bez poslednjeg koji je postavljen na pravu
82     poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
     kracim prefiksima niza, cime se jedan po jedan istiskuju
84     elemenenti na svoje prave pozicije. */
  void bubblesort(int a[], int n)
86  {
      int i, j;
88    int ind;

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

92      /* Poput "mehurica" potiskuje se najveci element medju elementima
           od a[0] do a[i-1] na poziciju i-1 uporedjujuci susedne
94         elemente niza i potiskujuci veci u desno */
        for (j = 0, ind = 0; j < i - 1; j++)
```

```
96        if (a[j] > a[j + 1]) {
            int temp = a[j];
98          a[j] = a[j + 1];
            a[j + 1] = temp;
100
            /* Promenljiva ind registruje da je bilo premestanja. Samo u
102            tom slucaju ima smisla ici na sledecu iteraciju, jer ako
               nije bilo premestanja, znaci da su svi elementi vec u
104            dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
               niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
106            ispravno, ali manje efikasano, jer bi se cesto nepotrebno
               vrsila mnoga uporedjivanja, kada je vec jasno da je
108            sortiranje zavrseno. */
            ind = 1;
110       }
  }
112
  /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
114    dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
       sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
116    prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji
       od n (sto omogucuje razmenu udaljenih elemenata) i tako se dobija
118    h-sortiran niz, tj. niz u kome su elementi na rastojanju h
       sortirani, mada susedni elementi to ne moraju biti. U drugom
120    prolazu kroz isti algoritam sprovodi se isti postupak ali za manji
       korak h. Sa prolazima se nastavlja sve do koraka h = 1, u kome se
122    dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
       nacina njegovog smanjivanja menja u nekim slucajevima brzinu
124    algoritma, ali bilo koja vrednost ce rezultovati ispravnim
       sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
126    vrednost 1. */
  void shellsort(int a[], int n)
128 {
    int h = n / 2, i, j;
130 while (h > 0) {
      /* Insertion sort sa korakom h */
132   for (i = h; i < n; i++) {
        int temp = a[i];
134     j = i;
        while (j >= h && a[j - h] > temp) {
136       a[j] = a[j - h];
          j -= h;
138     }
        a[j] = temp;
140   }
      h = h / 2;
142 }
  }
144
  #define MAX 1000000
146
  /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
```

```
148    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 d je
150    jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
       dve polovine, levu i desnu, koje zatim rekurzivno sortira. Od ova
152    dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
       istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
154    elementa koji se smesta u pomocni niz. Na kraju algoritma,
       sortirani elementi su u pomocnom nizu, koji se kopira u originalni
156    niz. */
   void mergesort(int a[], int l, int d)
158 {
     int s;
160  static int b[MAX];            /* Pomocni niz */
     int i, j, k;
162
     /* Izlaz iz rekurzije */
164  if (l >= d)
       return;
166
     /* Odredjivanje sredisnjeg indeksa */
168  s = (l + d) / 2;

170  /* Rekurzivni pozivi */
     mergesort(a, l, s);
172  mergesort(a, s + 1, d);

174  /* Inicijalizacija indeksa. Indeks i prolazi krozi levu polovinu
        niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
176     prolazi kroz pomocni niz b[] */
     i = l;
178  j = s + 1;
     k = 0;
180
     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
182  while (i <= s && j <= d) {
       if (a[i] < a[j])
184      b[k++] = a[i++];
       else
186      b[k++] = a[j++];
     }
188
     /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive j
190     iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
        polovine niza */
192  while (i <= s)
       b[k++] = a[i++];
194
     /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive i
196     iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
        polovine niza */
198  while (j <= d)
       b[k++] = a[j++];
```

```
200
    /* Prepisuje se "ucesljani" niz u originalni niz */
202   for (k = 0, i = l; i <= d; i++, k++)
        a[i] = b[k];
204 }

206 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
    void swap(int a[], int i, int j)
208 {
    int tmp = a[i];
210   a[i] = a[j];
    a[j] = tmp;
212 }

214
    /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r. Njena
216     ideja sortiranja je izbor jednog elementa niza, koji se naziva
      pivot, i koji se dovodi na svoje mesto. Posle ovog koraka, svi
218     elementi levo od njega bice manji, a svi desno bice veci od njega.
      Kako je pivot doveden na svoje mesto, da bi niz bio kompletno
220     sortiran, potrebno je sortirati elemente levo (manje) od njega, i
      elemente desno (vece). Kako su dimenzije ova dva podniza manje od
222     dimenzije pocetnog niza koji je trebalo sortirati, ovaj deo moze
      se uraditi rekurzivno. */
224 void quicksort(int a[], int l, int r)
    {
226   int i, pivot_position;

228   /* Izlaz iz rekurzije -- prazan niz */
    if (l >= r)
230     return;

232
    /* Particionisanje niza. Svi elementi na pozicijama levo od
234     pivot_position (izuzev same pozicije l) su strogo manji od
      pivota. Kada se pronadje neki element manji od pivota, uvecava
236     se promenljiva pivot_position i na tu poziciju se premesta
      nadjeni element. Na kraju ce pivot_position zaista biti pozicija
238     na koju treba smestiti pivot, jer ce svi elementi levo od te
      pozicije biti manji a desno biti veci ili jednaki od pivota. */
240   pivot_position = l;
    for (i = l + 1; i <= r; i++)
242     if (a[i] < a[l])
        swap(a, ++pivot_position, i);
244
    /* Postavljanje pivota na svoje mesto */
246   swap(a, l, pivot_position);

248   /* Rekurzivno sortiranje elementa manjih od pivota */
    quicksort(a, l, pivot_position - 1);
250   /* Rekurzivno sortiranje elementa vecih od pivota */
    quicksort(a, pivot_position + 1, r);
```

```
}
```

```
#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;
52        break;
      case 'm':
54        tip_sortiranja = 4;
          break;
56      case 'q':
          tip_sortiranja = 5;
58        break;
      case 'r':
60        tip_niza = 1;
          break;
62      case 'o':
          tip_niza = 2;
64        break;
      default:
66        printf("Pogresna opcija -%c\n", argv[i][1]);
          return 1;
68        break;
      }
70  }
    /* Ako je u pitanju argument, onda je to duzina niza koji treba
72     da se sortira */
    else {
74    dimenzija = atoi(argv[i]);
      if (dimenzija <= 0 || dimenzija > MAX) {
76      fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
        exit(EXIT_FAILURE);
78    }
  }
80}

82/* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
    niza dobijenom iz komandne linije. srandom() funkcija
84  obezbedjuje novi seed za pozivanje random funkcije, i kako
    generisani niz ne bi uvek bio isti seed je postavljen na tekuce
86  vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
    izmedju 0 i 99 */
88srandom(time(NULL));
if (tip_niza == 0)
90  for (i = 0; i < dimenzija; i++)
      niz[i] = random();
92else if (tip_niza == 1)
  for (i = 0; i < dimenzija; i++)
94  niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
else
96  for (i = 0; i < dimenzija; i++)
      niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
98
/* Ispisivanje elemenata niza */
100/*************************************************************
  Ovaj deo je iskomentarisan jer sledeci ispis ne treba da se nadje
102 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)
    selectionsort(niz, dimenzija);
  else if (tip_sortiranja == 1)
    insertionsort(niz, dimenzija);
  else if (tip_sortiranja == 2)
    bubblesort(niz, dimenzija);
  else if (tip_sortiranja == 3)
    shellsort(niz, dimenzija);
  else if (tip_sortiranja == 4)
    mergesort(niz, 0, dimenzija - 1);
  else
    quicksort(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]);
  ****************************************************************/

  return 0;
}
```

Rešenje 3.17

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

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 */
```

```
10    if (dim3 < dim1 + dim2)
        return -1;
12
    /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
14     od njih */
    while (i < dim1 && j < dim2) {
16    if (niz1[i] < niz2[j])
        niz3[k++] = niz1[i++];
18    else
        niz3[k++] = niz2[j++];
20  }
    /* Ostatak prvog niza prepisujemo u treci */
22  while (i < dim1)
      niz3[k++] = niz1[i++];
24
    /* Ostatak drugog niza prepisujemo u treci */
26  while (j < dim2)
      niz3[k++] = niz2[j++];
28  return dim1 + dim2;
}
30
int main()
32{
    int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
34  int i = 0, j = 0, k, dim3;

36  /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
       Pretpostavka je da na ulazu nece biti vise od MAX_DIM elemenata
       */
38  printf("Unesite elemente prvog niza: ");
    while (1) {
40    scanf("%d", &niz1[i]);
      if (niz1[i] == 0)
42      break;
      i++;
44  }
    printf("Unesite elemente drugog niza: ");
46  while (1) {
      scanf("%d", &niz2[j]);
48    if (niz2[j] == 0)
        break;
50    j++;
    }
52
    /* Poziv trazene funkcije */
54  dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

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

```
  return 0;
}
```

Rešenje 3.18

```
#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) {
    if (strcmp(ime1, ime2) < 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);

  return 0;
}
```

Rešenje 3.19

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#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
   (0,0) */
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)) {
53      min = j;
      }
55    }
    if (min != i) {
57    tmp = t[i];
      t[i] = t[min];
59    t[min] = tmp;
    }
61  }
}
63
/* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
65 void sortiraj_po_y(Tacka t[], int n)
{
67  int min, i, j;
  Tacka tmp;
69
  for (i = 0; i < n - 1; i++) {
71  min = i;
    for (j = i + 1; j < n; j++) {
73    if (abs(t[j].y) < abs(t[min].y)) {
        min = j;
75    }
    }
77  if (min != i) {
      tmp = t[i];
79    t[i] = t[min];
      t[min] = tmp;
81  }
  }
83 }

85 int main(int argc, char *argv[])
{
87  FILE *ulaz;
  FILE *izlaz;
89  Tacka tacke[MAX_BR_TACAKA];
  int i, n;
91
  /* Proveravanje broja argumenata komandne linije: ocekuje se ime
93    izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
      datoteke, tj. 4 argumenta */
95  if (argc != 4) {
    fprintf(stderr,
97            "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
    return 0;
99  }

101 /* Otvaranje datoteke u kojoj su zadate tacke */
  ulaz = fopen(argv[2], "r");
103 if (ulaz == NULL) {
```

```
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
105             argv[2]);
    return 0;
107 }

109 /* Otvaranje datoteke u koju treba upisati rezultat */
    izlaz = fopen(argv[3], "w");
111 if (izlaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
113             argv[3]);
    return 0;
115 }

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

125 /* Ukupan broj procitanih tacaka */
    n = i;
127
    /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
129    "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
       (karakter -), a karakter argv[1][1] odredjuje kriterijum
131    sortiranja */
    switch (argv[1][1]) {
133 case 'x':
    /* Sortiranje po vrednosti x koordinate */
135 sortiraj_po_x(tacke, n);
    break;
137 case 'y':
    /* Sortiranje po vrednosti y koordinate */
139 sortiraj_po_y(tacke, n);
    break;
141 case 'o':
    /* Sortiranje po udaljenosti od koorinatnog pocetka */
143 sortiraj_po_rastojanju(tacke, n);
    break;
145 }

147 /* Upisivanje dobijenog niza u izlaznu datoteku */
    for (i = 0; i < n; i++) {
149 fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
    }
151
    /* Zatvaranje otvorenih datoteka */
153 fclose(ulaz);
    fclose(izlaz);
155
```

```
  return 0;
}
```

Rešenje 3.20

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

#define MAX 1000
#define MAX_DUZINA 16

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

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

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

/* 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. */
48      min = i;
        for (j = i + 1; j < n; j++)
50        if (strcmp(a[j].prezime, a[min].prezime) < 0)
            min = j;
52      /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
           su (i) i min razliciti, inace je nepotrebno. */
54      if (min != i) {
          pom = a[i];
56        a[i] = a[min];
          a[min] = pom;
58      }
    }
60  }

62  /* Pretraga niza Gradjana */
    int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64  {
      int i;
66    for (i = 0; i < n; i++)
        if (strcmp(a[i].ime, x->ime) == 0
68          && strcmp(a[i].prezime, x->prezime) == 0)
          return i;
70    return -1;
    }
72


74  int main()
    {
76    Gradjanin spisak1[MAX], spisak2[MAX];
      int isti_rbr = 0;
78    int i, n;
      FILE *fp = NULL;
80
      /* Otvaranje datoteke */
82    if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
        fprintf(stderr,
84              "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
        exit(EXIT_FAILURE);
86    }

88    /* Citanje sadrzaja */
      for (i = 0;
90         fscanf(fp, "%s %s", spisak1[i].ime,
                   spisak1[i].prezime) != EOF; i++)
92      spisak2[i] = spisak1[i];
      n = i;
94
      /* Zatvaranje datoteke */
96    fclose(fp);

98    sort_ime(spisak1, n);
```

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

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

  sort_prezime(spisak2, n);

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

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

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

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

  /* Ispis rezultata */
  printf("%d\n", isti_rbr);

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.22

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

#define MAX_BR_RECI 128
#define MAX_DUZINA_RECI 32


/* Funkcija koja izracunava broj suglasnika u reci */
```

```
int broj_suglasnika(char s[])
{
  char c;
  int i;
  int suglasnici = 0;
  /* Prolaz karakter po karakter kroz zadatu nisku */
  for (i = 0; s[i]; i++) {
    /* Ako je u pitanju slovo */
    if (isalpha(s[i])) {
      /* Konvertuje se u veliko da bi bio pokriven slucaj i malih i
         velikih suglasnika */
      c = toupper(s[i]);
      /* Ukoliko slovo nije samoglasnik */
      if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U') {
        /* Uvecava se broj suglasnika */
        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
          /* A 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 kojem 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);
```

```
115   return 0;
}
```

Rešenje 3.23

```
#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)
{
```

```
46    int i, j;
      int min;
48    Artikal pom;

50    for (i = 0; i < n - 1; i++) {
        min = i;
52      for (j = i + 1; j < n; j++)
          if (a[j].kod < a[min].kod)
54          min = j;
        if (min != i) {
56        pom = a[i];
          a[i] = a[min];
58        a[min] = pom;
        }
60    }
  }
62
  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. */
104   selection_sort(asortiman, n);

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

114   kod = 0;
      while (1) {
116     printf("---------------------------\n");
        printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
118     printf("- Za nov racun unesite kod artikla!\n\n");
        /* Unos bar koda provog artikla sledeceg kupca */
120     if (scanf("%ld", &kod) == EOF)
          break;
122     /* Trenutni racun novog kupca */
        racun = 0;
124     /* Za sve artikle trenutnog kupca */
        while (1) {
126       /* Vrsi se njihov pronalazak u nizu */
          if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
128         printf("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
          } else {
130         printf("\tTrazili ste:\t%s %s %12.2f\n",
                  asortiman[i].naziv, asortiman[i].proizvodjac,
132               asortiman[i].cena);
            /* I dodavanje na ukupan racun */
134         racun += asortiman[i].cena;
          }
136       /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
             nema vise artikla */
138       printf("Unesite kod artikla [ili 0 za prekid]: \t");
          scanf("%ld", &kod);
140       if (kod == 0)
            break;
142     }
        /* Stampanje ukupnog racuna trenutnog kupca */
144     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
      }
146
      printf("Kraj rada kase!\n");
148
```

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

Rešenje 3.24

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

#define MAX 500

/* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
  char ime[20];
  char prezime[25];
  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;
```

```
47      for (j = i + 1; j < n; j++)
          if (niz[j].zadaci > niz[max].zadaci)
49          max = j;
          else if (niz[j].zadaci == niz[max].zadaci
51                   && strlen(niz[j].ime) < strlen(niz[max].ime))
            max = j;
53      if (max != i) {
          pom = niz[max];
55        niz[max] = niz[i];
          niz[i] = pom;
57      }
    }
59  }

61  /* Funkcija za sortiranje niza struktura po broju casova na kojima
       su bili 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


93  int main(int argc, char *argv[])
    {
95    Student praktikum[MAX];
    int i, br_studenata = 0;
97

    FILE *fp = NULL;
```

```
99
   /* Otvaranje datoteke za citanje */
101 if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
      fprintf(stderr, "Neupesno otvaranje datoteke aktivnost.txt.\n");
103   exit(EXIT_FAILURE);
    }
105
   /* Ucitavanje sadrzaja */
107 for (i = 0;
         fscanf(fp, "%s%s%d%d", praktikum[i].ime,
109             praktikum[i].prezime, &praktikum[i].prisustvo,
                &praktikum[i].zadaci) != EOF; i++);
111 /* Zatvaranje datoteke */
    fclose(fp);
113 br_studenata = i;

115 /* Kreiranje prvog spiska studenata po prvom kriterijumu */
    sort_ime_leksikografski(praktikum, br_studenata);
117 /* Otvaranje datoteke za pisanje */
    if ((fp = fopen("dat1.txt", "w")) == NULL) {
119   fprintf(stderr, "Neupesno otvaranje datoteke dat1.txt.\n");
      exit(EXIT_FAILURE);
121 }
    /* Upis niza u datoteku */
123 fprintf
        (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
125 for (i = 0; i < br_studenata; i++)
      fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
127           praktikum[i].prezime, praktikum[i].prisustvo,
              praktikum[i].zadaci);
129 /* Zatvaranje datoteke */
    fclose(fp);
131
   /* Kreiranje drugog spiska studenata po drugom kriterijumu */
133 sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
135 if ((fp = fopen("dat2.txt", "w")) == NULL) {
      fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
137   exit(EXIT_FAILURE);
    }
139 /* Upis niza u datoteku */
    fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
141 fprintf(fp, "pa po duzini imena rastuce:\n");
    for (i = 0; i < br_studenata; i++)
143   fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
145         praktikum[i].zadaci);
    /* Zatvaranje datoteke */
147 fclose(fp);

149 /* Kreiranje treceg spiska studenata po trecem kriterijumu */
    sort_prisustvo_pa_zadatke_pa_prezimena(praktikum, br_studenata);
```

```
  /* Otvaranje datoteke za pisanje */
  if ((fp = fopen("dat3.txt", "w")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke dat3.txt.\n");
    exit(EXIT_FAILURE);
  }
  /* 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);

  return 0;
}
```

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)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return strcmp(p1->izvodjac, p2->izvodjac);
}


int main(int argc, char *argv[])
{
  FILE *ulaz;
  Pesma *pesme;                 /* Pokazivac na deo memorije za
                                   cuvanje pesama */
  int alocirano_za_pesme;       /* Broj mesta alociranih za pesme */
  int i;                        /* Redni broj pesme cije se
                                   informacije citaju */
  int n;                        /* Ukupan broj pesama */
  int j, k;
  char c;
  int alocirano;                /* Broj mesta alociranih za propratne
                                   informacije o pesmama */
  int broj_gledanja;

  /* Priprema datoteke za citanje */
  ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
  if (ulaz == NULL) {
    printf("Greska pri otvaranju ulazne datoteke!\n");
    return 0;
  }

  /* Citanje informacija o pesmama */
  pesme = NULL;
  alocirano_za_pesme = 0;
  i = 0;

  while (1) {

    /* Proverava da li je dostignut kraj datoteke */
    c = fgetc(ulaz);
    if (c == EOF) {
      /* Nema vise sadrzaja za citanje */
      break;
    } else {
      /* Inace, vracamo procitani karakter nazad */
      ungetc(c, ulaz);
    }

```

```
84      /* Provera da li postoji dovoljno memorije za citanje nove pesme
        */
        if (alocirano_za_pesme == i) {
86
          /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
88           novih KORAK mesta */
          alocirano_za_pesme += KORAK;
90        pesme =
              (Pesma *) realloc(pesme,
92                              alocirano_za_pesme * sizeof(Pesma));

94        /* Proverava da li je nova memorija uspesno realocirana */
          if (pesme == NULL) {
96          /* Ako nije ispisuje se obavestenje */
            printf("Problem sa alokacijom memorije!\n");
98          /* I oslobadja sva memorija zauzeta do ovog koraka */
            for (k = 0; k < i; k++) {
100           free(pesme[k].izvodjac);
              free(pesme[k].naslov);
102         }
            free(pesme);
104         return 0;
          }
106
        }
108
        /* Ako jeste, nastavlja se sa citanjem pesama ... */
110     /* Cita se ime izvodjaca */

112     j = 0;                        /* Pozicija na koju treba smestiti
                                         procitani karakter */
114     alocirano = 0;                /* Broj alociranih mesta */
        pesme[i].izvodjac = NULL;     /* Memorija za smestanje procitanih
116                                      karaktera */

118     /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
           izvodjaca) citaju se karakteri iz datoteke */
120     while ((c = fgetc(ulaz)) != ' ') {
          /* Proverav da li postoji dovoljno memorije za smestanje
122          procitanog karaktera */
          if (j == alocirano) {
124
            /* Ako ne, ako je potrosena sva alocirana memorija, alocira
126              se novih KORAK mesta */
            alocirano += KORAK;
128         pesme[i].izvodjac =
                (char *) realloc(pesme[i].izvodjac,
130                              alocirano * sizeof(char));

132         /* Provera da li je nova alokacija uspesna */
            if (pesme[i].izvodjac == NULL) {
```

```
134           /* Ako nije oslobadja se sva memorija zauzeta do ovog
                 koraka */
136           for (k = 0; k < i; k++) {
                free(pesme[k].izvodjac);
138             free(pesme[k].naslov);
              }
140           free(pesme);
              /* I prekida sa izvrsavanjem programa */
142           return 0;
            }
144
        }
146     /* Ako postoji dovoljno memorije, smestamo procitani karakter
      */
        pesme[i].izvodjac[j] = c;
148     j++;
        /* I nastavlja se sa citanjem */
150   }

152   /* Upis terminirajuce nule na kraj reci */
      pesme[i].izvodjac[j] = '\0';
154


156   /* Preskace se karakter - */
      fgetc(ulaz);
158
      /* Preskace se razmak */
160   fgetc(ulaz);

162   /* Cita se naslov pesme */
      j = 0;                        /* Pozicija na koju treba smestiti
164                                    procitani karakter */
      alocirano = 0;                /* Broj alociranih mesta */
166   pesme[i].naslov = NULL;       /* Memorija za smestanje procitanih
                                     karaktera */
168
      /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
170      karakteri iz datoteke */

172   while ((c = fgetc(ulaz)) != ',') {
        /* Provera da li postoji dovoljno memorije za smestanje
174         procitanog karaktera */
        if (j == alocirano) {
176       /* Ako ne, ako je potrosena sva alocirana memorija, alocira
             se novih KORAK mesta */
178       alocirano += KORAK;
          pesme[i].naslov =
180           (char *) realloc(pesme[i].naslov,
                               alocirano * sizeof(char));
182
          /* Provera da li je nova alokacija uspesna */
184       if (pesme[i].naslov == NULL) {
```

```
            /* Ako nije, oslobadja se sva memorija zauzeta do ovog
186             koraka */
            for (k = 0; k < i; k++) {
188           free(pesme[k].izvodjac);
              free(pesme[k].naslov);
190         }
            free(pesme[i].izvodjac);
192         free(pesme);

194         /* I prekida izvrsavanje programa */
            return 0;
196       }

198     }
        /* Ako postoji dovoljno memorije, smesta se procitani karakter
      */
200     pesme[i].naslov[j] = c;
        j++;
202     /* I nastavlja dalje sa citanjem */
      }
204   /* Upisuje se terminirajuca nula na kraj reci */
      pesme[i].naslov[j] = '\0';
206
      /* Preskace se razmak */
208   fgetc(ulaz);

210   /* Cita se broj gledanja */
      broj_gledanja = 0;
212
      /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
214      datoteke */
      while ((c = fgetc(ulaz)) != '\n') {
216     broj_gledanja = broj_gledanja * 10 + (c - '0');
      }
218   pesme[i].broj_gledanja = broj_gledanja;

220   /* Prelazi se na citanje sledece pesme */
      i++;
222 }

224 /* Informacija o broju procitanih pesama */
    n = i;
226 /* Zatvaranje nepotrebne datoteke */
    fclose(ulaz);
228
    /* Analiza argumenta komandne linije */
230 if (argc == 1) {
      /* Nema dodatnih opcija => sortiranje po broju gledanja */
232   qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
    } else {
234   if (argc == 2 && strcmp(argv[1], "-n") == 0) {
        /* Sortiranje po naslovu */
```

```
236       qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
      } else {
238       if (argc == 2 && strcmp(argv[1], "-i") == 0) {
            /* Sortirnje po izvodjacu */
240         qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
          } else {
242         printf("Nedozvoljeni argumenti!\n");
            free(pesme);
244         return 0;
          }
246     }
    }
248
    /* Ispis rezultata */
250 for (i = 0; i < n; i++) {
      printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
252           pesme[i].broj_gledanja);
    }
254
    /* Oslobadjanje memorije */
256 for (i = 0; i < n; i++) {
      free(pesme[i].izvodjac);
258   free(pesme[i].naslov);
    }
260 free(pesme);

262 return 0;
}
```

Rešenje 3.28

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
   /* Funkcija poredjenja dva cela broja */
9  int compare_int(const void *a, const void *b)
   {
11   /* Konvertovanje void pokazivaca u int pokazivace koji se zatim
        dereferenciraju, a dobijeni int-ovi se oduzimaju i razlika
13      vraca. */

15   /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
        oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */
17
     int b1 = *((int *) a);
19   int b2 = *((int *) b);
```

```
21    if (b1 > b2)
        return 1;
23    else if (b1 < b2)
        return -1;
25    else
        return 0;
27  }

29  int compare_int_desc(const void *a, const void *b)
    {
31    /* Za obrnuti poredak treba samo oduzimati a od b */
      /* return *((int *)b) - *((int *)a); */
33
      /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
35       funkcija */
      return -compare_int(a, b);
37  }

39  int main()
    {
41    size_t n;
      int i, x;
43    int a[MAX], *p = NULL;

45    /* Unos dimenzije */
      printf("Uneti dimenziju niza: ");
47    scanf("%ld", &n);
      if (n > MAX)
49      n = MAX;

51    /* Unos elementa niza */
      printf("Uneti elemente niza:\n");
53    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
55
      /* Sortiranje niza celih brojeva */
57    qsort(a, n, sizeof(int), &compare_int);

59    /* Prikaz sortiranog niz */
      printf("Sortirani niz u rastucem poretku:\n");
61    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
63    putchar('\n');

65    /* Pretrazivanje niza */
      /* Vrednost koja ce biti trazena u nizu */
67    printf("Uneti element koji se trazi u nizu: ");
      scanf("%d", &x);
69
      /* Binarna pretraga */
71    printf("Binarna pretraga: \n");
      p = bsearch(&x, a, n, sizeof(int), &compare_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), &compare_int);
  if (p == NULL)
    printf("Elementa nema u nizu!\n");
  else
    printf("Element je nadjen na poziciji %ld\n", p - a);

  return 0;
}
```

Rešenje 3.29

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

#define MAX 100

/* Funkcija racuna broj delilaca broja x */
int no_of_deviders(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;
  /* 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
     jos jednog delioca */
  if (i * i == x)
    br++;

  return br;
```

```
}

/* Funkcija poredjenja dva cela broja po broju delilaca */
int compare_no_deviders(const void *a, const void *b)
{
  int ak = *(int *) a;
  int bk = *(int *) b;
  int n_d_a = no_of_deviders(ak);
  int n_d_b = no_of_deviders(bk);

  if (n_d_a > n_d_b)
    return 1;
  else if (n_d_a < n_d_b)
    return -1;
  else
    return 0;
}

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

  /* Unos dimenzije */
  printf("Uneti dimenziju niza: ");
  scanf("%ld", &n);
  if (n > MAX)
    n = MAX;

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

  /* Sortiranje niza celih brojeva prema broju delilaca */
  qsort(a, n, sizeof(int), &compare_no_deviders);

  /* Prikaz sortiranog niza */
  printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
  putchar('\n');

  return 0;
}
```

Rešenje 3.30

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

```c
#include <string.h>
#include <search.h>
#define MAX_NISKI 1000
#define MAX_DUZINA 30

/*****************************************************************
  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) {
    fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
    exit(EXIT_FAILURE);
  }

  /* Citanje sadrzaja datoteke */
```

```
55    for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);

57    /* Zatvaranje datoteke */
      fclose(fp);
59    n = i;

61    /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
         prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
63       niske po duzini */
      qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);
65
      printf("Leksikografski sortirane niske:\n");
67    for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
69    printf("\n");

71    /* Unos trazene niske */
      printf("Uneti trazenu nisku: ");
73    scanf("%s", x);

75    /* Binarna pretraga */
      /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
77       je niz vec sortiran leksikografski. */
      p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
79                  &poredi_leksikografski);

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

87    /* Linearna pretraga */
      p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
89                &poredi_leksikografski);

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

97    /* Sortiranje po duzini */
      qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
99
      printf("Niske sortirane po duzini:\n");
101   for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
103   printf("\n");

105   exit(EXIT_SUCCESS);
    }
```

Rešenje 3.31

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>
#define MAX_NISKI 1000
#define MAX_DUZINA 30

/*****************************************************************
  Niz pokazivaca na karaktere ovog potpisa
  char *niske[3];
  posle alokacije u main-u se moze graficki predstaviti ovako:
  -----                    -----------------
  | X | --------------->   | a | b | c | \0|
  -----                    =================
  | Y | --------------->   | d | e | \0|
  -----                    =================
  | Z | --------------->   | f | g | h | \0|
  -----                    -----------------
  Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
  reci. Njegov tip je char*.

  Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
  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
  moraju i kastovati. Da bi se leksikografski uporedili elementi X i
  Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
  u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
  kastovani na odgovarajuci tip.
*****************************************************************/
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);
}

/* 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) * 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]);
  printf("\n");

  /* Unos trazene niske */
  printf("Uneti trazenu nisku: ");
  scanf("%s", x);

  /* Binarna pretraga */
  p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
  if (p != NULL)
```

```
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
104           *p, p - niske);
  else
106   printf("Niska nije pronadjena u nizu\n");

108 /* Linearna pretraga */
  p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
110 if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
112           *p, p - niske);
  else
114   printf("Niska nije pronadjena u nizu\n");

116 /* Sortiramo po duzini */
  qsort(niske, n, sizeof(char *), &poredi_duzine);
118
  printf("Niske sortirane po duzini:\n");
120 for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
122 printf("\n");

124 /* Oslobadjanje zauzete memorije */
  for (i = 0; i < n; i++)
126   free(niske[i]);

128 exit(EXIT_SUCCESS);
}
```

Rešenje 3.32

```
#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;
14
/* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16    istim brojem bodova se dodatno sortiraju leksikografski po
    prezimenu */
18 int compare(const void *a, const void *b)
{
20   Student *prvi = (Student *) a;
  Student *drugi = (Student *) b;
```

```
22
   if (prvi->bodovi > drugi->bodovi)
24   return -1;
   else if (prvi->bodovi < drugi->bodovi)
26   return 1;
   else
28   /* Ako su jednaki po broju bodova, treba ih uporediti po
        prezimenu */
30   return strcmp(prvi->prezime, drugi->prezime);
}
32
/* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
34    Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
    parametar je element niza ciji se bodovi porede. */
36 int compare_za_bsearch(const void *a, const void *b)
{
38   int bodovi = *(int *) a;
   Student *s = (Student *) b;
40   return s->bodovi - bodovi;
}
42
/* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
    parametar je element niza cije se prezime poredi. */
46 int compare_za_linearna_prezimena(const void *a, const void *b)
{
48   char *prezime = (char *) a;
   Student *s = (Student *) b;
50   return strcmp(prezime, s->prezime);
}
52

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

64   /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
        informacija korisniku kako se program koristi i prekida se
66      izvrsavanje. */
   if (argc < 2) {
68     fprintf(stderr,
             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
70           argv[0]);
     exit(EXIT_FAILURE);
72   }
```

```
74  /* Otvaranje datoteke */
    if ((fp = fopen(argv[1], "r")) == NULL) {
76    fprintf(stderr, "Neupesno otvaranje datoteke %s\n", argv[1]);
      exit(EXIT_FAILURE);
78  }

80  /* Ucitavanje sadrzaja */
    for (i = 0;
82       fscanf(fp, "%s%s%d", kolokvijum[i].ime,
               kolokvijum[i].prezime,
84             &kolokvijum[i].bodovi) != EOF; i++);

86  /* Zatvaranje datoteke */
    fclose(fp);
88  br_studenata = i;

90  /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
       studenata sa istim brojem bodova sortiranje vrsi po prezimenu */
92  qsort(kolokvijum, br_studenata, sizeof(Student), &compare);

94  printf("Studenti sortirani po broju poena opadajuce, ");
    printf("pa po prezimenu rastuce:\n");
96  for (i = 0; i < br_studenata; i++)
      printf("%s %s  %d\n", kolokvijum[i].ime,
98           kolokvijum[i].prezime, kolokvijum[i].bodovi);

100 /* Pretrazivanje studenata po broju bodova se vrsi binarnom
       pretragom jer je niz sortiran po broju bodova. */
102 printf("Unesite broj bodova: ");
    scanf("%d", &bodovi);
104
    nadjen =
106     bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
              &compare_za_bsearch);
108
    if (nadjen != NULL)
110   printf
        ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
112      nadjen->ime, nadjen->prezime, nadjen->bodovi);
    else
114   printf("Nema studenta sa unetim brojem bodova\n");

116 /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
       sortiran po bodovima. */
118 printf("Unesite prezime: ");
    scanf("%s", prezime);
120
    nadjen =
122     lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
            &compare_za_linearna_prezimena);
124
    if (nadjen != NULL)
```

```
126     printf
            ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
128          nadjen->ime, nadjen->prezime, nadjen->bodovi);
    else
130     printf("Nema studenta sa unetim prezimenom\n");

132   return 0;
}
```

Rešenje 3.33

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

5  #define MAX 128

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

13 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
   int anagrami(char s[], char t[])
15 {
     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
17   if (strlen(s) != strlen(t))
       return 0;
19
     /* Sortiranje niski */
21   qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);
23
     /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
25      suprotnom, nisu */
     return !strcmp(s, t);
27 }

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

33   /* Unos niski */
     printf("Unesite prvu nisku: ");
35   scanf("%s", s);
     printf("Unesite drugu nisku: ");
37   scanf("%s", t);

39   /* Ispituje se da li su niske anagrami */
     if (anagrami(s, t))
```

```
41      printf("jesu\n");
    else
43      printf("nisu\n");

45  return 0;
}
```

Rešenje 3.34

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

5  #define MAX 10
   #define MAX_DUZINA 32
7
   /* Funkcija porenjenja */
9  int uporedi_niske(const void *pa, const void *pb)
   {
11   return strcmp((char *) pa, (char *) pb);
   }
13
   int main()
15 {
     int i, n;
17   char S[MAX][MAX_DUZINA];

19   /* Unos broja niski */
     printf("Unesite broj niski:");
21   scanf("%d", &n);

23   /* Unos niza niski */
     printf("Unesite niske:\n");
25   for (i = 0; i < n; i++)
       scanf("%s", S[i]);
27
     /* Sortiranje niza niski */
29   qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

31   /*************************************************************
       Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
33     sortiranih niski. Koriscen je samo u fazi testiranja programa.

35     printf("Sortirane niske su:\n");
       for(i = 0; i < n; i++)
37       printf("%s ", S[i]);
     *************************************************************/
39
     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
41      niza biti jedna do druge */
     for (i = 0; i < n - 1; i++)
```

```
43      if (strcmp(S[i], S[i + 1]) == 0) {
          printf("ima\n");
45        return 0;
        }
47
    printf("nema\n");
49  return 0;
}
```

Rešenje 3.35

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

5   #define MAX 21

7   /* Struktura koja predstavlja jednog studenta */
    typedef struct student {
9     char nalog[8];
      char ime[MAX];
11    char prezime[MAX];
      int poeni;
13  } Student;


15
    /* Funkcija poredi studente prema broju poena, rastuce */
17  int uporedi_poeni(const void *a, const void *b)
    {
19    Student s = *(Student *) a;
      Student t = *(Student *) b;
21    return s.poeni - t.poeni;
    }
23
    /* Funkcija poredi studente prvo prema godini, zatim prema smeru i
25     na kraju prema indeksu */
    int uporedi_nalog(const void *a, const void *b)
27  {
      Student s = *(Student *) a;
29    Student t = *(Student *) b;
      /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
31       broj indeksa */
      int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
33    int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
      char smer1 = s.nalog[1];
35    char smer2 = t.nalog[1];
      int indeks1 =
37        (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
          s.nalog[6] - '0';
39    int indeks2 =
          (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
```

```
41        t.nalog[6] - '0';
  if (godina1 != godina2)
43      return godina1 - godina2;
  else if (smer1 != smer2)
45      return smer1 - smer2;
  else
47      return indeks1 - indeks2;
}
49
int uporedi_bsearch(const void *a, const void *b)
51{
  /* Nalog studenta koji se trazi */
53  char *nalog = (char *) a;
  /* Kljuc pretrage */
55  Student s = *(Student *) b;

57  int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
  int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
59  char smer1 = nalog[1];
  char smer2 = s.nalog[1];
61  int indeks1 =
      (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
    ;
63  int indeks2 =
      (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
65      s.nalog[6] - '0';
  if (godina1 != godina2)
67    return godina1 - godina2;
  else if (smer1 != smer2)
69    return smer1 - smer2;
  else
71    return indeks1 - indeks2;
}
73
int main(int argc, char **argv)
75{
  Student *nadjen = NULL;
77  char nalog_trazeni[8];
  Student niz_studenata[100];
79  int i = 0, br_studenata = 0;
  FILE *in = NULL, *out = NULL;
81
  /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
83     korisnik nije ispravno pozvao program i prijavljuje se greska.
    */
  if (argc != 2 && argc != 3) {
85    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
    );
87    exit(EXIT_FAILURE);
  }
89
```

```
  /* Otvaranje datoteke za citanje */
91 in = fopen("studenti.txt", "r");
   if (in == NULL) {
93   fprintf(stderr,
             "Greska prilikom otvarnja datoteke studenti.txt!\n");
95   exit(EXIT_FAILURE);
   }
97
   /* Otvaranje datoteke za pisanje */
99 out = fopen("izlaz.txt", "w");
   if (out == NULL) {
101  fprintf(stderr,
             "Greska prilikom otvaranja datoteke izlaz.txt!\n");
103  exit(EXIT_FAILURE);
   }
105
   /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
107 while (fscanf
         (in, "%s %s %s %d", niz_studenata[i].nalog,
109       niz_studenata[i].ime, niz_studenata[i].prezime,
          &niz_studenata[i].poeni) != EOF)
111  i++;

113 br_studenata = i;

115 /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
   if (strcmp(argv[1], "-p") == 0)
117  qsort(niz_studenata, br_studenata, sizeof(Student),
         &uporedi_poeni);
119 /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
   else if (strcmp(argv[1], "-n") == 0)
121  qsort(niz_studenata, br_studenata, sizeof(Student),
         &uporedi_nalog);
123
   /* Sortirani studenti se ispisuju u izlaznu datoteku */
125 for (i = 0; i < br_studenata; i++)
   fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
127          niz_studenata[i].ime, niz_studenata[i].prezime,
          niz_studenata[i].poeni);
129
   /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
131    studenta... */
   if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
133  strcpy(nalog_trazeni, argv[2]);

135  /* ... pronalazi se student sa tim nalogom... */
   nadjen =
137      (Student *) bsearch(nalog_trazeni, niz_studenata,
                           br_studenata, sizeof(Student),
139                        &uporedi_bsearch);

141  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);

  return 0;
}
```

Rešenje 3.37

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

/* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
   standardnog ulaza */
void ucitaj_matricu(int **a, int n, int m)
{
  printf("Unesite elemente matrice po vrstama:\n");
  int i, j;

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

/* 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;
38    for (j = i + 1; j < n; j++) {
      if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
40      min = j;
      }
42    }
    if (min != i) {
44      int *tmp;
      tmp = a[i];
46      a[i] = a[min];
      a[min] = tmp;
48    }
  }
50}

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

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


66
/* Funkcija koja alocira memoriju za matricu dimenzija nxm */
68int **alociraj_memoriju(int n, int m)
{
70  int i, j;
  int **a;
72
  a = (int **) malloc(n * sizeof(int *));
74  if (a == NULL) {
    fprintf(stderr, "Problem sa alokacijom memorije!\n");
76    exit(EXIT_FAILURE);
  }
78  /* Za svaku vrstu ponaosob */
  for (i = 0; i < n; i++) {
80    /* Alocira se memorija */
    a[i] = (int *) malloc(m * sizeof(int));
82    /* Proverava se da li je doslo do greske prilikom alokacije */
    if (a[i] == NULL) {
84      /* Ako jeste, ispisuje se poruka */
      fprintf(stderr, "Problem sa alokacijom memorije!\n");
86      /* I oslobadja memorija zauzeta do ovog koraka */
      for (j = 0; j < i; j++) {
88        free(a[i]);
```

```
      }
      free(a);
      exit(EXIT_FAILURE);
    }
  }

  return a;
}

/* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije nxm
    */
void oslobodi_memoriju(int **a, int n, int m)
{
  int i;
  for (i = 0; i < n; i++) {
    free(a[i]);
  }
  free(a);
}



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_memoriju(n, m);

  /* Ucitavanje elementa matrice */
  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 */
  oslobodi_memoriju(a, n, m);

  return 0;
}
```

Glava 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. 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 polja novog čvora 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 `void dodaj_iza(Cvor* tekuci, Cvor* novi)` koja uvezuje u postojeću listu čvor `novi` iza čvora `tekuci`.

(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 se vrednost zadaje kao argument funkcije.

(k) Napisati funkciju `Cvor* pretrazi_sortiranu_listu(Cvor* glava, int broj)` koja proverava da li se u listi nalazi čvor čija se vrednost zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.

(l) Napisati funkciju `void obrisi_cvor(Cvor** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost koja se zadaje kao argument funkcije.

(m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost koja se zadaje kao argument funkcije, pri čemu se pretpostavlja da se pretraživanje vrši nad neopadajuće uređenoj listi.

(n) Napisati funkciju `void oslobodi_listu(Cvor** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

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

```
Poziv: ./a.out

Interakcija programa:
 Unosite brojeve: (za kraj unesite 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]
 3
 Lista: [3, 3, 5, 14, 3, 2]
 17
 Lista: [17, 3, 3, 5, 14, 3, 2]
 3
 Lista: [3, 17, 3, 3, 5, 14, 3, 2]
 1
 Lista: [1, 3, 17, 3, 3, 5, 14, 3, 2]
 9
 Lista: [9, 1, 3, 17, 3, 3, 5, 14, 3, 2]

 Unesite broj koji se trazi u listi:  17
 Trazeni broj 17 je u listi!
```

Primer 2

```
Poziv: ./a.out

Interakcija programa:
 Unosite brojeve: (za kraj unesite CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [35, 14, 23]

 Unesite broj koji se trazi u listi:  8
 Broj 8 se ne nalazi u listi!
```

Primer 3

```
Poziv: ./a.out

Interakcija programa:
 Unosite brojeve: (za kraj unesite CTRL+D)

 Unesite broj koji se trazi u listi:  1
 Broj 1 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

```
Poziv: ./a.out

Interakcija programa:
 Unosite brojeve: (za kraj unesite CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 5
 Lista: [2, 3, 14, 5]
 3
 Lista: [2, 3, 14, 5, 3]
 3
 Lista: [2, 3, 14, 5, 3, 3]
 17
 Lista: [2, 3, 14, 5, 3, 3, 17]
 3
 Lista: [2, 3, 14, 5, 3, 3, 17, 3]
 1
 Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1]
 3
 Lista: [2, 3, 14, 5, 3, 3, 17, 3, 1, 3]

 Unesite broj koji se brise iz liste:  3
 Lista nakon brisanja: [2, 14, 5, 17, 1]
```

Primer 2

```
Poziv: ./a.out

Interakcija programa:
 Unesite brojeve: (za kraj unesite CTRL+D)
 23
 Lista: [23]
 14
 Lista: [23, 14]
 35
 Lista: [23, 14, 35]

 Unesite broj koji se brise iz liste:  3
 Lista nakon brisanja: [23, 14, 35]
```

Primer 3

```
Poziv: ./a.out

Interakcija programa:
 Unesite brojeve: (za kraj unesite CTRL+D)

 Unesite broj koji se brise iz liste:  12
 Lista nakon brisanja: []
```

(3) U glavnom programu se učitani celi brojevi dodaju u listu tako da je vrednosti budu uređene u neopadajućem poretku. Unosi ceo broj koji se traži u unetoj listi, i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja cvora liste koristiti činjenicu da je lista uređena.*

Primer 1

```
POZIV: ./a.out

INTERAKCIJA PROGRAMA:
 Unesite brojeve: (za kraj unesite CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 5
 Lista: [2, 3, 5, 14]
 3
 Lista: [2, 3, 3, 5, 14]
 3
 Lista: [2, 3, 3, 3, 5, 14]
 17
 Lista: [2, 3, 3, 3, 5, 14, 17]
 3
 Lista: [2, 3, 3, 3, 3, 5, 14, 17]
 1
 Lista: [1, 2, 3, 3, 3, 3, 5, 14, 17]
 9
 Lista: [1, 2, 3, 3, 3, 3, 5, 9, 14, 17]

 Unesite broj koji se trazi u listi:  5
 Trazeni broj 5 je u listi!

 Unesite broj koji se brise iz liste:  3
 Lista nakon brisanja: [1, 2, 5, 9, 14, 17]
```

Primer 3

```
POZIV: ./a.out

INTERAKCIJA PROGRAMA:
 Unesite brojeve: (za kraj unesite CTRL+D)

 Unesite broj koji se trazi u listi:  1
 Broj 1 se ne nalazi u listi!

 Unesite broj koji se brise iz liste:  12
 Lista nakon brisanja: []
```

Primer 2

```
Poziv: ./a.out

Interakcija programa:
 Unesite brojeve: (za kraj unesite CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [14, 23, 35]

 Unesite broj koji se trazi u listi:  8
 Broj 8 se ne nalazi u listi!

Unesite broj koji se brise iz liste:  3
Lista nakon brisanja: [14, 23, 35]
```

[Rešenje 4.1]

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 iste* `main` *programe i upotrebe programa iz zadatka 4.1.*

[Rešenje 4.2]

Zadatak 4.3 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* tekuci)` koja iz liste čija se glava nalazi na adresi `adresa_glave` briše čvor na koji pokazuje pokazivač `tekuci`.

(b) Napisati funkciju `void ispisi_listu_u_nazad(Cvor* glava)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Napomena: *Koristiti iste* `main` *programe i upotrebe programa iz zadatka 4.1. Ove programe dopuniti pozivom funkcije koja ispisuje listu u nazad.*

[Rešenje 4.3]

Zadatak 4.4 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

```
Poziv: ./a.out

izraz.txt
 {[23 + 5344] * (24 - 234)} - 23

Izlaz:
 Zagrade su ispravno uparene.
```

Test 2

```
Poziv: ./a.out

izraz.txt
 {[23 + 5] * (9 * 2)} - {23}

Izlaz:
 Zagrade su ispravno uparene.
```

Test 3

```
Poziv: ./a.out

izraz.txt
 {[2 + 54) / (24 * 87)} + (234 + 23)

Izlaz:
 Zagrade nisu ispravno uparene.
```

Test 4

```
Poziv: ./a.out

izraz.txt
 {(2 - 14) / (23 + 11)}} * (2 + 13)

Izlaz:
 Zagrade nisu ispravno uparene.
```

Test 5

```
Poziv: ./a.out

izraz.txt
 Datoteka je prazna.

Izlaz:
 Zagrade su ispravno uparene.
```

Test 6

```
Poziv: ./a.out

izraz.txt
 Datoteka ne postoji.

Izlaz:
 Greska prilikom otvaranja
 datoteke izraz.txt!
```

[Rešenje 4.4]

Zadatak 4.5 Napisati program koji proverava ispravnost uparivanja etiketa u `HTML` datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. Uputstvo: *Za rešavanje problema koristiti stek implementiran preko liste čiji su čvorovi `HTML` etikete.*

Test 1

```
Poziv: ./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

```
Poziv: ./a.out datoteka.html

datoteka.html
  Datoteka ne postoji.

Izlaz:
  Greska prilikom otvaranja
  datoteke datoteka.html.
```

Test 3

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
 <html>
  <head>
  <title>Primer</title>
  </head>
  <body>
  <h1>Naslov</h1>
  Danas je lep i suncan dan. <br>
  A sutra ce biti jos lepsi.
  <a link="http://www.google.com» Link 1</a>
  <a link="http://www.math.rs» Link 2</a>
  </body>
 </html>

Izlaz:
 Etikete su pravilno uparene!
```

Test 4

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
 <html>
  <head>
  <title>Primer</title>
  </head>
  <body>
 </html>

Izlaz:
 Etikete nisu pravilno uparene
 (nadjena etiketa </html>, a poslednja
 otvorena etiketa je <body>)
```

Test 5

```
Poziv: ./a.out

Izlaz:
 Greska! Program se poziva
 sa: ./a.out datoteka.html!
```

Test 6

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
 Datoteka je prazna.

Izlaz:
 Etikete su pravilno uparene!
```

[Rešenje 4.5]

Zadatak 4.6 Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike. Službenik evidentira 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, tj. službeniku se 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`, tada 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 5 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. Uputstvo: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

Primer 1

```
Poziv: ./a.out

Interakcija programa:
 Sluzbenik evidentira korisnicke zahteve unosenjem
 njihovog JMBG broja i opisa potrebne usluge:
 Novi zahtev [CTRL+D za kraj]
  JMBG:   1234567890123
  Opis problema:   Otvaranje racuna

 Novi zahtev [CTRL+D za kraj]
  JMBG:   2345678901234
  Opis problema:   Podizanje novca

 Novi zahtev [CTRL+D za kraj]
  JMBG:   3456789012345
  Opis problema:   Reklamacija

 Novi zahtev [CTRL+D za kraj]
  JMBG:   4567890123456
  Opis problema:   Zatvaranje racuna

 Novi zahtev [CTRL+D za kraj]
  JMBG:

 Sledeci je korisnik sa JMBG brojem: 1234567890123
 sa zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG brojem: 2345678901234
 sa zahtevom: Podizanje novca
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG brojem: 3456789012345
 sa zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG brojem: 4567890123456
 sa zahtevom: Zatvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG brojem: 1234567890123
 sa zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Da li je kraj smene? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG brojem: 3456789012345
 sa zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

izvestaj.txt
   JMBG: 2345678901234 Zahtev: Podizanje novca
   JMBG: 4567890123456 Zahtev: Zatvaranje racuna
   JMBG: 1234567890123 Zahtev: Otvaranje racuna
   JMBG: 3456789012345 Zahtev: Reklamacija
```

[Rešenje 4.6]

Zadatak 4.7 Napisati program koji prebrojava pojavljivanja etiketa `HTML` datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. 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

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
 <html>
  <head><title>Primer</title></head>
  <body>
  <h1>Naslov</h1>
  Danas je lep i suncan dan. <br>
  A sutra ce biti jos lepsi.
  <a link="http://www.google.com» Link 1</a>
  <a link="http://www.math.rs» Link 2</a>
  </body>
 </html>

Izlaz:
 a - 4
 br - 1
 h1 - 2
 body - 2
 title - 2
 head - 2
 html - 2
```

Test 2

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
 Datoteka ne postoji.

Izlaz:
 Greska prilikom otvaranja
 datoteke datoteka.html.
```

Test 3

```
Poziv: ./a.out

Izlaz:
 Greska! Program se poziva
 sa: ./a.out datoteka.html!
```

Test 4

```
Poziv: ./a.out datoteka.html

DATOTEKA.HTML
 Datoteka je prazna.
```

[Rešenje 4.7]

Zadatak 4.8 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 studenata, 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). UPUTSTVO: *Dodavanje novog studenta izdvojiti u funkciju* `void dodaj_na_pocetak_liste(Cvor** glava, char* broj_indeksa, char* ime, char* prezime)`. *Napisati rekurzivnu funkciju* `Cvor* pretrazi_listu(Cvor* glava, char* broj_indeksa)` *koja određuje da li student sa zadatim brojem indeksa pripada listi ili ne. Nakon završenog pretraživanja rekurzivnom funkcijom* `void oslobodi_listu(Cvor** glava)` *osloboditi memoriju koju je lista sa studentima zauzimala. Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaketera maksimalna dužina bilo imena bilo prezimena studenta.*

Primer 1

```
POZIV: ./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 PROGRAMA:
  3/2014   da: Ana Sokic
  235/2008   ne
  41/2009   da: Marija Zaric
```

Primer 2

```
POZIV: ./a.out studenti.txt

STUDENTI.TXT
  Datoteka je prazna.

INTERAKCIJA PROGRAMA:
  3/2014   ne
  235/2008   ne
  41/2009   ne
```

Primer 3

```
POZIV: ./a.out

IZLAZ:
  Greska! Program se poziva sa:
  ./a.out studenti.txt!
```

Primer 4

```
POZIV: ./a.out studenti.txt

STUDENTI.TXT
  Datoteka ne postoji.

IZLAZ:
  Greska prilikom otvaranja datoteke
  studenti.txt.
```

[Rešenje 4.8]

Zadatak 4.9 Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

Test 1

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
 2 4 6 10 15

dat2.txt
 5 6 11 12 14 16

Izlaz:
 2 4 5 6 6 10 11 12 14 15 16
```

Test 2

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
 2 4 6 10 15

dat2.txt
 Datoteka ne postoji.

Izlaz:
 Greska prilikom otvaranja datoteke
 dat2.txt.
```

Test 3

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
 Datoteka ne postoji.

dat2.txt
 5 6 11 12 14 16

Izlaz:
 Greska prilikom otvaranja datoteke
 dat1.txt.
```

Test 4

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
 2 4 6 10 15

dat2.txt
 Datoteka je prazna.

Izlaz:
 2 4 6 10 15
```

Test 5

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
 Datoteka je prazna.

dat2.txt
 5 6 11 12 14 16

Izlaz:
 5 6 11 12 14 16
```

Test 6

```
Poziv: ./a.out

Izlaz:
 Greska! Program se poziva sa:
 ./a.out dat1.txt dat2.txt!
```

Test 7

```
Poziv: ./a.out dat1.txt

Izlaz:
 Greska! Program se poziva sa:
 ./a.out dat1.txt dat2.txt!
```

[Rešenje 4.9]

Zadatak 4.10 Date su dve jednostruko povezane liste `L1` i `L2`. Napisati funkciju koja od tih lista formira novu listu `L` koja sadrži alternirajući raspoređene čvorove lista `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 čvorove rasporediti

u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 7 za zadatak 4.9.*

Test 1

```
POZIV: ./a.out dat1.txt dat2.txt

DAT1.TXT
 2 4 6 10 15

DAT2.TXT
 5 6 11 12 14 16

IZLAZ:
 2 5 4 6 6 11 10 12 15 14 16
```

Zadatak 4.11 Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

(a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

(b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `Rezultat.txt` upisuje nađeni strogo rastući podniz.

Test 1

```
POZIV: ./a.out

BROJEVI.TXT
 43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
 12 15 16
```

Test 2

```
POZIV: ./a.out

BROJEVI.TXT
 Datoteka ne postoji.

IZLAZ:
REZULTAT.TXT
 Greska prilikom otvaranja
 datoteke brojevi.txt.
```

Test 3

```
POZIV: ./a.out

BROJEVI.TXT
 Datoteka je prazna.

IZLAZ:
REZULTAT.TXT
 Rezultat.txt ce biti prazna.
```

Zadatak 4.12 Grupa od n plesača na kostimima imaju brojeve od 1 do n, redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k-ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k-ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n, k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni

izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. Uputstvo: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

Test 1

```
Poziv: ./a.out

Ulaz:
 5 3

Izlaz:
 3 1 5 2 4
```

Test 2

```
Poziv: ./a.out

Ulaz:
 8 4

Izlaz:
 4 8 5 2 1 3 7 6
```

Test 3

```
Poziv: ./a.out

Ulaz:
 3 8

Izlaz:
 n mora biti uvek vece
 od k, a 3 < 8!
```

Test 4

```
Poziv: ./a.out

Ulaz:
 0 0

Izlaz:
 Broj plesaca mora biti
 veci od 0!
```

Zadatak 4.13 Grupa od n plesača na kostimima imaju brojeve od 1 do n, redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k-ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k-ti plesač. Odbrojavanje počinje od sledećeg suseda prethodno izbačenog, 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.* Napomena: *Iskoristiti 3. i 4. test za 4.12. zadatak.*

Test 1

```
Poziv: ./a.out

Ulaz:
 5 3

Izlaz:
 3 5 4 2 1
```

Test 2

```
Poziv: ./a.out

Ulaz:
 8 4

Izlaz:
 4 8 5 7 6 3 2 1
```

4.2 Stabla

Zadatak 4.14 Napisati program za rad sa binarnim pretraživačkim stablima.

(a) Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.

(b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.

(c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.

(d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.

(e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.

(f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.

(g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.

(h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.

(i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.

(j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.

(k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

Test Test 1

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa):
  7 2 1 9 32 18
Izlaz:
  Infiksni ispis: 1 2 7 9 18 32
  Prefiksni ispis: 7 2 1 9 32 18
  Postfiksni ispis: 1 2 18 32 9 7
  Trazi se broj: 11
  Broj se ne nalazi u stablu!
  Brise se broj: 7
  Rezultujuce stablo: 1 2 9 18 32
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa):
  8 -2 6 13 24 -3
Izlaz:
  Infiksni ispis:  -3 -2 6 8 13 24
  Prefiksni ispis: 8 -2 -3 6 13 24
  Postfiksni ispis: -3 6 -2 24 13 8
  Trazi se broj: 6
  Broj se nalazi u stablu!
  Brise se broj: 14
  Rezultujuce stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

Zadatak 4.15 Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog 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 poruku `Nedostaje ime ulazne datoteke!`. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

Test Test 1

```
Poziv: ./a.out test.txt
Datoteka test.txt:
  Sunce utorak raCunar SUNCE programiranje
     jabuka PROGramiranje sunCE JABUka
Izlaz:
  jabuka: 2
  programiranje: 2
  racunar: 1
  sunce: 3
  utorak: 1

  Najcesca rec: sunce (pojavljuje se 3 puta)
```

Test Test 2

```
Poziv: ./a.out suma.txt
Datoteka suma.txt:
  lipa zova hrast ZOVA breza LIPA
Izlaz:
  breza: 1
  hrast: 1
  lipa: 2
  zova: 2

  Najcesca rec: lipa (pojavljuje se 2 puta)
```

Test Test 3

```
Poziv: ./a.out
Izlaz:
  Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

Zadatak 4.16 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. `Pera Peric 064/123-4567`. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči `KRAJ`, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Test Upotreba programa 1

```
Poziv: ./a.out
Datoteka imenik.txt:
   Pera Peric 011/3240-987
   Marko Maric 064/1234-987
   Mirko Maric 011/589-333
   Sanja Savkovic 063/321-098
   Zika Zikic 021/759-858
Ulaz:
  Unesite ime datoteke: imenik.txt
  Unesite ime i prezime: Pera Peric
  Broj je: 011/3240-987
  Unesite ime i prezime: Marko Markovic
  Broj nije u imeniku!
  Unesite ime i prezime: KRAJ
```

[Rešenje 4.16]

Zadatak 4.17 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niz najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

Test Test 1

```
Poziv: ./a.out
Datoteka prijemni.txt:
   Marko Markovic 45.4 12.3 11
   Milan Jevremovic 35.2 1.3 9
   Maja Agic 60 19 20
   Nadica Zec 54.2 10 15.8
   Jovana Milic 23.3 2 5.6
Izlaz:
  1. Maja Agic 60.0 19.0 20.0 99.0
  2. Nadica Zec 54.2 10.0 15.8 80.0
  3. Marko Markovic 45.4 12.3 11.0 68.7
  4. Milan Jevremovic 35.2 1.3 9.0 45.5
  ----------------------------------
  5. Jovana Milic 23.3 2.0 5.6 30.9
```

[Rešenje 4.17]

*** Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY`.

Test Upotreba programa 1

```
Poziv: a.out
Datoteka rodjendani.txt:
  Marko Markovic 12.12.1990.
  Milan Jevremovic 04.06.1989.
  Maja Agic 23.04.2000.
  Nadica Zec 01.01.1993.
  Jovana Milic 05.05.1990.
Ulaz:
  Unesite datum: 23.04.
Izlaz:
  Slavljenik: Maja Agic
Ulaz:
  Unesite datum: 01.01.
Izlaz:
  Slavljenik: Nadica Zec
Ulaz:
  Unesite datum: 01.05.
Izlaz:
  Slavljenik: Jovana Milic 05.05.
Ulaz:
  Unesite datum: CTRL+D
```

[Rešenje 4.18]

Zadatak 4.19 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. NAPOMENA: *Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka stabla.h predstavljati popis funkcija biblioteke, a datoteka stabla.c njihove implementacije. Programe koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

Test Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo:
  10 5 15 3 2 4 30 12 14 13 0
  Drugo stablo:
  10 15 5 3 4 2 12 14 13 30 0
Izlaz:
  Stabla jesu identicna.
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo:
  10 5 15 4 3 2 30 12 14 13 0
  Drugo stablo:
  10 15 5 3 4 2 12 14 13 30 0
Izlaz:
  Stabla nisu identicna.
```

[Rešenje 4.19]

*** Zadatak 4.20** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

Test Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 1 7 8 9 2 2
  Drugo stablo: 3 9 6 11 1
Izlaz:
  Unija: 1 1 2 2 3 6 7 8 9 9 11
  Presek: 1 9
  Razlika: 2 2 7 8
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 2 7 5
  Drugo stablo: 4 3 3 7
Izlaz:
  Unija: 2 3 3 4 5 7 7 11
  Presek: 7
  Razlika: 2 5 11
```

[Rešenje 4.20]

Zadatak 4.21 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

Test Test 1

```
Poziv: ./a.out
Ulaz:
  n: 7
  a: 1 11 8 6 37 25 30
Izlaz:
  1 6 8 11 25 30 37
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  n: 55
Izlaz:
  Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

Zadatak 4.22 Dato je binarno pretraživačko stablo celih brojeva.

(a) Napisati funkciju koja izračunava broj čvorova stabla.

(b) Napisati funkciju koja izračunava broj listova stabla.

(c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.

(d) Napisati funkciju koja izračunava zbir čvorova stabla.

(e) Napisati funkciju koja izračunava najveći element stabla.

(f) Napisati funkciju koja izračunava dubinu stabla.

(g) Napisati funkciju koja izračunava broj čvorova na i-tom nivou stabla.

(h) Napisati funkciju koja ispisuje sve elemente na i-tom nivou stabla.

(i) Napisati funkciju koja izračunava maksimalnu vrednost na i-tom nivou stabla.

(j) Napisati funkciju koja izračunava zbir čvorova na i-tom nivou stabla.

(k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x.

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije.

Test Test 1

```
Poziv: ./a.out 2 15
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  broj cvorova: 10
  broj listova: 4
  pozitivni listovi: 2 4 13 30
  zbir cvorova: 108
  najveci element: 30
  dubina stabla: 5
  broj cvorova na 2. nivou: 3
  elementi na 2. nivou: 3 12 30
  maksimalni na 2. nivou: 30
  zbir na 2. nivou: 45
  zbir elemenata manjih ili jednakih od 15: 7
```

[Rešenje 4.22]

Zadatak 4.23 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

Test Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  0.nivo: 10
  1.nivo: 5 15
  2.nivo: 3 12 30
  3.nivo: 2 4 14
  4.nivo: 13
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  6 11 8 3 -2
Izlaz:
  0.nivo: 6
  1.nivo: 3 11
  2.nivo: -2 8
```

[Rešenje 4.23]

*** Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Test Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 14 30 1 0
Izlaz:
  Stabla su slicna kao u ogledalu.
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 20 15 0
Izlaz:
  Stabla nisu slicna kao u ogledalu.
```

Zadatak 4.25 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.

Test Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 2 11 16 1 13
Izlaz:
  7
```

Test Test 2

```
Poziv: ./a.out
Ulaz:
  16 30 40 24 10 18 45 22
Izlaz:
  6
```

[Rešenje 4.25]

Zadatak 4.26 Binarno stablo se naziva HEAP ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva HEAP. Napisati zatim i glavni program koji kreira stablo kao na slici ..., poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

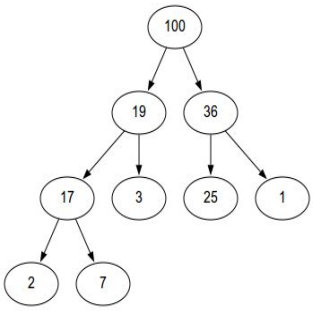
[Rešenje 4.26]

Zadatak 4.27 Dato je binarno stablo celih brojeva.

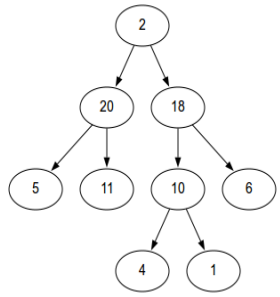
(a) Napisati funkciju koja pronalazi čvor u stablu sa maksimalnim proizvodom vrednosti iz desnog podstabla.

(b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjom sumom vrednosti iz levog podstabla.

(c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.



Slika 4.1: *zadatak 4.27*



Slika 4.2: *zadatak 4.23*

(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 ...

4.3 Rešenja

Rešenje 4.1

```
#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;

Cvor *napravi_cvor(int broj);

void oslobodi_listu(Cvor ** adresa_glave);

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

Cvor *pronadji_poslednji(Cvor * glava);

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
```

```
22
   void dodaj_iza(Cvor * tekuci, Cvor * novi);
24
   int dodaj_sortirano(Cvor ** adresa_glave, int broj);
26
   Cvor *pretrazi_listu(Cvor * glava, int broj);
28
   Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
30
   void obrisi_cvor(Cvor ** adresa_glave, int broj);
32
   void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
34
   void ispisi_listu(Cvor * glava);
36
   #endif
```

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
      cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
7     novom cvoru postavlja na NULL. Funkcija vraca pokazivac na
      novokreirani cvor ili NULL ako alokacija nije uspesno
9     izvrsena.  */
   Cvor *napravi_cvor(int broj)
11 {
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
13   if (novi == NULL)
       return NULL;
15
     novi->vrednost = broj;
17   novi->sledeci = NULL;
     return novi;
19 }

21 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove
      liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
23 void oslobodi_listu(Cvor ** adresa_glave)
   {
25   Cvor *pomocni = NULL;

27   /* Ako lista nije prazna, onda treba osloboditi memoriju. */
     while (*adresa_glave != NULL) {
29     /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
          osloboditi cvor koji predstavlja glavu liste */
31     pomocni = (*adresa_glave)->sledeci;
       free(*adresa_glave);
33     /* Sledeci cvor je nova glava liste. */
       *adresa_glave = pomocni;
35   }
```

```
}

/* Funkcija dodaje broj na pocetak liste. Kreira novi cvor
   koriscenjem funkcije napravi_cvor i uvezuje ga na pocetak.
   Funkcija treba da vrati 0 ukoliko je sve bilo u redu, odnosno
   1 ukoliko je doslo do greske prilikom alokacije memorije za
   nov cvor. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se nov cvor i proverava se da li je bilo greske pri
     alokaciji */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak, i tako postaje nova glave
     liste. */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  return 0;
}

/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste,
   ili NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava)
{
  /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
  if (glava == NULL)
    return NULL;

  /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
     pokazivac glava se pomera na sledeci cvor. Nakon izlaska iz
     petlje, glava ce pokazivati na cvor liste koji nema
     sledeceg, tj, poslednji je cvor liste, vraca se vrednost
     pokazivaca glava. Pokazivac glava je argument funkcije i
     njegove promene nece se odraziti na vrednost pokazivaca
     glava u pozivajucoj funkciji. */
  while (glava->sledeci != NULL)
    glava = glava->sledeci;

  return glava;
}

/* Funkcija dodaje broj na kraj liste. Ukoliko dodje do greske
   pri alokaciji memorije vratice 1, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

```

```
  /* U slucaju prazne liste, glava nove liste je upravo novi
     cvor i ujedno i cela lista. Azurira se vrednost na koju
     pokazuje adresa_glave i tako se azurira i pokazivacka
     promenljivu u pozivajucoj funkciji. */
  if (*adresa_glave == NULL) {
    *adresa_glave = novi;
    return 0;
  }

  /* Ako lista nije prazna, pronalazi se poslednji cvor i novi
     cvor se dodaje na kraj liste kao sledbenik poslednjeg. */
  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
  poslednji->sledeci = novi;

  return 0;
}

/* Pomocna funkcija pronalazi cvor u listi iza koga treba
   umetnuti nov cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
  /* U praznoj listi nema takvog mesta i vraca se NULL. */
  if (glava == NULL)
    return NULL;

  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
     pokazivala na cvor ciji je sledeci ili ne postoji ili ima
     vrednost vecu ili jednaku vrednosti novog cvora.

     Zbog izracunavanja izraza u C-u prvi deo konjukcije mora
     biti provera da li se doslo do poslednjeg cvora liste pre
     nego sto se proveri vrednost njegovog sledeceg cvora, jer u
     slucaju poslednjeg, sledeci ne postoji, pa ni njegova
     vrednost. */
  while (glava->sledeci != NULL
         && glava->sledeci->vrednost < broj)
    glava = glava->sledeci;

  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
     poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima
     vrednost vecu od broj. */
  return glava;
}

/* Funkcija uvezuje cvor novi iza cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi)
{
  /* Novi cvor se dodaje iza tekuceg cvora. */
  novi->sledeci = tekuci->sledeci;
  tekuci->sledeci = novi;
}
```

```
/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greski pri alokaciji
   memorije, inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* U slucaju prazne liste glava nove liste je novi cvor. */
  if (*adresa_glave == NULL) {
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
      return 1;
    *adresa_glave = novi;
    return 0;
  }

  /* Lista nije prazna. */
  /* 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 da se
     umetne nov broj. */
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Uvezuje se novi cvor iza pomocnog. */
  dodaj_iza(pomocni, novi);
  return 0;
}

/* 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)
{
  for (; glava != NULL; glava = glava->sledeci)
    if (glava->vrednost == broj)
      return glava;

  /* Nema trazenog broja u listi i vraca se NULL. */
  return NULL;
}

/* 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. Funkcija se u pretrazi oslanja na cinjenicu da je
```

```
   lista koja se pretrazuje neopadajuce sortirana. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
  /* U uslovu ostanka u petlji, bitan je redosled u konjukciji.*/
  for (; glava != NULL && glava->vrednost <= broj;
       glava = glava->sledeci)
    if (glava->vrednost == broj)
      return glava;

  return NULL;
}

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
   Funkcija azurira pokazivac na glavu liste, koji moze biti
   promenjen u slucaju da se obrise stara 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 toga lista ostala prazna, izlazi se iz
     funkcije. */
  if (*adresa_glave == NULL)
    return;

  /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje
     na cvor cija vrednost je 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, ili, ako je razlicit, prelazi se na
     sledeci cvor. Ovaj postupak se ponavlja dok se ne dodje do
     poslednjeg cvora. */
  tekuci = *adresa_glave;
  while (tekuci->sledeci != NULL)
    if (tekuci->sledeci->vrednost == broj) {
      /* tekuci->sledeci treba obrisati, zbog toga se njegova
         adresa prvo cuva u pomocni. */
      pomocni = tekuci->sledeci;
      /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
         njegovog trenutnog sledeceg. Njegov novi sledeci ce
         biti sledeci od ovog cvor koji se brise. */
```

```
      tekuci->sledeci = pomocni->sledeci;
      /* Sada treba osloboditi cvor sa vrednoscu broj. */
      free(pomocni);
    } else {
      /* Ne treba brisati sledeceg od tekuceg i pokazivac se
         pomera dalje. */
      tekuci = tekuci->sledeci;
    }
  return;
}

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
   neopadajuce. Funkcija azurira pokazivac na glavu liste, koji
   moze biti promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(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 toga lista ostala prazna, funkcija se prekida.
     Isto se radi i ukoliko glava liste sadrzi vrednost koja je
     veca od broja, jer kako je lista sortirana rastuce nema
     potrebe broj traziti u repu liste. */
  if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
    return;

  /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci
     pokazuje na cvor cija vrednost je manja od trazenog broja,
     kao i svim cvorovima levo od njega. Cvor se brise ako je
     jednak, ili, ako je razlicit, prelazi se na sledeci cvor.
     Ovaj postupak se ponavlja dok se ne dodje do poslednjeg
     cvora ili prvog cvora cija vrednost je veca od trazenog
     broja. */
  tekuci = *adresa_glave;
  while (tekuci->sledeci != NULL
         && tekuci->sledeci->vrednost <= broj)
    if (tekuci->sledeci->vrednost == broj) {
      pomocni = tekuci->sledeci;
      tekuci->sledeci = tekuci->sledeci->sledeci;
      free(pomocni);
```

```
    } else {
297   /* Ne treba brisati sledeceg od tekuceg, jer je manji od
         trazenog i prelazi se na sledeci. */
299   tekuci = tekuci->sledeci;
    }
301 return;
}
303
/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
305   kraju liste. Ne salje joj se adresa promenljive koja cuva
   glavu liste, jer ova funkcija nece menjati listu, pa nema ni
307   potrebe da azuriza pokazivac na glavu liste iz pozivajuce
   funkcije. */
309 void ispisi_listu(Cvor * glava)
{
311 putchar('[');
  for (; glava != NULL; glava = glava->sledeci) {
313   printf("%d", glava->vrednost);
    if (glava->sledeci != NULL)
315     printf(", ");
  }
317
  printf("]\n");
319 }
```

```
1 #include <stdio.h>
  #include <stdlib.h>
3 #include "lista.h"

5 int main()
{
7   /* Lista je prazna na pocetku. */
  Cvor *glava = NULL;
9   Cvor *trazeni = NULL;
  int broj;
11
  /* Testiranje dodavanja novog broja na pocetak liste. */
13 printf("Unosite brojeve: (za kraj unesite CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
15   /* Ako je funkcija vratila 1 onda je bilo greske pri
       alokaciji memorije za nov cvor. Memoriju alociranu za
17     cvorove liste treba osloboditi pre napustanja programa. */
    if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
19     fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
21     exit(EXIT_FAILURE);
    }
23   printf("\tLista: ");
    ispisi_listu(glava);
25 }

27 printf("\nUnesite broj koji se trazi u listi: ");
```

```
  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);

  oslobodi_listu(&glava);

  return 0;
}
```

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

int main()
{
  Cvor *glava = NULL;
  int broj;

  /* Testiranje dodavanja novog broja na kraj liste. */
  printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
  printf("\tLista: ");
  ispisi_listu(glava);

  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);
  }

  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, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  oslobodi_listu(&glava);

```

```
  return 0;
}
```

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

int main()
{
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se dodavanje u listu tako da ona bude neopadajuce
     uredjena */
  printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
  printf("\tLista: ");
  ispisi_listu(glava);

  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);
  }

  printf("\nUnesite broj koji se trazi u listi: ");
  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);

  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_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  oslobodi_listu(&glava);
```

```
  return 0;
}
```

Rešenje 4.2

```
#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;

Cvor *napravi_cvor(int broj);

void oslobodi_listu(Cvor ** adresa_glave);

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

Cvor *pronadji_poslednji(Cvor * glava);

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

void dodaj_iza(Cvor * tekuci, Cvor * novi);

int dodaj_sortirano(Cvor ** adresa_glave, int broj);

Cvor *pretrazi_listu(Cvor * glava, int broj);

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

void obrisi_cvor(Cvor ** adresa_glave, int broj);

void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

void ispisi_vrednosti(Cvor * glava);

void ispisi_listu(Cvor * glava);

#endif
```

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

```c
/* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
   cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
   novom cvoru postavlja na NULL. Funkcija vraca pokazivac na
   novokreirani cvor ili NULL ako alokacija nije uspesno
   izvrsena.  */
Cvor *napravi_cvor(int broj)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  novi->vrednost = broj;
  novi->sledeci = NULL;
  return novi;
}

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove
   liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave)
{
  /* Lista je vec prazna */
  if (*adresa_glave == NULL)
    return;

  /* Ako lista nije prazna, treba osloboditi memoriju. Pre
     oslobadjanja memorije za glavu liste, treba osloboditi rep
     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;
}

/* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
   koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.
   Funkcija vraca 1 ukoliko je doslo do greske pri alokaciji,
   inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se nov cvor i proverava se da li je bilo greske pri
     alokaciji */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak, i tako postaje nova glave
     liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;
  return 0;
```

```
}

/* Funkcija dodaje novi cvor na kraj liste. Prilikom dodavanja u
   listu na kraj u velikoj vecini slucajeva nov broj se dodaje u
   rep liste u rekurzivnom pozivu. U slucaju da je u rekurzivnom
   pozivu doslo do greske pri alokaciji, funkcija vraca 1 visem
   rekurzivnom pozivu koji tu informaciju vraca u rekurzivni
   poziv iznad, sve dok se ne vrati u main. Ako je funkcija
   vratila 0, onda nije bilo greske. Tek je iz main funkcije
   moguce pristupiti pravom pocetku liste i osloboditi je celu,
   ako ima potrebe. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  if (*adresa_glave == NULL) {
    /* Glava liste je upravo novi cvor i ujedno i cela lista. */
    Cvor *novi = napravi_cvor(broj);
    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;
    return 0;
  }

  /* Ako lista nije prazna, broj se dodaje u rep liste. */
  return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
}

/* Funkcija dodaje broj u rastuce sortiranu listu tako da nova
   lista ostane sortirana. Vraca 0, ako je alokacija novog cvora
   prosla bez greske, inace vraca 1 da bi ta vrednost bila
   propagirala nazad do prvog poziva. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* U slucaju prazne liste adresa_glave nove liste je upravo
     novi cvor. */
  if (*adresa_glave == NULL) {
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
      return 1;

    *adresa_glave = novi;
    return 0;
  }

  /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
     glavi liste, onda se dodaje na pocetak liste i vraca se
     informacija o uspesnosti alokacije. */
  if ((*adresa_glave)->vrednost >= broj)
    return dodaj_na_pocetak_liste(adresa_glave, broj);
```

```
109
    /* Inace, broj treba dodati u rep, tako da rep i sa novim
111     cvorom bude sortirana lista. */
    return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
113 }

115 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
     broju. Vraca pokazivac na cvor liste u kome je sadrzan
117     trazeni broj ili NULL u slucaju da takav cvor ne postoji u
     listi. */
119 Cvor *pretrazi_listu(Cvor * glava, int broj)
    {
121   /* U praznoj listi ga sigurno nema */
    if (glava == NULL)
123     return NULL;

125   /* Ako glava liste sadrzi trazeni broj */
    if (glava->vrednost == broj)
127     return glava;

129   /* Ako nije nijedna od prethodnih situacija, pretraga se
        nastavlja u repu liste. */
131   return pretrazi_listu(glava->sledeci, broj);
    }
133
    /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
135     broju. Funkcija se u pretrazi oslanja na cinjenicu da je
     lista koja se pretrazuje neopadajuce sortirana. Vraca
137     pokazivac na cvor liste u kome je sadrzan trazeni broj ili
     NULL u slucaju da takav cvor ne postoji u listi. */
139 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
    {
141   /* Trazenog broja nema ako je lista prazna ili ako je broj
        manji od vrednosti u glavi liste, jer je lista neopadajuce
143     sortirana. */
    if (glava == NULL || glava->vrednost > broj)
145     return NULL;

147   /* Ako glava liste sadrzi trazeni broj, vraca se glava. */
    if (glava->vrednost == broj)
149     return glava;

151   /* Ako nije nijedna od prethodnih situacija, pretraga se
        nastavlja u repu. */
153   return pretrazi_listu(glava->sledeci, broj);
    }
155
    /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
157     Funkcija azurira pokazivac na glavu liste, koji moze biti
     promenjen u slucaju da se obrise stara glava liste. */
159 void obrisi_cvor(Cvor ** adresa_glave, int broj)
    {
```

```
161   /* U praznoj listi, nema cvorova za brisanje. */
      if (*adresa_glave == NULL)
163     return;

165   /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj. */
      obrisi_cvor(&(*adresa_glave)->sledeci, broj);
167
      /* Preostaje provera da li glavu liste treba obrisati. */
169   if ((*adresa_glave)->vrednost == broj) {
        /* pomocni pokazuje na cvor koji treba da se obrise. */
171     Cvor *pomocni = *adresa_glave;
        /* Azurira se pokazivac na glavu da pokazuje na sledeci u
173        listi i brise se cvor koji je bio glava liste. */
        *adresa_glave = (*adresa_glave)->sledeci;
175     free(pomocni);
      }
177 }

179 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
       oslanjajuci se na cinjenicu da je prosledjena lista sortirana
181    neopadajuce. Funkcija azurira pokazivac na glavu liste, koji
       moze biti promenjen ukoliko se obrise stara glava liste. */
183 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
    {
185   /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
         veca od broja, kako je lista sortirana rastuce nema potrebe
187      broj traziti u repu liste i zato se funkcija prekida. */
      if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
189     return;

191   /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
      obrisi_cvor(&(*adresa_glave)->sledeci, broj);
193
      /* Preostaje provera da li glavu liste treba obrisati. */
195   if ((*adresa_glave)->vrednost == broj) {
        /* pomocni pokazuje na cvor koji treba da se obrise. */
197     Cvor *pomocni = *adresa_glave;
        /* Azurira se pokazivac na glavu da pokazuje na sledeci u
199        listi i brise se cvor koji je bio glava liste. */
        *adresa_glave = (*adresa_glave)->sledeci;
201     free(pomocni);
      }
203 }

205 /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
       zapetama. */
207 void ispisi_vrednosti(Cvor * glava)
    {
209   /* Prazna lista */
      if (glava == NULL)
211     return;
```

```
213   /* Ispisuje se vrednost u glavi liste. */
      printf("%d", glava->vrednost);
215
      /* Ako rep nije prazan, ispisuje se znak ',' i razmak.
217      Rekurzivno se poziva ista funkcija za ispis ostalih. */
      if (glava->sledeci != NULL) {
219     printf(", ");
        ispisi_vrednosti(glava->sledeci);
221   }
}
223
/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
225    kraju liste. Ne salje joj se adresa promenljive koja cuva
      glavu liste, jer ova funkcija nece menjati listu, pa nema ni
227    potrebe da azuriza pokazivac na glavu liste iz pozivajuce
      funkcije. Ova funkcija ispisuje samo zagrade, a rekurzivno
229    ispisivanje vrednosti u listi prepusta rekurzivnoj pomocnoj
      funkciji ispisi_vrednosti, koja ce ispisati elemente
231    razdvojene zapetom i razmakom. */
void ispisi_listu(Cvor * glava)
233 {
  putchar('[');
235   ispisi_vrednosti(glava);
  printf("]\n");
237 }
```

Rešenje 4.3

```
1 #ifndef _LISTA_H
  #define _LISTA_H
3
  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5    podatak vrednost i pokazivace na sledeci i prethodni cvor
     liste. */
7 typedef struct cvor {
    int vrednost;
9   struct cvor *sledeci;
    struct cvor *prethodni;
11 } Cvor;

13 Cvor *napravi_cvor(int broj);

15 void oslobodi_listu(Cvor ** adresa_glave);

17 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

19 Cvor *pronadji_poslednji(Cvor * glava);

21 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

23 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
```

```
25 void dodaj_iza(Cvor * tekuci, Cvor * novi);

27 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

29 Cvor *pretrazi_listu(Cvor * glava, int broj);

31 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

33 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci);

35 void obrisi_cvor(Cvor ** adresa_glave, int broj);

37 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

39 void ispisi_listu(Cvor * glava);

41 void ispisi_listu_u_nazad(Cvor * glava);

43 #endif
```

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  /* Pomocna funkcija koja kreira cvor. Funkcija vrednost novog
      cvora inicijalizuje na broj, dok pokazivac na sledeci cvor u
7     novom cvoru postavlja na NULL. Funkcija vraca pokazivac na
      novokreirani cvor ili NULL ako alokacija nije uspesno
9     izvrsena. */
   Cvor *napravi_cvor(int broj)
11 {
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
13   if (novi == NULL)
       return NULL;
15
     novi->vrednost = broj;
17   novi->sledeci = NULL;
     return novi;
19 }

21 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove
      liste ciji se pocetni cvor nalazi na adresi adresa_glave. */
23 void oslobodi_listu(Cvor ** adresa_glave)
   {
25   Cvor *pomocni = NULL;

27   /* Ako lista nije prazna, onda treba osloboditi memoriju. */
     while (*adresa_glave != NULL) {
29     /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
          osloboditi memoriju cvora koji predstavlja glavu liste. */
31     pomocni = (*adresa_glave)->sledeci;
```

```
    free(*adresa_glave);
    /* Sledeci cvor je nova glava liste. */
    *adresa_glave = pomocni;
  }
}

/* Funkcija dodaje novi cvor na pocetak liste. Kreira novi cvor
   koriscenjem funkcije napravi_cvor() i uvezuje ga na pocetak.
   Vraca 1 ukoliko je bilo greski pri alokaciji memorije, inace
   vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Sledbenik novog cvora je glava stare liste */
  novi->sledeci = *adresa_glave;
  /* Ako stara lista nije bila prazna, onda prethodni od glave
     treba da bude nov cvor. */
  if (*adresa_glave != NULL)
    (*adresa_glave)->prethodni = novi;
  /* Novi cvor je nova glava liste. */
  *adresa_glave = novi;

  return 0;
}

/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste,
   ili NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava)
{
  /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
  if (glava == NULL)
    return NULL;

  /* Sve dok glava ne pokazuje na cvor koji nema sledeceg,
     pokazivac glava se pomera na sledeci cvor. Nakon izlaska iz
     petlje, glava ce pokazivati na cvor liste koji nema
     sledeceg, tj, poslednji je cvor liste, vraca se vrednost
     pokazivaca glava. Pokazivac glava je argument funkcije i
     njegove promene nece se odraziti na vrednost pokazivaca
     glava u pozivajucoj funkciji. */
  while (glava->sledeci != NULL)
    glava = glava->sledeci;

  return glava;
}

/* Funkcija dodaje broj na kraj liste. Ukoliko dodje do greske
   pri alokaciji memorije vratice 1, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
```

```
{
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* U slucaju prazne liste, glava nove liste je upravo novi
     cvor i ujedno i cela lista. Azurira se vrednost na koju
     pokazuje adresa_glave i tako se azurira i pokazivacka
     promenljivu u pozivajucoj funkciji. */
  if (*adresa_glave == NULL) {
    *adresa_glave = novi;
    return 0;
  }

  /* Kako lista nije prazna, pronalazi se poslednji cvor i novi
     cvor se dodaje na kraj liste kao sledbenik poslednjeg. */
  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
  poslednji->sledeci = novi;
  novi->prethodni = poslednji;

  return 0;
}

/* Pomocna funkcija pronalazi cvor u listi iza koga treba
   umetnuti nov cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
  /* U praznoj listi nema takvog mesta i vraca se NULL. */
  if (glava == NULL)
    return NULL;

  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
     pokazivala na cvor ciji je sledeci ili ne postoji ili ima
     vrednost vecu ili jednaku vrednosti novog cvora.

     Zbog izracunavanja izraza u C-u prvi deo konjukcije mora
     biti provera da li se doslo do poslednjeg cvora liste pre
     nego sto se proveri vrednost njegovog sledeceg cvora, jer u
     slucaju poslednjeg, sledeci ne postoji, pa ni njegova
     vrednost. */
  while (glava->sledeci != NULL
         && glava->sledeci->vrednost < broj)
    glava = glava->sledeci;

  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
     poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima
     vrednost vecu od broj. */
  return glava;
}

/* Funkcija uvezuje cvor novi iza cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi)
```

```
{
  novi->sledeci = tekuci->sledeci;
  novi->prethodni = tekuci;

  /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje
     prethodnik i tekuci dobija novog sledeceg postavljanjem
     pokazivaca na ispravne adrese. */
  if (tekuci->sledeci != NULL)
    tekuci->sledeci->prethodni = novi;
  tekuci->sledeci = novi;
}

/* Fukcija dodaje u listu nov cvor na odgovarajuce mesto, tako
   sto pronalazi cvor u listi iza kod treba uvezati nov cvor.
   Ukoliko dodje do greske pri alokaciji memorije vratice 1,
   inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna, glava nove liste je novi cvor. */
  if (*adresa_glave == NULL) {
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
      return 1;
    *adresa_glave = novi;
    return 0;
  }

  /* Ukoliko je vrednost glave liste veca ili jednaka od nove
     vrednosti onda nov cvor treba staviti na pocetak liste. */
  if ((*adresa_glave)->vrednost >= broj) {
    dodaj_na_pocetak_liste(adresa_glave, broj);
    return 0;
  }

  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Pronazi se cvor iza koga treba uvezti nov cvor. */
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  dodaj_iza(pomocni, novi);

  return 0;
}

/* 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)
{
  for (; glava != NULL; glava = glava->sledeci)
```

```
    if (glava->vrednost == broj)
189       return glava;

191   /* Nema trazenog broja u listi i vraca se NULL. */
    return NULL;
193 }

195 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
    broju. Funkcija se u pretrazi oslanja na cinjenicu da je
197    lista koja se pretrazuje neopadajuce sortirana. Vraca
    pokazivac na cvor liste u kome je sadrzan trazeni broj ili
199    NULL u slucaju da takav cvor ne postoji u listi.  */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
201 {
    /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
203   for (; glava != NULL && glava->vrednost <= broj;
         glava = glava->sledeci)
205     if (glava->vrednost == broj)
        return glava;
207
    /* Nema trazenog broja u listi i bice vraceno NULL. */
209   return NULL;
}
211
/* Funkcija brise u listi na koju pokazuje pokazivac glava onaj
213    cvor na koji pokazuje pokazivac tekuci. Obratiti paznju da je
    kod dvostruke liste ovo mnogo lakse uraditi jer cvor tekuci
215    sadrzi pokazivace na svog sledbenika i prethodnika u listi. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
217 {
    /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
219   if (tekuci == NULL)
      return;
221
    /* Ako postoji prethodnik od tekuceg, onda se postavlja da
223       njegov sledeci bude sledeci od tekuceg. */
    if (tekuci->prethodni != NULL)
225     tekuci->prethodni->sledeci = tekuci->sledeci;

227   /* Ako postoji sledbenik tekuceg (cvora koji se brise), onda
       njegov prethodnik treba da bude prethodnik tekuceg. */
229   if (tekuci->sledeci != NULL)
      tekuci->sledeci->prethodni = tekuci->prethodni;
231
    /* Ako je glava cvor koji se brise, glava nove liste ce biti
233       sledbenik od stare glave. */
    if (tekuci == *adresa_glave)
235     *adresa_glave = tekuci->sledeci;

237   /* Oslobadja se dinamicki alociran prostor za cvor tekuci. */
    free(tekuci);
239 }
```

```
241 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
       Funkcija azurira pokazivac na glavu liste, koji moze biti
243    promenjen u slucaju da se obrise stara glava. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj)
245 {
      Cvor *tekuci = *adresa_glave;
247
      while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
249     obrisi_tekuci(adresa_glave, tekuci);
    }
251
    /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
253    oslanjajuci se na cinjenicu da je prosledjena lista
       neopadajuce sortirana. Funkcija azurira pokazivac na glavu
255    liste, koji moze biti promenjen ukoliko se obrise stara glava
       liste. */
257 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
    {
259   Cvor *tekuci = *adresa_glave;

261   while ((tekuci =
              pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
263     obrisi_tekuci(adresa_glave, tekuci);
    }
265
    /* Funkcija prikazuje cvorove liste pocev od glave ka kraju
267    liste. Ne salje joj se adresa promenljive koja cuva glavu
       liste, jer ova funkcija nece menjati listu, pa nema ni
269    potrebe da azuriza pokazivac na glavu liste iz pozivajuce
       funkcije. */
271 void ispisi_listu(Cvor * glava)
    {
273   putchar('[');
      for (; glava != NULL; glava = glava->sledeci) {
275     printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
277       printf(", ");
      }
279
      printf("]\n");
281 }

283 /* Funkcija prikazuje cvorove liste pocev od kraja ka glavi
       liste. */
285 void ispisi_listu_u_nazad(Cvor * glava)
    {
287   putchar('[');
      if (glava == NULL) {
289     printf("]\n");
        return;
291   }
```

```
293   glava = pronadji_poslednji(glava);

295   for (; glava != NULL; glava = glava->prethodni) {
        printf("%d", glava->vrednost);
297     if (glava->prethodni != NULL)
          printf(", ");
299   }
      printf("]\n");
301 }
```

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  int main()
   {
7    /* Lista je prazna na pocetku. */
     Cvor *glava = NULL;
9    Cvor *trazeni = NULL;
     int broj;
11
     /* Testiranje dodavanja novog broja na pocetak liste. */
13   printf("Unosite brojeve: (za kraj unesite CTRL+D)\n");
     while (scanf("%d", &broj) > 0) {
15     /* Ako je funkcija vratila 1 onda je bilo greske pri
          alokaciji memorije za nov cvor. Memoriju alociranu za
17        cvorove liste treba osloboditi pre napustanja programa. */
       if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
19       fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
         oslobodi_listu(&glava);
21       exit(EXIT_FAILURE);
       }
23     printf("\tLista: ");
       ispisi_listu(glava);
25   }

27   printf("\nUnesite broj koji se trazi u listi: ");
     scanf("%d", &broj);
29
     trazeni = pretrazi_listu(glava, broj);
31   if (trazeni == NULL)
       printf("Broj %d se ne nalazi u listi!\n", broj);
33   else
       printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
35
     printf("\nLista ispisana u nazad: ");
37   ispisi_listu_u_nazad(glava);

39   oslobodi_listu(&glava);

41   return 0;
```

```
}
```

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

int main()
{
  Cvor *glava = NULL;
  int broj;

  /* Testiranje dodavanja novog broja na kraj liste. */
  printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
  printf("\tLista: ");
  ispisi_listu(glava);

  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);
  }

  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, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  printf("\nLista ispisana u nazad: ");
  ispisi_listu_u_nazad(glava);

  oslobodi_listu(&glava);

  return 0;
}
```

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

```

```c
int main()
{
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se dodavanje u listu tako da ona bude neopadajuce
     uredjena */
  printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
  printf("\tLista: ");
  ispisi_listu(glava);

  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);
  }

  printf("\nUnesite broj koji se trazi u listi: ");
  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);

  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_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  printf("\nLista ispisana u nazad: ");
  ispisi_listu_u_nazad(glava);

  oslobodi_listu(&glava);

  return 0;
}
```

Rešenje 4.4

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

typedef struct cvor {
  char zagrada;
  struct cvor *sledeci;
} Cvor;

int main()
{
  Cvor *stek = NULL;
  FILE *ulaz = NULL;
  char c;
  Cvor *pomocni = NULL;

  ulaz = fopen("izraz.txt", "r");
  if (ulaz == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke izraz.txt!\n");
    exit(EXIT_FAILURE);
  }

  while ((c = fgetc(ulaz)) != EOF) {
    /* Ako je ucitana otvorena zagrada, stavlja se na stek. */
    if (c == '(' || c == '{' || c == '[') {
      pomocni = (Cvor *) malloc(sizeof(Cvor));
      if (pomocni == NULL) {
        fprintf(stderr, "Greska prilikom alokacije memorije!\n");
        return 1;
      }
      pomocni->zagrada = c;
      pomocni->sledeci = stek;
      stek = pomocni;
    }
    /* Ako je ucitana zatvorena zagrada, proverava se da li je
       stek je prazan i ako nije, da li se na vrhu steka nalazi
       odgovarajuca otvorena zagrada. */
    else {
      if (c == ')' || c == '}' || c == ']') {
        if (stek != NULL && ((stek->zagrada == '(' && c == ')')
                             || (stek->zagrada == '{' && c
                                 == '}')
                             || (stek->zagrada == '[' && c
                                 == ']'))) {
          /* Sa vrha steka se uklanja otvorena zagrada */
          pomocni = stek->sledeci;
          free(stek);
          stek = pomocni;
        } else {
          /* Zagrade u izrazu nisu ispravno uparene. */
```

```
            break;
52        }
        }
54    }
  }
56
  /* Ako je na kraju stek prazan i procitana je cela datoteka,
58     zagrade su ispravno uparene, u suprotnom, nisu. */
  if (stek == NULL && c == EOF)
60    printf("Zagrade su ispravno uparene.\n");
  else {
62    printf("Zagrade nisu ispravno uparene.\n");

64    while (stek != NULL) {
        /* U slucaju neispravnog uparivanja treba osloboditi
66         memoriju koja je ostala zauzeta stekom. */
      pomocni = stek->sledeci;
68    free(stek);
      stek = pomocni;
70    }
  }
72
  fclose(ulaz);
74  return 0;
}
```

Rešenje 4.5

```
1 #include <stdio.h>
  #include <stdlib.h>
3 #include <string.h>
  #include <ctype.h>
5
  #define MAX 100
7
  #define OTVORENA 1
9 #define ZATVORENA 2

11 #define VAN_ETIKETE 0
  #define PROCITANO_MANJE 1
13 #define U_ETIKETI 2

15 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete
     i pokazivac na sledeci cvor. */
17 typedef struct cvor {
    char etiketa[MAX];
19  struct cvor *sledeci;
  } Cvor;
21    /* Funkcija kreira novi cvor, upisuje u njega etiketu i
       vraca njegovu adresu ili NULL ako alokacija nije bila
23     uspesna. */
```

```
Cvor *napravi_cvor(char *etiketa)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja u novom cvoru */
  if (strlen(etiketa) >= MAX) {
    fprintf(stderr,
            "Etiketa koju biste stavili na stek je preduga, \
bice skracena .\n");
    etiketa[MAX - 1] = '\0';
  }
  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;
  return novi;
}

/* Funkcija prazni stek */
void oslobodi_stek(Cvor ** adresa_vrha)
{
  Cvor *pomocni;
  while (*adresa_vrha != NULL) {
    pomocni = *adresa_vrha;
    *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);
  }
}

/* Funkcija proverava uspesnost alokacije memorije za cvor novi
   i ukoliko alokacija nije bila uspesna, oslobadja se sva
   prethodno zauzeta memorija za listu cija pocetni cvor se
   nalazi na adresi adresa_vrha. */
void proveri_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
{
  if (novi == NULL) {
    fprintf(stderr, "Neuspela alokacija za nov cvor\n");
    oslobodi_stek(adresa_vrha);
    exit(EXIT_FAILURE);
  }
}

/* Funkcija postavlja na vrh steka novu etiketu. */
void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
{
  Cvor *novi = napravi_cvor(etiketa);
  proveri_alokaciju(adresa_vrha, novi);
  novi->sledeci = *adresa_vrha;
  *adresa_vrha = novi;
}

/* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
```

```
   pokazivac razlicit od NULL, tada u niz karaktera na koji on
77   pokazuje upisuje ime etikete koja je upravo skinuta sa steka
   dok u suprotnom ne radi nista. Funkcija vraca 0 ako je stek
79   prazan (pa samim tim nije bilo moguce skinuti vrednost sa
   steka) ili 1 u suprotnom. */
81 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
{
83   Cvor *pomocni;

85   /* Pokusaj skidanja vrednost sa vrha praznog steka rezultuje
        greskom i vraca se 0. */
87   if (*adresa_vrha == NULL)
       return 0;
89
   /* Ako adresa na koju se smesta etiketa nije NULL, onda se na
91      tu adresu kopira etiketa sa vrha steka. */
   if (etiketa != NULL)
93     strcpy(etiketa, (*adresa_vrha)->etiketa);

95   /* Element sa vrha steka se uklanja. */
   pomocni = *adresa_vrha;
97   *adresa_vrha = (*adresa_vrha)->sledeci;
   free(pomocni);
99
   return 1;
101 }

103 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na
    vrhu steka. Ukoliko je stek prazan, vraca NULL. */
105 char *vrh_steka(Cvor * vrh)
{
107   if (vrh == NULL)
       return NULL;
109   return vrh->etiketa;
}
111
/* Funkcija prikazuje stek pocev od vrha prema dnu. */
113 void prikazi_stek(Cvor * vrh)
{
115   for (; vrh != NULL; vrh = vrh->sledeci)
       printf("<%s>\n", vrh->etiketa);
117 }

119 /* Funkcija iz fajla na koji pokazuje f cita sledecu etiketu, i
    njeno ime upisuje u niz na koji pokazuje pokazivac etiketa.
121   Funkcija vraca EOF u slucaju da se dodje do kraja fajla pre
    nego sto se procita etiketa, vraca OTVORENA ako je procitana
123   otvorena etiketa, odnosno ZATVORENA ako je procitana
    zatvorena etiketa. */
125 int uzmi_etiketu(FILE * f, char *etiketa)
{
127   int c;
```

```
  int i = 0;
129
  /* 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. U HTML-u etikete BODY i body imaju isto znacenje,
     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 zatvarac */
        tip = ZATVORENA;
      } else {
        if (isalpha(c)) {
          /* Cita se otvarac */
          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 premasena
           dozvoljena duzina etikete, procitani karakter se
           smanjuje i smesta u etiketu. */
        etiketa[i++] = tolower(c);
      } else {
        /* U suprotnom, staje se sa citanjem etikete i stanje se
           menja. Korektno se zavrsava niska koja sadrzi
           procitanu etiketu i vraca se njen tip. */
        stanje = VAN_ETIKETE;
        etiketa[i] = '\0';
        return tip;
      }
      break;
    }
  }

  /* Doslo se do kraja datoteke pre nego sto je procitana
```

```
     naredna etiketa i vraca se EOF. */
  return EOF;
}

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(0);
  }

  /* Datoteka se otvara za citanje. */
  if ((f = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
            argv[1]);
    exit(1);
  }

  /* 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, tako da
       ih nije potrebno zatvoriti. NAPOMENA: U HTML-u postoje
       jos neke etikete koje koje nemaju sadrzaj (npr link).
       Pretpostavlja se da njih nema u HTML dokumentu, zbog
       jednostavnosti. */
    if (tip == OTVORENA) {
      if (strcmp(etiketa, "br") != 0
          && strcmp(etiketa, "hr") != 0
          && strcmp(etiketa, "meta") != 0)
        potisni_na_stek(&vrh, etiketa);
    }
    /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti
       da je u pitanju zatvaranje etikete koja je poslednja
       otvorena, a jos uvek nije zatvorena. Ova etiketa se mora
       nalaziti na vrhu steka. Ako je taj uslov ispunjen, skida
       se sa steka, jer je zatvorena. U suprotnom, pronadjena je
       nepravilnost i etikete nisu pravilno uparene. */
    else if (tip == ZATVORENA) {
      if (vrh_steka(vrh) != NULL
          && strcmp(vrh_steka(vrh), etiketa) == 0)
        skini_sa_steka(&vrh, NULL);
```

```
      else {
233     printf("Etikete nisu pravilno uparene\n(nadjena\
 etiketa </%s>", etiketa);
235     if (vrh_steka(vrh) != NULL)
          printf(", a poslednja otvorena etiketa je <%s>)\n",
237             vrh_steka(vrh));
        else
239       printf(" koja nije otvorena)\n");
        uparene = 0;
241     break;
      }
243   }
  }
245 /* Zavrseno je citanje datoteke i zatvara se. */
  fclose(f);
247
  /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi
249    trebalo da bude prazan. Ukoliko nije, tada postoje etikete
     koje su ostale otvorene. */
251 if (uparene) {
    if (vrh_steka(vrh) == NULL)
253   printf("Etikete su pravilno uparene!\n");
    else {
255   printf("Etikete nisu pravilno uparene\n(etiketa <%s> \
nije zatvorena)\n", vrh_steka(vrh));
257   /* Oslobadja se memorija zauzeta stekom. */
      oslobodi_stek(&vrh);
259 }
  }
261 return 0;
}
```

Rešenje 4.6

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

7 #define MAX 1000
  #define JMBG_DUZINA 14
9
  /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG
11    korisnika i opis njegovog zahteva. */
  typedef struct {
13  char jmbg[JMBG_DUZINA];
    char opis[MAX];
15 } Zahtev;

17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
```

```
   korisnika i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  Zahtev nalog;
  struct cvor *sledeci;
} Cvor;

Cvor *napravi_cvor(Zahtev * zahtev);

void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

void proveri_alokaciju(Cvor ** adresa_pocetka,
                       Cvor ** adresa_kraja, Cvor * novi);

void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                 Zahtev * zahtev);

int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

Zahtev *pocetak_reda(Cvor * pocetak);

void prikazi_red(Cvor * pocetak);

#endif
```

```
#include "red.h"

/* 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. Ako je doslo do greske,
   trebalo bi osloboditi ceo red. To je ostavljeno da to uradi
   funkcija koja je pozvala funkciju napravi_cvor, a gresku ce
   biti signalizirana vracanjem NULL. Funkciji se prosledjuje
   pokazivac na zahtev koji treba smestiti u nov 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)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  novi->nalog = *zahtev;
  novi->sledeci = NULL;
  return novi;
}

/* Funkcija prazni red */
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
  Cvor *pomocni = NULL;
```

```
29    while (*pocetak != NULL) {
        pomocni = *pocetak;
31      *pocetak = (*pocetak)->sledeci;
        free(pomocni);
33    }
      *kraj = NULL;
35  }

37  /* Funkcija proverava uspesnost alokacije memorije za cvor novi
       i ukoliko alokacija nije bila uspesna, oslobadja se sva
39     prethodno zauzeta memorija za listu cija pocetni cvor se
       nalazi na adresi adresa_pocetka. */
41  void proveri_alokaciju(Cvor ** adresa_pocetka,
                           Cvor ** adresa_kraja, Cvor * novi)
43  {
      if (novi == NULL) {
45      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_red(adresa_pocetka, adresa_kraja);
47      exit(EXIT_FAILURE);
      }
49  }

51  /* Funkcija dodaje na kraj reda novi fajl. */
    void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
53                   Zahtev * zahtev)
    {
55    Cvor *novi = napravi_cvor(zahtev);
      proveri_alokaciju(adresa_pocetka, adresa_kraja, novi);
57
      /* U red se uvek dodaje na kraj, ali zbog postojanja
59       pokazivaca na kraj, dodavanje na kraj je podjednako
         efikasno kao dodavanje na pocetak. */
61    if (*adresa_kraja != NULL) {
        (*adresa_kraja)->sledeci = novi;
63      *adresa_kraja = novi;
      } else {
65      /* Ako je red bio ranije prazan */
        *adresa_pocetka = novi;
67      *adresa_kraja = novi;
      }
69  }

71  /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji
       argument pokazivac razlicit od NULL, tada se u strukturu na
73     koju on pokazuje upisuje zahtev koji je upravo skinut sa reda
       dok u suprotnom ne upisuje nista. Funkcija vraca 0 ako je red
75     bio prazan ili 1 u suprotnom. */
    int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
77                    Zahtev * zahtev)
    {
79    Cvor *pomocni = NULL;
```

```
81    if (*adresa_pocetka == NULL)
        return 0;
83
      if (zahtev != NULL)
85      *zahtev = (*adresa_pocetka)->nalog;

87    pomocni = *adresa_pocetka;
      *adresa_pocetka = (*adresa_pocetka)->sledeci;
89    free(pomocni);

91    if (*adresa_pocetka == NULL)
        *adresa_kraja = NULL;
93
      return 1;
95  }

97  /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev
       korisnika na pocetku reda. Ukoliko je red prazan, vraca NULL.
99   */
    Zahtev *pocetak_reda(Cvor * pocetak)
101 {
      if (pocetak == NULL)
103     return NULL;

105   return &(pocetak->nalog);
    }
107
    /* Funkcija prikazuje red. */
109 void prikazi_red(Cvor * pocetak)
    {
111   for (; pocetak != NULL; pocetak = pocetak->sledeci)
        printf("%s %s\n",
113            (pocetak->nalog).jmbg, (pocetak->nalog).opis);

115   printf("\n");
    }
```

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

6   #define VREME_ZA_PAUZU 5

8   int main(int argc, char **argv)
    {
10    /* Red je prazan. */
      Cvor *pocetak = NULL, *kraj = NULL;
12    Zahtev nov_zahtev;
      Zahtev *sledeci = NULL;
14    char odgovor[3];
```

```
  int broj_usluzenih = 0;
  FILE *izlaz = fopen("izvestaj.txt", "a");

  if (izlaz == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke \
izvestaj.txt\n");
    exit(EXIT_FAILURE);
  }

  /* Sluzbenik evidentira korisnicke zahteve. */
  printf("Sluzbenik evidentira korisnicke zahteve unosenjem \
njihovog JMBG broja i opisa potrebne usluge:\n[CTRL+D za \
kraj]\n");

  /* Neophodan je poziv funkcije getchar da bi se i nov red
     nakon JMBG broja procitao i da bi fgets nakon toga
     procitala ispravan red sa opisom zahteva. */
  printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
  while (scanf("%s", nov_zahtev.jmbg) != EOF) {
    getchar();
    printf("\tOpis problema: ");
    fgets(nov_zahtev.opis, MAX - 1, stdin);
    /* Ako je poslednji karakter nov red, eliminise se. */
    if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
      nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
    dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
    printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
  }

  /* Datoteka vise nije potrebna i treba je zatvoriti. */
  fclose(izlaz);

  /* Dokle god ima korisnika u redu, treba ih usluziti. */
  while (1) {
    sledeci = pocetak_reda(pocetak);
    /* Ako nema nikog vise u redu, prekida se petlja. */
    if (sledeci == NULL)
      break;

    printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
           sledeci->jmbg);
    printf("sa zahtevom: %s\n", sledeci->opis);

    skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

    broj_usluzenih++;

    printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
    scanf("%s", odgovor);

    if (strcmp(odgovor, "Da") == 0)
      dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
```

```
    else
      fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
              nov_zahtev.jmbg, nov_zahtev.opis);

    if (broj_usluzenih == VREME_ZA_PAUZU) {
      printf("\nDa li je kraj smene? [Da/Ne] ");
      scanf("%s", odgovor);

      if (strcmp(odgovor, "Da") == 0)
        break;
      else
        broj_usluzenih = 0;
    }
  }

  /* Prethodno usluzivanje korisnika moze da se izvrsi i na
     sledeci nacin */
  /* while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
     printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
     nov_zahtev.jmbg); printf("sa zahtevom: %s\n",
     nov_zahtev.opis);

     broj_usluzenih++;

     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
     scanf("%s", odgovor);

     if (strcmp(odgovor, "Da") == 0) dodaj_u_red(&pocetak,
     &kraj, &nov_zahtev); else fprintf(izlaz, "JMBG: %s\tZahtev:
     %s\n", nov_zahtev.jmbg, nov_zahtev.opis);

     if (broj_usluzenih == VREME_ZA_PAUZU) { printf("\nDa li je
     kraj smene? [Da/Ne] "); scanf("%s", odgovor);

     if (strcmp(odgovor, "Da") == 0) break; else broj_usluzenih
     = 0; } } */

  /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
     neusluzenih korisnika, u tom slucaju treba osloboditi
     memoriju koju zauzima red sa neobradjenim zahtevima
     korisnika. */
  oslobodi_red(&pocetak, &kraj);

  return 0;
}
```

Rešenje 4.7

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

```
4  #define MAX_DUZINA 20

6  typedef struct _Element {
     unsigned broj_pojavljivanja;
8    char etiketa[20];
     struct _Element *sledeci;
10 } Element;

12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi
      cvor ili NULL ako alokacija nije uspesno izvrsena. */
14 Element *napravi_cvor(unsigned br, char *etiketa)
   {
16   Element *novi = (Element *) malloc(sizeof(Element));
     if (novi == NULL)
18     return NULL;

20   novi->broj_pojavljivanja = br;
     strcpy(novi->etiketa, etiketa);
22   novi->sledeci = NULL;
     return novi;
24 }

26 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente
      liste. */
28 void oslobodi_listu(Element ** glava)
   {
30   Element *pomocni = NULL;

32   while (*glava != NULL) {
       pomocni = (*glava)->sledeci;
34     free(*glava);
       *glava = pomocni;
36   }
   }
38
   /* Funkcija proverava uspesnost alokacije memorije za cvor novi
40    i ukoliko alokacija nije bila uspesna, oslobadja se sva
      prethodno zauzeta memorija za listu cija pocetni cvor se
42    nalazi na adresi glava. */
   void provera_alokacije(Element * novi, Element ** glava)
44 {
     if (novi == NULL) {
46     fprintf(stderr, "malloc() greska u funkciji \
   napravi_cvor()!\n");
48     oslobodi_listu(glava);
       exit(EXIT_FAILURE);
50   }
   }
52
   /* Funkcija dodaje novi cvor na pocetak liste. */
54 void dodaj_na_pocetak_liste(Element ** glava, unsigned br,
                              char *etiketa)
```

```
{
  Element *novi = napravi_cvor(br, etiketa);
  provera_alokacije(novi, glava);
  novi->sledeci = *glava;
  *glava = novi;
}

/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu.
   (NULL u suprotnom) */
Element *pretrazi_listu(Element * glava, char etiketa[])
{
  Element *tekuci;
  for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
    if (strcmp(tekuci->etiketa, etiketa) == 0)
      return tekuci;
  return NULL;
}

/* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Element * glava)
{
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s - %u\n", glava->etiketa,
           glava->broj_pojavljivanja);
}

int main(int argc, char **argv)
{
  if (argc != 2) {
    fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
datoteka.html!\n");
    exit(EXIT_FAILURE);
  }

  FILE *in = NULL;
  in = fopen(argv[1], "r");
  if (in == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  char c;
  int i = 0;
  char a[MAX_DUZINA];

  Element *glava = NULL;
  Element *trazeni = NULL;

  while ((c = fgetc(in)) != EOF) {

    if (c == '<') {
```

```
      /* Cita se zatvarac. */
      if ((c = fgetc(in)) == '/') {
        i = 0;
        while ((c = fgetc(in)) != '>')
          a[i++] = c;
      }
      /* Cita se otvarac. */
      else {
        i = 0;
        a[i++] = c;
        while ((c = fgetc(in)) != ' ' && c != '>')
          a[i++] = c;
      }
      a[i] = '\0';

      /* Trazi se ucitana etiketa medju postojecim cvorovima
         liste. Ukoliko ne postoji, dodaje se novi cvor za
         ucitanu etiketu sa brojem pojavljivanja 1, inace
         uvecava se broj pojavljivanja etikete. */
      trazeni = pretrazi_listu(glava, a);
      if (trazeni == NULL)
        dodaj_na_pocetak_liste(&glava, 1, a);
      else
        trazeni->broj_pojavljivanja++;
    }
  }

  fclose(in);

  ispisi_listu(glava);
  oslobodi_listu(&glava);

  return 0;
}
```

Rešenje 4.8

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

#define MAX_INDEKS 11
#define MAX_IME_PREZIME 21

typedef struct _Cvor {
  char broj_indeksa[MAX_INDEKS];
  char ime[MAX_IME_PREZIME];
  char prezime[MAX_IME_PREZIME];
  struct _Cvor *sledeci;
} Cvor;

```

```
/* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac
   na nov cvor ili NULL ukoliko alokacija nije prosla. */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;
  strcpy(novi->broj_indeksa, broj_indeksa);
  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
  novi->sledeci = NULL;
  return novi;
}

/* Funkcija oslobadja memoriju zauzetu za elemente liste. */
void oslobodi_listu(Cvor ** glava)
{
  if (*glava == NULL)
    return;
  oslobodi_listu(&(*glava)->sledeci);
  free(*glava);
  *glava = NULL;
}

void proveri_alokaciju(Cvor ** glava, Cvor * novi)
{
  if (novi == NULL) {
    fprintf(stderr, "Neuspela alokacija za nov cvor\n");
    oslobodi_listu(glava);
    exit(EXIT_FAILURE);
  }
}

/* Funkcija dodaje novi cvor na pocetak liste. */
void dodaj_na_pocetak_liste(Cvor ** glava, char *broj_indeksa,
                            char *ime, char *prezime)
{
  Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
  proveri_alokaciju(glava, novi);
  novi->sledeci = *glava;
  *glava = novi;
}

void ispisi_listu(Cvor * glava)
{
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
           glava->prezime);
}

/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu,
   u suprotnom vraca NULL. */
```

```
Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
{
  if (glava == NULL)
    return NULL;
  if (!strcmp(glava->broj_indeksa, broj_indeksa))
    return glava;
  return pretrazi_listu(glava->sledeci, broj_indeksa);
}

int main(int argc, char **argv)
{
  if (argc != 2) {
    fprintf(stderr, "Greska! Program se poziva sa: ./a.out \
studenti.txt!\n");
    exit(EXIT_FAILURE);
  }

  FILE *in = NULL;
  in = fopen(argv[1], "r");
  if (in == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
  char broj_indeksa[MAX_INDEKS];
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;

  /* Ucitavanje vrednosti u listu. */
  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) !=
         EOF)
    dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);

  fclose(in);

  while (scanf("%s", broj_indeksa) != EOF) {
    trazeni = pretrazi_listu(glava, broj_indeksa);
    if (trazeni == NULL)
      printf("ne\n");
    else
      printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
  }

  oslobodi_listu(&glava);

  return 0;
}
```

Rešenje 4.9

```c
#include<stdio.h>
#include<stdlib.h>
#include "601/lista.h"

Cvor *objedini(Cvor ** glava1, Cvor ** glava2)
{
  Cvor *l3 = NULL;
  Cvor **tek = &l3;

  if (*glava1 == NULL && *glava2 == NULL)
    return NULL;

  /* Ako je prva lista prazna, rezultat je druga lista. */
  if (*glava1 == NULL)
    return *glava2;

  /* Ako je druga lista prazna, rezultat je prva lista. */
  if (*glava2 == NULL)
    return *glava1;

  /* l3 pokazuje na pocetak nove liste, tj. na manji od brojeva
     sadrzanih u cvorovima na koje pokazuju glava1 i glava2. */
  l3 = ((*glava1)->vrednost < (*glava2)->vrednost) ? *glava1 :
       *glava2;


  while (*glava1 != NULL && *glava2 != NULL) {
    if ((*glava1)->vrednost < (*glava2)->vrednost) {
      *tek = *glava1;
      *glava1 = (*glava1)->sledeci;
    } else {
      *tek = *glava2;
      *glava2 = (*glava2)->sledeci;
    }
    tek = &((*tek)->sledeci);
  }

  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste,
     na rezultujucu listu treba nadovezati ostatak druge liste. */
  if (*glava1 == NULL)
    *tek = *glava2;

  else if (*glava2 == NULL)
    *tek = *glava1;

  return l3;
}

int main(int argc, char **argv)
{
  if (argc != 3) {
```

```
    fprintf(stderr,
53              "Greska! Program se poziva sa: ./a.out dat1.txt dat2.txt
    !\n");
    exit(EXIT_FAILURE);
55  }

57  FILE *in1 = NULL;
    in1 = fopen(argv[1], "r");
59  if (in1 == NULL) {
      fprintf(stderr,
61              "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
      exit(EXIT_FAILURE);
63  }

65  FILE *in2 = NULL;
    in2 = fopen(argv[2], "r");
67  if (in2 == NULL) {
      fprintf(stderr,
69              "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
      exit(EXIT_FAILURE);
71  }

73  int broj;
    Cvor *glava1 = NULL;
75  Cvor *glava2 = NULL;
    Cvor *l3 = NULL;
77
    /* Ucitavanje listi */
79  while (fscanf(in1, "%d", &broj) != EOF)
      dodaj_na_kraj_liste(&glava1, broj);
81  while (fscanf(in2, "%d", &broj) != EOF)
      dodaj_na_kraj_liste(&glava2, broj);
83
    l3 = objedini(&glava1, &glava2);
85
    /* Ispis rezultujuce liste. */
87  ispisi_listu(l3);
    oslobodi_listu(&l3);
89
    fclose(in1);
91  fclose(in2);
    return 0;
93  }
```

Rešenje 4.14

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

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   /* Alociramo memoriju */
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
18   if (novi == NULL)
       return NULL;
20
     /* Inicijalizujemo polja novog cvora. */
22   novi->broj = broj;
     novi->levo = NULL;
24   novi->desno = NULL;

26   /* Vracamo adresu novog cvora. */
     return novi;
28 }


30
   /* Funkcija koja proverava uspesnost kreiranja novog cvora
32    stabla */
   void proveri_alokaciju(Cvor * novi_cvor)
34 {
     /* Ukoliko je cvor neuspesno kreiran */
36   if (novi_cvor == NULL) {

38     /* Ispisuje se odgovarajuca poruka i prekida izvrsavanje
          programa */
40     fprintf(stderr, "Malloc greska za novi cvor!\n");
       exit(EXIT_FAILURE);
42   }
   }
44


46 /* c) Funkcija koja dodaje zadati broj u stablo */
   void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
48 {
     /* Ako je stablo prazno */
50   if (*adresa_korena == NULL) {

52     /* Kreiramo novi cvor */
       Cvor *novi = napravi_cvor(broj);
54     proveri_alokaciju(novi);

56     /* I proglasavamo ga korenom stabla */
       *adresa_korena = novi;
```

```
58      return;
    }
60
    /* U suprotnom trazimo odgovarajucu poziciju za zadati broj */
62
    /* Ako je zadata vrednost manja od vrednosti korena */
64  if (broj < (*adresa_korena)->broj)

66    /* Dodajemo broj u levo podstablo */
      dodaj_u_stablo(&(*adresa_korena)->levo, broj);
68
    else
70    /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa
         ga dodajemo u desno podstablo */
72    dodaj_u_stablo(&(*adresa_korena)->desno, broj);
}
74

76 /* d) Funkcija koja proverava da li se zadati broj nalazi u
      stablu */
78 Cvor *pretrazi_stablo(Cvor * koren, int broj)
{
80   /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
      */
82   if (koren == NULL)
       return NULL;
84
     /* Ako je trazena vrednost sadrazana u korenu */
86   if (koren->broj == broj) {

88     /* Prekidamo pretragu */
       return koren;
90   }

92   /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
     if (broj < koren->broj)
94
       /* Pretragu nastavljamo u levom podstablu */
96     return pretrazi_stablo(koren->levo, broj);

98   else
       /* U suprotnom, pretragu nastavljamo u desnom podstablu */
100    return pretrazi_stablo(koren->desno, broj);
}
102

104 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u
      stablu */
106 Cvor *pronadji_najmanji(Cvor * koren)
{
108   /* Ako je stablo prazno, prekidamo pretragu */
      if (koren == NULL)
```

```
110     return NULL;

112   /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze
          se levo od njega */
114
      /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
116        najmanju vrednost */
      if (koren->levo == NULL)
118     return koren;

120   /* Inace, pretragu treba nastaviti u levom podstablu */
      return pronadji_najmanji(koren->levo);
122 }


124
    /* f) Funkcija pronalazi cvor koji sadrzi najvecu vrednost u
126      stablu */
    Cvor *pronadji_najveci(Cvor * koren)
128 {
      /* Ako je stablo prazno, prekidamo pretragu */
130   if (koren == NULL)
        return NULL;
132
      /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze
134        se desno od njega */

136   /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
          najvecu vrednost */
138   if (koren->desno == NULL)
        return koren;
140
      /* Inace, pretragu treba nastaviti u desnom podstablu */
142   return pronadji_najveci(koren->desno);
    }
144


146 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
    void obrisi_element(Cvor ** adresa_korena, int broj)
148 {
      Cvor *pomocni_cvor = NULL;
150
      /* ako je stablo prazno, brisanje nije primenljivo pa mozemo
152        prekinuti rad funkcije */
      if (*adresa_korena == NULL)
154     return;

156   /* Ako je vrednost koju treba obrisati manja od vrednosti u
          korenu stabla, ona se eventualno nalazi u levom podstablu,
158       pa treba rekurzivno primeniti postupak na levo podstablo.
          Koren ovako modifikovanog stabla je nepromenjen. */
160   if (broj < (*adresa_korena)->broj) {
        obrisi_element(&(*adresa_korena)->levo, broj);
```

```
162       return;
          }
164
          /* Ako je vrednost koju treba obrisati veca od vrednosti u
166          korenu stabla, ona se eventualno nalazi u desnom podstablu
             pa treba rekurzivno primeniti postupak na desno podstablo.
168          Koren ovako modifikovanog stabla je nepromenjen. */
          if ((*adresa_korena)->broj < broj) {
170         obrisi_element(&(*adresa_korena)->desno, broj);
            return;
172       }

174       /* Slede podslucajevi vezani za slucaj kada je vrednost u
             korenu jednaka broju koji se brise (tj. slucaj kada treba
176          obrisati koren) */

178       /* Ako koren nema sinova, tada se on prosto brise, i rezultat
             je prazno stablo (vracamo NULL) */
180       if ((*adresa_korena)->levo == NULL
              && (*adresa_korena)->desno == NULL) {
182         free(*adresa_korena);
            *adresa_korena = NULL;
184         return;
          }
186
          /* Ako koren ima samo levog sina, tada se brisanje vrsi tako
188          sto obrisemo koren, a novi koren postaje levi sin */
          if ((*adresa_korena)->levo != NULL
190           && (*adresa_korena)->desno == NULL) {
            pomocni_cvor = (*adresa_korena)->levo;
192         free(*adresa_korena);
            *adresa_korena = pomocni_cvor;
194         return;
          }
196
          /* Ako koren ima samo desnog sina, tada se brisanje vrsi tako
198          sto obrisemo koren, a novi koren postaje desni sin */
          if ((*adresa_korena)->desno != NULL
200           && (*adresa_korena)->levo == NULL) {
            pomocni_cvor = (*adresa_korena)->desno;
202         free(*adresa_korena);
            *adresa_korena = pomocni_cvor;
204         return;
          }
206
          /* Slucaj kada koren ima oba sina. Tada se brisanje vrsi na
208          sledeci nacin: - najpre se potrazi sledbenik korena (u
             smislu poretka) u stablu. To je upravo po vrednosti
210          najmanji cvor u desnom podstablu. On se moze pronaci npr.
             funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
212          vrednost tog cvora, a u taj cvor se smesti vrednost korena
             (tj. broj koji se brise).  - Onda se prosto rekurzivno
```

```
214       pozove funkcija za brisanje na desno podstablo. S obzirom
          da u njemu treba obrisati najmanji element, a on zasigurno
216       ima najvise jednog potomka, jasno je da ce njegovo brisanje
          biti obavljeno na jedan od jednostavnijih nacina koji su
218       gore opisani. */
    pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
220   (*adresa_korena)->broj = pomocni_cvor->broj;
    pomocni_cvor->broj = broj;
222   obrisi_element(&(*adresa_korena)->desno, broj);
  }
224

226 /* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
      postablo - Koren - Desno podstablo ) */
228 void ispisi_stablo_infiksno(Cvor * koren)
  {
230   /* Ako stablo nije prazno */
    if (koren != NULL) {
232
      /* Prvo ispisujemo sve cvorove levo od korena */
234     ispisi_stablo_infiksno(koren->levo);

236     /* Ispisujemo vrednost u korenu */
      printf("%d ", koren->broj);
238
      /* Na kraju ispisujemo cvorove desno od korena */
240     ispisi_stablo_infiksno(koren->desno);
    }
242 }

244
  /* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren -
246    Levo podstablo - Desno podstablo ) */
  void ispisi_stablo_prefiksno(Cvor * koren)
248 {
    /* Ako stablo nije prazno */
250   if (koren != NULL) {

252     /* Prvo ispisujemo vrednost u korenu */
      printf("%d ", koren->broj);
254
      /* Ispisujemo sve cvorove levo od korena */
256     ispisi_stablo_prefiksno(koren->levo);

258     /* Na kraju ispisujemo sve cvorove desno od korena */
      ispisi_stablo_prefiksno(koren->desno);
260   }
  }
262

264 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
    podstablo - Desno postablo - Koren) */
```

```
266 void ispisi_stablo_postfiksno(Cvor * koren)
    {
268   /* Ako stablo nije prazno */
      if (koren != NULL) {
270
        /* Prvo ispisujemo sve cvorove levo od korena */
272     ispisi_stablo_postfiksno(koren->levo);

274     /* Ispisujemo sve cvorove desno od korena */
        ispisi_stablo_postfiksno(koren->desno);
276
        /* Na kraju ispisujemo vrednost u korenu */
278     printf("%d ", koren->broj);
      }
280 }


282
    /* k) Funkcija koja oslobadja memoriju zauzetu stablom.  */
284 void oslobodi_stablo(Cvor ** adresa_korena)
    {
286   /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
      if (*adresa_korena == NULL)
288     return;

290   /* U suprotnom rekurzivno oslobadjamo memoriju koje zauzima
         najpre levo, a zatim i desno podstablo */
292   oslobodi_stablo(&(*adresa_korena)->levo);
      oslobodi_stablo(&(*adresa_korena)->desno);
294
      /* Oslobadjamo memoriju koju zauzima koren */
296   free(*adresa_korena);

298   /* I proglasavamo stablo praznim */
      *adresa_korena = NULL;
300 }

302 int main()
    {
304   Cvor *koren;
      int n;
306   Cvor *trazeni_cvor;

308   /* Proglasavamo stablo praznim */
      koren = NULL;
310
      /* Dodajemo vrednosti u stablo */
312   printf("Unesite brojeve (CTRL+D za kraj unosa): ");
      while (scanf("%d", &n) != EOF) {
314     dodaj_u_stablo(&koren, n);
      }
316
      /* Generisemo trazene ispise: */
```

```
318   printf("\nInfiksni ispis: ");
      ispisi_stablo_infiksno(koren);
320   printf("\nPrefiksni ispis: ");
      ispisi_stablo_prefiksno(koren);
322   printf("\nPostfiksni ispis: ");
      ispisi_stablo_postfiksno(koren);
324
      /* Demonstriramo rad funkcije za pretragu */
326   printf("\nTrazi se broj: ");
      scanf("%d", &n);
328   trazeni_cvor = pretrazi_stablo(koren, n);
      if (trazeni_cvor == NULL)
330     printf("Broj se ne nalazi u stablu!\n");
      else
332     printf("Broj se nalazi u stablu!\n");

334   /* Demonstriramo rad funkcije za brisanje */
      printf("Brise se broj: ");
336   scanf("%d", &n);
      obrisi_element(&koren, n);
338   printf("Rezultujuce stablo: ");
      ispisi_stablo_infiksno(koren);
340   printf("\n");

342   /* Oslobadjamo memoriju zauzetu stablom */
      oslobodi_stablo(&koren);
344
      /* Prekidamo sa programom */
346   return 0;
}
```

Rešenje 4.15

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <string.h>
   #include <ctype.h>
5
   #define MAX 50
7
   /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
9     pojavljivanja i redom pokazivace na levo i desno podstablo */
   typedef struct cvor {
11   char *rec;
     int brojac;
13   struct cvor *levo;
     struct cvor *desno;
15 } Cvor;

17 /* Funkcija koja kreira novi cvora stabla */
   Cvor *napravi_cvor(char *rec)
```

```
{
  /* Alociramo memoriju za novi cvor */
  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
  if (novi_cvor == NULL)
    return NULL;

  /* Alociramo memoriju za zadatu rec: potrebno je rezervisati
     memoriju za svaki karakter reci ukljucujuci i terminirajucu
     nulu */
  novi_cvor->rec =
      (char *) malloc((strlen(rec) + 1) * sizeof(char));
  if (novi_cvor->rec == NULL) {
    free(novi_cvor);
    return NULL;
  }

  /* Inicijalizujemo polja u novom cvoru */
  strcpy(novi_cvor->rec, rec);
  novi_cvor->brojac = 1;
  novi_cvor->levo = NULL;
  novi_cvor->desno = NULL;

  /* Vracamo adresu novog cvora */
  return novi_cvor;
}

/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
void proveri_alokaciju(Cvor * novi_cvor)
{
  /* Ukoliko je cvor neuspesno kreiran */
  if (novi_cvor == NULL) {
    /* Ispisuje se odgovarajuca poruka i prekida izvrsavanje
       programa */
    fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
  }
}

/* Funkcija koja dodaje novu rec u stablo. */
void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
    /* Kreiramo novi cvor */
    Cvor *novi = napravi_cvor(rec);
    proveri_alokaciju(novi);

    /* i proglasavamo ga korenom stabla */
    *adresa_korena = novi;
    return;
  }
```

```
71
   /* U suprotnom trazimo odgovarajucu poziciju za novu rec */
73
   /* Ako je rec leksikografski manju od reci u korenu ubacujemo
75      je u levo podstablo */
   if (strcmp(rec, (*adresa_korena)->rec) < 0)
77   dodaj_u_stablo(&(*adresa_korena)->levo, rec);

79 else
     /* Ako je rec leksikografski veca od reci u korenu ubacujemo
81      je u desno podstablo */
   if (strcmp(rec, (*adresa_korena)->rec) > 0)
83   dodaj_u_stablo(&(*adresa_korena)->desno, rec);

85 else
     /* Ako je rec jednaka reci u korenu, uvecavamo njen broj
87      pojavljivanja */
     (*adresa_korena)->brojac++;
89 }

91 /* Funkcija koja oslobadja memoriju zauzetu stablom */
   void oslobodi_stablo(Cvor ** adresa_korena)
93 {
   /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
95 if (*adresa_korena == NULL)
     return;
97
   /* Inace ... */
99 /* Oslobadjamo memoriju zauzetu levim podstablom */
   oslobodi_stablo(&(*adresa_korena)->levo);
101
   /* Oslobadjamo memoriju zauzetu desnim podstablom */
103 oslobodi_stablo(&(*adresa_korena)->desno);

105 /* Oslobadjamo memoriju zauzetu korenom */
   free((*adresa_korena)->rec);
107 free(*adresa_korena);

109 /* Proglasavamo stablo praznim */
   *adresa_korena = NULL;
111 }


113
   /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
115    (rec sa najvecim brojem pojavljivanja) */
   Cvor *nadji_najfrekventniju_rec(Cvor * koren)
117 {
   Cvor *max, *max_levo, *max_desno;
119
   /* Ako je stablo prazno, prekidamo sa pretragom */
121 if (koren == NULL)
     return NULL;
```

```
123
    /* Pronalazimo najfrekventniju reci u levom podstablu */
125   max_levo = nadji_najfrekventniju_rec(koren->levo);

127   /* Pronalazimo najfrekventniju reci u desnom podstablu */
    max_desno = nadji_najfrekventniju_rec(koren->desno);
129
    /* Trazimo maksimum vrednosti pojavljivanja reci iz levog
131    podstabla, korena i desnog podstabla */
    max = koren;
133   if (max_levo != NULL && max_levo->brojac > max->brojac)
      max = max_levo;
135   if (max_desno != NULL && max_desno->brojac > max->brojac)
      max = max_desno;
137
    /* Vracamo adresu cvora sa najvecim brojacem */
139   return max;
}
141

143 /* Funkcija koja ispisuje reci iz stabla u leksikografskom
     poretku pracene brojem pojavljivanja */
145 void prikazi_stablo(Cvor * koren)
{
147   /* Ako je stablo prazno, zavrsavamo sa ispisom */
    if (koren == NULL)
149     return;

151   /* Zbog leksikografskog poretka, prvo ispisujemo sve reci iz
      levog podstabla */
153   prikazi_stablo(koren->levo);

155   /* Zatim ispisujemo rec iz korena */
    printf("%s: %d\n", koren->rec, koren->brojac);
157
    /* I nastavljamo sa ispisom reci iz desnog podstabla */
159   prikazi_stablo(koren->desno);
}
161

163 /* Funkcija ucitava sledecu rec iz zadate datoteke i upisuje je
     u niz rec. Maksimalna duzina reci je odredjena argumentom
165    max. Funkcija vraca EOF ako nema vise reci ili 0 u suprotnom.
     Rec je niz malih ili velikih slova. */
167 int procitaj_rec(FILE * f, char rec[], int max)
{
169   /* karakter koji citamo */
    int c;
171
    /* indeks pozicije na koju se smesta procitani karakter */
173   int i = 0;
```

```
175   /* Sve dok ima mesta za jos jedan karakter u nizu i dokle god
         nismo stigli do kraja datoteke... */
177   while (i < max - 1 && (c = fgetc(f)) != EOF) {
        /* Proveravamo da li je procitani karakter slovo */
179     if (isalpha(c))

181       /* Ako jeste, smestamo ga u niz - pritom vrsimo konverziju
             u mala slova jer program treba da bude neosetljiv na
183          razliku izmedju malih i velikih slova */
          rec[i++] = tolower(c);
185
        else
187       /* Ako nije, proveravamo da li smo procitali barem jedno
             slovo nove rece */
189       /* Ako jesmo prekidamo sa citanjem */
        if (i > 0)
191       break;

193     /* U suprotnom idemo na sledecu iteraciju */
      }
195
      /* Dodajemo na rec terminirajucu nulu */
197   rec[i] = '\0';

199   /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */
      return i > 0 ? 0 : EOF;
201 }

203 int main(int argc, char **argv)
    {
205   Cvor *koren = NULL, *max;
      FILE *f;
207   char rec[MAX];

209   /* Proveravamo da li je navedeno ime datoteke prilikom
         pokretanja programa */
211   if (argc < 2) {
        fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
213     exit(EXIT_FAILURE);
      }
215
      /* Otvaramo datoteku iz koje citamo reci */
217   if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "fopen() greska pri otvaranju %s\n",
219             argv[1]);
        exit(EXIT_FAILURE);
221   }

223   /* Ucitavamo reci iz datoteke i smestamo u binarno stablo
         pretrage. */
225   while (procitaj_rec(f, rec, MAX) != EOF)
        dodaj_u_stablo(&koren, rec);
```

```
227
  /* Posto smo zavrsili sa citanjem reci zatvaramo datoteku */
229 fclose(f);

231 /* Prikazujemo sve reci iz teksta i brojeve njihovih
       pojavljivanja. */
233 prikazi_stablo(koren);

235 /* Pronalazimo najfrekventniju rec */
  max = nadji_najfrekventniju_rec(koren);
237
  /* Ako takve reci nema... */
239 if (max == NULL)

241   /* Ispisujemo odgovarajuce obavestenje */
    printf("U tekstu nema reci!\n");
243
  else
245   /* Inace, ispisujemo broj pojavljivanja reci */
    printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
247          max->rec, max->brojac);

249 /* Oslobadjamo dinamicki alociran prostor za stablo */
  oslobodi_stablo(&koren);
251
  /* Zavrsavamo sa programom */
253 return 0;
}
```

Rešenje 4.16

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

6 #define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
     broj telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
   char ime_i_prezime[MAX_IME_I_PREZIME];
14 char telefon[MAX_CIFARA];
   struct cvor *levo;
16 struct cvor *desno;
} Cvor;
18
/* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
```

```
{
  /* Alociramo memoriju za novi cvor */
  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
  if (novi_cvor == NULL)
    return NULL;

  /* Inicijalizujemo polja u novom cvoru */
  strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
  strcpy(novi_cvor->telefon, telefon);
  novi_cvor->levo = NULL;
  novi_cvor->desno = NULL;

  /* Vracamo adresu novog cvora */
  return novi_cvor;
}


/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
void proveri_alokaciju(Cvor * novi_cvor)
{
  /* Ukoliko je cvor neuspesno kreiran */
  if (novi_cvor == NULL) {
    /* Ispisuje se odgovarajuca poruka i prekida izvrsavanje
       programa */
    fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
  }
}


/* Funkcija koja dodaje novu osobu i njen broj telefona u
   stablo. */
void
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
               char *telefon)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
    /* Kreiramo novi cvor */
    Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
    proveri_alokaciju(novi);

    /* i proglasavamo ga korenom stabla */
    *adresa_korena = novi;
    return;
  }

  /* U suprotnom trazimo odgovarajucu poziciju za novi unos */
  /* Kako pretragu treba vrsiti po imenu i prezimenu, stablo
     treba da bude pretrazivacko po ovom polju */
  /* Ako je zadato ime i prezime leksikografski manje od imena i
```

```
     prezimena sadrzanog u korenu, podatke dodajemo u levo
     podstablo */
  if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
      < 0)
    dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
                   telefon);

  else
    /* Ako je zadato ime i prezime leksikografski vece od imena
       i prezimena sadrzanog u korenu, podatke kodajemo u desno
       podstablo */
  if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
    dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
                   telefon);
}


/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
  if (*adresa_korena == NULL)
    return;

  /* Inace ... */
  /* Oslobadjamo memoriju zauzetu levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);

  /* Oslobadjamo memoriju zauzetu desnim podstablom */
  oslobodi_stablo(&(*adresa_korena)->desno);

  /* Oslobadjamo memoriju zauzetu korenom */
  free(*adresa_korena);

  /* Proglasavamo stablo praznim */
  *adresa_korena = NULL;
}


/* Funkcija koja ispisuje imenik u leksikografskom poretku */
/* Napomena: ova funkcija nije trazena u zadatku ali se moze
   koristiti za proveru da li je stablo lepo kreirano ili ne */
void prikazi_stablo(Cvor * koren)
{
  /* Ako je stablo prazno, zavrsavamo sa ispisom */
  if (koren == NULL)
    return;

  /* Zbog leksikografskog poretka, prvo ispisujemo podatke iz
     levog podstabla */
  prikazi_stablo(koren->levo);

```

```
  /* Zatim ispisujemo podatke iz korena */
  printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

  /* I nastavljamo 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)
{
  int c;
  int i = 0;

  /* Linije datoteke koje obradjujemo su formata Ime Prezime
     BrojTelefona */
  /* Preskacemo eventualne praznine sa pocetka linije datoteke */
  while ((c = fgetc(f)) != EOF && isspace(c));

  /* Prvo procitano slovo upisujemo 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 cemo citanje forsirati sve dok ne naidjemo
     na cifru. Pri tom cemo voditi racuna da li ima dovoljno
     mesta za smestanje procitanog karaktera i da slucajno ne
     dodjemo 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;
  }

  /* Upisujemo terminirajucu nulu na mesto poslednjeg procitanog
     blanko karaktera */
  ime_i_prezime[--i] = '\0';

  /* I pocinjemo sa citanjem broja telefona */
  i = 0;

  /* Upisujemo cifru koju smo vec procitali */
  telefon[i++] = c;

```

```
  /* I citamo peostale cifre - naznaka kraja ce nam biti pojava
178      karaktera cije prisustvo nije dozvoljeno u broju telefona */
  while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
180   if (c == '/' || c == '-' || isdigit(c))
        telefon[i++] = c;
182
      else
184     break;

186 /* Upisujemo terminirajucu nulu */
  telefon[i] = '\0';
188
  /* Vracamo 0 ako smo procitali kontakt, EOF u suprotnom */
190 return !feof(f) ? 0 : EOF;
}
192

194 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
     prezimenom */
196 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
{
198 /* Ako je imenik prazan, zavrsavamo sa pretragom */
  if (koren == NULL)
200   return NULL;

202 /* Ako je trazeno ime i prezime sadrzano u korenu, takodje
      zavrsavamo sa pretragom */
204 if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
      return koren;
206
  /* Ako je zadato ime i prezime leksikografski manje od
208      vrednosti u korenu pretragu nastavljamo levo */
  if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
210   return pretrazi_imenik(koren->levo, ime_i_prezime);

212 else
      /* u suprotnom, pretragu nastavljamo desno */
214   return pretrazi_imenik(koren->desno, ime_i_prezime);
}
216
int main(int argc, char **argv)
218 {
  char ime_datoteke[MAX_IME_DATOTEKE];
220 Cvor *koren = NULL;
  Cvor *trazeni;
222 FILE *f;
  char ime_i_prezime[MAX_IME_I_PREZIME];
224 char telefon[MAX_CIFARA];
  char c;
226 int i;

228 /* Ucitavamo ime datoteke i pripremamo je za citanje */
```

```
  printf("Unesite ime datoteke: ");
  scanf("%s", ime_datoteke);
  if ((f = fopen(ime_datoteke, "r")) == NULL) {
    fprintf(stderr, "fopen() greska prilikom otvaranja
%s\n", ime_datoteke);
    exit(EXIT_FAILURE);
  }

  /* Ucitavamo podatke iz datoteke i smestamo kontakte u binarno
     stablo pretrage. */
  while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
    dodaj_u_stablo(&koren, ime_i_prezime, telefon);

  /* Posto smo zavrsili sa citanjem podataka zatvaramo datoteku */
  fclose(f);

  /* Omogucavamo pretragu imenika */
  while (1) {
    /* Ucitavamo ime i prezime */
    printf("Unesite ime i prezime: ");
    i = 0;
    while ((c = getchar()) != '\n')
      ime_i_prezime[i++] = c;
    ime_i_prezime[i] = '\0';

    /* Ako je korisnik uneo naznaku za kraj pretrage,
       obustavljamo funkcionalnost */
    if (strcmp(ime_i_prezime, "KRAJ") == 0)
      break;

    /* Inace, ispisujemo rezultat pretrage */
    trazeni = pretrazi_imenik(koren, ime_i_prezime);
    if (trazeni == NULL)
      printf("Broj nije u imeniku!\n");

    else
      printf("Broj je: %s \n", trazeni->telefon);
  }

  /* Oslobadjamo memoriju zauzetu imenikom */
  oslobodi_stablo(&koren);

  /* Zavrsavamo sa programom */
  return 0;
}
```

Rešenje 4.17

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

```
4
#define MAX 51
6
/* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8    studenta, ukupan uspsh, uspeh iz matematike, uspeh iz
     maternjeg jezika i redom pokazivace na levo i desno podstablo
10  */
typedef struct cvor_stabla {
12   char ime[MAX];
   char prezime[MAX];
14   double uspeh;
   double matematika;
16   double jezik;
   struct cvor_stabla *levo;
18   struct cvor_stabla *desno;
} Cvor;
20
/* Funkcija kojom se kreira cvor stabla */
22 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                     double matematika, double jezik)
24 {
   /* Alociramo memoriju za novi cvor */
26   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
   if (novi == NULL)
28     return NULL;

30   /* Inicijalizujemo polja strukture */
   strcpy(novi->ime, ime);
32   strcpy(novi->prezime, prezime);
   novi->uspeh = uspeh;
34   novi->matematika = matematika;
   novi->jezik = jezik;
36   novi->levo = NULL;
   novi->desno = NULL;
38
   /* Vracamo adresu kreiranog cvora */
40   return novi;
}
42
/* Funkcija kojom se proverava uspesnost alociranja memorije */
44 void proveri_alokaciju(Cvor * novi_cvor)
{
46   /* Ako alokacije nije uspesna */
   if (novi_cvor == NULL) {
48     /* Ispisujemo poruku i prekidamo sa izvrsavanjem */
     fprintf(stderr, "Malloc greska za novi cvor!\n");
50     exit(EXIT_FAILURE);
   }
52
}
54
/* Funkcija kojom se oslobadja memorija zauzeta stablom */
```

```
56 void oslobodi_stablo(Cvor ** koren)
   {
58   /* Ako je stablo prazno, nema potrebe za oslobadjanjem
        memorije */
60   if (*koren == NULL)
       return;
62
     /* oslobadjamo memoriju zauzetu levim podstablom */
64   oslobodi_stablo(&(*koren)->levo);

66   /* oslobadjamo memoriju zauzetu desnim podstablom */
     oslobodi_stablo(&(*koren)->desno);
68
     /* oslobadjamo memoriju zauzetu korenom */
70   free(*koren);

72   /* proglasavamo stablo praznim */
     *koren = NULL;
74 }

76 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
   void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
78                       double uspeh, double matematika,
                        double jezik)
80 {
     /* Ako je stablo prazno */
82   if (*koren == NULL) {
       /* Kreiramo novi cvor */
84     Cvor *novi =
           napravi_cvor(ime, prezime, uspeh, matematika, jezik);
86     proveri_alokaciju(novi);

88     /* I proglasavamo ga korenom stabla */
       *koren = novi;
90
       return;
92   }

94   /* Inace, dodajemo cvor u stablo tako da bude sortiran po
        ukupnom broju poena */
96   if (uspeh + matematika + jezik >
        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
98     dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                      matematika, jezik);
100  else
       dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
102                   matematika, jezik);
   }
104

106 /* Funkcija ispisuje sadrzaj stabla - ukoliko je vrednost
       argumenta polozili jednaka 0 ispisuju se informacije o
```

```
108    ucenicima koji nisu polozili prijemni, a ako je vrednost
       argumenta razlicita od nule, ispisuju se informacije o
110    ucenicima koji su polozili prijemni */
void stampaj(Cvor * koren, int polozili)
112 {
  /* Stablo je prazno - prekidamo sa ispisom */
114   if (koren == NULL)
    return;
116
  /* Stampamo informacije iz levog podstabla */
118   stampaj(koren->levo, polozili);

120   /* Stampamo informacije iz korenog cvora */
  if (polozili && koren->matematika + koren->jezik >= 10)
122     printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
            koren->prezime, koren->uspeh, koren->matematika,
124         koren->jezik,
            koren->uspeh + koren->matematika + koren->jezik);
126   else if (!polozili && koren->matematika + koren->jezik < 10)
    printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
128         koren->prezime, koren->uspeh, koren->matematika,
            koren->jezik,
130         koren->uspeh + koren->matematika + koren->jezik);

132   /* Stampamo informacije iz desnog podstabla */
  stampaj(koren->desno, polozili);
134 }


136
/* Funkcija koja odredjuje koliko studenata nije polozilo
138    prijemni ispit */
int nisu_polozili(Cvor * koren)
140 {
  /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
142   if (koren == NULL)
    return 0;
144
  /* Pretragu vrsimo i u levom i u desnom podstablu - ako uslov
146     za polaganje nije ispunjen za koreni cvor, broj studenata
     uvecavamo za 1 */
148   if (koren->matematika + koren->jezik < 10)
    return 1 + nisu_polozili(koren->levo) +
150       nisu_polozili(koren->desno);

152   return nisu_polozili(koren->levo) +
      nisu_polozili(koren->desno);
154 }

156 int main(int argc, char **argv)
{
158   FILE *in;
  Cvor *koren;
```

```
160   char ime[MAX], prezime[MAX];
      double uspeh, matematika, jezik;
162
      /* Otvaramo datoteku sa rezultatima sa prijemnog za citanje */
164   in = fopen("prijemni.txt", "r");
      if (in == NULL) {
166     fprintf(stderr, "Greska prilikom citanja podataka!\n");
        exit(EXIT_FAILURE);
168   }

170   /* Citamo podatke i dodajemo ih u stablo */
      koren = NULL;
172   while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
                  &matematika, &jezik) != EOF) {
174     dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
                       jezik);
176   }

178   /* Zatvaramo datoteku */
      fclose(in);
180
      /* Stampamo prvo podatke o ucenicima koji su polozili prijemni
182    */
      stampaj(koren, 1);
184
      /* Liniju iscrtavamo samo ako postoje ucenici koji nisu
186       polozili prijemni */
      if (nisu_polozili(koren) != 0)
188     printf("---------------------------------------\n");

190   /* Stampamo podatke o ucenicima koji nisu polozili prijemni */
      stampaj(koren, 0);
192
      /* Oslobadjamo memoriju zauzetu stablom */
194   oslobodi_stablo(&koren);

196   /* Zavrsavamo sa programom */
      return 0;
198 }
```

Rešenje 4.18

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

5  #define MAX_NISKA 51
   #define MAX_DATUM 3
7
   /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i
9     prezime osobe, dan, mesec i godinu 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;
  int godina;
  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, int godina)
{
  /* Alociramo memoriju */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizujemo polja strukture */
  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
  novi->dan = dan;
  novi->mesec = mesec;
  novi->godina = godina;
  novi->levo = NULL;
  novi->desno = NULL;

  /* Vracamo adresu novog cvora */
  return novi;
}

/* Funkcija koja proverava uspesnost alokacije */
void proveri_alokaciju(Cvor * novi_cvor)
{
  /* Ako memorija nije uspesno alocirana */
  if (novi_cvor == NULL) {
    /* Ispisujemo poruku i prekidamo izvrsavanje programa */
    fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
  }
}

/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** koren)
{
  /* Stablo je prazno */
  if (*koren == NULL)
    return;

  /* Oslobadjamo memoriju zauzetu levim podstablom (ako postoji)
```

```
   */
63  if ((*koren)->levo)
      oslobodi_stablo(&(*koren)->levo);
65
    /* Oslobadjamo memoriju zauzetu desnim podstablom (ako
67     postoji) */
    if ((*koren)->desno)
69    oslobodi_stablo(&(*koren)->desno);

71  /* Oslobadjamo memoriju zauzetu korenom */
    free(*koren);
73
    /* Proglasavamo stablo praznim */
75  *koren = NULL;
}
77
/* Funkcija koja dodaje novi cvor u stablo - stablo treba da
79    bude uredjeno po datumu */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
81                    int dan, int mesec, int godina)
{
83  /* Ako je stablo prazno */
    if (*koren == NULL) {
85
      /* Kreiramo novi cvor */
87    Cvor *novi_cvor =
          napravi_cvor(ime, prezime, dan, mesec, godina);
89    proveri_alokaciju(novi_cvor);

91    /* I proglasavamo ga korenom */
      *koren = novi_cvor;
93
      return;
95  }

97  /* Kako se ne unosi godina za pretragu, stablo uredjujemo samo
       po mesecu (i danu u okviru istog meseca) */
99  if (mesec < (*koren)->mesec)
      dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
101                  godina);
    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
103   dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
                    godina);
105 else
      dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
107                  godina);
}
109
/* Funkcija vrsi pretragu stabla i vraca cvor sa trazenim
111   datumom (null ako takav ne postoji). u promenljivu pom ce
      biti smesten prvi datum (dan i mesec) veci od trazenog datuma
113   (null ako takav ne postoji)
```

```
115  */
     Cvor *pretrazi(Cvor * koren, int dan, int mesec)
117  {
       /* Stablo je prazno, obustavljamo pretragu */
119    if (koren == NULL)
         return NULL;
121
       /* Nasli smo trazeni datum u stablu */
123    if (koren->dan == dan && koren->mesec == mesec)
         return koren;
125
       /* Ako je mesec trazenog datuma manji od meseca sadrzanog u
127       korenu ili ako su meseci isti ali je dan trazenog datuma
          manji od aktuelnog datuma, pretrazujemo levo podstablo -
129       pre toga svakako proveravamo da li leva grana postoji - ako
          ne postoji treba da vratimo prvi sledeci, a to je bas
131       vrednost uocenog korena */
       if (mesec < koren->mesec
133        || (mesec == koren->mesec && dan < koren->dan)) {
         if (koren->levo == NULL)
135        return koren;
         else
137        return pretrazi(koren->levo, dan, mesec);
       }
139
       /* inace, nastavljamo pretragu u desnom delu */
141    return pretrazi(koren->desno, dan, mesec);
     }
143

145  int main(int argc, char **argv)
     {
147    FILE *in;
       Cvor *koren;
149    Cvor *slavljenik;
       char ime[MAX_NISKA], prezime[MAX_NISKA];
151    int dan, mesec, godina;

153    /* Proveravamo da li je zadato ime ulazne datoteke */
       if (argc < 2) {
155      /* Ako nije, ispisujemo poruku i prekidamo sa izvrsavanjem
            programa */
157      printf("Nedostaje ime ulazne datoteke!\n");
         return 0;
159    }

161    /* Inace, pripremamo datoteku za citanje */
       in = fopen(argv[1], "r");
163    if (in == NULL) {
         fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
165      exit(EXIT_FAILURE);
```

```
  }

  /* I popunjavamo podacima stablo */
  koren = NULL;
  while (fscanf
         (in, "%s %s %d.%d.%d.", ime, prezime, &dan, &mesec,
          &godina) != EOF)
    dodaj_u_stablo(&koren, ime, prezime, dan, mesec, godina);

  /* I zatvaramo datoteku */
  fclose(in);

  /* Omogucavamo pretragu podataka */
  while (1) {

    /* Ucitavamo novi datum */
    printf("Unesite datum: ");
    if (scanf("%d.%d.", &dan, &mesec) == EOF)
      break;

    /* Pretrazujemo stablo */
    slavljenik = pretrazi(koren, dan, mesec);

    /* Ispisujemo pronadjene podatke */
    if (slavljenik == NULL) {
      printf("Nema podataka o ovim ni o sledecem rodjendanu.\n");
      continue;
    }

    /* Slucaj kada smo pronasli prave podatke */
    if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
      printf("Slavljenik: %s %s\n", slavljenik->ime,
             slavljenik->prezime);
      continue;
    }

    /* Slucaj kada smo pronasli podatke o prvom sledecem
       rodjendanu */
    printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
           slavljenik->prezime, slavljenik->dan,
           slavljenik->mesec);
  }

  /* Oslobadjamo memoriju zauzetu stablom */
  oslobodi_stablo(&koren);

  /* Prekidamo sa izvrsavanjem programa */
  return 0;
}
```

Rešenje 4.19

```c
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
typedef struct cvor {
  int broj;
  struct cvor *levo, *desno;
} Cvor;

/* Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
Cvor *napravi_cvor(int broj);

/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla. */
void proveri_alokaciju(Cvor * novi_cvor);

/* Funkcija koja dodaje zadati broj u stablo */
void dodaj_u_stablo(Cvor ** adresa_korena, int broj);

/* Funkcija koja proverava da li se zadati broj nalazi u stablu */
Cvor *pretrazi_stablo(Cvor * koren, int broj);

/* Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
Cvor *pronadji_najmanji(Cvor * koren);

/* Funkcija koja pronalazi cvor koji sadrzi najveci vrednost u
   stablu */
Cvor *pronadji_najveci(Cvor * koren);

/* Funkcija koja brise cvor stabla koji sadrzi zadati broj */
void obrisi_element(Cvor ** adresa_korena, int broj);

/* Funkcija koja ispisuje sadrzaj stabla u infiksnoj notaciji
   (levo podstablo - koren - desno podstablo) */
void prikazi_stablo(Cvor * koren);

/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena);

#endif
```

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

/* Funkcija kojom se kreira novi cvor stabla koji sadrzi zadatu
   vrednost */
Cvor *napravi_cvor(int broj)
```

```
{
  /* Alociramo memoriju za novi cvor */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;
  /* Inicijalizujemo polja cvora */
  novi->broj = broj;
  novi->levo = NULL;
  novi->desno = NULL;
  /* Vracamo adresu novog cvora */
  return novi;
}

/* Funkcija koja proverava uspesnost kreiranja novog cvora
   stabla */
void proveri_alokaciju(Cvor * novi_cvor)
{
  /* Ukoliko je cvor neuspesno kreiran */
  if (novi_cvor == NULL) {
    /* Ispisuje se odgovarajuca poruka i prekida izvrsavanje
       programa */
    fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
  }
}

/* Funkcija koja dodaje novi broj u stablo. */
void dodaj_u_stablo(Cvor ** koren, int broj)
{
  /* Ako je stablo prazno */
  if (*koren == NULL) {
    /* Kreiramo novi cvor */
    Cvor *novi = napravi_cvor(broj);
    proveri_alokaciju(novi);
    /* i proglasavamo ga korenom stabla */
    *koren = novi;
    return;
  }
  /* U suprotnom trazimo odgovarajucu poziciju za novi broj */
  /* Ako je broj manji od vrednosti sadrzane u korenu, ubacujemo
     ga u levo podstablo */
  if (broj < (*koren)->broj)
    dodaj_u_stablo(&(*koren)->levo, broj);
  else
    /* Inace, ubacujemo broj u desno podstablo */
    dodaj_u_stablo(&(*koren)->desno, broj);
}

/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** koren)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
```

```
  if (*koren == NULL)
61    return;
  /* Inace ... */
63  /* Oslobadjamo memoriju zauzetu levom podstablom */
  if ((*koren)->levo)
65    oslobodi_stablo(&(*koren)->levo);
  /* Oslobadjamo memoriju zauzetu desnom podstablom */
67  if ((*koren)->desno)
    oslobodi_stablo(&(*koren)->desno);
69  /* Oslobadjamo memoriju zauzetu korenom */
  free(*koren);
71  /* Proglasavamo stablo praznim */
  *koren = NULL;
73}

75Cvor *pronadji_najmanji(Cvor * koren)
{
77  /* ako je stablo prazno, prekidamo pretragu */
  if (koren == NULL)
79    return NULL;
  /* vrednosti koje su manje od vrednosti u korenu stabla nalaze
81     se levo od njega */
  /* ako je koren cvor koji nema levo podstablo, onda on sadrzi
83     najmanju vrednost */
  if (koren->levo == NULL)
85    return koren;
  /* inace, pretragu treba nastaviti u levom podstablu */
87  return pronadji_najmanji(koren->levo);
}
89
Cvor *pronadji_najveci(Cvor * koren)
91{
  /* ako je stablo prazno, prekidamo pretragu */
93  if (koren == NULL)
    return NULL;
95  /* vrednosti koje su vece od vrednosti u korenu stabla nalaze
     se desno od njega */
97  /* ako je koren cvor koji nema desno podstablo, onda on sadrzi
     najvecu vrednost */
99  if (koren->desno == NULL)
    return koren;
101 /* inace, pretragu treba nastaviti u desnom podstablu */
  return pronadji_najveci(koren->desno);
103}

105/* Funkcija brise element iz stabla ciji je broj upravo jednak
   broju n. Funkcija azurira koren stabla u pozivajucoj
107   funkciji, jer u ovoj funkciji koren moze biti promenjen u
   funkciji. */
109void obrisi_element(Cvor ** adresa_korena, int n)
{
111 Cvor *pomocni = NULL;
```

```
  /* Izlaz iz rekurzije: ako je stablo prazno, nema sta da se
113     brise */
  if (*adresa_korena == NULL)
115   return;
  /* Ako je vrednost broja veca od vrednosti u korenu stablua,
117     tada se broj eventualno nalazi u desnom podstablu, pa treba
      rekurzivno primeniti postupak na desno podstablo. Koren
119     ovako modifikovanog stabla je nepromenjen. */
  if ((*adresa_korena)->broj < n) {
121   obrisi_element(&(*adresa_korena)->desno, n);
    return;
123 }
  /* Ako je vrednost broja manja od vrednosti korena, tada se
125     broj eventualno nalazi u levom podstablu, pa treba
      rekurzivno primeniti postupak na levo podstablo. Koren
127     ovako modifikovanog stabla je nepromenjen. */
  if ((*adresa_korena)->broj > n) {
129   obrisi_element(&(*adresa_korena)->levo, n);
    return;
131 }
  /* Slede podslucajevi vezani za slucaj kada je vrednost u
133     korenu jednaka broju koji se brise (tj. slucaj kada treba
      obrisati koren) */
135 /* Ako koren nema sinova, tada se on prosto brise, i rezultat
      je prazno stablo (vracamo NULL) */
137 if ((*adresa_korena)->levo == NULL
      && (*adresa_korena)->desno == NULL) {
139   free(*adresa_korena);
    *adresa_korena = NULL;
141   return;
  }
143 /* Ako koren ima samo levog sina, tada se brisanje vrsi tako
      sto obrisemo koren, a novi koren postaje levo sin */
145 if ((*adresa_korena)->levo != NULL
      && (*adresa_korena)->desno == NULL) {
147   pomocni = (*adresa_korena)->levo;
    free(*adresa_korena);
149   *adresa_korena = pomocni;
    return;
151 }
  /* Ako koren ima samo desnog sina, tada se brisanje vrsi tako
153     sto obrisemo koren, a novi koren postaje desno sin */
  if ((*adresa_korena)->desno != NULL
155     && (*adresa_korena)->levo == NULL) {
    pomocni = (*adresa_korena)->desno;
157   free(*adresa_korena);
    *adresa_korena = pomocni;
159   return;
  }
161 /* Slucaj kada koren ima oba sina. Tada se brisanje vrsi na
      sledeci nacin: - najpre se potrazi sledbenik korena (u
163     smislu poretka) u stablu. To je upravo po vrednosti
```

```
     najmanji cvor u desnom podstablu. On se moze pronaci npr.
165  funkcijom pronadji_najmanji(). - Nakon toga se u koren
     smesti vrednost tog cvora, a u taj cvor se smesti vrednost
167  korena (tj. broj koji se brise). - Onda se prosto
     rekurzivno pozove funkcija za brisanje na desno podstablo.
169  S obzirom da u njemu treba obrisati najmanji element, a on
     definitivno ima najvise jednog potomka, jasno je da ce
171  njegovo brisanje biti obavljeno na jedan od jednostavnijih
     nacina koji su gore opisani. */
173  pomocni = pronadji_najmanji((*adresa_korena)->desno);
  (*adresa_korena)->broj = pomocni->broj;
175  pomocni->broj = n;
  obrisi_element(&(*adresa_korena)->desno, n);
177 }

179 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje
    elemente u rastucem poretku) */
181 void prikazi_stablo(Cvor * koren)
 {
183  /* izlaz iz rekurzije */
  if (koren == NULL)
185    return;
  prikazi_stablo(koren->levo);
187  printf("%d ", koren->broj);
  prikazi_stablo(koren->desno);
189 }

191 Cvor *pretrazi_stablo(Cvor * koren, int broj)
 {
193  /* ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
     */
195  if (koren == NULL)
    return NULL;
197  /* ako je trazena vrednost sadrazana u korenu */
  if (koren->broj == broj) {
199    /* prekidamo pretragu */
    return koren;
201  }
  /* inace, ako je broj manji od vrednosti sadrzane u korenu */
203  if (broj < koren->broj)
    /* pretragu nastavljamo u levom podstablu */
205    return pretrazi_stablo(koren->levo, broj);
  else
207    /* u suprotnom, pretragu nastavljamo u desnom podstablu */
    return pretrazi_stablo(koren->desno, broj);
209 }
```

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Ukljucujemo biblioteku za rad sa stablima - pogledati uvodni
5     zadatak ove glave */
```

```c
#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, mozemo da zakljucimo da se razlikuju */
  if (koren1->broj != koren2->broj)
    return 0;

  /* inace, proveravamo da li vazi i jednakost u levih
     podstabala i desnih podstabala */
  return (identitet(koren1->levo, koren2->levo)
          && identitet(koren1->desno, koren2->desno));
}

int main()
{
  int broj;
  Cvor *koren1, *koren2;

  koren1 = NULL;
  /* ucitavamo elemente prvog stabla */
  printf("Prvo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    dodaj_u_stablo(&koren1, broj);
    scanf("%d", &broj);
  }

  koren2 = NULL;
  /* ucitavamo elemente drugog stabla */
  printf("Drugo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    dodaj_u_stablo(&koren2, broj);
    scanf("%d", &broj);
  }

  /* pozivamo funkciju koja ispituje identitet stabala */
```

```
  if (identitet(koren1, koren2))
59  printf("Stabla jesu identicna.\n");
  else
61  printf("Stabla nisu identicna.\n");

63 /* oslobadjamo memoriju zauzetu stablima */
  oslobodi_stablo(&koren1);
65 oslobodi_stablo(&koren2);

67 /* zavrsavamo sa radom programa */
  return 0;
69 }
```

Rešenje 4.20

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

4 /* Uklucujemo biblioteku za rad sa stablima */
#include "stabla.h"
6
/* Funkcija kreira novo stablo identicno stablu koje je dato
8    korenom. */
void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
10 {
  /* Izlaz iz rekurzije: ako je stablo prazno nema sta da se
12    kopira */
  if (koren == NULL) {
14   *duplikat = NULL;
    return;
16  }

18  /* Dupliramo koren stabla i postavljamo ga da bude koren novog
    stabla */
20  *duplikat = napravi_cvor(koren->broj);
  proveri_alokaciju(*duplikat);
22
  /* Rekurzivno dupliramo levo podstablo i njegovu adresu cuvamo
24    u pokazivacu na levo podstablo korena duplikata. */
  kopiraj_stablo(koren->levo, &(*duplikat)->levo);
26
  /* Rekurzivno dupliramo desno podstablo i njegovu adresu
28    cuvamo u pokazivacu na desno podstablo korena duplikata. */
  kopiraj_stablo(koren->desno, &(*duplikat)->desno);
30 }

32 /* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
   dobija modifikacijom prvog stabla */
34 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
{
36  /* Ako drugo stablo nije prazno */
```

```
  if (koren2 != NULL) {
    /* dodajemo njegov koren u prvo stablo */
    dodaj_u_stablo(adresa_korena1, koren2->broj);

    /* rekurzivno racunamo uniju levog i desnog podstabla drugog
       stabla sa prvim stablom */
    kreiraj_uniju(adresa_korena1, koren2->levo);
    kreiraj_uniju(adresa_korena1, koren2->desno);
  }
}

/* Funkcija izracunava presek dva stabla - rezultujuce stablo se
   dobija modifikacijom prvog stabla */
void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
{
  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
   */
  if (*adresa_korena1 == NULL)
    return;

  /* Kreiramo presek levog i desnog podstabla sa drugim stablom,
     tj. iz levog i desnog podstabla prvog stabla brisemo sve
     one elemente koji ne postoje u drugom stablu */
  kreiraj_presek(&(*adresa_korena1)->levo, koren2);
  kreiraj_presek(&(*adresa_korena1)->desno, koren2);

  /* Ako se koren prvog stabla ne nalazi u drugom stablu tada ga
     uklanjamo iz prvog stabla */
  if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
    obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
}

/* Funkcija izracunava razliku dva stabla - rezultujuce stablo
   se dobija modifikacijom prvog stabla */
void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
{
  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo
   */
  if (*adresa_korena1 == NULL)
    return;

  /* Kreiramo razliku levog i desnog podstabla sa drugim
     stablom, tj. iz levog i desnog podstabla prvog stabla
     brisemo sve one elemente koji postoje i u drugom stablu */
  kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
  kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

  /* Ako se koren prvog stabla nalazi i u drugom stablu tada ga
     uklanjamo iz prvog stabla */
  if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
    obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
}
```

```
90 int main()
   {
92   Cvor *koren1;
     Cvor *koren2;
94   Cvor *pomocni = NULL;
     int n;
96
     /* Ucitavamo elemente prvog stabla: */
98   koren1 = NULL;
     printf("Prvo stablo: ");
100  while (scanf("%d", &n) != EOF) {
       dodaj_u_stablo(&koren1, n);
102  }

104  /* Ucitavamo elemente drugog stabla: */
     koren2 = NULL;
106  printf("Drugo stablo: ");
     while (scanf("%d", &n) != EOF) {
108    dodaj_u_stablo(&koren2, n);
     }
110
     /* Kreiramo uniju stabala: prvo napravimo kopiju prvog stabla
112     kako bi mogli da ga iskoristimo i za preostale operacije */
     kopiraj_stablo(koren1, &pomocni);
114  kreiraj_uniju(&pomocni, koren2);
     printf("Unija: ");
116  prikazi_stablo(pomocni);
     putchar('\n');
118
     /* Oslobadjamo stablo za rezultatom operacije */
120  oslobodi_stablo(&pomocni);

122  /* Kreiramo presek stabala: prvo napravimo kopiju prvog stabla
        kako bi mogli da ga iskoristimo i za preostale operacije; */
124  kopiraj_stablo(koren1, &pomocni);
     kreiraj_presek(&pomocni, koren2);
126  printf("Presek: ");
     prikazi_stablo(pomocni);
128  putchar('\n');

130  /* Oslobadjamo stablo za rezultatom operacije */
     oslobodi_stablo(&pomocni);
132
     /* Kreiramo razliku stabala: prvo napravimo kopiju prvog
134     stabla kako bi mogli da ga iskoristimo i za preostale
        operacije; */
136  kopiraj_stablo(koren1, &pomocni);
     kreiraj_razliku(&pomocni, koren2);
138  printf("Razlika: ");
     prikazi_stablo(pomocni);
140  putchar('\n');
```

```
142   /* Oslobadjamo stablo za rezultatom operacije */
      oslobodi_stablo(&pomocni);
144
      /* Oslobadjamo i polazna stabla */
146   oslobodi_stablo(&koren2);
      oslobodi_stablo(&koren1);
148
      /* Zavrsavamo sa programom */
150   return 0;
}
```

Rešenje 4.21

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Ukljucujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  #define MAX 50

9  /* Funkcija koja obilazi stablo sa leva na desno i smesta
      vrednosti cvorova u niz. Povratna vrednost funkcije je broj
11    vrednosti koje su smestene u niz. */
   int kreiraj_niz(Cvor * koren, int a[])
13 {
     int r, s;
15
     /* Stablo je prazno - u niz je smesteno 0 elemenata */
17   if (koren == NULL)
       return 0;
19
     /* Dodajemo u niz elemente iz levog podstabla */
21   r = kreiraj_niz(koren->levo, a);

23   /* Tekuca vrednost promenljive r je broj elemenata koji su
        upisani u niz i na osnovu nje mozemo odrediti indeks novog
25      elementa */

27   /* Smestamo vrednost iz korena */
     a[r] = koren->broj;
29
     /* Dodajemo elemente iz desnog podstabla */
31   s = kreiraj_niz(koren->desno, a + r + 1);

33   /* Racunamo indeks na koji treba smestiti naredni element */
     return r + s + 1;
35 }

37 /* Funkcija sortira niz tako sto najpre elemente niza smesti u
```

```
   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva
   u desno.

   Ovaj nacin sortiranja primer sortiranja koje nije "u mestu "
   kao sto je to slucaj sa ostalim prethodno opisanim
   algoritmima sortiranja, jer se sortiranje vrsi u pomocnoj
   dinamickoj strukturi, a ne razmenom elemenata niza. */
void sortiraj(int a[], int n)
{
  int i;
  Cvor *koren;

  /* Kreiramo stablo smestanjem elemenata iz niza u stablo */
  koren = NULL;
  for (i = 0; i < n; i++)
    dodaj_u_stablo(&koren, a[i]);

  /* Infiksnim obilaskom stabla elemente iz stabla prepisujemo u
     niz a */
  kreiraj_niz(koren, a);

  /* Vise nam stablo nije potrebno i oslobadjamo memoriju */
  oslobodi_stablo(&koren);
}

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

  /* Ucitavamo dimenziju i elemente niza */
  printf("n: ");
  scanf("%d", &n);
  if (n < 0 || n > MAX) {
    printf("Greska: pogresna dimenzija niza!\n");
    return 0;
  }

  printf("a: ");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Pozivamo funkciju za sortiranje */
  sortiraj(a, n);

  /* Ispisujemo rezultat */
  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\n");

  /* Prekidamo sa programom */
  return 0;
```

```
}
```

Rešenje 4.22

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

/* Ukljucujemo biblioteku 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 dodajemo 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;

  /* Proveravamo da li je tekuci cvor list */
  if (koren->levo == NULL && koren->desno == NULL)
    /* i ako jeste vracamo 1 - to ce kasnije zbog rekurzivnih
       poziva uvecati broj listova za 1 */
    return 1;

  /* U suprotnom prebrojavamo listove koje se nalaze u
     podstablima */
  return broj_listova(koren->levo) + broj_listova(koren->desno);
}

/* c) Funckija 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)
    /* Stampamo ga */
    printf("%d ", koren->broj);

  /* Nastavljamo sa stampanjem pozitivnih listova u podstablima */
  pozitivni_listovi(koren->levo);
  pozitivni_listovi(koren->desno);
}

/* d) Funkcija koja izracunava zbir cvorova stabla */
int zbir_cvorova(Cvor * koren)
{
  /* Ako je stablo prazno, zbir cvorova je 0 */
  if (koren == NULL)
    return 0;

  /* Inace, zbir cvorova stabla izracunavamo kao zbir korena i
     svih elemenata u podstablima */
  return koren->broj + zbir_cvorova(koren->levo) +
      zbir_cvorova(koren->desno);
}

/* e) Funckija koja izracunava najveci element stabla. */
Cvor *najveci_element(Cvor * koren)
{
  /* Ako je stablo prazno, obustavljamo pretragu */
  if (koren == NULL)
    return NULL;

  /* Zbog prirode pretrazivackog stabla, sigurni smo da su
     vrednosti vece od korena u desnom podstablu */

  /* Ako desnog podstabla nema */
  if (koren->desno == NULL)
    /* Najveca vrednost je koren */
    return koren;

  /* Inace, najvecu vrednost trazimo jos desno */
  return najveci_element(koren->desno);
}

/* f) Funckija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
    return 0;

  /* Izracunavamo dubinu levog podstabla */
  int dubina_levo = dubina_stabla(koren->levo);

  /* Izracunavamo dubinu desnog podstabla */
```

```
  int dubina_desno = dubina_stabla(koren->desno);

  /* dubina stabla odgovara vecoj od dubina podstabala - 1
     dodajemo jer racunamo i koren */
  return dubina_levo >
      dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* g) Funckija koja izracunava broj cvorova na i-tom nivou */
int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
{
  /* ideja je da ste spustamo kroz stablo sve dok ne stignemo do
     trazenog nivoa */

  /* Ako nema vise cvorova, ne mozemo da se spustamo niz stablo */
  if (koren == NULL)
    return 0;

  /* Ako smo stigli do trazenog nivoa, vracamo 1 - to ce kasnije
     zbog rekurzivnih poziva uvecati broj pojavljivanja za 1 */
  if (i == 0)
    return 1;

  /* inace, spustamo se jedan nivo nize i u levom i u desnom
     postablu */
  return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
      + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
}

/* h) Funckija koja ispisuje sve elemente na i-tom nivou */
void ispis_nivo(Cvor * koren, int i)
{
  /* ideja je slicna ideji iz prethodne funkcije */
  /* nema vise cvorova, ne mozemo da se spustamo kroz stablo */
  if (koren == NULL)
    return;

  /* ako smo na trazenom nivou - ispisujemo vrednost */
  if (i == 0) {
    printf("%d ", koren->broj);
    return;
  }
  /* inace, spustamo se jedan nivo nize i u levom i u desnom
     podstablu */
  ispis_nivo(koren->levo, i - 1);
  ispis_nivo(koren->desno, i - 1);
}

/* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
   nivou stabla */
Cvor *max_nivo(Cvor * koren, int i)
{
```

```
  /* Ako je stablo prazno, obustavljamo pretragu */
  if (koren == NULL)
    return NULL;

  /* Ako smo na trazenom nivou, takodje prekidamo pretragu */
  if (i == 0)
    return koren;

  /* Pronalazimo maksimum sa i-tog nivoa levog podstabla */
  Cvor *a = max_nivo(koren->levo, i - 1);

  /* Pronalazimo maksimum sa i-tog nivoa desnog podstabla */
  Cvor *b = max_nivo(koren->desno, i - 1);

  /* Trazimo i vracamo maksimum izracunatih vrednosti */
  if (a == NULL && b == NULL)
    return NULL;
  if (a == NULL)
    return b;
  if (b == NULL)
    return a;
  return a->broj > b->broj ? a : b;
}

/* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
int zbir_nivo(Cvor * koren, int i)
{
  /* Ako je stablo prazno, zbir je nula */
  if (koren == NULL)
    return 0;

  /* Ako smo na trazenom nivou, vracamo vrednost */
  if (i == 0)
    return koren->broj;

  /* Inace, spustamo se jedan nivo nize i trazimo sume iz levog
     i desnog podstabla */
  return zbir_nivo(koren->levo, i - 1) + zbir_nivo(koren->desno,
                                                   i - 1);
}


/* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje
   su manje ili jednake od date vrednosti x */
int suma(Cvor * koren, int x)
{
  /* Ako je stablo prazno, zbir je nula */
  if (koren == NULL)
    return 0;

  /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
     prirode pretrazivackog stabla treba obici i levo i desno
```

```
     podstablo */
  if (koren->broj < x)
    return koren->broj + suma(koren->levo,
                              x) + suma(koren->desno, x);

  /* Inace, racunamo samo sumu vrednosti iz levog podstabla jer
     medju njima jedino moze biti onih koje zadovoljavaju uslov */
  return suma(koren->levo, x);
}

int main(int argc, char **argv)
{
  /* Analiziramo argumente komandne linije */
  if (argc != 3) {
    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out nivo
    broj_za_pretragu\n");
    exit(EXIT_FAILURE);
  }
  int i = atoi(argv[1]);
  int x = atoi(argv[2]);

  /* Kreiramo stablo */
  Cvor *koren = NULL;
  int broj;
  while (scanf("%d", &broj) != EOF)
    dodaj_u_stablo(&koren, broj);

  /* ispisujemo rezultat rada funkcija */
  printf("broj cvorova: %d\n", br_cvorova(koren));
  printf("broj listova: %d\n", br_listova(koren));
  printf("pozitivni listovi: ");
  pozitivni_listovi(koren);
  printf("zbir cvorova: %d\n", suma_cvorova(koren));

  if (najveci_element(koren) == NULL)
    printf("najveci element: ne postoji\n");
  else
    printf("najveci element: %d\n",
           najveci_element(koren)->broj);

  printf("dubina stabla: %d\n", dubina_stabla(koren));
  printf("\n");
  printf("broj cvorova na %d. nivou: %d\n", i,
         cvorovi_nivo(koren, i));
  printf("elementi na %d. nivou: ", i);
  ispis_nivo(koren, i);
  printf("\n");
  if (max_nivo(koren, i) == NULL)
    printf("Nema elemenata na %d. nivou!\n", i);
  else
    printf("maksimalni na %d. nivou: %d\n", i,
```

```
255             max_nivo(koren, i)->broj);

257   printf("zbir na %d. nivou: %d\n", i, zbir_nivo(koren, i));
      printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
259           suma(koren, x));

261   /* Oslobadjamo memoriju zauzetu stablom */
      oslobodi_stablo(&koren);
263
      /* Prekidamo izvrsavanje programa */
265   return 0;
}
```

Rešenje 4.23

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Ukljucujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* Funckija koja izracunava dubinu stabla */
   int dubina_stabla(Cvor * koren)
9  {
     /* Dubina praznog stabla je 0 */
11   if (koren == NULL)
       return 0;
13
     /* Izracunavamo dubinu levog podstabla */
15   int dubina_levo = dubina_stabla(koren->levo);

17   /* Izracunavamo dubinu desnog podstabla */
     int dubina_desno = dubina_stabla(koren->desno);
19
     /* Dubina stabla odgovara vecoj od dubina podstabala - 1
21      dodajemo jer racunamo i koren */
     return dubina_levo >
23       dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
   }
25
   /* Funckija koja ispisuje sve elemente na i-tom nivou */
27 void ispisi_nivo(Cvor * koren, int i)
   {
29   /* Ideja je slicna ideji iz prethodne funkcije */

31   /* Nema vise cvorova, ne mozemo da se spustamo kroz stablo */
     if (koren == NULL)
33     return;

35   /* Ako smo na trazenom nivou - ispisujemo vrednost */
     if (i == 0) {
```

```
    printf("%d ", koren->broj);
    return;
  }
  /* Inace, spustamo se 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 izracunavamo dubinu stabla */
  int dubina;
  dubina = dubina_stabla(koren);

  /* Ispisujemo 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;

  /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    dodaj_u_stablo(&koren, broj);
  }

  /* Ispisujemo stablo po nivoima */
  ispisi_stablo_po_nivoima(koren);

  /* Oslobadjamo memoriju zauzetu stablom */
  oslobodi_stablo(&koren);

  /* Prekidamo izvrsavanje programa */
  return 0;
}
```

Rešenje 4.24

Rešenje 4.25

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

/* Ukljucujemo biblioteku za rad sa stablima */
#include "stabla.h"

/* Funckija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
    return 0;

  /* Izracunavamo dubinu levog podstabla */
  int dubina_levo = dubina_stabla(koren->levo);

  /* Izracunavamo dubinu desnog podstabla */
  int dubina_desno = dubina_stabla(koren->desno);

  /* Dubina stabla odgovara vecoj od dubina podstabala - 1
     dodajemo jer racunamo i koren */
  return dubina_levo >
      dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
   AVL stablo */
int avl(Cvor * koren)
{
  int dubina_levo, dubina_desno;

  /* Ako je stablo prazno, zaustavljamo brojanje */
  if (koren == NULL) {
    return 0;
  }

  /* Izracunavamo dubinu levog podstabla korena */
  dubina_levo = dubina_stabla(koren->levo);

  /* Izracunavamo dubinu desnog podstabla korena */
  dubina_desno = dubina_stabla(koren->desno);

  /* Ako je uslov za AVL stablo ispunjen */
  if (abs(dubina_desno - dubina_levo) <= 1) {
    /* Racunamo broj avl cvorova u levom i desnom podstablu i
       uvecavamo za jedan iz razloga sto koren ispunjava uslov */
    return 1 + avl(koren->levo) + avl(koren->desno);
  } else {
    /* Inace, racunamo samo broj avl cvorova u podstablima */
    return avl(koren->levo) + avl(koren->desno);
  }
```

```
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int broj;

  /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    dodaj_u_stablo(&koren, broj);
  }

  /* Racunamo i ispisujemo broj AVL cvorova */
  printf("%d\n", avl(koren));

  /* Oslobadjamo memoriju zauzetu stablom */
  oslobodi_stablo(&koren);

  /* Prekidamo izvrsavanje programa */
  return 0;
}
```

Rešenje 4.26

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

/* Ukljucujemo biblioteku za rad sa stablima */
#include "stabla.h"

/* Funkcija proverava da li je zadato binarno stablo celih
   pozitivnih brojeva heap. Ideja koju cemo implementirati 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 heap. Zato ce funkcija vracati
   maksimalne vrednosti iz uocenog podstabala ili vrednost -1
   ukoliko zakljucimo da stablo nije heap. */
int heap(Cvor * koren)
{
  int max_levo, max_desno;

  /* Prazno sablo je heap. */
  if (koren == NULL) {
    /* posto je 0 najmanji pozitivan broj, moze nam posluziti
       kao indikator */
    return 0;

  }
  /* Ukoliko je stablo list ... */
```

```
27    if (koren->levo == NULL && koren->desno == NULL) {
        /* ... vracamo njegovu vrednost */
29      return koren->broj;
      }
31
      /* Proveravamo svojstvo za levo podstablo. */
33    max_levo = heap(koren->levo);

35    /* Proveravamo svojstvo za desno podstablo. */
      max_desno = heap(koren->desno);
37    /* Ako levo ili desno podstablo uocenog cvora nije heap, onda
         nije ni celo stablo. */
39    if (max_levo == -1 || max_desno == -1) {
        return -1;
41    }

43    /* U suprotonom proveravamo da li svojstvo vazi za uoceni
         cvor. */
45    if (koren->broj > max_levo && koren->broj > max_desno) {
        /* ako vazi, vracamo vrednost korena */
47      return koren->broj;
      }
49
      /* u suprotnom zakljucujemo da stablo nije heap */
51    return -1;
  }
53
  int main(int argc, char **argv)
55{
    Cvor *koren;
57  int heap_indikator;

59  /* Kreiramo stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2 7
     */
61  koren = NULL;
    koren = napravi_cvor(100);
63  koren->levo = napravi_cvor(19);
    koren->levo->levo = napravi_cvor(17);
65  koren->levo->levo->levo = napravi_cvor(2);
    koren->levo->levo->desno = napravi_cvor(7);
67  koren->levo->desno = napravi_cvor(3);
    koren->desno = napravi_cvor(36);
69  koren->desno->levo = napravi_cvor(25);
    koren->desno->desno = napravi_cvor(1);
71
    /* pozivamo funkciju kojom proveravamo da li je stablo heap */
73  heap_indikator = heap(koren);

75  /* i ispisujemo rezultat */
    if (heap_indikator == -1) {
77    printf("Zadato tablo nije heap\n");
    } else {
```

```
79      printf("Zadato stablo je heap!\n");
    }
81
    /* Oslobadjamo memoriju zauzetu stablom. */
83  oslobodi_stablo(&koren);

85  /* Zavrsavamo sa programom */
    return 0;
87  }
```

Rešenje 4.27

Glava 5

Ispitni rokovi

5.1 Programiranje 2, praktični deo ispita, jun 2015.

Zadatak 5.1

Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od `50` karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost `-1` i prekinuti izvršavanje programa.

Test Test 1

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
 5
 Programiranje
 Matematika
 12345
 dInAmiCnArEc
 Ispit
Izlaz:
 Ispit
 Matematika
 Programiranje
```

Test Test 2

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
  2
  maksimalano
  poena
Izlaz:
```

Test Test 3

```
Poziv: ./a.out ulaz.txt
Problem: datoteka
ulaz.txt ne postoji
Izlaz:
  -1
```

Test Test 4

```
Poziv: ./a.out
Izlaz:
  -1
```

[Rešenje 5.1]

Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na n-tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj n, a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test Test 1

```
Ulaz:
 2 8 10 3 6 14 13 7 4 0
Izlaz:
 20
```

Test Test 2

```
Ulaz:
 0 50 14 5 2 4 56 8 52 7 1 0
Izlaz:
 50
```

[Rešenje 5.2]

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A, a zatim i elementi matrice A. Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od `50`. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

Test 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 Test 2

```
Ulaz:
 2 3
 0 0 -5
 1 2 -4
 Izlaz:
 2
```

Test Test 3

```
Ulaz:
 -2
Izlaz (na stderr):
 -1
```

[Rešenje 5.3]

5.2 Programiranje 2, praktični deo ispita, jul 2015.

Zadatak 5.4

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 Test 1

```
Poziv:  ./a.out ulaz.txt test
ulaz.txt:   Ovo je test primer.
        U njemu se rec test javlja
        vise puta. testtesttest
Izlaz:  5
```

Test Test 2

```
Poziv:   ./a.out
Izlaz  (na stderr):  -1
```

Test Test 3

```
Poziv:   ./a.out ulaz.txt foo
ulaz.txt:   (ne postoji)
Izlaz (na stderr):    -1
```

Test Test 4

```
Poziv:  ./a.out ulaz.txt .
ulaz.txt:   (prazna)
Izlaz:    0
```

[Rešenje 5.4]

Zadatak 5.5 Jelena: Ukljuceno resenje prethodnog zadatka. Dodati resenje ovog zadatka. 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 Test 1

```
Datoteka:  4
           0 0 0 1.2 1 0
           0.3 0.3 0.5 0.5 0.9 1
           -2 0 0 0 0 1
           2 0 2 2 -1 -1
Izlaz:     2 0 2 2 -1 -1
           -2 0 0 0 0 1
           0 0 0 1.2 1 0
           0.3 0.3 0.5 0.5 0.9 1
```

Test Test 2

```
Datoteka:  3
           1.2 3.2 1.1 4.3
Izlaz:     -1
```

Test Test 3

```
Datoteka:   (nema datoteke)
Izlaz:     -1
```

Test Test 4

```
Datoteka:   0
Izlaz:
```

[Rešenje 5.5]

Zadatak 5.6 Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju
`int f3(Cvor *koren, int n)`
koja u datom stablu prebrojava čvorove na n-tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou `0`. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

Test Test 1

```
Ulaz:
 1 5 3 6 1 4 7 9
Izlaz:
 1
```

Test Test 2

```
Ulaz:
 2 5 3 6 1 0 4 7 9
Izlaz:
 2
```

Test Test 3

```
Ulaz:
 0 4 2 5
Izlaz:
 0
```

Test Test 4

```
Ulaz:
  3
Izlaz:
 0
```

Test Test 5

```
Ulaz:
  -1 4 5 1 7
Izlaz:
 0
```

[Rešenje 5.6]

5.3 Rešenja

Rešenje 5.1

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 50

void greska()
{
  printf("-1\n");
  exit(EXIT_FAILURE);
}

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

  FILE *ulaz;
  char **linije;
  int i, j, n;

  /* Proveravamo argumente komandne linije. */
  if (argc != 2) {
    greska();
  }

  /* Otvaramo datoteku cije ime je navedeno kao argument
     komandne linije neposredno nakon imena programa koji se
     poziva. */
  ulaz = fopen(argv[1], "r");
  if (ulaz == NULL) {
    greska();
  }

  /* Ucitavamo broj linija. */
  fscanf(ulaz, "%d", &n);

  /* Alociramo memoriju na osnovu ucitanog broja linija. */
  linije = (char **) malloc(n * sizeof(char *));
  if (linije == NULL) {
    greska();
  }
  for (i = 0; i < n; i++) {
    linije[i] = malloc(MAX * sizeof(char));
    if (linije[i] == NULL) {
      for (j = 0; j < i; j++) {
        free(linije[j]);
      }
      free(linije);
```

```
47        greska();
      }
49    }

51    /* Ucitavamo svih n linija iz datoteke. */
      for (i = 0; i < n; i++) {
53      fscanf(ulaz, "%s", linije[i]);
      }
55
      /* Ispisujemo u odgovarajucem poretku ucitane linije koje
57       zadovoljavaju kriterijum. */
      for (i = n - 1; i >= 0; i--) {
59      if (isupper(linije[i][0])) {
          printf("%s\n", linije[i]);
61      }
      }
63
      /* Oslobadjamo memoriju koju smo dinamicki alocirali. */
65    for (i = 0; i < n; i++) {
        free(linije[i]);
67    }

69    free(linije);

71    /* Zatvaramo datoteku. */
      fclose(ulaz);
73
      /* Zavrsavamo sa programom. */
75    return 0;

77  }
```

Rešenje 5.2

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


4
int sumirajN (Cvor * koren, int n){
6       if(koren==NULL){
                return 0;
8       }

10      if(n==0){
            return koren->broj;
12      }

14      return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
}
16
```

```c
int main(){
    Cvor* koren=NULL;
    int n;
    int nivo;

    /* Citamo vrednost nivoa */
    scanf("%d", &nivo);


    while(1){

        /* Citamo broj sa standardnog ulaza */
        scanf("%d", &n);

        /* Ukoliko je korisnik uneo 0, prekidamo dalje citanje. */
        if(n==0){
            break;
        }

        /* A ako nije, dodajemo procitani broj u stablo. */
        dodaj_u_stablo(&koren, n);

    }

    /* Ispisujemo rezultat rada trazene funkcije */
    printf("%d\n", sumirajN(koren,nivo));

    /* Oslobadjamo memoriju */
    oslobodi_stablo(&koren);


    /* Prekidamo izvrsavanje programa */
    return 0;
}
```

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

Cvor* napravi_cvor(int b ) {
    Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
    if( novi == NULL)
        return NULL;

    /* Inicijalizacija polja novog Cvora */
    novi->broj = b;
    novi->levo = NULL;
    novi->desno = NULL;

    return novi;
}

```

```
19 void oslobodi_stablo(Cvor** adresa_korena)  {
       /* Prazno stablo i nema sta da se oslobadja */
21     if( *adresa_korena == NULL)
           return;
23
       /* Rekurzivno oslobadjamo najpre levo, a onda i desno podstablo*/
25     if( (*adresa_korena)->levo )
           oslobodi_stablo(&(*adresa_korena)->levo);
27     if( (*adresa_korena)->desno)
           oslobodi_stablo(&(*adresa_korena)->desno);
29
       free(*adresa_korena);
31     *adresa_korena =NULL;
   }
33
35 void proveri_alokaciju( Cvor* novi) {
       if( novi == NULL) {
37         fprintf(stderr, "Malloc greska za nov cvor!\n");
           exit(EXIT_FAILURE);
39     }
   }
41
   void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
43     /* Postojece stablo je prazno*/
       if( *adresa_korena == NULL){
45         Cvor* novi = napravi_cvor(broj);
           proveri_alokaciju(novi);
47         *adresa_korena = novi;  /*  Kreirani Cvor novi ce biti od
       sada koren stabla*/
           return;
49     }

51     /* Brojeve smestamo u uredjeno binarno stablo, pa
       ako je broj koji ubacujemo manji od broja koji je u korenu  */
53     if( broj < (*adresa_korena)->broj)
       /* Dodajemo u levo podstablo */
55         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
       /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
        dodajemo nov Cvor desno od korena */
57     else
           dodaj_u_stablo(&(*adresa_korena)->desno, broj);
59 }
```

```
   #ifndef __STABLA_H__
2  #define __STABLA_H__ 1

4  /* Struktura kojom se predstavlja Cvor stabla */
   typedef struct dcvor{
6      int broj;
       struct dcvor* levo, *desno;
```

```
} Cvor;

/* Funkcija alocira prostor za novi Cvor stabla, inicijalizuje polja
    strukture i vraca pokazivac na nov Cvor */
Cvor* napravi_cvor(int b );

/* Oslobadjamo dinamicki alociran prostor za stablo
 * Nakon oslobadjanja se u pozivajucoj funkciji koren
 * postavljana NULL, jer je stablo prazno  */
void oslobodi_stablo(Cvor** adresa_korena);


/* Funkcija proverava da li je novi Cvor ispravno alociran,
 * i nakon toga prekida program */
void proveri_alokaciju( Cvor* novi);


/* Funkcija dodaje nov Cvor u stablo i
 * azurira vrednost korena stabla u pozivajucoj funkciji.
 */
void dodaj_u_stablo(Cvor** adresa_korena, int broj);

#endif
```

Rešenje 5.3

```
#include <stdio.h>
#define MAX 50


int main()
{
  int m[MAX][MAX];
  int v, k;
  int i, j;
  int max_broj_negativnih, max_indeks_kolone;
  int broj_negativnih;

  /* Ucitavamo dimenzije matrice */
  scanf("%d", &v);
  if (v < 0 || v > MAX) {
    fprintf(stderr, "-1\n");
    return 0;
  }

  scanf("%d", &k);
  if (k < 0 || k > MAX) {
    fprintf(stderr, "-1\n");
    return 0;
  }
```

```
  /* Ucitavamo elemente matrice */
  for (i = 0; i < v; i++) {
    for (j = 0; j < k; j++) {
      scanf("%d", &m[i][j]);
    }
  }

  /* Pronalazimo kolonu koja sadrzi najveci broj negativnih
     elemenata */
  max_indeks_kolone = 0;

  max_broj_negativnih = 0;
  for (i = 0; i < v; i++) {
    if (m[i][0] < 0) {
      max_broj_negativnih++;
    }

  }

  for (j = 0; j < k; j++) {
    broj_negativnih = 0;
    for (i = 0; i < v; i++) {
      if (m[i][j] < 0) {
        broj_negativnih++;
      }
      if (broj_negativnih > max_broj_negativnih) {
        max_indeks_kolone = j;
      }
    }

  }

  /* Ispisujemo trazeni rezultat */
  printf("%d\n", max_indeks_kolone);

  /* Zavrsavamo program */
  return 0;
}
```

Rešenje 5.4

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 128

int main(int argc, char **argv)
{
  FILE *f;
  int brojac = 0;
  char linija[MAX], *p;
```

```
11
   if (argc != 3) {
13   fprintf(stderr, "-1\n");
     exit(EXIT_FAILURE);
15 }

17 if ((f = fopen(argv[1], "r")) == NULL) {
     fprintf(stderr, "-1\n");
19   exit(EXIT_FAILURE);
   }
21
   while (fgets(linija, MAX, f) != NULL) {
23   p = linija;
     while (1) {
25     p = strstr(p, argv[2]);
       if (p == NULL)
27       break;
       brojac++;
29     p = p + strlen(argv[2]);
     }
31 }

33 fclose(f);

35 printf("%d\n", brojac);

37 return 0;
}
```

Rešenje 5.5

```
#include <stdio.h>
2 #include <stdlib.h>
#include <math.h>
4
typedef struct _trougao {
6   double xa, ya, xb, yb, xc, yc;
} trougao;
8
double duzina(double x1, double y1, double x2, double y2) {
10  return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
12
double povrsina(trougao t) {
14  double a = duzina(t.xb, t.yb, t.xc, t.yc);
    double b = duzina(t.xa, t.ya, t.xc, t.yc);
16  double c = duzina(t.xa, t.ya, t.xb, t.yb);
    double s = (a + b + c) / 2;
18  return sqrt(s * (s - a) * (s - b) * (s - c));
}
20
```

```
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;

  if ((f = fopen("trouglovi.txt", "r")) == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  if (fscanf(f, "%d", &n) != 1) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  if ((niz = malloc(n * sizeof(trougao))) == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  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);
    }
  }

  qsort(niz, n, sizeof(trougao), &poredi);

  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);

  free(niz);
  fclose(f);
```

```
  return 0;
}
```

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

Cvor *napravi_cvor(int broj)
{

/* Dinamicki kreiramo cvor */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

/* U slucaju greske ... */
    if (novi == NULL) {
  fprintf(stderr, "-1\n");
  exit(1);
    }

/* Inicijalizacija */
    novi->vrednost = broj;
    novi->levi = NULL;
    novi->desni = NULL;

/* Vracamo adresu novog cvora */
    return novi;
}

void dodaj_u_stablo(Cvor **koren, int broj)
{

/* Izlaz iz rekurzije: ako je stablo bilo prazno,
   novi koren je upravo novi cvor */
  if (*koren == NULL) {
    *koren = napravi_cvor(broj);
    return;
  }

/* Ako je stablo neprazno, i koren sadrzi manju vrednost
   od datog broja, broj se umece u desno podstablo,
   rekurzivnim pozivom */
  if ((*koren)->vrednost < broj)
    dodaj_u_stablo(&(*koren)->desni, broj);
/* Ako je stablo neprazno, i koren sadrzi vecu vrednost
   od datog broja, broj se umece u levo podstablo,
   rekurzivnim pozivom */
  else if ((*koren)->vrednost > broj)
    dodaj_u_stablo(&(*koren)->levi, broj);

}
```

```c
void prikazi_stablo(Cvor * koren)
{
/* Izlaz iz rekurzije */
    if (koren == NULL)
  return;

    prikazi_stablo(koren->levi);
    printf("%d ", koren->vrednost);
    prikazi_stablo(koren->desni);
}

Cvor* ucitaj_stablo() {
  Cvor *koren = NULL;
  int x;
  while (scanf("%d", &x) == 1)
    dodaj_u_stablo(&koren, x);
  return koren;
}

void oslobodi_stablo(Cvor **koren)
{

/* Izlaz iz rekurzije */
    if (*koren == NULL)
  return;

    oslobodi_stablo(&(*koren)->levi);
    oslobodi_stablo(&(*koren)->desni);
    free(*koren);

    *koren = NULL;
}
```

```c
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura koja predstavlja cvor stabla */
typedef struct cvor {
    int vrednost;   /* Vrednost koja se cuva */
    struct cvor *levi;    /* Pokazivac na levo podstablo */
    struct cvor *desni;   /* Pokazivac na desno podstablo */
} Cvor;

/* Pomocna funkcija za kreiranje cvora. Cvor se kreira
   dinamicki, funkcijom malloc(). U slucaju greske program
   se prekida i ispisuje se poruka o gresci. U slucaju
   uspeha inicijalizuje se vrednost datim brojem, a pokazivaci
   na podstabla se inicijalizuju na NULL. Funkcija vraca
   adresu novokreiranog cvora */
Cvor *napravi_cvor(int broj);

/* Funkcija dodaje novi cvor u stablo sa datim korenom.
```

```
   Ukoliko broj vec postoji u stablu, ne radi nista.
   Cvor se kreira funkcijom napravi_cvor(). */
void dodaj_u_stablo(Cvor **koren, int broj);

/* Funkcija prikazuje stablo s leva u desno (tj.
   prikazuje elemente u rastucem poretku) */
void prikazi_stablo(Cvor * koren);

/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
    vraca
   pokazican na njegov koren */
Cvor* ucitaj_stablo();

/* Funkcija oslobadja prostor koji je alociran za
   cvorove stabla. */
void oslobodi_stablo(Cvor **koren);

#endif
```

Rešenje 5.6

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

int f3(Cvor * koren, int n)
{
  if (koren == NULL || n < 0)
    return 0;
  if (n == 0) {
    if (koren->levi == NULL && koren->desni != NULL)
      return 1;
    if (koren->levi != NULL && koren->desni == NULL)
      return 1;
    return 0;
  }
  return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
}

int main()
{
  Cvor *koren;
  int n;

  scanf("%d", &n);
  koren = ucitaj_stablo();

  printf("%d\n", f3(koren, n));

  oslobodi_stablo(&koren);

  return 0;
```

```
}
```