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 1

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

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

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

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

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

[Rešenje 1.1]

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

Test 1

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

[Rešenje 1.2]

Zadatak 1.3 Napisati malu biblioteku za rad sa polinomima.

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

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

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

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

(e) Napisati funkciju koja sabira dva polinoma.

(f) Napisati funkciju koja množi dva polinoma.

Napisati program koji testira prethodno napisane funkcije tako što se najpre unosi polinom `p` (stepen polinoma, a zatim i koeficijenti) i ispisuje na standardan izlaz u odgovarajućem obliku. Nakon toga se od korisnika traži da unese tačku u kojoj se računa vrednost tog polinoma a zatim se ispisuje 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`.

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 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 2

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

Test 3

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

Test 4

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

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 1

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

Test 2

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

Test 3

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

Test 4

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

Test 4

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

[Rešenje 1.6]

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

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

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.* Jelena: Da li je u redu da na izlazu bude ovakvo poravnanje?

Test 1

```
Ulaz:   235 5 10 127
Izlaz:
  Broj    235                                  = 00000000000
  reset(  235,    5,   10)                     = 00000000000
  set(  235,    5,   10)                       = 00000000000
  get_bits(  235,    5,   10)                  = 00000000000
  y =                                      127 = 00000000000
  set_n_bits(  235,    5,   10,  127)         = 00000000000
  invert(  235,    5,   10)                   = 00000000000
```

[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 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 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 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.11]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.12]

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

Test 1

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

Test 2

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

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 1

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

Test 2

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

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 1

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

Test 2

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

Test 3

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

Zadatak 1.16 ++ Napisati funkciju koja 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 1

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

Test 2

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

Test 3

```
Ulaz:   100328
Izlaz:  1
```

1.3 Rekurzija

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

Test 1

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

[Rešenje 1.17]

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

- 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 1

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

Test 2

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

[Rešenje 1.18]

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

Test 1

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

[Rešenje 1.19]

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

$$F(0) = 0$$

$$F(1) = 1$$

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

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

Test 1

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

[Rešenje 1.20]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

Test 5

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

[Rešenje 1.21]

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

Test 1

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

[Rešenje 1.22]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.23]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.24]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.25]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.26]

Zadatak 1.27 Napisati rekurzivnu funkciju koja vraća broj bitova koji su postavljeni na 1, u binarnoj reprezentaciji njenog celobrojnog argumenta. Napi-

sati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.

Test 1
```
Ulaz:  0x7F
Izlaz:  7
```

Test 2
```
Ulaz:  0x80
Izlaz:  1
```

Test 3
```
Ulaz:  0x00FF00FF
Izlaz:  16
```

Test 4
```
Ulaz:  0xFFFFFFFF
Izlaz:  32
```

[Rešenje 1.27]

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

Test 1
```
Ulaz:       10
Izlaz:      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 1
```
Ulaz:  5
Izlaz: 5
```

Test 2
```
Ulaz:  125
Izlaz: 7
```

Test 3
```
Ulaz:  8
Izlaz: 1
```

Test 4
```
Ulaz:  10
Izlaz: 2
```

[Rešenje 1.29]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

[Rešenje 1.30]

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

Test

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

[Rešenje 1.31]

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

Test 1

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

[Rešenje 1.32]

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

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5   10  10  5   1
```

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

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

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

Test 1

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

8
```

Test 2

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

32
```

[Rešenje 1.33]

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

Test 1

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

Zadatak 1.35 ++ *Hanojske kule*: Data su tri vertikalna štapa, na jednom se nalazi n diskova poluprečnika 1,2,3,... do n, tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, a preostali štap se koristi kao pomoćni štap prilikom premeštanja.
Napisati program koji za proizvoljnu vrednost n, koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

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

1.4 Rešenja

Rešenje 1.1

```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;
```

```c
/* 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",
    realan_deo(z1), imaginaran_deo(z1), moduo(z1));
    printf("\n");

    /* Ispisujemo drugi kompleksan broj, a zatim i racunamo i
    ispisujemo konjugovano kompleksan broj od z2 */
    ispisi_kompleksan_broj(z2);
    printf("\nNjegov konjugovano kompleksan broj: ");
    ispisi_kompleksan_broj( konjugovan(z2) );

    /* Testiramo funkciju koja racuna argument kompleksnih brojeva */
    printf("\nArgument kompleksnog broja: %f\n", argument(z2) );
    printf("\n");

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

    /* Testiramo oduzimanje kompleksnih brojeva */
    printf("\n");
    ispisi_kompleksan_broj(z1);
    printf(" - ");
    ispisi_kompleksan_broj(z2);
    printf("  =  ");
    ispisi_kompleksan_broj(oduzmi(z1, z2));
    printf("\n");

    /* Testiramo mnozenje kompleksnih brojeva */
    printf("\n");
    ispisi_kompleksan_broj(z1);
    printf(" * ");
    ispisi_kompleksan_broj(z2);
    printf("  =  ");
    ispisi_kompleksan_broj(mnozi(z1, z2));
    printf("\n");
    /* Program se zavrsava uspesno, tj, bez greske */
    return 0;
}
```

Rešenje 1.2

```c
/* Ukljucujemo zaglavlje neophodno za rad sa kompleksnim brojevima
 * 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.
```

```
7     Kako bi ustedeli kopiranje cele strukture, polinom prenosimo po
       adresi */
  void ispisi(const Polinom * p)
9 {
      int i;
11    for (i = p->stepen; i >= 0; i--) {
    if (p->koef[i]) {
13      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;
```

```
}

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

    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)
{
    int i;
    Polinom r;

    if (n < 0) {
  fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
  exit(EXIT_FAILURE);
    }

    if (n == 0)
  return *p;

    r = izvod(p);
    for (i = 1; i < n; i++)
  r = izvod(&r);

    return r;
}
```

```

/* Ovim pretrocesorskim direktivama zakljucavamo zaglavlje i time
    onemogucujemo
   da se sadrzaj zaglavlja vise puta ukljuci
*/
#ifndef  _POLINOM_H
#define _POLINOM_H

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

/* Maksimalni stepen polinoma */
#define MAX_STEPEN 20


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

/* Funkcija koja ispisuje polinom na stdout u citljivom obliku
```

```
   Polinom prenosimo po adresi, da bi ustedeli kopiranje cele
    strukture,
   vec samo prenosimo adresu na kojoj se nalazi polinom kog
    ispisujemo */
void ispisi(const Polinom * p);

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

/* 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
```

```
#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));

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

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

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

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

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

Rešenje 1.5

```
#include <stdio.h>

/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji.
Bitove u zapisu broja treba da ispisujemo sa leva na desno, tj. od
    bita najvece tezine ka bitu najmanje tezine. Iz tog razloga, za
    pocetnu vrednost maske uzimamo vrednost ciji binarni zapis je
    takav da je bit najvece tezine 1, a svi ostali nule.
  Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u desno,
    kako bismo ocitali naredni bit, gledano s leva na desno.
    Odgovarajuci karakter, ('0' ili '1'), ispisuje se na ekranu.
 Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
    postavljen na 1, neophodno je da maska bude neoznacen ceo broj i
```

```
    da se siftovanjem u desno ova 1 ne bi smatrala znakom i
    prepisivala, vec da bi nam se svakim siftovanjem  sa levog kraja
    binarnog zapisa pojavljivale 0.*/
void print_bits(unsigned x)  {

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

    return 0;
}
```

Rešenje 1.6

```
#include <stdio.h>

/* 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');
}

/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
    pomeranjem broja x*/
int count_bits(int x) {
    int br=0;
    unsigned wl = sizeof(unsigned)*8 -1;

    /* Formiramo masku 100000...0000000, koja sluzi za
        ocitavanje bita najvece tezine. U svakoj iteraciji
        maska se pomera u desno za 1 mesto, i ocitavamo sledeci
        bit. Petlja se zavrsava kada vise nema jedinica tj.
```

```
        kada maska postane nula. */
    unsigned maska= 1 << wl;
    for( ;  maska!=0 ; maska>>=1 )
        x & maska ? br++ : 1;

    return br;
}


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

    return 0;
}
```

```
#include <stdio.h>

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

    /* maska koju cemo koristiti za "ocitavanje" bitova */
    unsigned maska;

    /* Bitove u zapisu broja treba da ispisujemo sa leva na desno, tj
    . od bita najvece tezine ka
      bitu najmanje tezine. Iz tog razloga, za pocetnu vrednost maske
     uzimamo vrednost
      ciji binarni zapis je takav da je bit najvece tezine 1, a svi
    ostali nule.
      Nakon toga, u svakoj iteraciji cemo tu jedinicu pomerati u
    desno, kako bismo ocitali
      naredni bit, gledano s leva na desno. Odgovarajuci karakter,
    ('0' ili '1'), ispisuje se na ekranu.

      Zbog siftovanja maske u desno koja na pocetku ima najvisi bit
    postavljen na 1,
      neophodno je da maska bude neoznacen ceo broj i da se
    siftovanjem u desno ova 1
      ne bi smatrala znakom i prepisivala, vec da bi nam se svakim
    siftovanjem  sa levog kraja
      binarnog zapisa pojavljivale 0. */

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

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

```c
/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
    pomeranjem broja x*/
int count_bits1(int x) {
    int br=0;
    unsigned wl = sizeof(int)*8 -1;

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

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

    return br;
}


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

    return 0;
}
```

Rešenje 1.7

```c
#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;
```

```
14      /* 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 ) {
16          /* Za svaku jedinicu, potiskujemo jednu
                novu jedinicu sa leva u rezultat */
18          if( x & maska) {
                rezultat >>= 1;
20              rezultat |= maska;
            }
22      }

24      return rezultat;
    }
26
    /* Funkcija vraca najmanji neoznacen broj
28     sa istim binarnim ciframa kao i x */
    unsigned najmanji(unsigned x)  {
30      /* Inicijalizujemo rezultat na 0 */
        unsigned rezultat = 0;
32
        /* Dokle god imamo jedinice u broju x, pomeramo ga udesno. */
34      for( ; x!=0; x >>= 1){
            /* Za svaku jedinicu, potiskujemo jednu novu jedinicu sa
        desna u rezultat */
36          if(x & 1) {
                rezultat <<= 1;
38              rezultat |= 1;
            }
40      }

42      return rezultat;
    }
44
    /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
         celog broja u memoriji */
46  void print_bits( int x)  {
        unsigned velicina = sizeof(int)*8;
48      unsigned maska;

50      for(maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
            putchar(x & maska ? '1' : '0');
52
        putchar('\n');
54  }

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

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

```
62
   printf("Najmanji:\n");
64 print_bits( najmanji(broj) );

66    return 0;
}
```

Rešenje 1.8

```
1  #include <stdio.h>

3  /* Funckija postavlja na nulu n bitova pocev od pozicije p.
     Pozicije se broje pocev od pozicije najnizeg bita,
5    pri cemu se broji od nule .
     Npr, za n=5,  p=10
7        1010 1011 1100 1101 1110 1010 1110 0111
         1010 1011 1100 1101 1110 1000 0010 0111
9  */
   unsigned reset(unsigned x, unsigned n, unsigned p ) {
11     /* 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,
13       dok su svi ostali postavljeni na 1.

15       Na primer, za n=5 i p=10
         formiramo masku oblika
17       1111 1111 1111 1111 1111 1000 0011 1111
         To postizemo na sledeci nacin:
19       ~0                        1111 1111 1111 1111 1111 1111 1111
       1111
         (~0 << n)            1111 1111 1111 1111 1111 1111 1110 0000
21       ~(~0 << n)          0000 0000 0000 0000 0000 0000 0001 1111
         (~(~0 << n) << ( p-n+1))              0000 0000 0000 0000 0000
       0111 1100 0000
23       ~(~(~0 << n) << ( p-n+1))           1111 1111 1111 1111 1111
       1000 0011 1111
        */
25     unsigned maska =  ~(~(~0 << n) << ( p-n+1));

27     return x & maska;
   }
29
   /* Funckija postavlja na 1 n bitova pocev od pozicije p.
31   Pozicije se broje pocev od pozicije najnizeg bita,
     pri cemu se broji od nule .
33   Npr, za n=5,  p=10
         1010 1011 1100 1101 1110 1010 1110 0111
35       1010 1011 1100 1101 1110 1111 1110 0111
   */
37 unsigned set(unsigned x, unsigned n, unsigned p ) {
```

```
    /* Kako zelimo da samo odredjenih n bitova postavimo na 1, dok
    ostali treba da ostanu netaknuti.
39    Na primer, za n=5 i p=10
      formiramo masku oblika  0000 0000 0000 0000 0000 0111 1100 0000
41    prateci vrlo slican postupak kao za prethodnu funkciju
     */
43  unsigned maska = ~(~0 << n) << (p-n+1);

45  return x|maska;
}
47
/* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
    predstavlja n bitova
49  pocev od pozicije p u binarnoj reprezentaciji broja x, pri cemu se
    pozicija broji
  sa desna ulevo, gde je pocetna pozicija 0.
51  Na primer za n = 5 i p = 10 i broj
    1010 1011 1100 1101 1110 1010 1110 0111
53  0000 0000 0000 0000 0000 0000 0000 1011
 */
55 unsigned get_bits(unsigned x, unsigned n, unsigned p ) {
    /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a ostali
     su 0.
57    Na primer za n=5
      0000 0000 0000 0000 0000 0000 0001 1111
59   */
    unsigned maska = ~(~0 << n);
61
    /* Pomeramo sadrzaj u desno tako da trazeno polje bude uz desni
    kraj.
63    Zatim maskiramo ostale bitove, sem zeljenih n i vracamo
    vrednost */
    return maska & ( x >> (p-n+1));
65 }

67
/* Funkcija vraca broj x kome su n bitova pocev od pozicije p
    postavljeni na
69  vrednosti n bitova najnize tezine binarne reprezentacije broja y
    */
unsigned set_n_bits(unsigned x, unsigned n, unsigned p , unsigned y)
    {
71   /* Kreiramo masku kod kod koje su poslednjih n bitova 1, a
    ostali su 0.
      Na primer za n=5
73    0000 0000 0000 0000 0000 0000 0001 1111
     */
75  unsigned last_n_1 = ~(~0 << n);

77  /* Kao ranije u funkciji reset, kreiramo masku koja ima n bitova
    postavljenih
      na 0 pocevsi od pozicije p, dok su ostali bitovi 1.
```

```
79        Na primer za n=5 i p =10
          1111 1111 1111 1111 1111 1000 0011 1111
81       */
      unsigned middle_n_0 = ~(~(~0 << n ) << (p-n+1));
83
      /* x sa resetovanih n bita na pozicijama pocev od p */
85    unsigned x_reset = x & middle_n_0;

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

93    return x_reset ^ y_shift_middle;
}
95

97 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
       njih n*/
   unsigned invert(unsigned x, unsigned n, unsigned p )
99 {
     /* Formiramo masku sa n jedinica pocev od pozicije p
101       Na primer za n=5 i p=10
          0000 0000 0000 0000 0000 0111 1100 0000
103      */
     unsigned maska = ~(~0 << n) << (p-n+1);
105
     /* Operator ekskluzivno ili invertuje sve bitove gde je
        odgovarajuci
107        bit maske 1. Ostali bitovi ostaju nepromenjeni. */
     return maska ^ x;
109 }

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

117   for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
          putchar( x & maska ? '1' : '0' );
119
      putchar('\n');
121 }

123

125 int main() {
      unsigned broj, p, n, y;
127   scanf("%u%u%u%u", &broj, &n, &p, &y);
```

```
    printf("Broj %5u %25s= ", broj, "");
    print_bits(broj);

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

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

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

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

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

  return 0;
}
```

Rešenje 1.9

```
#include <stdio.h>

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

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

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

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

    return x;
}

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


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

        if( x<0 )
            /* Siftovanjem u desno broja koji je negativan dobijamo 1
     na najvisoj
             * poziciji.  Na primer ako je x
             * 1010 1011 1100 1101 1110 0001 0010 001b
             * (sa b oznacavamo u primeru 1 ili 0 na najnizoj
    poziciji)
             *
             * last_bit je
             * 0000 0000 0000 0000 0000 0000 0000 000b
             * nakon Siftovanja za 1 u desno
```

```
62              * 1101 0101 1110 0110 1111 0000 1001 0001
                *
64              * da bismo najvisu 1 u x postavili na b nije dovoljno da
     ga siftujemo
                * na najvisu poziciju jer bi se time dobile 0, a nama su
     potrebne 1
66              * zbog bitovskog &
                * zato prvo komplementiramo, pa tek onda siftujemo
68              * ~last_bit << (sizeof(int)*8 -1)
                * B000 0000 0000 0000 0000 0000 0000 0000
70              * (B oznacava ~b )
                * i ponovo komplementiramo da bismo imali b na najvisoj
    poziciji
72              * i sve 1 na ostalim pozicijama
                * ~(~last_bit << (sizeof(int)*8 -1))
74              * b111 1111 1111 1111 1111 1111 1111 1111
                */
76             x = (x >>1) & ~(~last_bit << (sizeof(int)*8 -1));
           else
78             x = (x >>1) | last_bit<< (sizeof(int)*8 -1);
    }
80
    return x;
82 }


84
/* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
86 void print_bits( int x)  {
    unsigned velicina = sizeof(int)*8;
88  unsigned maska;
    for( maska = 1 << (velicina -1); maska!=0 ; maska >>= 1)
90      putchar( x & maska ? '1' : '0' );

92  putchar('\n');
}
94
int main() {
96   unsigned x, k;
    scanf("%x%x", &x, &k);
98  printf("x %36s = ", "");
    print_bits(x);
100 printf("rotate_left(%7u,%6u)%8s = ", x, k, "");
    print_bits( rotate_left(x, k));
102
    printf("rotate_right(%7u,%6u)%7s = ", x, k, "");
104 print_bits( rotate_right(x, k));

106 printf("rotate_right_signed(%7u,%6u) = ", x, k);
    print_bits( rotate_right_signed(x, k));
108
    return 0;
```

```
}
```

Rešenje 1.10

```
#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
 * 10101011110011011110010010003 0011
 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);
```

```
44    /*Ispisujemo binarnu reprezentaciju broja dobijenog
          pozivom funkcije mirror
46    */
        print_bits( mirror(broj));
48
      return 0;
50  }
```

Rešenje 1.11

```
    #include <stdio.h>
2
    /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
         jedinica veci od broja nula.
4    U suprotnom funkcija vraca 0 */
    int Broj01(unsigned int n){
6
      int broj_nula, broj_jedinica;
8     unsigned int maska;

10    broj_nula=0;
      broj_jedinica=0;
12
      /* Postavljamo masku tako da pocinjemo sa analiziranjem bita
        najvece tezine */
14    maska=1<<(sizeof(unsigned int)*4-1);

16    /* Dok ne obidjemo sve bitove u zapisu broj n */
      while(maska!=0){
18
        /* Proveravamo da li se na poziciji koju odredjuje maska nalazi 0
         ili 1 i uvecavamo odgovarajuci brojac */
20      if(n&maska){
          broj_jedinica++;
22      }
        else{
24        broj_nula++;
        }
26
        /* Pomeramo masku u desnu stranu tako da mozemo da ocitamo
        vrednost narednog bita */
28      maska=maska>>1;
      }
30
      /* Ako je broj jedinica veci od broja nula vracamo 1, u suprotnom
        vracamo 0 */
32    return (broj_jedinica>broj_nula)? 1: 0;

34  }

36  int main(){
```

```
  unsigned int n;

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

  /* Ispsujemo vrednost funkcije */
  printf("%d\n", Broj01(n));

  return 0;
}
```

Rešenje 1.12

```
#include <stdio.h>

int broj_parova(unsigned int x){

  int broj_parova;
  unsigned int maska;

  /* Postavljamo broj parova na 0 */
  broj_parova=0;

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



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

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

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


  return broj_parova;

}

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

```
36
   /* Ucitavamo broj */
38 scanf("%u", &x);

40 /* Ispsujemo vrednost funkcije */
   printf("%d\n", broj_parova(x));
42
   return 0;
44 }
```

Rešenje 1.13

Rešenje 1.14

```
#include <stdio.h>
2
/*
4   Niska koju formiramo je duzine
   (sizeof(unsigned int)*8)/4 +1
6   jer za svaku heksadekadnu cifru nam trebaju 4 binarne cifre i 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;
   unsigned int maska;
20 int vrednost;

22 /* Heksadekadni zapis broja 15 je 000...0001111  - ovo nam odgovara
       ako hocemo da citamo 4 uzastopne cifre */
   maska=15;
24
   /*
26   Broj cemo citati od pozicije najmanje tezine ka poziciji najvece
     tezine;
     npr. za broj 00000000001101000100001111010101
28   u prvom koraku cemo procitati bitove:
     00000000001101000100001111010101<0101> (bitove izdvojene sa <...>)
     u drugom koraku cemo procitati:
     00000000001101000100001101<1101>0101
30   u trecem koraku cemo procitati:
     00000000001101000100<0011>11010101
```

```
    i tako redom

    indeks i oznacava poziciju na koju smestamo vrednost

  */
  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

```c
#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.
 */

/*  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

```
#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

```
#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.

  Kod ovih funkcija se po zavrsetku za tekuci rekurzivni poziv
  umesto skoka na adresu povratka skace na adresu
  povratka za prethodni poziv, odnosno za poziv na manjoj
  dubini. Time se stedi i prostor i vreme.

  Ovakve funkcije se mogu lako zameniti odgovarajucom
  iterativnom funkcijom,
  cime se smanjuje prostorna slozenost algoritma.
 */

/* PomoCna funkcija koja izracunava n! * result.
  Koristi repnu rekurziju. */

/* Result je argument u kom cemo akumulirati do tada izracunatu
  vrednost faktorijela.
  Kada zavrsimo, tj. kada dodjemo do izlaza iz rekurzije potrebno je
  da vratimo result. */
int faktorijelRepna(int n, int result) {
    if (n == 0)
        return result;

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

/* Sada zelimo da se oslobodimo repne rekurzije koja postoji u
  funkciji faktorijelRepna, koristeci algoritam sa predavanja.

  Najpre cemo vrednost argumenta funkcije postaviti na vrednost
  koja bi se prosledjivala rekurzivnom pozivu i pomocu goto naredbe
  vratiti se na pocetak tela funkcije.
 */

int faktorijelRepna_v1(int n, int result) {
    pocetak:
    if (n == 0)
        return result;

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

```
}

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

    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

```c
#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.
     */
    if(n<=0 )
            return 0;

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

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

    return a[0] + sumaNiza2(a+1,n-1);
}

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

    /* Ucitavamo broj elemenata niza */
    scanf("%d", &n);

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

    printf("Suma elemenata je %d\n", sumaNiza(a, n));
  // printf("Suma elemenata je %d\n",sumaNiza2(a, n));

    return 0;
```

```
52 }
```

Rešenje 1.23

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

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

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

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

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

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

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

Rešenje 1.24

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

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

```
8
      /* Na osnovu reSenja problema dimenzije n-1, resavamo problem
      dimenzije n */
10    else return a[n-1] * b[n-1] + skalarno(a,b,n-1);
  }
12
  int main()
14{
      int i, a[MAX_DIM], b[MAX_DIM], n;
16
      /* Unosimo dimenziju nizova, */
18    scanf("%d",&n);

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

24    for(i=0;i<n;i++)
        scanf("%d",&b[i]);
26
      /* Ispisujemo rezultat skalarnog proizvoda dva ucitana niza. */
28    printf("%d\n", skalarno(a,b,n));

30    return 0;
  }
```

Rešenje 1.25

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

4 int br_pojave(int x, int a[], int n)
  {
6     /* Izlazak iz rekurzije */
      if(n==1) return a[0]==x ? 1 : 0;
8
      int bp = br_pojave(x, a, n-1);
10    return a[n-1]==x ? 1 + bp : bp;
  }
12
  int main()
14{
      int x, a[MAX_DIM];
16    int n, i=0;

18    /* UCitavamo broj koji se trazi */
      scanf("%d", &x);
20
      /* Sve dok ne dodjemo do kraja ulaza, ucitavamo brojeve u niz; i
      predstavlja indeks tekuceg broja */
22    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

```
#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 prosleduje 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

```c
#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

```c
#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.
69              */
               permutacija(a,m+1,n);
71          }
        }
73  }

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

79      printf("Unesite duzinu permutacije: ");
        scanf("%d", &n);
81      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);
83          exit(EXIT_FAILURE);
        }
85

        permutacija(a,1,n);
87

        exit(EXIT_SUCCESS);
89  }
```

Rešenje 1.33

```
1   #include <stdio.h>
    #include <stdlib.h>
3
    /* Rekurzivna funkcija za racunanje binomnog koeficijenta.     */
5   /* 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)
        */
7   int binomniKoeficijent(int n, int k) {
        return (0<k && k<n) ? binomniKoeficijent (n-1, k-1) +
        binomniKoeficijent (n-1, k) : 1;
9   }
    /* Iterativno izracunavanje datog binomnog koeficijenta.
11
    int binomniKoeficijent (int n, int k) {
13    int i, j, b;
      for (b=i=1, j=n; i<=k; b=b*j--/i++);
15    return b;
    }
17
    */
19
    /* Prostim opaZanjem se uocava da se svaki element n-te hipotenuze (
        osim ivicnih 1)
21    dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz pomenute dve nove
        ivicne jedinice lako se zakljucuje
```

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Test 1

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

Test 2

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

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.

Test 1

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

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

Test 1

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

Test 2

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

Test 3

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

Test 4

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

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Test 1

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

Test 2

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

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

Test 1

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

Test 2

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

Test 3

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

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

Test 1

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

Test 2

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

Test 3

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

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

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

Test 1

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

Test 2

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

Test 3

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

Test 3

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

2.2 Višedimenzioni nizovi

Zadatak 2.8 Data je kvadratna matrica dimenzije n.

(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 1

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

Test 2

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

Zadatak 2.9 Date su dve kvadratne matrice istih dimenzija n.

(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 1

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

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

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

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

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

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

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

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

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

Test 1

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

Zadatak 2.11 Data je kvadratna matrica dimenzije n.

(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 1

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

Test 2

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

Test 3

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

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

Test 1

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

Test 2

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

Test 3

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

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

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati

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 1

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

Test 2

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

Test 3

```
Ulaz:  11 4
Izlaz: Greska: 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 1

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

2.3 Dinamička alokacija memorije

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

Test 1

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

Test 2

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

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

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

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

Test 1

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

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 1

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

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

Test 1

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

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

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

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

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

Test 1

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

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

Test 1

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

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

Test 2

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

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

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

Test 1

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

Test 2

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

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

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

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. Umesto "void obrni_niz(int *a, int n)" potpis
   metode bi mogao da bude i "void obrni_niz(int a[], int n)".
   U oba slucaja se argument funkcije "a" tumaci kao pokazivac,
   ili tacnije, kao adresa prvog elementa niza. U odnosu na
   njega se odredjuju adrese ostalih elemenata u nizu */
void obrni_niz_v2(int *a, int n)
{
  /* Pokazivaci na elemente niza a */
  int *prvi, *poslednji;


  for(prvi = a, poslednji = a + n - 1;
                prvi < poslednji; prvi++, poslednji--) {
    int t = *prvi;
    *prvi = *poslednji;
    *poslednji = t;
  }
}

/* Funkcija obrce elemente niza koriscenjem pokazivacke
   sintakse - modifikovano koriscenje pokazivaca */
void obrni_niz_v3(int *a, int n)
{
  /* Pokazivaci na elemente niza a */
  int *prvi, *poslednji;

  /* Obrcemo niz */
  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;
  }
```

```
}

int main()
{
  /* Deklaracija niza a od najvise MAX elemenata */
  int a[MAX];

  /* Broj elemenata niza a */
  int n;

  /* Pokazivac na elemente niza a */
  int *p;

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

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

  /* Unosimo elemente niza */
  for(p = a; p - a < n; p++)
    scanf("%d", p);

  obrni_niz_v1(a,n);
  // obrni_niz_v2(a,n);
  // obrni_niz_v3(a,n);

  /* Prikazujemo sadrzaj niza nakon obrtanja */
  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 racuna zbir elemenata niza */
double zbir(double *a, int n)
{
  double s = 0;
  int i;
```

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

14    return s;
}
16
/* Funkcija racuna proizvod elemenata niza */
18 double proizvod(double a[], int n)
{
20    double p = 1;

22    for(; n; n--)
        p *= *a++;
24
      return p;
26 }

28 /* Funkcija racuna najmanji element niza */
   double min(double *a, int n)
30 {
      /* Za najmanji element se najpre postavlja prvi element */
32    double min = a[0];
      int i;
34
      /* Ispitujemo da li se medju ostalim elementima niza
36       nalazi najmanji */
      for(i = 1; i < n; i++)
38      if ( a[i] < min )
          min = a[i];
40
       return min;
42 }

44 /* Funkcija racuna najveci element niza */
   double max(double *a, int n)
46 {
      /* Za najveci element se najpre postavlja prvi element */
48    double max = *a;

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

56    return max;
   }
58


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

```
64
   /* Ucitavamo dimenziju niza */
66   scanf("%d", &n);

68   /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
   if(n <= 0 || n > MAX) {
70     fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
     exit(EXIT_FAILURE);
72   }

74   /* Unosimo elemente niza */
   for(i = 0; i < n; i++)
76     scanf("%lf", a + i);

78   /* Testiramo definisane funkcije */
   printf("zbir = %5.3f\n", zbir(a, n));
80   printf("proizvod = %5.3f\n", proizvod(a, n));
   printf("min = %5.3f\n", min(a, n));
82   printf("max = %5.3f\n", max(a, n));

84   return 0;
}
```

Rešenje 2.3

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

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

14   while( prvi < poslednji ){

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

20   /* Pokazivac prvi se pomera na sledeci element */
   prvi++;
22

     /* Smanjuje se vrednost elementa na koji pokazuje
24     pokazivac poslednji */
   (*poslednji)--;
26
```

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

void povecaj_smanji_sazetije(int *a , int n)
{
  int *prvi = a;
  int *poslednji = a+n-1;

  while( prvi < poslednji ){
    (*prvi++)++;
    (*poslednji--)--;
  }
}

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

  /* Unosimo broj elemenata */
  scanf("%d", &n);

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

  /* Unosimo elemente niza */
  for(p = a; p - a < n; p++)
    scanf("%d", p);

  povecaj_smanji(a,n);
  /*  povecaj_smanji_sazetije(a,n); */

  /* Prikaz niza nakon modifikacije */
  for(p = a; p - a < n; p++)
    printf("%d ", *p);
  printf("\n");

  return 0;
}
```

Rešenje 2.4

```
#include <stdio.h>

/* Argumenti funkcije main mogu da budu broj argumenta komandne
```

```
   linije (int argc) i niz arugmenata komandne linije
   (niz niski) (char *argv[]  <=>  char** argv) */
int main(int argc, char *argv[])
{
  int i;

  /* Ispisujemo broj argumenata komandne linije */
  printf("%d\n", argc);

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

  /* koristeci pokazivacku sintaksu */
  i=argc;
  for (; argc>0; argc--)
    printf("%d %s\n", i-argc, *argv++);


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

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

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

  /* koristeci pokazivacku sintaksu */

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

  return 0;
}
```

Rešenje 2.5

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

```

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

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

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

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

Rešenje 2.6

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

#define MAX_KARAKTERA 100

/* 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, i = 0, n;
  FILE *in;

  /* Ako korisnik nije uneo trazene argumente,
     prijavljujemo gresku */
  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);
  }

  /* Otvaramo datoteku 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("%d\n", br);

  /* Zatvaramo datoteku */
  fclose(in);
  return 0;
}
```

Rešenje 2.7

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

#define MAX_KARAKTERA 100

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

/* Funkcija 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;
```

```
21    return *(s+i) - *(t+i);
}

/* Funkcija strlen() iz standardne biblioteke */
int duzina_niske(char *s)
{
27    int i;
    for(i = 0; *(s+i); i++)
29      ;
    return i;
31  }

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

/* Funkcija ispituje da li je niska zadata drugim argumentom
43     funkcije prefiks niske zadate prvi argumentom funkcije */
int prefiks_niske(char *niska, char *prefiks) {
    int i;
47    if(duzina_niske(prefiks) <= duzina_niske(niska)) {
      for(i=0; i<duzina_niske(prefiks); i++)
49      if(*(prefiks+i) != *(niska+i))
          return 0;
51      return 1;
    }
53    else return 0;
}
55
int main(int argc, char **argv)
57  {
    /* Ako korisnik nije uneo trazene argumente,
59       prijavljujemo gresku */
    if(argc < 4) {
61      printf("Greska: ");
    printf("Nedovoljan broj argumenata komandne linije.\n");
63    printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
                                                    argv[0]);
65      exit(EXIT_FAILURE);
    }
67
    FILE *in;
69  int br = 0, i = 0, n;
    char rec[MAX_KARAKTERA];
71
    in = fopen(*(argv+1), "r");
```

```
  if(in == NULL) {
    fprintf(stderr, "Greska: ");
    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
                                              argv[1]);
    exit(EXIT_FAILURE);
  }

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

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

Rešenje 2.8

```
#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 */
```

```
28    for(i = 0; i<n; i++)
        for (j=0 ; j<n; j++)
30        scanf("%d",&A[i][j]);

32    /* Ispis elemenata matrice koriscenjem indeksne sintakse.
         Ispis vrsimo vrstu po vrstu */
34    for(i = 0; i<n; i++) {
        /* Ispisujemo elemente i-te vrste */
36      for ( j=0 ; j<n; j++)
          printf("%d ", A[i][j]);
38      printf("\n");
      }
40
      /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
42       Kod ovako definisane matrice, elementi su uzastopno
       smesteni u memoriju, kao na traci. To znaci da su svi
44     elementi prve vrste redom smesteni jedan iza drugog. Odmah
       iza poslednjeg elementa prve vrste smesten je prvi element
46     druge vrste za kojim slede svi elementi te vrste
       i tako dalje redom */
48    /*
      for( i = 0; i<n; i++) {
50      for ( j=0 ; j<n; j++)
          printf("%d ", *(*(A+i)+j));
52      printf("\n");
      }
54    */

56    int tr = trag(A,n);
      printf("trag = %d\n",tr);
58
      printf("euklidska norma = %.2f\n",euklidska_norma(A,n));
60    printf ("vandijagonalna norma = %d\n",
                           gornja_vandijagonalna_norma(A,n));
62
      return 0;
64  }

66  /* Definisemo funkcije koju smo ranije deklarisali */

68  /* Funkcija izracunava trag matrice */
    int trag(int M[][MAX], int n)
70  {
      int trag = 0,i;
72    for(i=0; i<n; i++)
        trag += M[i][i];
74    return trag;
    }
76
    /* Funkcija izracunava euklidsku normu matrice */
78  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>

#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");
```

```
26    }
  }
28
  /* Funkcija proverava da li su zadate kvadratne matrice a i b
30       dimenzije n jednake */
  int jednake_matrice(int a[][MAX], int b[][MAX], int n) {
32    int i, j;

34    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
36      /* Nasli smo elemente na istim pozicijama u matricama
           koji se razlikuju */
38        if(a[i][j]!=b[i][j])
          return 0;
40
    /* Prosla je provera jednakosti za sve parove elemenata koji
42       su na istim pozicijama sto znaci da su matrice jednake */
    return 1;
44  }

46  /* Funkcija izracunava zbir dve kvadratne matice */
  void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
48  {
    int i, j;
50
    for(i=0; i<n; i++)
52    for(j=0; j<n; j++)
        c[i][j] = a[i][j] + b[i][j];
54  }

56  /* Funkcija izracunava proizvod dve kvadratne matice */
  void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
58  {
    int i, j, k;
60
    for(i=0; i<n; i++)
62    for(j=0; j<n; j++) {
    /* Mnozimo i-tu vrstu prve sa j-tom kolonom druge matrice */
64  c[i][j] = 0;
    for(k=0; k<n; k++)
66    c[i][j] += a[i][k] * b[k][j];
    }
68  }

70  int main()
  {
72    /* Matrice ciji se elementi zadaju sa ulaza */
    int  a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
74
    /* Matrice zbira i proizvoda */
76  int zbir[MAX][MAX], proizvod[MAX][MAX];
```

```
78    /* Dimenzija matrica */
      int n;
80    int i, j;

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

86    /* Proveravamo da li je prekoraceno ogranicenje */
      if( n > MAX || n <= 0) {
88      fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
      fprintf(stderr, "matrica.\n");
90      exit(EXIT_FAILURE);
      }
92
      /* Ucitavamo matrice */
94    ucitaj_matricu(a, n);
      ucitaj_matricu(b, n);
96
      /* Izracunavamo zbir i proizvod matrica */
98    saberi(a, b, zbir, n);
      pomnozi(a, b, proizvod, n);
100
      /* Ispisujemo rezultat */
102   if(jednake_matrice(a, b, n) == 1)
        printf("da\n");
104   else
        printf("ne\n");
106
      printf("Zbir matrica je:\n");
108   ispisi_matricu(zbir, n);

110   printf("Proizvod matrica je:\n");
      ispisi_matricu(proizvod, n);
112
      return 0;
114 }
```

Rešenje 2.10

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

4 #define MAX 64

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

```
12    int i;

14    /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
         dijagonali vazi da je indeks vrste jednak indeksu kolone */
16    for(i=0; i<n; i++) {
        if(m[i][i] != 1)
18        return 0;
      }
20
      return 1;
22  }

24  /* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono
       je odredjeno matricom koja sadrzi sve elemente polazne matrice
26     dopunjene jedinicama na glavnoj dijagonali */
    void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
28  {
      int i, j;
30
      /* Prepisujemo vrednosti elemenata matrice pocetne matrice */
32    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
34        zatvorenje[i][j] = m[i][j];

36    /* Postavljamo na glavnoj dijagonali jedinice */
      for(i=0; i<n; i++)
38      zatvorenje[i][i] = 1;
    }
40
    /* Funkcija proverava da li je relacija simetricna. Relacija je
42     simetricna ako za svaki par elemenata vazi: ako je element
       "i" u relaciji sa elementom "j", onda je i element "j" u
44     relaciji sa elementom "i". Ovakve matrice su simetricne u
       odnosu na glavnu dijagonalu */
46  int simetricnost (int m[][MAX], int n)
    {
48    int i, j;

50    /* Obilazimo elemente ispod glavne dijagonale matrice i
         uporedjujemo ih sa njima simetricnim elementima */
52    for(i=0; i<n; i++)
        for(j=0; j<i; j++)
54        if(m[i][j] != m[j][i])
          return 0;
56
      return 1;
58  }

60  /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono
       je odredjeno matricom koja sadrzi sve elemente polazne matrice
62     dopunjene tako da matrica postane simetricna u odnosu na
       glavnu dijagonalu */
```

```
64 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
   {
66   int i, j;

68   /* Prepisujemo vrednosti elemenata matrice m */
     for(i=0; i<n; i++)
70     for(j=0; j<n; j++)
         zatvorenje[i][j] = m[i][j];
72
     /* Odredjujemo simetricno zatvorenje matrice */
74   for(i=0; i<n; i++)
       for(j=0; j<n; j++)
76       if(zatvorenje[i][j] == 1)
         zatvorenje[j][i] = 1;
78 }


80
   /* Funkcija proverava da li je relacija tranzitivna. Relacija je
82    tranzitivna ako ispunjava sledece svojstvo: ako je element "i"
      u relaciji sa elementom "j" i element "j" u relaciji sa
84    elementom "k", onda je i element "i" u relaciji sa elementom
      "k" */
86 int tranzitivnost (int m[][MAX], int n)
   {
88   int i, j, k;

90   for(i=0; i<n; i++)
       for(j=0; j<n; j++)
92       /* Pokusavamo da pronadjemo element koji narusava
        * tranzitivnost */
94       for(k=0; k<n; k++)
         if(m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
96         return 0;

98   return 1;
   }
100

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

108   /* Kopiramo pocetnu matricu u matricu rezultata */
     for(i=0; i<n; i++)
110      for(j=0; j<n; j++)
         zatvorenje[i][j] = m[i][j];
112
     /* Primenom Varsalovog algoritma odredjujemo
114     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);
```

```
168   }

170   /* Ucitavamo element po element matrice */
      for(i=0; i<n; i++)
172     for(j=0; j<n; j++)
          fscanf(ulaz, "%d", &m[i][j]);
174
      /* Ispisujemo trazene vrednosti */
176   printf("Refleksivnost: %s\n",
              refleksivnost(m, n) == 1 ? "da" : "ne");
178
      printf("Simetricnost: %s\n",
180           simetricnost(m, n) == 1 ? "da" : "ne");

182   printf("Tranzitivnost: %s\n",
              tranzitivnost(m, n) == 1 ? "da" : "ne");
184
      printf("Refleksivno zatvorenje:\n");
186   ref_zatvorenje(m, n, pomocna);
      pisi_matricu(pomocna, n);
188
      printf("Simetricno zatvorenje:\n");
190   sim_zatvorenje(m, n, pomocna);
      pisi_matricu(pomocna, n);
192
      printf("Refleksivno-tranzitivno zatvorenje:\n");
194   tran_zatvorenje(m, n, pomocna);
      pisi_matricu(pomocna, n);
196
      /* Zatvaramo datoteku */
198   fclose(ulaz);

200   return 0;
}
```

Rešenje 2.11

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 32
5
   int max_sporedna_dijagonala(int m[][MAX], int n)
7  {
     int i, j;
9    /* Trazimo najveci element na sporednoj dijagonali. Za
        elemente sporedne dijagonale vazi da je zbir indeksa vrste
11     i indeksa kolone jednak n-1. Za pocetnu vrednost maksimuma
       uzimamo element u gornjem desnom uglu */
13   int max_na_sporednoj_dijagonali = m[0][n-1];
     for(i=1; i<n; i++)
```

```
15      if(m[i][n-1-i] > max_na_sporednoj_dijagonali)
          max_na_sporednoj_dijagonali = m[i][n-1-i];
17
    return max_na_sporednoj_dijagonali;
19  }

21  /* Funkcija izracunava indeks kolone najmanjeg elementa */
    int indeks_min(int m[][MAX], int n)
23  {
    int i, j;
25  /* Za pocetnu vrednost minimuma uzimamo element u gornjem
       levom uglu */
27  int min=m[0][0], indeks_kolone=0;

29  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
31      /* Ako je tekuci element manji od minimalnog */
        if(m[i][j]<min) {
33      /* cuvamo njegovu vrednost */
        min=m[i][j];
35      /* i cuvamo indeks kolone u kojoj se nalazi */
        indeks_kolone=j;
37      }
    return indeks_kolone;
39  }

41  /* Funkcija izracunava indeks vrste najveceg elementa */
    int indeks_max(int m[][MAX], int n) {
43    int i, j;
    /* Za maksimalni element uzimamo gornji levi ugao */
45  int max=m[0][0], indeks_vrste=0;

47  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
49      /* Ako je tekuci element manji od minimalnog */
        if(m[i][j]>max) {
51      /* cuvamo njegovu vrednost */
        max=m[i][j];
53      /* i cuvamo indeks vrste u kojoj se nalazi */
        indeks_vrste=i;
55      }
    return indeks_vrste;
57  }

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

63  int broj_negativnih=0;

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

```
67        if(m[i][j]<0)
            broj_negativnih++;
69    return broj_negativnih;
  }
71
  int main(int argc, char* argv[])
73{
    int m[MAX][MAX];
75  int n;
    int i, j;
77
    /* Proveravamo broj argumenata komandne linije */
79  if(argc < 2) {
      printf("Greska: ");
81  printf("Nedovoljan broj argumenata komandne linije.\n");
    printf("Program se poziva sa %s dim_matrice.\n", argv[0]);
83    exit(EXIT_FAILURE);
    }
85
    /* Ucitavamo vrednost dimenzije i proveravamo njenu
87     korektnost */
    n = atoi(argv[1]);
89
    if( n > MAX || n <= 0) {
91    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
      fprintf(stderr, "matrice.\n");
93    exit(EXIT_FAILURE);
    }
95
    /* Ucitavamo element po element matrice */
97  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
99      scanf("%d", &m[i][j]);

101 int max_sd = max_sporedna_dijagonala(m, n);
    int i_min = indeks_min(m, n);
103 int i_max = indeks_max(m, n);
    int bn = broj_negativnih(m, n);
105
    /* Ispisujemo rezultat */
107 printf("%d %d %d %d\n", max_sd, i_min, i_max, bn);

109 /* Prekidamo izvrsavanje programa */
    return 0;
111}
```

Rešenje 2.12

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

```c
#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");
  }
}

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

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

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

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

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

  /* Proveravamo uslov ortogonalnosti, odnosno da li je proizvod
     dve bilo koje razlicite vrste matrice jednak nuli */
  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 */
72    return 1;
  }
74
  int main()
76{
    int A[MAX][MAX];
78  int n;

80  /* Ucitavamo vrednost dimenzije i proveravamo njenu
       korektnost */
82  scanf("%d", &n);

84  if( n > MAX || n <= 0) {
      fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
86    fprintf(stderr, "matrice.\n");
      exit(EXIT_FAILURE);
88  }

90  /* Ucitavamo matricu */
    ucitaj_matricu(A, n);
92
    /* Ispisujemo rezultat rada funkcije */
94  if(ortonormirana(A,n))
      printf("da\n");
96  else
      printf("ne\n");
98
    return 0;
100 }
```

Rešenje 2.13

```
#include <stdio.h>
2 #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++;

    if(krajIspisa(top,bottom,left,right))
      break;

    for(i=top; i<=bottom; i++ )
      printf("%d ",a[i][right]);

    /* Pomeramo desnu kolonu za naredni krug ispisa
       blize levom kraju */
    right--;

    if(krajIspisa(top,bottom,left,right))
      break;

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

    /* Podizemo donju vrstu za naredni krug ispisa */
    bottom--;

    if(krajIspisa(top,bottom,left,right))
      break;

    /* 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  }

64  void 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
8    koji pokazuje na prvi element niza tipa T. Na dalje se
```

```
   elementima moze pristupati na isti nacin kao da nam
   je dato ime niza (koje se tako i ponasa - kao pokazivac
   na element tipa T koji je prvi u nizu) */
int main()
{
  int *p = NULL;
  int i, n;

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

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

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

  /* 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;
```

```
10  /* Velicina alocirane memorije  */
    int alocirano;
12
    /* Broj elemenata niza */
14  int n;

16  /* Broj koji se ucitava sa ulaza */
    int x;
18  int i;
    int* b = NULL;
20
    /* Inicijalizacija */
22  alocirano = n = 0;

24  /* Unosimo brojeve sa ulaza */
    scanf("%d", &x);
26
    /* Sve dok je procitani broj razlicit od nule... */
28  while(x!=0) {

30    /* Ako broj ucitanih elemenata niza odgovara broju
         alociranih mesta, za smestanje novog elemenata treba
32       obezbediti dodatni prostor. Da se ne bi za svaki sledeci
         element pojedinacno alocirala memorija, prilikom
34       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
46         promenljivoj 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 pocetnom
70           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(): ");
88          fprintf(stderr, "greska pri alokaciji memorije.\n");
            exit(EXIT_FAILURE);
90        }
          */
92      }
94      /* Smestamo element u niz */
        a[n++] = x;
96
        /* i ucitavamo sledeci element */
98      scanf("%d", &x);
    }
100
    /* Ispisujemo brojeve u obrnutom poretku */
102 for(n--; n>=0; n--)
      printf("%d ", a[n]);
104 printf("\n");
106 /* Oslobadjamo dinamicki alociranu memoriju */
    free(a);
108
    /* Program se zavrsava */
110 exit(EXIT_SUCCESS);
}
```

Rešenje 2.17

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

#define MAX 1000

/* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
   Ovako nesto se moze improvizovati tako sto se u funkciji
   dinamicki kreira niz potrebne velicine, popuni se potrebnim
   informacijama, a zatim se vrati njegova adresa. Imajuci u
   vidu cinjenicu da dinamicki kreiran objekat ne nestaje
   kada se izadje iz funkcije koja ga je kreirala, vraceni
   pokazivac se kasnije u pozivajucoj funkciji moze koristiti
   za pristup "vracenom" nizu. Medjutim, pozivajuca funkcija
   ima odgovornost i da se brine o dealokaciji istog prostora */

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

  /* Proveravamo uspeh alokacije */
  if (p == NULL) {
    fprintf(stderr, "malloc(): ");
    fprintf(stderr, "greska pri alokaciji memorije.\n");
    exit(EXIT_FAILURE);
  }

  /* Kopiramo i nadovezujemo stringove */

  /* Resenje bez koriscenja biblioteckih funkcija */
  /*
  int i,j;
  for(i=j=0; s[j]!='\0'; i++, j++)
    p[i]=s[j];

  for(j=0; t[j]!='\0'; i++, j++)
    p[i]=t[j];

  p[i]='\0';
  */

  /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
     string.h */
  strcpy(p, s);
  strcat(p, t);

```

```
  /* Vracamo pokazivac p */
  return p;
}

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

  /* Ucitavamo dve niske koje cemo da nadovezemo */
  scanf("%s", s1);
  scanf("%s", s2);

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

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

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

  return 0;
}
```

Rešenje 2.18

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

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 */
```

```
26    if(A == NULL) {
        fprintf(stderr, "malloc(): ");
28      fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
30    }

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

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

44        exit( EXIT_FAILURE);
        }
46    }

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

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

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

70    return 0;
  }
```

Rešenje 2.19

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

```c
#include <math.h>

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

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

void ispisi_elemente_ispod_dijagonale(int ** M, int n, int m)
{
  int i, j;

  for(i=0; i<n; i++) {
    /* Ispisujemo elemente ispod glavne dijagonale matrice */
    for(j=0; j<=i; j++)
      printf("%d ", M[i][j]);
    printf("\n");
  }
}

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

  /* 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);
    }
  }
```

```
55    ucitaj_matricu(matrica, n, m);

57    ispisi_elemente_ispod_dijagonale(matrica, n, m);

59    /* Oslobadjamo dinamicki alociranu memoriju za matricu.
         Prvo oslobadjamo prostor rezervisan za svaku vrstu */
61    for( j=0; j<n; j++)
        free(matrica[j]);
63
      /* Zatim oslobadjamo memoriju za niz pokazivaca na vrste
65       matrice */
      free(matrica);
67
      return 0;
69  }
```

Rešenje 2.20

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

Rešenje 2.21

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include <math.h>
4
  /* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni (float** a, int n)
  {
8   int i, j;

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

```
}

/* 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]);
}

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

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

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

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

```
76    /* Za svaku vrstu matrice */
      for(i=0; i<n; i++) {
78      /* Alociramo memoriju */
        m[i] = (float*) malloc(n * sizeof(float));
80
        /* Proveravamo da li je doslo do greske pri alokaciji */
82      if(m[i] == NULL) {
          /* Ako jeste, ispisujemo poruku */
84        printf("malloc(): neuspela alokacija memorije!\n");

86        /* Oslobadjamo memoriju zauzetu do ovog koraka */
          for(j=0; j<i; j++)
88        free(m[i]);
          free(m);
90        exit(EXIT_FAILURE);
        }
92    }
      return m;
94  }

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

102   for(i=0; i<n; i++)
        free(m[i]);
104   free(m);
    }
106
    int main(int argc, char* argv[])
108 {
      FILE* ulaz;
110   float** a;
      int n;
112
     /* Ako korisnik nije uneo trazene argumente,
114       prijavljujemo gresku */
      if(argc < 2) {
116     printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
118     printf("Program se poziva sa %s ime_dat.\n", argv[0]);
        exit(EXIT_FAILURE);
120   }

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

```
                                                argv[1]);
128       exit(EXIT_FAILURE);
    }
130
    /* citamo dimenziju matrice */
132 fscanf(ulaz, "%d", &n);

134 /* Alociramo memoriju */
    a = alociraj_memoriju(n);
136
    /* Ucitavamo elemente matrice */
138 ucitaj_matricu(ulaz, a, n);

140 float zbir = zbir_ispod_sporedne_dijagonale(a, n);

142 /* Pozivamo funkciju za modifikovanje elemenata */
    izmeni(a, n);
144
    /* Ispisujemo rezultat */
146 printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
    printf("je %.2f.\n", zbir);
148
    printf("Transformisana matrica je:\n");
150 ispisi_matricu(a,n);

152 /* Oslobadjamo memoriju */
    oslobodi_memoriju(a, n);
154
    /* Zatvaramo datoteku */
156 fclose(ulaz);

158 /* i prekidamo sa izvrsavanjem programa */
    return 0;
160 }
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

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

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

/* NAPOMENA:
   Zaglavlje math.h sadrzi deklaracije raznih matematickih
   funkcija. đIzmeu ostalog, to su ćsledee funkcije:
   double sin(double x);
   double cos(double x);
   double tan(double x);
   double asin(double x);
   double acos(double x);
   double atan(double x);
   double atan2(double y, double x);
   double sinh(double x);
   double cosh(double x);
   double tanh(double x);
   double exp(double x);
   double log(double x);
   double log10(double x);
   double pow(double x, double y);
   double sqrt(double x);
   double ceil(double x);
   double floor(double x);
   double fabs(double x);
*/

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

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

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

```

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

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

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

  /* Inicijalizujemo čpokaziva na funkciju koja treba da se
     tabelira */
  if(strcmp(ime_fje, "sin") == 0)
    fp=&sin;
  else if(strcmp(ime_fje, "cos") == 0)
    fp=&cos;
  else if(strcmp(ime_fje, "tan") == 0)
    fp=&tan;
  else if(strcmp(ime_fje, "atan") == 0)
    fp=&atan;
  else if(strcmp(ime_fje, "acos") == 0)
    fp=&acos;
  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), 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
---------------------------------
```

[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.

Test 1

```
Ulaz: 11 2 5 6 8 10 11 23
Izlaz:
  Linearna pretraga
  Pozicija elementa je 5.
  Binarna pretraga
  Pozicija elementa je 5.
  Interpolaciona pretraga
  Pozicija elementa je 5.
```

Test 2

```
Ulaz: 14 10 32 35 43 66 89 100
Izlaz:
  Linearna pretraga
  Element se ne nalazi u nizu.
  Binarna pretraga
  Element se ne nalazi u nizu.
  Interpolaciona pretraga
  Element se ne nalazi u nizu.
```

[Rešenje 3.2]

Zadatak 3.3 Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks studenta čije informacije se potom prikazuju na ekranu. Zatim, korisnik učitava prezime studenta i prikazuju mu se informacije o prvom studentu sa unetim prezimenom. 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.

Test 1

```
Datoteka:
  20140003 Marina Petrovic
  20140012 Stefan Mitrovic
  20140032 Dejan Popovic
  20140049 Mirko Brankovic
  20140076 Sonja Stevanovic
  20140104 Ivan Popovic
  20140187 Vlada Stankovic
  20140234 Darko Brankovic
Ulaz:
  20140076
  Popovic
Izlaz:
  Indeks: 20140076, Ime i prezime: Sonja Stevanovic
  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
Datoteka:
  12 53
  2.342 34.1
  -0.3 23
  -1 23.1
  123.5 756.12
Izlaz: -0.3 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: Korisiti 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 niz elemenata koji se učitavaju sa standardnog ulaza. Niz neće biti duži od 256, i njegovi elementi se unose sve do kraja ulaza.

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.7]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.8]

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

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

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

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

Test 1
```
Ulaz:  10
Izlaz: 3 3
```

Test 2
```
Ulaz:  4
Izlaz: 2 2
```

Test 3
```
Ulaz:  17
Izlaz: 4 4
```

Test 4
```
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* $[0, 90°]$.

Test 1
```
Ulaz:
  2
  2 0 2 1
  1 2 2 2
Izlaz: 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 standardni izlaz.

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

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

Test 1

```
Ulaz:  R
Izlaz: Dusko Radovic
```

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
```

[Rešenje 3.12]

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

Test 1

```
Ulaz:   anagram ramgana
Izlaz:  jesu
```

Test 2

```
Ulaz:   anagram anagrm
Izlaz:  nisu
```

[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
```

[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.*

Test 1

```
Ulaz:   34 134 4 1 6 30 23
Izlaz:  da
```

Test 2

```
Ulaz:   12 53 1 43 3 56 13
Izlaz:  ne
```

[Rešenje 3.15]

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

Test 1

```
Ulaz:
Unesite elemente prvog niza:
3 6 7 11 14 35 0
Unesite elemente drugog niza:
3 5 8 0
Izlaz:
3 3 5 6 7 8 11 14 35
```

Test 2

```
Ulaz:
Unesite elemente prvog niza:
1 4 7 0
Unesite elemente drugog niza:
9 11 23 54 75 0
Izlaz:
1 4 7 9 11 23 54 75
```

[Rešenje 3.16]

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

Test 1

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

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

Izlazna datoteka (ceo-tok.txt):

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

[Rešenje 3.17]

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

Upotreba programa 1

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

Upotreba programa 2

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

Upotreba programa 3

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

[Rešenje 3.18]

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

(a) njihovog rastojanja od koordinatnog početka,

(b) `x` koordinata tačaka,

(c) `y` koordinata tačaka.

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

Test 1

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

Test 2

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

[Rešenje 3.19]

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

Test 1

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

[Rešenje 3.20]

Zadatak 3.21 Definisana je struktura podataka

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

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

Test 1

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

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

Test 1

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

[Rešenje 3.22]

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

Upotreba programa 1

```
Ulazna datoteka (artikli.txt):
  1001 Keks Jaffa 120
  2530 Napolitanke Bambi 230
  0023 Medeno_srce Pionir 150
  2145 Pardon Marbo 70
Interakcija programa:
  Asortiman:
  KOD                Naziv artikla     Ime proizvodjaca       Cena
           23           Medeno_srce               Pionir     150.00
         1001                 Keks                Jaffa     120.00
         2145               Pardon                Marbo      70.00
         2530           Napolitanke                Bambi     230.00
  ---------------------------
  - Za kraj za kraj rada kase, pritisnite CTRL+D!
  - Za nov racun unesite kod artikla!

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

    UKUPNO: 270.00 dinara.

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

  232
    GRESKA: Ne postoji proizvod sa 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

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

[Rešenje 3.24]

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

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

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

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

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.25]

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

Test 1

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

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

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

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

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.28 Napisati program koji ilustruje upotrebu bibiliotečkih 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 standardni izlaz ispisati odgovarajuću poruku.

Upotreba programa 1

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

Upotreba programa 2

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

[Rešenje 3.28]

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

Upotreba programa 1

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

[Rešenje 3.29]

Zadatak 3.30 Korišćenjem 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 standardni izlaz.

Test 1

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

[Rešenje 3.30]

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

[Rešenje 3.31]

Zadatak 3.32 Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova ili poruka ukoliko nema takvog. Potom, sa standardnom 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.

Test 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 karaktera). Sortirati niz `S` (bibliotečkom funkcijom `qsort`) i proveriti da li u njemu ima identičnih stringova.

Test 1

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

Test 2

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

[Rešenje 3.34]

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

Test 1

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

[Rešenje 3.35]

Zadatak 3.36 Definisana je struktura:

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

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

Test 1

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

Zadatak 3.37 Za 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 standardni izlaz.

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
```

[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 standardni izlaz.

3.4 Rešenja

Rešenje 3.1

```
1 #include <stdio.h>
  #include <stdlib.h>
3 #include <time.h>
  #define MAX 1000000
5
```

```c
/* 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) {
    /* Racunamo srednji indeks */
    srednji = (levi + desni) / 2;
    /* Ako je srednji element veci od x, tada se x mora nalaziti
       u levoj polovini niza */
    if (x < a[srednji])
      desni = srednji - 1;
    /* Ako je srednji element manji od x, tada se x mora
       nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
      levi = srednji + 1;
    else
      /* Ako je srednji element jednak x, tada smo pronasli x na
         poziciji srednji */
      return srednji;
  }
  /* Ako nije pronadjen vracamo -1 */
  return -1;
}

/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
   indeks pozicije nadjenog elementa ili -1, ako element nije
   pronadjen */
int interpolaciona_pretraga(int a[], int n, int x)
{
```

```
  int levi = 0;
59  int desni = n - 1;
  int srednji;
61  /* Dokle god je indeks levi levo od indeksa desni...  */
  while (levi <= desni) {
63    /* Ako je element manji od pocetnog ili veci od poslednjeg
         clana u delu niza a[levi],...,a[desni] tada nije u tom
65       delu niza. Ova provera je neophodna, da se ne bi dogodilo
         da se prilikom izracunavanja indeksa srednji izadje izvan
67       opsega indeksa [levi,desni] */
    if (x < a[levi] || x > a[desni])
69      return -1;
    /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
71       a[levi] i a[desni] jednaki, tada je jasno da je x jednako
         ovim vrednostima, pa vracamo indeks levi (ili indeks
73       desni. Ova provera je neophodna, zato sto bismo inace
         prilikom izracunavanja srednji imali deljenje nulom. */
75  else if (a[levi] == a[desni])
      return levi;
77  /* Racunamo srednji indeks */
    srednji =
79      levi +
        ((double) (x - a[levi]) / (a[desni] - a[levi])) *
81      (desni - levi);
    /* Napomena: Indeks srednji je uvek izmedju levi i desni,
83       ali ce verovatno biti blize trazenoj vrednosti nego da
         smo prosto uvek uzimali srednji element. Ovo se moze
85       porediti sa pretragom recnika: ako neko trazi rec na
         slovo 'B', sigurno nece da otvori recnik na polovini, vec
87       verovatno negde blize pocetku. */
    /* Ako je srednji element veci od x, tada se x mora nalaziti
89       u levoj polovini niza */
    if (x < a[srednji])
91    desni = srednji - 1;
    /* Ako je srednji element manji od x, tada se x mora
93       nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
95    levi = srednji + 1;
    else
97    /* Ako je srednji element jednak x, tada smo pronasli x na
         poziciji srednji */
99    return srednji;
  }
101 /* Ako nije pronadjen vracamo -1 */
  return -1;
103 }

105 /* Funkcija main */
int main(int argc, char **argv)
107 {
  int a[MAX];
109 int n, i, x;
```

```
  struct timespec time1, time2, time3, time4, time5, time6;
111 FILE *f;

113 /* Provera argumenata komandne linije */
  if (argc != 3) {
115   fprintf(stderr,
          "koriscenje programa: %s dim_niza trazeni_br\n",
117       argv[0]);
    exit(EXIT_FAILURE);
119 }

121 /* Dimenzija niza */
  n = atoi(argv[1]);
123 if (n > MAX || n <= 0) {
    fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
125   exit(EXIT_FAILURE);
  }
127
  /* Broj koji se trazi */
129 x = atoi(argv[2]);

131 /* Elemente niza odredjujemo slucajno, tako da je svaki
     sledeci veci od prethodnog. srandom() funkcija obezbedjuje
133    novi seed za pozivanje random() funkcije. Kako nas niz ne
     bi uvek isto izgledao seed smo postavili na tekuce vreme u
135    sekundama od Nove godine 1970. random()%100 daje brojeve
     izmedju 0 i 99 */
137 srandom(time(NULL));
  for (i = 0; i < n; i++)
139   a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;

141 /* Lineara pretraga */
  printf("Linearna pretraga\n");
143 /* Racunamo vreme proteklo od Nove godine 1970 */
  clock_gettime(CLOCK_REALTIME, &time1);
145 /* Pretrazujemo niz */
  i = linearna_pretraga(a, n, x);
147 /* Racunamo novo vreme i razlika predstavlja vreme utroseno za
     lin pretragu */
149 clock_gettime(CLOCK_REALTIME, &time2);
  if (i == -1)
151   printf("Element nije u nizu\n");
  else
153   printf("Element je u nizu na poziciji %d\n", i);
  printf("---------------------------------\n");
155
  /* Binarna pretraga */
157 printf("Binarna pretraga\n");
  clock_gettime(CLOCK_REALTIME, &time3);
159 i = binarna_pretraga(a, n, x);
  clock_gettime(CLOCK_REALTIME, &time4);
161 if (i == -1)
```

```
    printf("Element nije u nizu\n");
  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");

  /* Upisujemo podatke o izvrsavanju programa 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);

  /* Zatvaramo datoteku */
  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;
  /* Srednja pozicija na kojoj se trazi vrednost x */
  srednji = (l + d) / 2;
  /* Ako je sredisnji element trazeni */
  if (a[srednji] == x)
    return srednji;
  /* Ako je trazeni broj veci od srednjeg, pretrazujemo desnu
     polovinu niza */
  if (a[srednji] < x)
    return bin_pretgraga_rek(a, srednji + 1, d, x);
  /* Ako je trazeni broj manji od srednjeg, pretrazujemo levu
     polovinu 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
64    return interp_pretgraga_rek(a, l, p - 1, x);
}
66
#define MAX 1024
68
int main()
70 {
  int a[MAX];
72  int x;
  int i, indeks;
74
  /* Ucitavamo trazeni broj */
76  printf("Unesite trazeni broj: ");
  scanf("%d", &x);
78
  /* Ucitavamo elemente niza sve do kraja ulaza - ocekujemo da
80     korisnik pritisne CTRL+D za naznaku kraja */
  i = 0;
82  printf("Unesite sortiran niz elemenata: ");
  while (scanf("%d", &a[i]) == 1) {
84    i++;
  }
86
  /* Linearna pretraga */
88  printf("Linearna pretraga\n");
  indeks = lin_pretgraga_rek_sufiks(a, i, x);
90  if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
92  else
    printf("Pozicija elementa je %d.\n", indeks);
94
  /* Binarna pretraga */
96  printf("Binarna pretraga\n");
  indeks = bin_pretgraga_rek(a, 0, i - 1, x);
98  if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
100 else
    printf("Pozicija elementa je %d.\n", indeks);
102
  /* Interpolaciona pretraga */
104 printf("Interpolaciona pretraga\n");
  indeks = interp_pretgraga_rek(a, 0, i - 1, x);
106 if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
108 else
    printf("Pozicija elementa je %d.\n", indeks);
110
  return 0;
112 }
```

Rešenje 3.3

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

/* O svakom studentu imamo 3 informacije i njih objedinjujemo u
   strukturu kojom cemo predstavljati svakog studenta. */
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 prema indeksu, jer cemo
   ih, redom, kako citamo smestati u niz, a u datoteci su vec
   smesteni sortirani rastuce prema broju indeksa. Iz tog
   razloga pretragu po indeksu cemo vrsiti binarnom pretragom,
   dok pretragu po prezimenu moramo vrsiti linearno, jer nemamo
   garancije da postoji uredjenje po prezimenu. */

/* Funkcija trazi u sortiranom nizu studenata a[] duzine n
   studenta sa indeksom x. 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) {
    /* Racunamo srednji indeks */
    srednji = (levi + desni) / 2;
    /* Ako je srednji element veci od x, tada se x mora nalaziti
       u levoj polovini niza */
    if (x < a[srednji].indeks)
      desni = srednji - 1;
    /* Ako je srednji element manji od x, tada se x mora
       nalaziti u desnoj polovini niza */
    else if (x > a[srednji].indeks)
      levi = srednji + 1;
    else
      /* Ako je srednji element jednak x, tada smo pronasli x na
         poziciji srednji */
      return srednji;
  }
  /* Ako nije pronadjen vracamo -1 */
```

```
51    return -1;
}
53
/* Linearnom pretragom niza studenata trazimo prezime x */
55 int linearna_pretraga(Student a[], int n, char x[])
{
57    int i;
    for (i = 0; i < n; i++)
59      /* Poredimo prezime i-tog studenta i poslato x */
      if (strcmp(a[i].prezime, x) == 0)
61        return i;
    return -1;
63 }

65 /* Main funkcija mora imate argumente jer se ime datoteke dobija
      kao argument komandne linije */
67 int main(int argc, char *argv[])
{
69    /* Ucitacemo redom sve studente iz datoteke u niz. */
    Student dosije[MAX_STUDENATA];
71    FILE *fin = NULL;
    int i;
73    int br_studenata = 0;
    long trazen_indeks = 0;
75    char trazeno_prezime[MAX_DUZINA];

77    /* Proveravamo da li nam je korisnik prilikom poziva prosledio
        ime datoteke sa informacijama o studentima */
79    if (argc != 2) {
      fprintf(stderr,
81              "Greska: Program se poziva sa %s ime_datoteke\n",
              argv[0]);
83      exit(EXIT_FAILURE);
    }
85
    /* Otvaramo datoteku */
87    fin = fopen(argv[1], "r");
    if (fin == NULL) {
89      fprintf(stderr,
              "Neuspesno otvaranje datoteke %s za citanje\n",
91              argv[1]);
      exit(EXIT_FAILURE);
93    }

95    /* Citamo sve dok imamo red sa informacijama o studentu */
    i = 0;
97    while (1) {
      if (i == MAX_STUDENATA)
99        break;
      if (fscanf
101         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
           dosije[i].prezime) != 3)
```

```
103       break;
      i++;
105   }
    br_studenata = i;
107
    /* Nakon citanja datoteka nam vise nije neophodna i odmah je
109      zatvaramo */
    fclose(fin);
111
    /* Unos indeksa koji se binarno trazi u nizu */
113 printf("Unesite indeks studenta cije informacije zelite: ");
    scanf("%ld", &trazen_indeks);
115 i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
    /* Rezultat binarne pretrage */
117 if (i == -1)
      printf("Ne postoji student sa indeksom %ld\n",
119           trazen_indeks);
    else
121   printf("Indeks: %ld, Ime i prezime: %s %s\n",
          dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
123
    /* Unos prezimena koje se linearno trazi u nizu */
125 printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
127 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    /* Rezultat linearne pretrage */
129 if (i == -1)
      printf("Ne postoji student sa prezimenom %s\n",
131           trazeno_prezime);
    else
133   printf("Indeks: %ld, Ime i prezime: %s %s\n",
          dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135
    return 0;
137 }
```

Rešenje 3.4

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

5 #define MAX_STUDENATA 128
  #define MAX_DUZINA 16
7
  typedef struct {
9   long indeks;
    char ime[MAX_DUZINA];
11  char prezime[MAX_DUZINA];
  } Student;
13
```

```
int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
{
  /* Ako je indeks elementa na levom kraju veci od indeksa
     elementa na desnom kraju dela niza koji se pretrazuje, onda
     zapravo pretrazujemo prazan deo niza. U praznom nizu nema
     elementa koji trazimo i zato vracamo -1 */
  if (levi > desni)
    return -1;
  /* Racunamo indeks srednjeg elementa */
  int srednji = (levi + desni) / 2;
  /* Da li je srednji, bas onaj kog trazimo? */
  if (a[srednji].indeks == x) {
    return srednji;
  }
  /* Ako je trazeni indeks manji od indeksa srednjeg, onda
     potragu nastavljamo u levoj polovini niza jer znamo 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, vracamo -1, jer ga ne mozemo naci */
  if (n == 0)
    return -1;
  /* Kako trazimo prvog studenta sa trazenim prezimenom,
     pocinjemo 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, vracamo -1, jer ga ne mozemo naci */
  if (n == 0)
    return -1;
  /* Kako trazimo poslednjeg studenta sa trazenim prezimenom,
     pocinjemo 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 dobija
   kao argument komandne linije */
```

```
int main(int argc, char *argv[])
{
  /* Ucitacemo redom sve studente iz datoteke u niz. */
  Student dosije[MAX_STUDENATA];
  FILE *fin = NULL;
  int i;
  int br_studenata = 0;
  long trazen_indeks = 0;
  char trazeno_prezime[MAX_DUZINA];

  /* Proveravamo da li nam 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);
  }

  /* Otvaramo datoteku */
  fin = fopen(argv[1], "r");
  if (fin == NULL) {
    fprintf(stderr,
            "Neuspesno otvaranje datoteke %s za citanje\n",
            argv[1]);
    exit(EXIT_FAILURE);
  }

  /* Citamo sve dok imamo 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 je
     zatvaramo */
  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);

  printf("Unesite prezime studenta cije informacije zelite: ");
  scanf("%s", trazeno_prezime);
  i = linearna_pretraga_rekurzivna(dosije, br_studenata,
                                   trazeno_prezime);
  if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n",
            trazeno_prezime);
  else
    printf
        ("Poslednji 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;
}

/* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
   tacaka t dimenzije n */
Tacka najbliza_x_osi(Tacka t[], int n)
{

  Tacka najbliza;
  int i;
  najbliza = t[0];
  for (i = 1; i < n; i++) {
    if (fabs(t[i].x) < fabs(najbliza.x)) {
      najbliza = t[i];
    }
  }
  return najbliza;
}

/* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
   tacaka t dimenzije n */
Tacka najbliza_y_osi(Tacka t[], int n)
{
  Tacka najbliza;
  int i;
  najbliza = t[0];
  for (i = 1; i < n; i++) {
    if (fabs(t[i].y) < fabs(najbliza.y)) {
      najbliza = t[i];
    }
  }
  return najbliza;
}

#define MAX 1024

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

  /* Ocekujemo da korisnik unese barem ime izvrsne verzije
     programa i ime datoteke sa tackama */
  if (argc < 2) {
    fprintf(stderr,
            "koriscenje programa: %s ime_datoteke\n", argv[0]);
    return EXIT_FAILURE;
  }
```

```
82    /* Otvaramo datoteku za citanje */
      ulaz = fopen(argv[1], "r");
84    if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86              argv[1]);
        return EXIT_FAILURE;
88    }

90    /* Sve dok ima tacaka u datoteci, smestamo ih u niz sa
         tackama; i predstavlja indeks tekuce tacke */
92    i = 0;
      while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94      i++;
      }
96    n = i;

98    /* Proveravamo koji su dodatni argumenti komandne linije. Ako
         nema dodatnih argumenata */
100   if (argc == 2)
        /* Trazimo najblizu tacku u odnosu na koordinatni pocetak */
102     najbliza = najbliza_koordinatnom(tacke, n);
      /* Inace proveravamo koji je dodatni argument. Ako je u
104      pitanju opcija -x */
      else if (strcmp(argv[2], "-x") == 0)
106     /* Racunamo rastojanje u odnosu na x osu */
        najbliza = najbliza_x_osi(tacke, n);
108   /* Ako je u pitanju opcija -y */
      else if (strcmp(argv[2], "-y") == 0)
110     /* Racunamo rastojanje u odnosu na y osu */
        najbliza = najbliza_y_osi(tacke, n);
112   else {
        /* Ako nije zadata opcija -x ili -y, ispisujemo obavestenje
114        za korisnika i prekidamo izvrsavanje programa */
        fprintf(stderr, "Pogresna opcija\n");
116     return EXIT_FAILURE;
      }
118

      /* Stampamo koordinate trazene tacke */
120   printf("%g %g\n", najbliza.x, najbliza.y);

122   /* Zatvaramo datoteku */
      fclose(ulaz);
124

      return 0;
126 }
```

Rešenje 3.6

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

```c

/* Tacnost */
#define EPS 0.001

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

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

  /* Sve dok ne pronadjemo trazenu vrednost argumenta */
  while (1) {
    /* Pronalazimo sredinu intervala */
    s = (l + d) / 2;
    /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
       tacnosti, prekidamo pretragu */
    if (fabs(cos(s)) < EPS) {
      break;
    }
    /* Ako je nula u levom delu intervala, nastavljamo pretragu
       na intervalu [l, s] */
    if (cos(l) * cos(s) < 0)
      d = s;
    else
      /* Inace, nastavljamo pretragu na intervalu [s, d] */
      l = s;
  }

  /* Stampamo 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) {
    /* Racunamo sredisnju poziciju */
    s = (l + d) / 2;
```

```
    /* Ako je broj na toj poziciji veci od nule a eventualni
       njegov prethodnik manji ili jednak nuli */
    if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
      return s;
    /* Pretrazujemo desnu polovinu niza */
    if (niz[s] <= 0)
      l = s + 1;
    /* Pretrazujemo levu polovinu binarnog zapisa */
    else
      d = s - 1;
  }
  return -1;
}

#define MAX 256

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

  /* Unos niza */
  printf("Unesi rastuce sortiran niz celih brojeva: ");
  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) {
    /* Racunamo sredisnju poziciju */
    s = (l + d) / 2;
    /* Ako je broj na toj poziciji manji od nule a eventualni
       njegov prethodnik veci ili jednak nuli */
    if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
      return s;
    /* Pretrazujemo desnu polovinu niza */
```

```
    if (niz[s] >= 0)
      l = s + 1;
    /* Pretrazujemo levu polovinu binarnog zapisa */
    else
      d = s - 1;
  }
  return -1;
}

#define MAX 256

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

  /* Unos niza */
  printf("Unesi opadajuce sortiran niz celih brojeva: ");
  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 trazimo jedinicu u binarnom zapisu broja
     x najvece vaznosti, tj. najlevlju. Pretragu radimo od
     pozicije 0 do 31 */
  int d = 0, l = sizeof(unsigned int) * 8 - 1;
  int s;
  /* Sve dok je desna granica pretrage desnije od leve */
  while (d <= l) {
    /* Racunamo sredisnju poziciju */
```

```
    s = (l + d) / 2;
    /* Proveravamo da li je na toj poziciji trazena jedinica */
    if ((1 << s) <= x && (1 << (s + 1)) > x)
      return s;
    /* Pretrazujemo desnu polovinu binarnog zapisa */
    if ((1 << s) > x)
      l = s - 1;
    /* Pretrazujemo levu polovinu binarnog zapisa */
    else
      d = s + 1;
  }
  return s;
}

int main()
{
  unsigned int x;

  /* Unos podatka */
  printf("Unesi pozitivan ceo broj: ");
  scanf("%u", &x);

  /* Provera da li je uneti broj pozitivan */
  if (x == 0) {
    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
```

```
16      navedenih elemenata. */
    for (i = 0; i < n - 1; i++) {
18    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
         nalazi najmanji od elemenata a[i],...,a[n-1]. */
20    min = i;
      for (j = i + 1; j < n; j++)
22      if (a[j] < a[min])
          min = j;
24    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
         samo ako su (i) i min razliciti, inace je nepotrebno. */
26    if (min != i) {
        pom = a[i];
28      a[i] = a[min];
        a[min] = pom;
30    }
    }
32  }

34  /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja
       u sortiranom nizu celih brojeva */
36  int najmanje_rastojanje(int a[], int n)
    {
38    int i, min;
      min = a[1] - a[0];
40    for (i = 2; i < n; i++)
        if (a[i] - a[i - 1] < min)
42        min = a[i] - a[i - 1];
      return min;
44  }


46
    int main()
48  {
      int i, a[MAX];
50
      /* Ucitavaju se elementi niza sve do kraja ulaza */
52    i = 0;
      printf("Unesite elemente niza: ");
54    while (scanf("%d", &a[i]) != EOF)
        i++;
56
      /* Sortiranje */
58    selectionSort(a, i);

60    /* Ispis rezultata */
      printf("%d\n", najmanje_rastojanje(a, i));
62
      return 0;
64  }
```

Rešenje 3.13

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

#define MAX_DIM 128

/* Funkcija za sortiranje niza */
void selectionSort(char s[], int n)
{
  int i, j, min;
  char pom;
  for (i = 0; i < n; i++) {
    min = i;
    for (j = i + 1; j < n; 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 jesu anagrami; 0 - inace.
   pretpostavlja se da su niske s i t sortirane */
int anagrami(char s[], char t[], int n_s, int n_t)
{
  int i, n;

  /* Ako dve niske imaju razlicit broj elemenata onda nisu
     anagrami */
  if (n_s != n_t)
    return 0;

  n = n_s;

  /* Dve sortirane niske su anagrami akko su jednake */
  for (i = 0; i < n; i++)
    if (s[i] != t[i])
      return 0;
  return 1;
}

int main()
{
  char s[MAX_DIM], t[MAX_DIM];
  int n_s, n_t;

  /* Ucitavamo dve niske sa ulaza */
  printf("Unesite prvu nisku: ");
  scanf("%s", s);
```

```
52    printf("Unesite drugu nisku: ");
      scanf("%s", t);
54
      /* Odredjujemo duzinu niski */
56    n_s = strlen(s);
      n_t = strlen(t);
58
      /* Sortiramo niske */
60    selectionSort(s, n_s);
      selectionSort(t, n_t);
62
      /* Proveravamo da li su niske anagrami */
64    if (anagrami(s, t, n_s, n_t))
        printf("jesu\n");
66    else
        printf("nisu\n");
68    return 0;
}
```

Rešenje 3.14

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

4 /* Funkcija za sortiranje niza */
void selectionSort(int s[], int n)
6 {
    int i, j, min;
8   char pom;
    for (i = 0; i < n; i++) {
10    min = i;
      for (j = i + 1; j < n; j++)
12      if (s[j] < s[min])
          min = j;
14    if (min != i) {
        pom = s[i];
16      s[i] = s[min];
        s[min] = pom;
18    }
    }
20 }

22 /* Funkcija za odredjivanje onog elementa sortiranog niza koji
      se najvise puta pojavio u tom nizu */
24 int najvise_puta(int a[], int n)
{
26   int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
    /* Za i-ti element izracunavamo koliko se puta pojavio u nizu */
28  for (i = 0; i < n; i = j) {
      br_pojava = 1;
30    for (j = i + 1; j < n && a[i] == a[j]; j++)
```

```
      br_pojava++;
    /* Ispitujemo da li se do tog trenutka i-ti element pojavio
       najvise puta u nizu */
    if (br_pojava > max_br_pojava) {
      max_br_pojava = br_pojava;
      i_max_pojava = i;
    }
  }
  /* Vracamo element koji se najvise puta pojavio u nizu */
  return a[i_max_pojava];
}

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

  /* Ucitavaju se elementi niza sve do kraja ulaza */
  i = 0;
  printf("Unesite elemente niza: ");
  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. funkcija vraca: 1 - ako se
   element x nalazi u nizu; 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

```c
#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 */
  if (dim3 < dim1 + dim2)
    return -1;

  /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
     jednog od njih */
  while (i < dim1 && j < dim2) {
    if (niz1[i] < niz2[j])
      niz3[k++] = niz1[i++];
    else
      niz3[k++] = niz2[j++];
  }
  /* Ostatak prvog niza prepisujemo u treci */
  while (i < dim1)
    niz3[k++] = niz1[i++];

  /* Ostatak drugog niza prepisujemo u treci */
  while (j < dim2)
    niz3[k++] = niz2[j++];
  return dim1 + dim2;
}

int main()
{
  int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
  int i = 0, j = 0, k, dim3;

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

```
51      j++;
    }
53
    /* Poziv trazene funkcije */
55  dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

57  /* Ispis niza */
    for (k = 0; k < dim3; k++)
59    printf("%d ", niz3[k]);
    printf("\n");
61
    return 0;
63  }
```

Rešenje 3.17

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

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

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

```
36  /* Otvaranje datoteke za upis rezultata */
    fout = fopen("ceo-tok.txt", "w");
38  if (fout == NULL) {
      fprintf(stderr,
40            "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
      exit(EXIT_FAILURE);
42  }

44  /* Citamo narednog studenta iz prve datoteke */
    if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
46    kraj1 = 1;

48  /* Citamo narednog studenta iz druge datoteke */
    if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
50    kraj2 = 1;

52  /* Sve dok nismo dosli do kraja neke datoteke */
    while (!kraj1 && !kraj2) {
54    if (strcmp(ime1, ime2) < 0) {
        /* Ime i prezime iz prve datoteke je leksikografski
56         ranije, upisujemo ga u izlaznu datoteku */
        fprintf(fout, "%s %s\n", ime1, prezime1);
58      /* Citamo narednog studenta iz prve datoteke */
        if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
60        kraj1 = 1;
      } else {
62      /* Ime i prezime iz druge datoteke je leksikografski
           ranije, upisujemo ga u izlaznu datoteku */
64      fprintf(fout, "%s %s\n", ime2, prezime2);
        /* Citamo narednog studenta iz druge datoteke */
66      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
          kraj2 = 1;
68    }
    }
70
    /* Ako smo iz prethodne petlje izasli zato sto se doslo do
72     kraja druge datoteke, onda ima jos imena u prvoj datoteci,
       i prepisujemo ih, redom, jer su vec sortirani po imenu. */
74  while (!kraj1) {
      fprintf(fout, "%s %s\n", ime1, prezime1);
76    if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
        kraj1 = 1;
78  }

80  /* Ako smo iz prve petlje izasli zato sto se doslo do kraja
       prve datoteke, onda ima jos imena u drugoj datoteci, i
82     prepisujemo ih, redom, jer su vec sortirani po imenu. */
    while (!kraj2) {
84    fprintf(fout, "%s %s\n", ime2, prezime2);
      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
86      kraj2 = 1;
    }
```

```
88
   /* Zatvaramo datoteke */
90 fclose(fin1);
   fclose(fin2);
92 fclose(fout);

94 return 0;
}
```

Rešenje 3.18

```
1  /* Datoteka sort.h */
   #ifndef __SORT_H__
3  #define __SORT_H__ 1

5  /* Selection sort */
   void selectionsort(int a[], int n);
7  /* Insertion sort */
   void insertionsort(int a[], int n);
9  /* Bubble sort */
   void bubblesort(int a[], int n);
11 /* Shell sort */
   void shellsort(int a[], int n);
13 /* Merge sort */
   void mergesort(int a[], int l, int r);
15 /* Quick sort */
   void quicksort(int a[], int l, int r);
17
   #endif
```

```
   /* Datoteka sort.c */
2
   #include "sort.h"
4
   /* Funkcija sortira niz celih brojeva metodom sortiranja
6     izborom. Ideja algoritma je sledeca: U svakoj iteraciji
      pronalazimo najmanji element i postavljamo ga na pocetak
8     niza. Dakle, u prvoj iteraciji, pronalazimo najmanji element,
      i dovodomo ga na nulto mesto u nizu. U i-toj iteraciji
10    najmanjih i elemenata su vec na svojim pozicijama, pa od i+1
      do n-1 elementa trazimo najmanji, koji dovodimo na i+1
12    poziciju. */
   void selectionsort(int a[], int n)
14 {
     int i, j;
16   int min;
     int pom;
18
     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
20      medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
```

```
     poziciju i, dok se element na pozciji 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 sortira niz celih brojeva metodom sortiranja
   umetanjem. Ideja algoritma je sledeca: neka je na pocetku
   i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
   sortirano. U i-toj iteraciji zelimo da element a[i] umetnemo
   na pravu poziciju medju prvih i elemenata tako da dobijemo
   niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
   element najpre uporedimo sa njegovim prvim levim susedom
   (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i
   niz a[0],a[1],...,a[i] je sortiran, pa mozemo preci na
   sledecu iteraciju. Ako je a[i-1] vece, tada zamenjujemo a[i]
   i a[i-1], a zatim proveravamo da li je potrebno dalje
   potiskivanje elementa u levo, poredeci ga sa njegovim novim
   levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
   pravo mesto u nizu. */
void insertionsort(int a[], int n)
{
  int i, j;

  /* Na pocetku iteracije pretpostavljamo da je niz
     a[0],...,a[i-1] sortiran */
  for (i = 1; i < n; i++) {

    /* U ovoj petlji redom potiskujemo element a[i] u levo
       koliko je potrebno, dok ne zauzme pravo mesto, tako da
       niz a[0],...a[i] bude sortiran. Indeks j je trenutna
       pozicija na kojoj se element koji umecemo nalazi. Petlja
       se zavrsava ili kada element dodje do levog kraja (j==0)
       ili dok ne naidjemo na element a[j-1] koji je manji od
       a[j]. */
```

```
    for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
      int temp = a[j];
      a[j] = a[j - 1];
      a[j - 1] = temp;
    }
  }
}

/* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
   algoritma je sledeca: prolazimo kroz niz redom poredeci
   susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
   poretku. Ovim se najveci element poput mehurica istiskuje na
   "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je
   potrebno ovaj postupak ponoviti nad nizom a[0],...,a[n-2],
   tj. nad prvih n-1 elemenata niza bez poslednjeg koji je
   postavljen na pravu poziciju. Nakon toga se istu postupak
   ponavlja nad sve kracim i kracim prefiksima niza, cime se
   jedan po jedan istiskuju elemenenti na svoje prave pozicije. */
void bubblesort(int a[], int n)
{
  int i, j;
  int ind;

  for (i = n, ind = 1; i > 1 && ind; i--)

    /* Poput "mehurica" potiskujemo najveci element medju
       elementima od a[0] do a[i-1] na poziciju i-1 uporedjujuci
       susedne elemente niza i potiskujuci veci u desno */
    for (j = 0, ind = 0; j < i - 1; j++)
      if (a[j] > a[j + 1]) {
        int temp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = temp;

        /* Promenljiva ind registruje da je bilo premestanja.
           Samo u tom slucaju ima smisla ici na sledecu
           iteraciju, jer ako nije bilo premestanja, znaci da su
           svi elementi vec u dobrom poretku, pa nema potrebe
           prelaziti na kraci prefiks niza. Moglo je naravno i
           bez ovoga, algoritam bi radio ispravno, ali bi bio
           manje efikasan, jer bi cesto nepotrebno vrsio mnoga
           uporedjivanja, kada je vec jasno da je sortiranje
           zavrseno. */
        ind = 1;
      }
}

/* Selsort je jednostavno prosirenje sortiranja umetanjem koje
   dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
   sastoji u tome da se kroz algoritam umetanja prolazi vise
   puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
```

```
   koji je manji od n (sto omogucuje razmenu udaljenih
   elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
   elementi na rastojanju h sortirani, mada susedni elementi to
   ne moraju biti. U drugom 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 dobija potpuno
   sortirani niz. Izbor pocetne vrednosti za h, i nacina
   njegovog smanjivanja menja u nekim slucajevima brzinu
   algoritma, ali bilo koja vrednost ce rezultovati ispravnim
   sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
   vrednost 1. */
void shellsort(int a[], int n)
{
  int h = n / 2, i, j;
  while (h > 0) {
    /* Insertion sort sa korakom h */
    for (i = h; i < n; i++) {
      int temp = a[i];
      j = i;
      while (j >= h && a[j - h] > temp) {
        a[j] = a[j - h];
        j -= h;
      }
      a[j] = temp;
    }
    h = h / 2;
  }
}

#define MAX 1000000

/* Funkcija sortira niz celih brojeva a[] ucesljavanjem.
   Sortiranje se vrsi od elementa na poziciji l do onog na
   poziciji d. Na pocetku, da bismo dobili niz kompletno
   sortiran, l mora biti 0, a d je jednako poslednjem validnom
   indeksu u nizu. Funkcija niz podeli na dve polovine, levu i
   desnu, koje zatim rekurzivno sortira. Od ova dva sortirana
   podniza, dobijamo sortiran niz ucesljavanjem, tj.
   istovremenim prolaskom kroz oba niza i izborom trenutnog
   manjeg elementa koji se smesta u pomocni niz. Na kraju
   algoritma, sortirani elementi su u pomocnom nizu, koji se
   kopira u originalni niz. */
void mergesort(int a[], int l, int d)
{
  int s;
  static int b[MAX];            /* Pomocni niz */
  int i, j, k;

  /* Izlaz iz rekurzije */
  if (l >= d)
    return;

```

```
  /* Odredjujemo sredisnji indeks */
178   s = (l + d) / 2;

180   /* Rekurzivni pozivi */
      mergesort(a, l, s);
182   mergesort(a, s + 1, d);

184   /* Inicijalizacija indeksa. Indeks i prolazi krozi levu
         polovinu niza, dok indeks j prolazi kroz desnu polovinu
186      niza. Indeks k prolazi kroz pomocni niz b[] */
      i = l;
188   j = s + 1;
      k = 0;
190
      /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
192   while (i <= s && j <= d) {
        if (a[i] < a[j])
194       b[k++] = a[i++];
        else
196       b[k++] = a[j++];
      }
198
      /* U slucaju da se prethodna petlja zavrsila izlaskom
200      promenljive j iz dopustenog opsega u pomocni niz
         prepisujemo ostatak leve polovine niza */
202   while (i <= s)
        b[k++] = a[i++];
204
      /* U slucaju da se prethodna petlja zavrsila izlaskom
206      promenljive i iz dopustenog opsega u pomocni niz
         prepisujemo ostatak desne polovine niza */
208   while (j <= d)
        b[k++] = a[j++];
210
      /* Prepisujemo "ucesljani" niz u originalni niz */
212   for (k = 0, i = l; i <= d; i++, k++)
        a[i] = b[k];
214 }

216 /* Funkcija menja mesto i-tom i j-tom elementu niza a */
    void swap(int a[], int i, int j)
218 {
      int tmp = a[i];
220   a[i] = a[j];
      a[j] = tmp;
222 }


224
    /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r.
226      Njena ideja sortiranja je izbor jednog elementa niza, koga
         nazivamo pivot, koga cemo dovesti na svoje mesto. Posle ovog
228      koraka, svi elementi levo od njega bice manji, a svi desno
```

```
   bice veci od njega. Kako smo pivota doveli na svoje mesto, da
   bismo imali kompletno sortiran niz, treba sortirati elemente
   levo (manje) od njega, i elemente desno (vece). Kako je
   dimenzija ova dva podniza manja od dimenzije pocetgnom niza
   koji je trebalo sortirati, ovaj deo ce za nas uraditi
   rekurzija. */
void quicksort(int a[], int l, int r)
{
  int i, pivot_position;

  /* Izlaz iz rekurzije -- prazan niz */
  if (l >= r)
    return;


  /* Particionisanje niza. Svi elementi na pozicijama <=
     pivot_position (izuzev same pozicije l) su strogo manji od
     pivota. Kada se pronadje neki element manji od pivota,
     uvecava se pivot_position i na tu poziciju se premesta
     nadjeni element. Na kraju ce pivot_position zaista biti
     pozicija na koju treba smestiti pivot, jer ce svi elementi
     levo od te pozicije biti manji a desno biti veci ili
     jednaki od pivota. */
  pivot_position = l;
  for (i = l + 1; i <= r; i++)
    if (a[i] < a[l])
      swap(a, ++pivot_position, i);

  /* Postavljamo pivot na svoje mesto */
  swap(a, l, pivot_position);

  /* Rekurzivno sortiramo elemente manje od pivota */
  quicksort(a, l, pivot_position - 1);
  /* Rekurzivno sortiramo elemente vece 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
```

```
16      tip_sortiranja == 2 => bubblesort
                               -b opcija komandne linije
18      tip_sortiranja == 3 => shellsort
                               -s opcija komandne linije
20      tip_sortiranja == 4 => mergesort
                               -m opcija komandne linije
22      tip_sortiranja == 5 => quicksort
                               -q opcija komandne linije
24    ***********************************************/
    int tip_sortiranja = 0;
26  /***********************************************
       tip_niza == 0 => slucajno generisani nizovi
28                      (podrazumevano)
       tip_niza == 1 => rastuce sortirani nizovi
30                      -r opcija komandne linije
       tip_niza == 2 => opadajuce soritrani nizovi
32                      -o opcija komandne linije
    ***********************************************/
34  int tip_niza = 0;

36  /* Dimenzija niza koji se sortira */
    int dimenzija;
38  int i;
    int niz[MAX];
40
    /* Provera argumenata komandne linije */
42  if (argc < 2) {
      fprintf(stderr,
44            "Program zahteva bar 2 argumenta komandne linije!\n");
      exit(EXIT_FAILURE);
46  }

48  /* Ocitavamo opcije i argumente prilikom poziva programa */
    for (i = 1; i < argc; i++) {
50    /* Ako je u pitanju opcija... */
      if (argv[i][0] == '-') {
52      switch (argv[i][1]) {
        case 'i':
54        tip_sortiranja = 1;
          break;
56      case 'b':
          tip_sortiranja = 2;
58        break;
        case 's':
60        tip_sortiranja = 3;
          break;
62      case 'm':
          tip_sortiranja = 4;
64        break;
        case 'q':
66        tip_sortiranja = 5;
          break;
```

```
68        case 'r':
            tip_niza = 1;
70          break;
          case 'o':
72          tip_niza = 2;
            break;
74        default:
            printf("Pogresna opcija -%c\n", argv[i][1]);
76          return 1;
            break;
78        }
      }
80    /* Ako je u pitanju argument, onda je to duzina niza koji
         treba da se sortira */
82    else {
        dimenzija = atoi(argv[i]);
84      if (dimenzija <= 0 || dimenzija > MAX) {
          fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
86        exit(EXIT_FAILURE);
        }
88    }
  }
90
  /* Elemente niza odredjujemo slucajno, ali vodeci racuna o
92   tipu niza dobijenom iz komandni linije. srandom funkcija
     obezbedjuje novi seed za pozivanje random funkcije, i kako
94   nas niz ne bi uvek isto izgledao seed smo postavili na
     tekuce vreme u sekundama od Nove godine 1970. random()%100
96   daje brojeve izmedju 0 i 99 */
  srandom(time(NULL));
98 if (tip_niza == 0)
    for (i = 0; i < dimenzija; i++)
100   niz[i] = random();
  else if (tip_niza == 1)
102   for (i = 0; i < dimenzija; i++)
        niz[i] =
104         i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
  else
106   for (i = 0; i < dimenzija; i++)
        niz[i] =
108         i == 0 ? random() % 100 : niz[i - 1] - random() % 100;

110 /* Ispisujemo elemente niza */
  /************************************************
112   Ovaj deo je iskomentarisan jer ne zelimo da se sledeci ispis
      nadje na izlazu. Njegova svrha je samo bila provera da li je
114   niz generisan u skladu sa opcijama komandne linije.

116   printf("Niz koji sortiramo je:\n");
      for (i = 0; i < dimenzija; i++)
118     printf("%d\n", niz[i]);
  ************************************************/
```

```
120
122   /* Sortiramo niz na odgovarajuci nacin */
      if (tip_sortiranja == 0)
124     selectionsort(niz, dimenzija);
      else if (tip_sortiranja == 1)
126     insertionsort(niz, dimenzija);
      else if (tip_sortiranja == 2)
128     bubblesort(niz, dimenzija);
      else if (tip_sortiranja == 3)
130     shellsort(niz, dimenzija);
      else if (tip_sortiranja == 4)
132     mergesort(niz, 0, dimenzija - 1);
      else
134     quicksort(niz, 0, dimenzija - 1);

136   /* Ispisujemo elemente niza */
      /*****************************
138     Ovaj deo je iskomentarisan jer nismo zeleli da vreme potrebno
        za njegovo izvrsavanje bude ukljuceno u vreme izmereno
140     programom time. Takodje, kako je svrha ovog programa da prikaze
        vremena razlicitih algoritama sortiranja, dimenzije nizova ce
142     biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
        od toliko elemenata. Ovaj deo je koriscen u razvoju programa
144     zarad testiranja korektnosti.

146     printf("Sortiran niz je:\n");
        for (i = 0; i < dimenzija; i++)
148       printf("%d\n", niz[i]);
      *****************************/
150
      return 0;
152 }
```

Rešenje 3.19

```
1  #include <stdio.h>
   #include <string.h>
3  #include <math.h>
   #include <stdlib.h>
5
   #define MAX_BR_TACAKA 128
7
   /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
     int x;
11   int y;
   } Tacka;
13
   /* Funkcija racuna rastojanje zadate tacke od koordinatnog
15    pocetka (0,0) */
```

```
float rastojanje(Tacka A)
{
  return sqrt(A.x * A.x + A.y * A.y);
}

/* Funkcija koja sortira niz tacaka po rastojanju od
   koordinatnog pocetka */
void sortiraj_po_rastojanju(Tacka t[], int n)
{
  int min, i, j;
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (rastojanje(t[j]) < rastojanje(t[min])) {
        min = j;
      }
    }
    if (min != i) {
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}

/* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
{
  int min, i, j;
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (abs(t[j].x) < abs(t[min].x)) {
        min = j;
      }
    }
    if (min != i) {
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}

/* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
void sortiraj_po_y(Tacka t[], int n)
{
  int min, i, j;
```

```
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (abs(t[j].y) < abs(t[min].y)) {
        min = j;
      }
    }
    if (min != i) {
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}


int main(int argc, char *argv[])
{
  FILE *ulaz;
  FILE *izlaz;
  Tacka tacke[MAX_BR_TACAKA];
  int i, n;

  /* Proveravamo broj argumenata komandne linije: ocekujemo ime
     izvrsnog programa, opciju, ime ulazne datoteke i ime
     izlazne datoteke tj. ocekujemo 4 argumenta */
  if (argc != 4) {
    fprintf(stderr,
            "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
    return 0;
  }

  /* Otvaramo datoteku u kojoj su zadate tacke */
  ulaz = fopen(argv[2], "r");
  if (ulaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[2]);
    return 0;
  }

  /* Otvaramo datoteku u koju treba upisati rezultat */
  izlaz = fopen(argv[3], "w");
  if (izlaz == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
            argv[3]);
    return 0;
  }

  /* Sve dok ne stignemo do kraja ulazne datoteke ucitavamo
```

```
     koordinate tacaka i smestamo ih na odgovarajucu poziciju
     odredjenu brojacem i; prilikom ucitavanja oslanjamo se na
     svojstvo funkcije fscanf povratka EOF vrednosti kada stigne
     do kraja ulaza */
  i = 0;
  while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
    i++;
  }

  /* Cuvamo broj procitanih tacaka */
  n = i;

  /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
     "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
     (karakter -) i dalje proveravamo sta je na sledecoj
     poziciji tj. sta je argv[1][1] */

  switch (argv[1][1]) {
  case 'x':
    /* Ako je u pitanju karakter x, pozivamo funkciju za
       sortiranje po vrednosti x koordinate */
    sortiraj_po_x(tacke, n);
    break;
  case 'y':
    /* Ako je u pitanju karakter y, pozivamo funkciju za
       sortiranje po vrednosti y koordinate */
    sortiraj_po_y(tacke, n);
    break;
  case 'o':
    /* Ako je u pitanju karakter o, pozivamo funkciju za
       sortiranje po udaljenosti od koorinatnog pocetka */
    sortiraj_po_rastojanju(tacke, n);
    break;
  }

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

  /* Zatvaramo otvorene datoteke */
  fclose(ulaz);
  fclose(izlaz);

  /* Zavrsavamo sa programom */
  return 0;
}
```

Rešenje 3.20

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

#define MAX 1000
#define MAX_DUZINA 16

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

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

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

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

  for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
       nalazi najmanji od elemenata
       a[i].prezime,...,a[n-1].prezime. */
    min = i;
    for (j = i + 1; j < n; j++)
      if (strcmp(a[j].prezime, a[min].prezime) < 0)
        min = j;
```

```
    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
54     samo ako su (i) i min razliciti, inace je nepotrebno. */
    if (min != i) {
56    pom = a[i];
      a[i] = a[min];
58    a[min] = pom;
    }
60  }
}
62
/* Pretraga niza Gradjana */
64 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
{
66  int i;
  for (i = 0; i < n; i++)
68  if (strcmp(a[i].ime, x->ime) == 0
       && strcmp(a[i].prezime, x->prezime) == 0)
70    return i;
  return -1;
72 }

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

82  /* Otvaranje datoteke */
  if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
84  fprintf(stderr,
          "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
86  exit(EXIT_FAILURE);
  }
88
  /* Citanje sadrzaja */
90  for (i = 0;
       fscanf(fp, "%s %s", spisak1[i].ime,
92           spisak1[i].prezime) != EOF; i++)
    spisak2[i] = spisak1[i];
94  n = i;

96  /* Zatvaranje datoteke */
  fclose(fp);
98
  sort_ime(spisak1, n);
100
  /************************************************
102   Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih nizova. Koriscen je samo u fazi testiranja programa.
104
```

```
    printf("Biracki spisak [uredjen prema imenima]:\n");
    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;
```

```
  /* Obilazimo karakter po karakter zadate niske */
  for (i = 0; s[i]; i++) {
    /* Ako je u pitanju slovo */
    if (isalpha(s[i])) {
      /* Pretvaramo ga u veliko da bismo mogli da pokrijemo
         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') {
        /* Uvecavamo broj suglasnika */
        suglasnici++;
      }
    }
  }
  /* Vracamo izracunatu 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 uporedjujemo 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, reci imaju isti broj suglasnika uporedjujemo
           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, uporedjujemo ih leksikografski */
        if (duzina_j == duzina_min
            && strcmp(reci[j], reci[min]) < 0)
          min = j;
      }
    }
    if (min != i) {
```

```
      strcpy(tmp, reci[min]);
68    strcpy(reci[min], reci[i]);
      strcpy(reci[i], tmp);
70  }
  }
72}

74int main()
{
76
  FILE *ulaz;
78  int i = 0, n;

80  /* Niz u kojem ce biti smestane reci. Prvi broj oznacava broj
     reci, a drugi maksimalnu duzinu pojedinacne reci */
82  char reci[MAX_BR_RECI][MAX_DUZINA_RECI];

84  /* Otvaramo datoteku niske.txt za citanje */
  ulaz = fopen("niske.txt", "r");
86  if (ulaz == NULL) {
    fprintf(stderr,
88            "Greska prilikom otvaranja datoteke niske.txt!\n");
    return 0;
90  }

92  /* Sve dok mozemo da procitamo sledecu rec */
  while (fscanf(ulaz, "%s", reci[i]) != EOF) {
94    /* Proveravamo da li smo ucitali najvise dozvoljenih reci i
       ako jesmo, prekidamo ucitavanje */
96    if (i == MAX_BR_RECI)
      break;
98    /* Pripremamo brojac za narednu iteraciju */
    i++;
100 }

102 /* n je duzina naseg niza reci i predstavlja poslednju
     vrednost koriscenog brojaca */
104 n = i;
  /* Pozivamo funkciju za sortiranje reci - OPREZ: nacin
106    prosledjivanja niza reci */
  sortiraj_reci(reci, n);
108
  /* Ispisujemo sortirani niz reci */
110 for (i = 0; i < n; i++) {
    printf("%s ", reci[i]);
112 }
  printf("\n");
114
  /* Zatvaramo datoteku */
116 fclose(ulaz);

118 /* Zavrsavamo sa programom */
```

```
  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) {
    /* Racunamo sredisnji indeks */
    int srednji = (levi + desni) / 2;
    /* Ako je sredisnji element veci od x, tada se x mora
       nalaziti u levoj polovini niza */
    if (x < a[srednji].kod)
      desni = srednji - 1;
    /* Ako je sredisnji element manji od x, tada se x mora
       nalaziti u desnoj polovini niza */
    else if (x > a[srednji].kod)
      levi = srednji + 1;
    else
      /* Ako je sredisnji element jednak x, tada smo pronasli x
         na poziciji srednji */
      return srednji;
  }
  /* Ako nije pronadjen vracamo -1 */
  return -1;
}

/* Funkcija koja sortira niz artikala po bar kodovima rastuce */
void selection_sort(Artikal a[], int n)
{
  int i, j;
```

```
47    int min;
      Artikal pom;
49
      for (i = 0; i < n - 1; i++) {
51      min = i;
        for (j = i + 1; j < n; j++)
53        if (a[j].kod < a[min].kod)
            min = j;
55      if (min != i) {
          pom = a[i];
57        a[i] = a[min];
          a[min] = pom;
59      }
    }
61  }

63  int main()
    {
65    Artikal asortiman[MAX_ARTIKALA];
      long kod;
67    int i, n;
      float racun;
69
      FILE *fp = NULL;
71
      /* Otvaranje datoteke */
73    if ((fp = fopen("artikli.txt", "r")) == NULL) {
        fprintf(stderr,
75              "Neuspesno otvaranje datoteke artikli.txt.\n");
        exit(EXIT_FAILURE);
77    }

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

89    n = i;

91    /* Sortiracemo celokupan asortiman prodavnice prema kodovima
         jer ce pri kucanju racuna prodavac unositi kod artikla.
93       Prilikom kucanja svakog racuna pretrazuje se asortiman, da
         bi se utvrdila cena artikla. Kucanje racuna obuhvata vise
95       pretraga asortimana i u interesu nam je da ta operacija
         bude sto efikasnija. Zelimo da koristimo algoritam binarne
97       pretrage priliko pretrazivanje po kodu artikla. Iz tog
         razloga, potrebno je da nam asortiman bude sortiran po
```

```
99      kodovima i to cemo uraditi primenom selection sort
        algoritma. Sortiramo samo jednom na pocetku, ali zato posle
101     brzo mozemo da pretrazujemo prilikom kucanja proizvoljno
        puno racuna. Vreme koje se utrosi na sortiranje na pocetku
103     izvrsavanja programa, kasnije se isplati jer za brojna
        trazenja artikla mozemo umesto linearne da koristimo
105     efikasniju binarnu pretragu. */
    selection_sort(asortiman, n);
107
    /* Ispis stanja u prodavnici */
109 printf
        ("Asortiman:\nKOD                 Naziv artikla     Ime
      proizvodjaca       Cena\n");
111 for (i = 0; i < n; i++)
      printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113          asortiman[i].naziv, asortiman[i].proizvodjac,
             asortiman[i].cena);
115
    kod = 0;
117 while (1) {
      printf("----------------------------\n");
119   printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
      printf("- Za nov racun unesite kod artikla!\n\n");
121   /* Unos bar koda provog artikla sledeceg kupca */
      if (scanf("%ld", &kod) == EOF)
123     break;
      /* Trenutno racun novog kupca */
125   racun = 0;
      /* Za sve artikle trenutnog kupca */
127   while (1) {
        /* Nalazimo ih u nizu */
129     if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
          printf
131           ("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
        } else {
133       printf("\tTrazili ste:\t%s %s %12.2f\n",
                 asortiman[i].naziv, asortiman[i].proizvodjac,
135              asortiman[i].cena);
          /* I dodajemo na ukupan racun */
137       racun += asortiman[i].cena;
        }
139     /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0
           ako on nema vise artikla */
141     printf("Unesite kod artikla [ili 0 za prekid]: \t");
        scanf("%ld", &kod);
143     if (kod == 0)
          break;
145   }
      /* Stampanje ukupnog racuna trenutnog kupca */
147   printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
    }
149
```

```
  printf("Kraj rada kase!\n");

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.24

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

#define MAX 500

/* Struktura koja nam je neophodna za sve informacije o
   pojedinacnom studentu */
typedef struct {
  char ime[20];
  char prezime[25];
  int prisustvo;
  int zadaci;
} Student;

/* Funkcija kojom sortiramo niz 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 kojom sortiramo niz struktura po ukupnom broju
   uradjenih zadataka opadajuce, 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;
46  Student pom;
  for (i = 0; i < n - 1; i++) {
48    max = i;
    for (j = i + 1; j < n; j++)
50      if (niz[j].zadaci > niz[max].zadaci)
          max = j;
52      else if (niz[j].zadaci == niz[max].zadaci
                && strlen(niz[j].ime) < strlen(niz[max].ime))
54        max = j;
    if (max != i) {
56      pom = niz[max];
      niz[max] = niz[i];
58      niz[i] = pom;
    }
60  }
}
62
 /* Funkcija kojom sortiramo niz struktura po broju casova na
64    kojima su bili opadajuce, a ukoliko * neki studenti imaju
    isti broj casova, sortiraju se opadajuce po broju uradjenih
66    zadataka, * a ukoliko se i po broju zadataka poklapaju
    sortirati ih po prezimenu opadajuce. */
68 void sort_prisustvo_pa_zadatke_pa_prezimena(Student niz[], int n)
{
70  int i, j;
  int max;
72  Student pom;
  for (i = 0; i < n - 1; i++) {
74    max = i;
    for (j = i + 1; j < n; j++)
76      if (niz[j].prisustvo > niz[max].prisustvo)
          max = j;
78      else if (niz[j].prisustvo == niz[max].prisustvo
                && niz[j].zadaci > niz[max].zadaci)
80        max = j;
      else if (niz[j].prisustvo == niz[max].prisustvo
82              && niz[j].zadaci == niz[max].zadaci
                && strcmp(niz[j].prezime, niz[max].prezime) > 0)
84        max = j;
    if (max != i) {
86      pom = niz[max];
      niz[max] = niz[i];
88      niz[i] = pom;
    }
90  }
}
92


94
int main(int argc, char *argv[])
96 {
```

```
  Student praktikum[MAX];
98  int i, br_studenata = 0;

100 FILE *fp = NULL;

102 /* Otvaranje datoteke za citanje */
    if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
104   fprintf(stderr,
            "Neupesno otvaranje datoteke aktivnost.txt.\n");
106   exit(EXIT_FAILURE);
    }
108
    /* Ucitavanje sadrzaja */
110 for (i = 0;
         fscanf(fp, "%s%s%d%d", praktikum[i].ime,
112             praktikum[i].prezime, &praktikum[i].prisustvo,
                &praktikum[i].zadaci) != EOF; i++);
114 /* Zatvaranje datoteke */
    fclose(fp);
116 br_studenata = i;

118 /* Kreiramo prvi spisak studenata na kom su sortirani
       leksikografski po imenu rastuce */
120 sort_ime_leksikografski(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
122 if ((fp = fopen("dat1.txt", "w")) == NULL) {
      fprintf(stderr, "Neupesno otvaranje datoteke dat1.txt.\n");
124   exit(EXIT_FAILURE);
    }
126 /* Upis niza u datoteku */
    fprintf
128     (fp,
         "Studenti sortirani po imenu leksikografski rastuce:\n");
130 for (i = 0; i < br_studenata; i++)
      fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
132           praktikum[i].prezime, praktikum[i].prisustvo,
              praktikum[i].zadaci);
134 /* Zatvaranje datoteke */
    fclose(fp);
136
    /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
138    opadajuce, ukoliko neki studenti imaku isti broj uradjenih
       zadataka sortiraju se po duzini imena rastuce. */
140 sort_zadatke_pa_imena(praktikum, br_studenata);
    /* Otvaranje datoteke za pisanje */
142 if ((fp = fopen("dat2.txt", "w")) == NULL) {
      fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
144   exit(EXIT_FAILURE);
    }
146 /* Upis niza u datoteku */
    fprintf(fp,
148         "Studenti sortirani po broju zadataka opadajuce,\n");
```

```
  fprintf(fp, "pa po duzini imena rastuce:\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);

  /* Na trecem spisku su sortirani po broju casova na kojima su
     bili opadajuce, a ukoliko neki studenti imaju isti broj
     casova, sortiraju se opadajuce po broju uradjenih zadataka,
     a ukoliko se i po broju zadataka poklapaju sortirati ih po
     prezimenu opadajuce. */
  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)
```

```
16 {
     Pesma *p1 = (Pesma *) pp1;
18   Pesma *p2 = (Pesma *) pp2;

20   return p2->broj_gledanja - p1->broj_gledanja;
   }
22
   /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad
24    qsort funkcije) */
   int uporedi_naslove(const void *pp1, const void *pp2)
26 {
     Pesma *p1 = (Pesma *) pp1;
28   Pesma *p2 = (Pesma *) pp2;

30   return strcmp(p1->naslov, p2->naslov);
   }
32
   /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za
34    rad qsort funkcije) */
   int uporedi_izvodjace(const void *pp1, const void *pp2)
36 {
     Pesma *p1 = (Pesma *) pp1;
38   Pesma *p2 = (Pesma *) pp2;

40   return strcmp(p1->izvodjac, p2->izvodjac);
   }
42

44 int main(int argc, char *argv[])
   {
46   FILE *ulaz;
     Pesma *pesme;                  /* Pokazivac na deo memorije za
48                                     cuvanje pesama */
     int alocirano_za_pesme;        /* Broj mesta alociranih za
50                                     pesme */
     int i;                         /* Redni broj pesme cije se
52                                     informacije citaju */
     int n;                         /* Ukupan broj pesama */
54   int j, k;
     char c;
56   int alocirano;                 /* Broj mesta alociranih za
                                       propratne informacije o
58                                     pesmama */
     int broj_gledanja;
60
     /* Pripremamo datoteku za citanje */
62   ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
     if (ulaz == NULL) {
64     printf("Greska pri otvaranju ulazne datoteke!\n");
       return 0;
66   }
```

```
68  /* Citamo informacije o pesmama */
    pesme = NULL;
70  alocirano_za_pesme = 0;
    i = 0;
72
    while (1) {
74
      /* Proveravamo da li smo stigli do kraja datoteke */
76    c = fgetc(ulaz);
      if (c == EOF) {
78      /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
           sadrzaja za citanje */
80      break;
      } else {
82      /* Ako nismo, vracamo procitani karakter nazad */
        ungetc(c, ulaz);
84    }


86
      /* Proveravamo da li imamo dovoljno memorije za citanje nove
88       pesme */
      if (alocirano_za_pesme == i) {
90
        /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
92         alociramo novih KORAK mesta */
        alocirano_za_pesme += KORAK;
94      pesme =
            (Pesma *) realloc(pesme,
96                            alocirano_za_pesme * sizeof(Pesma));

98      /* Proveravamo da li je nova memorija uspesno realocirana */
        if (pesme == NULL) {
100       /* Ako nije ... */
          /* Ispisujemo obavestenje */
102       printf("Problem sa alokacijom memorije!\n");

104       /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
          for (k = 0; k < i; k++) {
106         free(pesme[k].izvodjac);
            free(pesme[k].naslov);
108       }
          free(pesme);
110       return 0;
        }
112
      }
114
      /* Ako jeste, nastavljamo sa citanjem pesama ... */
116   /* Citamo ime izvodjaca */

118   j = 0;                        /* Oznacava poziciju na koju
                                      treba smestiti procitani
```

```
120                                   karakter */
    alocirano = 0;                  /* Oznacava broj alociranih
122                                   mesta */
    pesme[i].izvodjac = NULL;       /* Memorija koju mozemo
124                                   koristiti za smestanje
                                      procitanih karaktera */
126
    /* Sve dok ne stignemo do prve beline u liniji - beline koja
128      se nalazi nakon imena izvodjaca - citamo karaktere iz
         datoteke */
130 while ((c = fgetc(ulaz)) != ' ') {

132   /* Proveravamo da li imamo dovoljno memorije za smestanje
         procitanog karaktera */
134   if (j == alocirano) {

136     /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
           alociramo novih KORAK mesta */
138     alocirano += KORAK;
        pesme[i].izvodjac =
140         (char *) realloc(pesme[i].izvodjac,
                             alocirano * sizeof(char));
142
        /* Proveravamo da li je nova alokacija uspesna */
144     if (pesme[i].izvodjac == NULL) {
          /* Ako nije... */
146       /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
          for (k = 0; k < i; k++) {
148         free(pesme[k].izvodjac);
            free(pesme[k].naslov);
150       }
          free(pesme);
152       /* I prekidamo sa izvrsavanjem programa */
          return 0;
154     }

156   }
      /* Ako imamo dovoljno memorije, smestamo procitani
158      karakter */
      pesme[i].izvodjac[j] = c;
160   j++;
      /* I nastavljamo sa citanjem */
162 }

164 /* Upisujemo terminirajucu nulu na kraju reci */
    pesme[i].izvodjac[j] = '\0';
166

168 /* Citamo - */
    fgetc(ulaz);
170
    /* Citamo razmak */
```

```
172     fgetc(ulaz);
174
        /* Citamo naslov pesme */
176     j = 0;                          /* Oznacava poziciju na koju
                                           treba smestiti procitani
178                                        karakter */
        alocirano = 0;                  /* Oznacava broj alociranih
180                                        mesta */
        pesme[i].naslov = NULL;         /* Memorija koju mozemo
182                                        koristiti za smestanje
                                           procitanih karaktera */
184
        /* Sve dok ne stignemo do zareza - zareza koji se nalazi
186        nakon naslova pesme - citamo karaktere iz datoteke */
188     while ((c = fgetc(ulaz)) != ',') {
          /* Proveravamo da li imamo dovoljno memorije za smestanje
190           procitanog karaktera */
          if (j == alocirano) {
192         /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
                alociramo novih KORAK mesta */
194         alocirano += KORAK;
            pesme[i].naslov =
196             (char *) realloc(pesme[i].naslov,
                                 alocirano * sizeof(char));
198
            /* Proveravamo da li je nova alokacija uspesna */
200         if (pesme[i].naslov == NULL) {
              /* Ako nije... */
202           /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
              for (k = 0; k < i; k++) {
204             free(pesme[k].izvodjac);
                free(pesme[k].naslov);
206           }
              free(pesme[i].izvodjac);
208           free(pesme);

210           /* I prekidamo izvrsavanje programa */
              return 0;
212         }

214       }
          /* Ako imamo dovoljno memorije, smestamo procitani
216           karakter */
          pesme[i].naslov[j] = c;
218       j++;
          /* I nastavljamo dalje sa citanjem */
220     }
        /* Upisujemo terminirajucu nulu na kraju reci */
222     pesme[i].naslov[j] = '\0';
```

```
224     /* Citamo razmak */
        fgetc(ulaz);
226
228     /* Citamo broj gledanja */

230     broj_gledanja = 0;

232     /* Sve dok ne stignemo do znaka za novi red - kraja linije -
           citamo karaktere iz datoteke */
234     while ((c = fgetc(ulaz)) != '\n') {
          broj_gledanja = broj_gledanja * 10 + (c - '0');
236     }
        pesme[i].broj_gledanja = broj_gledanja;
238
        /* Prelazimo na citanje sledece pesme */
240     i++;

242   }

244   /* Cuvamo informaciju o broju pesama koje smo procitali */
      n = i;
246
      /* Zatvaramo datoteku jer nam nece vise trebati */
248   fclose(ulaz);

250   /* Analiziramo argumente komandne linije */
      if (argc == 1) {
252
        /* Nema dodatnih opcija - sortiramo po broju gledanja */
254     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
      } else {
256
        if (argc == 2 && strcmp(argv[1], "-n") == 0) {
258       /* Sortiramo po naslovu */
          qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
260     } else {
          if (argc == 2 && strcmp(argv[1], "-i") == 0) {
262         /* Sortiramo po izvodjacu */
            qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
264       } else {
            printf("Nedozvoljeni argumenti!\n");
266         free(pesme);
            return 0;
268       }
        }
270   }

272   /* Ispisujemo rezultat */
      for (i = 0; i < n; i++) {
274     printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
                pesme[i].broj_gledanja);
```

```
276   }

278   /* Oslobadjamo memoriju */
      for (i = 0; i < n; i++) {
280     free(pesme[i].izvodjac);
        free(pesme[i].naslov);
282   }
      free(pesme);
284
      /* Prekidamo izvrsavanje programa */
286   return 0;
}
```

Rešenje 3.28

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
   /* Funkcija poredi dva cela broja */
9  int compare_int(const void *a, const void *b)
   {
11   /* Konvertujemo void pokazivace u int pokazivace koje zatim
        dereferenciramo, dobijamo int-ove koje oduzimamo i razliku
13      vracamo. */

15   /* Zbog moguceg prekoracenja opsega celih brojeva necemo ih
        oduzimati 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)
       /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
25     return -1;
     else
27     return 0;
   }
29
   int compare_int_desc(const void *a, const void *b)
31 {
     /* Za obrnuti poredak mozemo samo oduzimati a od b */
33   /* return *((int *)b) - *((int *)a); */

35   /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
        funkcija */
```

```
37    return -compare_int(a, b);
}
39
/* Test program */
41  int main()
{
43    size_t n;
    int i, x;
45    int a[MAX], *p = NULL;

47    /* Unosimo dimenziju */
    printf("Uneti dimenziju niza: ");
49    scanf("%ld", &n);
    if (n > MAX)
51      n = MAX;

53    /* Unosimo elemente niza */
    printf("Uneti elemente niza:\n");
55    for (i = 0; i < n; i++)
      scanf("%d", &a[i]);
57
    /* Sortiramo niz celih brojeva */
59    qsort(a, n, sizeof(int), &compare_int);

61    /* Prikazujemo sortirani niz */
    printf("Sortirani niz u rastucem poretku:\n");
63    for (i = 0; i < n; i++)
      printf("%d ", a[i]);
65    putchar('\n');

67    /* Pretrazivanje niza */
    /* Vrednost koju cemo traziti u nizu */
69    printf("Uneti element koji se trazi u nizu: ");
    scanf("%d", &x);
71
    /* Binarna pretraga */
73    printf("Binarna pretraga: \n");
    p = bsearch(&x, a, n, sizeof(int), &compare_int);
75    if (p == NULL)
      printf("Elementa nema u nizu!\n");
77    else
      printf("Element je nadjen na poziciji %ld\n", p - a);
79
    /* Linearna pretraga */
81    printf("Linearna pretraga (lfind): \n");
    p = lfind(&x, a, &n, sizeof(int), &compare_int);
83    if (p == NULL)
      printf("Elementa nema u nizu!\n");
85    else
      printf("Element je nadjen na poziciji %ld\n", p - a);
87
    return 0;
```

```
89 }
```

Rešenje 3.29

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

6 #define MAX 100

8 /* Funkcija racuna broj delilaca broja x */
int no_of_deviders(int x)
10 {
  int i;
12   int br;

14   /* Ako je argument negativan broj menjamo mu znak */
  if (x < 0)
16     x = -x;
  if (x == 0)
18     return 0;
  if (x == 1)
20     return 1;
  /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22   br = 2;
  /* Sve dok je */
24   for (i = 2; i < sqrt(x); i++)
    if (x % i == 0)
26       /* Ako i deli x onda su delioci: i, x/i */
       br += 2;
28   /* Ako je broj bas kvadrat, onda smo iz petlje izasli kada je
     i bilo bas jednako korenu od x, tada x ima jos jednog
30     delioca */
  if (i * i == x)
32     br++;

34   return br;
}
36
/* Funkcija poredjenja dva cela broja po broju delilaca */
38 int compare_no_deviders(const void *a, const void *b)
{
40   int ak = *(int *) a;
  int bk = *(int *) b;
42   int n_d_a = no_of_deviders(ak);
  int n_d_b = no_of_deviders(bk);
44
  if (n_d_a > n_d_b)
46     return 1;
  else if (n_d_a < n_d_b)
```

```
48      return -1;
    else
50      return 0;
}
52
/* Test program */
54 int main()
{
56   size_t n;
     int i;
58   int a[MAX];

60   /* Unosimo dimenziju */
     printf("Uneti dimenziju niza: ");
62   scanf("%ld", &n);
     if (n > MAX)
64     n = MAX;

66   /* Unosimo elemente niza */
     printf("Uneti elemente niza:\n");
68   for (i = 0; i < n; i++)
       scanf("%d", &a[i]);
70
     /* Sortiramo niz celih brojeva prema broju delilaca */
72   qsort(a, n, sizeof(int), &compare_no_deviders);

74   /* Prikazujemo sortirani niz */
     printf
76       ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
     for (i = 0; i < n; i++)
78     printf("%d ", a[i]);
     putchar('\n');
80
     return 0;
82 }
```

Rešenje 3.30

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include <search.h>
#define MAX_NISKI 1000
6 #define MAX_DUZINA 30

8 /*************************************************************
  Niz nizova karaktera ovog potpisa
10 char niske[3][4];
  se moze graficki predstaviti ovako:
12 ------------------------------------------------------
  | a | b | c |\0 || d | e | \0|   || f | g | h | \0||
```

```
14   ------------------------------------------------------
     Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu.
16   Za svaku je rezervisano po 4 karaktera ukljucujuci \0.
     Druga rec sa nalazi na adresi koja je za 4 veca od prve reci,
18   a za 4 manja od adrese na kojoj se nalazi treca rec.
     Adresa i-te reci je niske[i] i ona je tipa char*.
20
     Kako pokazivaci a i b u sledecoj funkciji sadrze adrese
22   elemenata koji trebaju biti uporedjeni, (npr. pri porecenju
     prve i poslednje reci, pokazivac a ce pokazivati na slovo 'a',
24   a pokazivac b na slovo 'f') kastujemo ih na char*, i pozivamo
     funkciju strcmp nad njima.
26 ************************************************************/
   int poredi_leksikografski(const void *a, const void *b)
28 {
     return strcmp((char *) a, (char *) b);
30 }

32 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
      leksikografski, vec po duzini */
34 int poredi_duzine(const void *a, const void *b)
   {
36   return strlen((char *) a) - strlen((char *) b);
   }
38
   int main()
40 {
     int i;
42   size_t n;
     FILE *fp = NULL;
44   char niske[MAX_NISKI][MAX_DUZINA];
     char *p = NULL;
46   char x[MAX_DUZINA];

48   /* Otvaranje datoteke */
     if ((fp = fopen("niske.txt", "r")) == NULL) {
50     fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
       exit(EXIT_FAILURE);
52   }

54   /* Citanje sadrzaja datoteke */
     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
56
     /* Zatvaranje datoteke */
58   fclose(fp);
     n = i;
60
     /* Sortiramo niske leksikografski, tako sto biblioteckoj
62      funkciji qsort prosledjujemo funkciju kojom se zadaje
        kriterijum poredjenja 2 niske po duzini */
64   qsort(niske, n, MAX_DUZINA * sizeof(char),
          &poredi_leksikografski);
```

```
66
   printf("Leksikografski sortirane niske:\n");
68 for (i = 0; i < n; i++)
     printf("%s ", niske[i]);
70 printf("\n");

72 /* Unosimo trazeni nisku */
   printf("Uneti trazenu nisku: ");
74 scanf("%s", x);

76 /* Binarna pretraga */
   /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
78    jer nam je niz sortiran leksikografski. */
   p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
80             &poredi_leksikografski);

82 if (p != NULL)
     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84          p, (p - (char *) niske) / MAX_DUZINA);
   else
86   printf("Niska nije pronadjena u nizu\n");

88 /* Linearna pretraga */
   /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90    jer nam je niz sortiran leksikografski. */
   p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
92           &poredi_leksikografski);

94 if (p != NULL)
     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
96          p, (p - (char *) niske) / MAX_DUZINA);
   else
98   printf("Niska nije pronadjena u nizu\n");

100 /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
       funkciju poredjenja */
102 qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

104 printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
106   printf("%s ", niske[i]);
    printf("\n");
108
    exit(EXIT_SUCCESS);
110 }
```

Rešenje 3.31

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

```c
#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 ih tako moramo i kastovati. Da bi uporedili
  leksikografski elemente X i Z, moramo uporediti stringove
  na koje oni pokazuju, pa zato u sledecoj funkciji pozivamo
  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 kastujemo i ne
   dereferenciramo. */
int poredi_leksikografski_b(const void *a, const void *b)
{
  return strcmp((char *) a, *(char **) b);
}
```

```
56 int main()
   {
58   int i;
     size_t n;
60   FILE *fp = NULL;
     char *niske[MAX_NISKI];
62   char **p = NULL;
     char x[MAX_DUZINA];
64
     /* Otvaranje datoteke */
66   if ((fp = fopen("niske.txt", "r")) == NULL) {
       fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
68     exit(EXIT_FAILURE);
     }
70
     /* Citanje sadrzaja datoteke */
72   i = 0;
     while (fscanf(fp, "%s", x) != EOF) {
74     /* Alociranje dovoljne memorije za i-tu nisku */
       if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
76       fprintf(stderr, "Greska pri alociranju niske\n");
         exit(EXIT_FAILURE);
78     }
       /* Kopiranje procitane niske na svoje mesto */
80     strcpy(niske[i], x);
       i++;
82   }

84   /* Zatvaranje datoteke */
     fclose(fp);
86   n = i;

88   /* Sortiramo niske leksikografski, tako sto biblioteckoj
        funkciji qsort prosledjujemo funkciju kojom se zadaje
90      kriterijum poredjenja 2 niske po duzini */
     qsort(niske, n, sizeof(char *), &poredi_leksikografski);
92
     printf("Leksikografski sortirane niske:\n");
94   for (i = 0; i < n; i++)
       printf("%s ", niske[i]);
96   printf("\n");

98   /* Unosimo trazeni nisku */
     printf("Uneti trazenu nisku: ");
100  scanf("%s", x);

102  /* Binarna pretraga */
     /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
104     jer nam je niz sortiran leksikografski. */
     p = bsearch(x, niske, n, sizeof(char *),
106              &poredi_leksikografski_b);
```

```
  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
           *p, p - niske);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Linearna pretraga */
  /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
     jer nam je niz sortiran leksikografski. */
  p = lfind(x, niske, &n, sizeof(char *),
            &poredi_leksikografski_b);

  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
           *p, p - niske);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
     funkciju poredjenja */
  qsort(niske, n, sizeof(char *), &poredi_duzine);

  printf("Niske sortirane po duzini:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  /* Oslobadjanje zauzete memorije */
  for (i = 0; i < n; i++)
    free(niske[i]);

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.32

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

#define MAX 500

/* Struktura koja nam je neophodna za sve informacije o
   pojedinacnom studentu */
typedef struct {
  char ime[21];
  char prezime[21];
  int bodovi;
} Student;
```

```
/* Funkcija poredjenja koju cemo koristiti za sortiranje po
   broju bodova, a studente sa istim brojevem bodova dodatno
   sortiramo leksikografski po prezimenu */
int compare(const void *a, const void *b)
{
  Student *prvi = (Student *) a;
  Student *drugi = (Student *) b;

  if (prvi->bodovi > drugi->bodovi)
    return -1;
  else if (prvi->bodovi < drugi->bodovi)
    return 1;
  else
    /* Jednaki su po broju bodova, treba ih uporediti po
       prezimenu */
    return strcmp(prvi->prezime, drugi->prezime);
}

/* Funkcija za poredjenje koje ce porediti samo po broju bodova
   Prvi parametar je ono sto trazimo u nizu, ovde je to broj
   bodova, a drugi parametar ce biti element niza ciji se bodovi
   porede. */
int compare_za_bsearch(const void *a, const void *b)
{
  int bodovi = *(int *) a;
  Student *s = (Student *) b;
  return s->bodovi - bodovi;
}

/* Funkcija za poredjenje koje ce porediti samo po prezimenu
   Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
   a drugi parametar ce biti element niza cije se prezime
   poredi. */
int compare_za_linearna_prezimena(const void *a, const void *b)
{
  char *prezime = (char *) a;
  Student *s = (Student *) b;
  return strcmp(prezime, s->prezime);
}


int main(int argc, char *argv[])
{
  Student kolokvijum[MAX];
  int i;
  size_t br_studenata = 0;
  Student *nadjen = NULL;
  FILE *fp = NULL;
  int bodovi;
  char prezime[21];

  /* Ako je program pozvan sa nedovoljnim brojem argumenata
```

```
68      informisemo korisnika kako se program koristi i prekidamo
        izvrsavanje. */
70    if (argc < 2) {
        fprintf(stderr,
72              "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
                argv[0]);
74      exit(EXIT_FAILURE);
      }
76
      /* Otvaranje datoteke */
78    if ((fp = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Neupesno otvaranje datoteke %s\n", argv[1]);
80      exit(EXIT_FAILURE);
      }
82
      /* Ucitavanje sadrzaja */
84    for (i = 0;
           fscanf(fp, "%s%s%d", kolokvijum[i].ime,
86                 kolokvijum[i].prezime,
                   &kolokvijum[i].bodovi) != EOF; i++);
88
      /* Zatvaranje datoteke */
90    fclose(fp);
      br_studenata = i;
92
      /* Sortiramo niz studenata po broju bodova, pa unutar grupe
94       studenata sa istim brojem bodova sortiranje se vrsi po
         prezimenu */
96    qsort(kolokvijum, br_studenata, sizeof(Student), &compare);

98    printf("Studenti sortirani po broju poena opadajuce, ");
      printf("pa po prezimenu rastuce:\n");
100   for (i = 0; i < br_studenata; i++)
        printf("%s %s  %d\n", kolokvijum[i].ime,
102            kolokvijum[i].prezime, kolokvijum[i].bodovi);

104   /* Pretrazivanje studenata po broju bodova se vrsi binarnom
         pretragom jer je niz sortiran po broju bodova. */
106   printf("Unesite broj bodova: ");
      scanf("%d", &bodovi);
108
      nadjen =
110       bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
                  &compare_za_bsearch);
112
      if (nadjen != NULL)
114     printf
            ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
116          nadjen->ime, nadjen->prezime, nadjen->bodovi);
      else
118     printf("Nema studenta sa unetim brojem bodova\n");
```

```
  /* Pretraga po prezimenu se mora vrsiti linearnom pretragom
     jer nam je niz sortiran po bodovima, globalno gledano. */
  printf("Unesite prezime: ");
  scanf("%s", prezime);

  nadjen =
      lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
            &compare_za_linearna_prezimena);

  if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
  else
    printf("Nema studenta sa unetim prezimenom\n");

  return 0;
}
```

Rešenje 3.33

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

#define MAX 128

/* Funkcija poredi dva karaktera */
int uporedi_char(const void *pa, const void *pb)
{
  return *(char *) pa - *(char *) pb;
}

/* Funkcija vraca: 1 - ako jesu anagrami 0 - inace */
int anagrami(char s[], char t[], int n_s, int n_t)
{
  /* Ako dve niske imaju razlicitu duzinu => nisu anagrami */
  if (n_s != n_t)
    return 0;

  /* Sortiramo niske */
  qsort(s, strlen(t) / sizeof(char), sizeof(char),
        &uporedi_char);
  qsort(t, strlen(t) / sizeof(char), sizeof(char),
        &uporedi_char);

  /* Ako su niske nakon sortiranja iste => jesu anagrami, u
     suprotnom, nisu */
  return !strcmp(s, t);
}
```

```
31 int main()
   {
33   char s[MAX], t[MAX];

35   /* Unose se dve niske sa ulaza */
     printf("Unesite prvu nisku: ");
37   scanf("%s", s);

39   printf("Unesite drugu nisku: ");
     scanf("%s", t);
41
     /* Ispituje se da li su niske anagrami */
43   if (anagrami(s, t, strlen(s), strlen(t)))
       printf("jesu\n");
45   else
       printf("nisu\n");
47
     return 0;
49 }
```

Rešenje 3.34

```
   #include <stdio.h>
2  #include <string.h>
   #include <stdlib.h>
4
   #define MAX 10
6  #define MAX_DUZINA 32

8  /* Funkcija poredi dve niske (stringove) */
   int uporedi_niske(const void *pa, const void *pb)
10 {
     return strcmp((char *) pa, (char *) pb);
12 }

14 int main()
   {
16   int i, n;
     char S[MAX][MAX_DUZINA];
18
     /* Unos broja niski */
20   printf("Unesite broj niski:");
     scanf("%d", &n);
22
     /* Unos niza niski */
24   printf("Unesite niske:\n");
     for (i = 0; i < n; i++)
26     scanf("%s", S[i]);

28   /* Sortiramo niz niski */
     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
```

```
30
   /*****************************************************
32   Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
     sortiranih niski. Koriscen je samo u fazi testiranja programa.
34
     printf("Sortirane niske su:\n");
36   for(i = 0; i < n; i++)
       printf("%s ", S[i]);
38 *****************************************************/

40 /* Ako postoje dve iste niske u nizu, onda ce one nakon
      sortiranja niza biti jedna do druge */
42 for (i = 0; i < n - 1; i++)
     if (strcmp(S[i], S[i + 1]) == 0) {
44     printf("ima\n");
       return 0;
46   }

48 printf("nema\n");
   return 0;
50 }
```

Rešenje 3.35

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

5  /* Struktura koja predstavlja jednog studenta */
   typedef struct student {
7    char nalog[8];
     char ime[21];
9    char prezime[21];
     int poeni;
11 } Student;


13
   /* Funkcija poredi studente prema broju poena, rastuce */
15 int uporedi_poeni(const void *a, const void *b)
   {
17
     Student s = *(Student *) a;
19   Student t = *(Student *) b;
     return s.poeni - t.poeni;
21 }

23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru
      i na kraju prema indeksu */
25 int uporedi_nalog(const void *a, const void *b)
   {
27   Student s = *(Student *) a;
```

```
  Student t = *(Student *) b;
  /* Za svakog studenta iz naloga izdvajamo godinu upisa, smer i
     broj indeksa */
  int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
  int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
  char smer1 = s.nalog[1];
  char smer2 = t.nalog[1];
  int indeks1 =
      (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
      s.nalog[6] - '0';
  int indeks2 =
      (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
      t.nalog[6] - '0';
  if (godina1 != godina2)
    return godina1 - godina2;
  else if (smer1 != smer2)
    return smer1 - smer2;
  else
    return indeks1 - indeks2;
}

int uporedi_bsearch(const void *a, const void *b)
{
  /* Nalog studenta koji se trazi */
  char *nalog = (char *) a;
  /* Kljuc pretrage */
  Student s = *(Student *) b;

  int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
  int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
  char smer1 = nalog[1];
  char smer2 = s.nalog[1];
  int indeks1 =
      (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
      '0';
  int indeks2 =
      (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
      s.nalog[6] - '0';
  if (godina1 != godina2)
    return godina1 - godina2;
  else if (smer1 != smer2)
    return smer1 - smer2;
  else
    return indeks1 - indeks2;
}

int main(int argc, char **argv)
{
  Student *nadjen = NULL;
  char nalog_trazeni[8];
  Student niz_studenata[100];
  int i = 0, br_studenata = 0;
```

```
  FILE *in = NULL, *out = NULL;
81
  /* Ako je broj argumenata komandne linije razlicit i od 2 i od
83    3, korisnik nije ispravno pozvao program i prijavljujemo
      gresku: */
85 if (argc != 2 && argc != 3) {
    fprintf(stderr,
87            "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\n
    ");
    exit(EXIT_FAILURE);
89 }

91 /* Otvaranje datoteke za citanje */
   in = fopen("studenti.txt", "r");
93 if (in == NULL) {
    fprintf(stderr,
95            "Greska prilikom otvarnja datoteke studenti.txt!\n");
    exit(EXIT_FAILURE);
97 }

99 /* Otvaranje datoteke za pisanje */
   out = fopen("izlaz.txt", "w");
101 if (out == NULL) {
    fprintf(stderr,
103           "Greska prilikom otvaranja datoteke izlaz.txt!\n");
    exit(EXIT_FAILURE);
105 }

107 /* Ucitavamo studente iz ulazne datoteke sve do njenog kraja */
   while (fscanf
109        (in, "%s %s %s %d", niz_studenata[i].nalog,
         niz_studenata[i].ime, niz_studenata[i].prezime,
111      &niz_studenata[i].poeni) != EOF)
    i++;
113
   br_studenata = i;
115
   /* Ako je student uneo opciju -p, vrsi se sortiranje po
117    poenima */
   if (strcmp(argv[1], "-p") == 0)
119   qsort(niz_studenata, br_studenata, sizeof(Student),
        &uporedi_poeni);
121 /* A ako je uneo opciju -n, vrsi se sortiranje po nalogu */
   else if (strcmp(argv[1], "-n") == 0)
123   qsort(niz_studenata, br_studenata, sizeof(Student),
        &uporedi_nalog);
125
   /* Sortirani studenti se ispisuju u izlaznu datoteku */
127 for (i = 0; i < br_studenata; i++)
    fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
129           niz_studenata[i].ime, niz_studenata[i].prezime,
           niz_studenata[i].poeni);
```

```
131
   /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
133     studenta... */
   if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
135   strcpy(nalog_trazeni, argv[2]);

137   /* ... pronalazi se student sa tim nalogom... */
      nadjen =
139       (Student *) bsearch(nalog_trazeni, niz_studenata,
                              br_studenata, sizeof(Student),
141                           &uporedi_bsearch);

143   if (nadjen == NULL)
        printf("Nije nadjen!\n");
145   else
        printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
147            nadjen->prezime, nadjen->poeni);
   }
149
   /* Zatvaranje datoteka */
151 fclose(in);
   fclose(out);
153
   return 0;
155 }
```

Rešenje 3.37

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

4 /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
     standardnog ulaza */
6 void ucitaj_matricu(int **a, int n, int m)
  {
8   printf("Unesite elemente matrice po vrstama:\n");
    int i, j;
10
    for (i = 0; i < n; i++) {
12    for (j = 0; j < m; j++) {
        scanf("%d", &a[i][j]);
14    }
    }
16 }

18 /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
     kolona */
20 int zbir_vrste(int **a, int v, int m)
  {
22   int i, zbir = 0;
```

```
24    for (i = 0; i < m; i++) {
        zbir += a[v][i];
26    }
    return zbir;
28  }

30  /* Funkcija koja sortira vrste matrice na osnovu zbira
       koriscenjem selection algoritma */
32  void sortiraj_vrste(int **a, int n, int m)
    {
34    int i, j, min;

36    for (i = 0; i < n - 1; i++) {
        min = i;
38      for (j = i + 1; j < n; j++) {
          if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
40          min = j;
          }
42      }
        if (min != i) {
44        int *tmp;
          tmp = a[i];
46        a[i] = a[min];
          a[min] = tmp;
48      }
    }
50  }

52  /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
       standardni izlaz */
54  void ispisi_matricu(int **a, int n, int m)
    {
56    int i, j;

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


66
    /* Funkcija koja alocira memoriju za matricu dimenzija nxm */
68  int **alociraj_memoriju(int n, int m)
    {
70    int i, j;
      int **a;
72
      a = (int **) malloc(n * sizeof(int *));
74    if (a == NULL) {
        fprintf(stderr, "Problem sa alokacijom memorije!\n");
```

```
76      exit(EXIT_FAILURE);
    }
78    /* Za svaku vrstu ponaosob */
    for (i = 0; i < n; i++) {
80
      /* Alociramo memoriju */
82      a[i] = (int *) malloc(m * sizeof(int));

84      /* Proveravamo da li je doslo do greske prilikom alokacije */
      if (a[i] == NULL) {
86        /* Ako jeste, ispisujemo poruku */
        fprintf(stderr, "Problem sa alokacijom memorije!\n");
88
        /* I oslobadjamo memoriju zauzetu do ovog koraka */
90        for (j = 0; j < i; j++) {
          free(a[i]);
92        }
        free(a);
94        exit(EXIT_FAILURE);
      }
96    }

98    return a;
}
100
/* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
102     nxm */
void oslobodi_memoriju(int **a, int n, int m)
104 {
    int i;
106
    for (i = 0; i < n; i++) {
108       free(a[i]);
    }
110   free(a);
}
112



114
int main(int argc, char *argv[])
116 {
    int **a;
118   int n, m;


120
    /* Citamo dimenzije matrice */
122   printf("Unesite dimenzije matrice: ");
    scanf("%d %d", &n, &m);
124
    /* Alociramo memoriju */
126   a = alociraj_memoriju(n, m);
```

```
128   /* Ucitavamo elemente matrice */
      ucitaj_matricu(a, n, m);
130
      /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */
132   sortiraj_vrste(a, n, m);

134   /* Ispisujemo rezultujucu matricu */
      printf("Sortirana matrica je:\n")
136       ispisi_matricu(a, n, m);

138   /* Oslobadjamo memoriju */
      oslobodi_memoriju(a, n, m);
140
      /* I prekidamo sa izvrsavanjem programa */
142   return 0;
}
```

Glava 4

Dinamičke strukture podataka

4.1 Liste

Zadatak 4.1 Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

(a) Definisati strukturu `Cvor` koja predstavlja čvor liste.

(b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vreća adresu novo kreiranog čvora.

(c) Napisati funkciju koja dodaje novi elemenat na početak liste.

(d) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama [,] i međusobno razdvojene zapetama.

(e) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.

(f) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.

(g) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funckije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i interativno rešenje

Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
  Unesite element koji se trazi u listi: 17
  Unesite element koji se brise iz liste: 3
Izlaz:
        Lista: []
        Lista: [2]
        Lista: [3, 2]
        Lista: [14, 3, 2]
        Lista: [5, 14, 3, 2]
        Lista: [3, 5, 14, 3, 2]
        Lista: [3, 3, 5, 14, 3, 2]
        Lista: [17, 3, 3, 5, 14, 3, 2]
        Lista: [3, 17, 3, 3, 5, 14, 3, 2]
        Lista: [1, 3, 17, 3, 3, 5, 14, 3, 2]
        Lista: [9, 1, 3, 17, 3, 3, 5, 14, 3, 2]
        Lista: [3, 9, 1, 3, 17, 3, 3, 5, 14, 3, 2]

        Trazeni broj 17 je u listi!
        Lista nakon brisanja:  [9, 1, 17, 5, 14, 2]
```

Upotreba programa 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 23 14 35
  Unesite element koji se trazi u listi: 8
  Unesite element koji se brise iz liste: 3
Izlaz:
  Lista: []
        Lista: [23]
        Lista: [14, 23]
        Lista: [35, 14, 23]

        Element NIJE u listi!
        Lista nakon brisanja:  [35, 14, 23]
```

Upotreba programa 3

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
  Unesite element koji se trazi u listi: 1
  Unesite element koji se brise iz liste: 12
Izlaz:
        Lista: []

        Element NIJE u listi!
        Lista nakon brisanja:  []
```

[Rešenje 4.1]

Zadatak 4.2 Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

(a) Definisati strukturu `Cvor` koja predstavlja čvor liste.

(b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vreća adresu novo kreiranog čvora.

(c) Napisati funkciju koja dodaje novi elemenat na kraj liste.

(d) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama [,] i međusobno razdvojene zapetama.

(e) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i interativno rešenje

Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
Izlaz:
        Lista: []
        Lista: [2]
        Lista: [2, 15]
        Lista: [2, 15, 4]
        Lista: [2, 15, 4, 8]
        Lista: [2, 15, 4, 8, 3]
        Lista: [2, 15, 4, 8, 3, 24]
        Lista: [2, 15, 4, 8, 3, 24, 11]
        Lista: [2, 15, 4, 8, 3, 24, 11, 17]
        Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4]
        Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4, 3]
        Lista: [2, 15, 4, 8, 3, 24, 11, 17, 4, 3, 7]
```

Upotreba programa 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
Izlaz:
        Lista: []
```

[Rešenje 4.2]

Zadatak 4.3 Napisati program koji koristi jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

(a) Definisati strukturu `Cvor` koja predstavlja čvor liste.

(b) Napisati funkciju koja kao argument dobija ceo broj, kreira nov čvor liste sa tom vrednosti i vreća adresu novo kreiranog čvora.

(c) Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.

(d) Napisati funkciju koja oslobađa memoriju koju je zauzela lista.

(e) Napisati funkciju koja ispisuje elemente liste, uokvirene zagradama

,

i međusobno razdvojene zapetama.

(f) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.

(g) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.

(h) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funkcije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno. Ana: Da li bi ovde trebalo da stoje dve reference na rešenja jer imamo nezavisno rekurzivno i interativno rešenje

Upotreba programa 1

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 2 3 14 5 3 3 17 3 1 9 3
  Unesite element koji se trazi u listi: 5
  Unesite element koji se brise iz liste: 3
Izlaz:
        Lista: []
        Lista: [2]
        Lista: [2, 3]
        Lista: [2, 3, 14]
        Lista: [2, 3, 5, 14]
        Lista: [2, 3, 3, 5, 14]
        Lista: [2, 3, 3, 3, 5, 14]
        Lista: [2, 3, 3, 3, 5, 14, 17]
        Lista: [2, 3, 3, 3, 3, 5, 14, 17]
        Lista: [1, 2, 3, 3, 3, 3, 5, 14, 17]
        Lista: [1, 2, 3, 3, 3, 3, 5, 9, 14, 17]
        Lista: [1, 2, 3, 3, 3, 3, 3, 5, 9, 14, 17]

        Trazeni broj 5 je u listi!
        Lista nakon brisanja:  [1, 2, 5, 9, 14, 17]
```

Upotreba programa 2

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D): 23 14 35
  Unesite element koji se trazi u listi: 8
  Unesite element koji se brise iz liste: 3
Izlaz:
  Lista: []
        Lista: [23]
        Lista: [14, 23]
        Lista: [14, 23, 35]

        Element NIJE u listi!
        Lista nakon brisanja:  [14, 23, 35]
```

Upotreba programa 3

```
Poziv: ./a.out
Ulaz:
  Unosite elemente liste! (za kraj unesite EOF tj. CTRL+D):
  Unesite element koji se trazi u listi: 1
  Unesite element koji se brise iz liste: 12
Izlaz:
        Lista: []

        Element NIJE u listi!
        Lista nakon brisanja:  []
```

[Rešenje 4.3]

Zadatak 4.4 Napisati program koji koristi dvostruko povezanu listu za

čuvanje celih brojeva koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu se prekida učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. I ovde isto mozda razdvojiti sortiranost od obične liste.

(a) Napisati funkciju koja dodaje novi elemenat na početak liste.

(b) Napisati funkciju koja dodaje novi elemenat na kraj liste.

(c) Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.

(d) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.

(e) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.

(f) Napisati funkciju koja oslobađa dinamički zauzetu memoriju za elemente liste.

Sve funckije za rad sa listom implementirati iterativno.

Zadatak 4.5 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

```
Datoteka: {[23 + 5344] * (24 - 234)} - 23
Izlaz:  Zagrade su ispravno uparene.
```

Test 2

```
Datoteka: {[23 + 5] * (9 * 2)} - {23}
Izlaz:  Zagrade su ispravno uparene.
```

Test 3

```
Datoteka: {[2 + 54) / (24 * 87)} + (234 + 23)
Izlaz:  Zagrade nisu ispravno uparene.
```

Test 3

```
Datoteka: {(2 - 14) / (23 + 11)}} * (2 + 13)
Izlaz:  Zagrade nisu ispravno uparene.
```

Test 4

```
Datoteka je prazna.
Izlaz:  Zagrade su ispravno uparene.
```

Test 5

```
Datoteka ne postoji.
Izlaz:  Greska prilikom otvaranja datoteke izraz.txt!
```

Zadatak 4.6 Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije . Milena: A sta ako se ne navede argument komandne linije? Uputstvo: za rešavanje problema koristiti stek implementiran preko listi čiji su čvorovi HTML etikete.

Test 1

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    A sutra ce biti jos lepsi.
    <a link="http://www.google.com"> Link 1</a>
    <a link="http://www.math.rs"> Link 2</a>
  </body>
</html>
Izlaz: Ispravno uparene etikete.
```

Test 2

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
</html>
Izlaz: Neispravno uparene etikete.
```

Test 3

```
Poziv: ./a.out datoteka.html
Datoteka.html:
<html>
  <head><title>Primer</title></head>
  <body>
  </body>
Izlaz: Neispravno uparene etikete.
```

Test 4

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out datoteka.html!
```

Test 5

```
Poziv: ./a.out datoteka.html
Datoteka.html ne postoji.
Izlaz: Greska prilikom otvaranja datoteke datoteka.html.
```

Test 6

```
Poziv: ./a.out datoteka.html
Datoteka.html je prazna
Izlaz: Ispravno uparene etikete.
```

Zadatak 4.7 Milena: Problem sa ovim zadatkom je sto je program najpre na usluzi korisnicima, a zatim na usluzi sluzbeniku i to nekako zbunjuje u formulaicji. Formulacija mi nije bila jasna bez citanja resenja, pokusala sam da je preciziran, u nastavku je izmenjena formulacija.
Medjutim, ja i dalje nisam bas zadovoljna i zato predlazem da se formulacija izmeni tako da je program stalno na usluzi sluzbeniku. Program ucitva podatke o prijavljenim korisnicima iz datoteke. Sluzbenik odlucuje da li ce da obradjuje redom korisnike, ili ce u nekim situacijama da odlozi rad sa korisnikom i stavi ga na kraj reda. Program ga uvek pita da na osnovu jmbg-a i zahteva odluci da li ce ga staviti na kraj reda, ako hoce, on ide na kraj reda, ako nece, onda sluzbenik daje odgovor na zahtev i jmbg, zahtev i odgovor se upisuju u izlaznu datoteku.

Napisati program kojim se simulira rad jednog šaltera na kojem se prvo zakazuju termini, a potom službenik uslužuje korisnike redom, kako su se prijavljivali.

Korisnik se prijavljuje unošenjem svog `jmbg` broja (niska koja sadrži 13 karaktera) i zahteva (niska koja sadrži najviše 999 karaktera). Prijavljivanje korisnika se prekida unošenjem karaktera za kraj ulaza (`EOF`).

Službenik redom proziva korisnike čitanjem njihovog `jmbg` broja, a zatim odlu-

čuje da li korisnika vraća na kraj reda ili ga odmah uslužuje. Službeniku se postavlja pitanje `Da li korisnika vracate na kraj reda?` i ukoliko on da odgovor `Da`, korisnik se vraća na kraj reda. Ukoliko odgovor nije `Da`, tada službenik čita korisnikov zahtev. Posle svakog 10 usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu.

Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.

Zadatak 4.8 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
Izlaz: Greska! Program se poziva sa: ./a.out datoteka.html!
```

Test 3

```
Poziv: ./a.out datoteka.html
Datoteka.html ne postoji.
Izlaz: Greska prilikom otvaranja datoteke datoteka.html.
```

Test 4

```
Poziv: ./a.out datoteka.html
Datoteka.html je prazna.
Izlaz:
```

Zadatak 4.9 Milena: malo me muci u ovom zadatku sto nema neki smisao. Naime, ako se samo vrsi ucitavanje iz datoteka i ispisivanje, onda su ove liste zapravo visak jer isti rezultat moze da se dobije i bez koriscenja listi. Zato mi fali da program uradi nesto sto ne bi mogao da uradi bez koriscenja listi, npr da na osnovu unetog broja ispisuje svaki n-ti broj rezultujuce liste pa to u nekoj petlji da korisnik moze da ispisuje za razlicite unete n ili tako nesto...

Napisati program koji objedinjuje dve sortirane liste. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

Test 1

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz: 2 4 5 6 6 10 11 12 14 15 16
```

Test 2

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out
     dat1.txt dat2.txt!
```

Test 3

```
Poziv: ./a.out dat1.txt
Izlaz: Greska! Program se poziva sa: ./a.out
     dat1.txt dat2.txt!
```

Test 4

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt ne postoji
Izlaz: Greska prilikom otvaranja datoteke dat2.
    txt.
```

Test 5

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt ne postoji
dat2.txt: 2 4 6 10 15
Izlaz: Greska prilikom otvaranja datoteke dat1.
    txt.
```

Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz: 2 4 6 10 15
```

Zadatak 4.10 Napisati funkciju koja formira listu studenata tako što se podaci o studentima učitavaju iz datoteke čije se ime zadaje kao argument komandne linije. U svakom redu datoteke nalaze se podaci o studentu i to broj indeksa, ime i prezime. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. Ispisati zatim odgovarajuću poruku i rekurzivno osloboditi memoriju koju je data lista zauzimala. Student se traži na osnovu broja indeksa, koji se zadaje sa standardnog ulaza.

Test 1

```
Poziv: ./a.out studenti.txt
Datoteka:                Ulaz:       Izlaz:
123/2014 Marko Lukic     3/2014      da: Ana Sokic
3/2014 Ana Sokic         235/2008    ne
43/2013 Jelena Ilic      41/2009     da: Marija Zaric
41/2009 Marija Zaric
13/2010 Milovan Lazic
```

Test 2

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out
    studenti.txt!
```

Test 5

```
Poziv: ./a.out  studenti.txt
studenti.txt ne postoji
Izlaz: Greska prilikom otvaranja datoteke
     studenti.txt
```

Test 5

```
Poziv: ./a.out  studenti.txt
studenti.txt prazna
Izlaz: ???  videti sta ce tacno biti
```

Zadatak 4.11 Neka su date dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajući raspoređene elemente lista L1 i L2 (prvi element iz L1, prvi element iz L2, drugi element L1, drugi element L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove rasporediti u jednu listu. Prva lista se učitava iz datoteke koja se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. Milena: Sta ako je neka lista duza? To precizirati. I ovde me muci sto nedostaje neki smisao zadatku, nesto sto ne bi moglo da se uradi da nismo koristili liste.

Test 1

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz:  2 5 4 6 6 11 10 12 15 14 16
```

Test 2

```
Poziv: ./a.out
Izlaz: Greska! Program se poziva sa: ./a.out
     dat1.txt dat2.txt!
```

Test 3

```
Poziv: ./a.out dat1.txt
Izlaz: Greska! Program se poziva sa: ./a.out
     dat1.txt dat2.txt!
```

Test 4

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt ne postoji
Izlaz: Greska prilikom otvaranja datoteke dat2.
    txt.
```

Test 5

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt ne postoji
dat2.txt: 2 4 6 10 15
Izlaz: Greska prilikom otvaranja datoteke dat1.
    txt.
```

Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz: 2 4 6 10 15
```

Zadatak 4.12 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 maximalan strogo rastući podniz.

Napisati program koji u datoteku `Rezultat.txt` upisuje nađeni strogo rastući podniz. Milena: I ovde me muci sto bi zadatak mogao da se resi i bez koriscenja listi...

Milena: Prirodni oblik testa ovde bi bio horizontalan, a ne ovako vertikalan.

Test 1

```
Poziv: ./a.out
brojevi.txt
       43 12 15 16 4 2 8
Izlaz:
Rezultat.txt : 12 15 16
```

Test 2

```
Poziv: ./a.out
brojevi.txt ne postoji
Izlaz:
Rezultat.txt : Greska prilikom otvaranja
     datoteke dat2.txt.
```

Test 3

```
Poziv: ./a.out
brojevi.txt prazna
Izlaz:
Rezultat.txt  je prazna.
```

Test 6

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt prazna
dat2.txt: 2 4 6 10 15
Izlaz: 2 4 6 10 15
```

Zadatak 4.13 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, 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: u implementaciji koristiti kružnu listu.

Test 1

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

Zadatak 4.14 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: u implementaciji koristiti dvostruko povezanu kružnu listu.

Test 1

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

Test 1

```
Ulaz: 2 7
Izlaz: n mora biti uvek
     vece od k, a 2 < 7!
```

4.2 Stabla

Zadatak 4.15 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 1

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa): 7 2 1 9 32 18
Izlaz:
  Infiksni ispis: 1 2 7 9 18 32
  Prefiksni ispis: 7 2 1 9 32 18
  Postfiksni ispis: 1 2 18 32 9 7
  Trazi se broj: 11
  Broj se ne nalazi u stablu!
  Brise se broj: 7
  Rezultujuce stablo: 1 2 9 18 32
```

Test 2

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa): 8 -2 6 13 24 -3
Izlaz:
  Infiksni ispis:  -3 -2 6 8 13 24
  Prefiksni ispis: 8 -2 -3 6 13 24
  Postfiksni ispis: -3 6 -2 24 13 8
  Trazi se broj: 6
  Broj se nalazi u stablu!
  Brise se broj: 14
  Rezultujuce stablo: -3 -2 6 8 13 24
```

[Rešenje 4.15]

Zadatak 4.16 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 prema rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku `Nedostaje ime ulazne datoteke!`. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

Test 1

```
Poziv: ./a.out test.txt
Datoteka test.txt:
  Sunce utorak raCunar SUNCE programiranje
     jabuka PROGramiranje sunCE JABUka
Izlaz:
  jabuka: 2
  programiranje: 2
  racunar: 1
  sunce: 3
  utorak: 1

  Najcesca rec: sunce (pojavljuje se 3 puta)
```

Test 2

```
Poziv: ./a.out suma.txt
Datoteka suma.txt:
  lipa zova hrast ZOVA breza LIPA
Izlaz:
  breza: 1
  hrast: 1
  lipa: 2
  zova: 2

  Najcesca rec: lipa (pojavljuje se 2 puta)
```

Test 3

```
Poziv: ./a.out
Izlaz:
  Nedostaje ime ulazne datoteke!
```

[Rešenje 4.16]

Zadatak 4.17 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. `Pera Peric 064/123-4567`. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči `KRAJ`, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Upotreba programa 1

```
Poziv: ./a.out
Datoteka imenik.txt:
   Pera Peric 011/3240-987
   Marko Maric 064/1234-987
   Mirko Maric 011/589-333
   Sanja Savkovic 063/321-098
   Zika Zikic 021/759-858
Ulaz:
  Unesite ime datoteke: imenik.txt
  Unesite ime i prezime: Pera Peric
Izlaz:
  Broj je: 011/3240-987
Ulaz:
  Unesite ime i prezime: Marko Markovic
Izlaz:
  Broj nije u imeniku!
Ulaz:
  Unesite ime i prezime: KRAJ
```

[Rešenje 4.17]

Zadatak 4.18 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niz najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang

listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

Test 1

```
Poziv: ./a.out
Datoteka prijemni.txt:
   Marko Markovic 45.4 12.3 11
   Milan Jevremovic 35.2 1.3 9
   Maja Agic 60 19 20
   Nadica Zec 54.2 10 15.8
   Jovana Milic 23.3 2 5.6
Izlaz:
  1. Maja Agic 60.0 19.0 20.0 99.0
  2. Nadica Zec 54.2 10.0 15.8 80.0
  3. Marko Markovic 45.4 12.3 11.0 68.7
  4. Milan Jevremovic 35.2 1.3 9.0 45.5
  ----------------------------------
  5. Jovana Milic 23.3 2.0 5.6 30.9
```

[Rešenje 4.18]

*** Zadatak 4.19** 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`.

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 4.19]

Zadatak 4.20 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. *Napomena: Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka stabla.h predstavljati popis funkcija biblioteke, a datoteka stabla.c njihove implementacije. Programe koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 10 5 15 3 2 4 30 12 14 13 0
  Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0
Izlaz:
  Stabla jesu identicna.
```

Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 10 5 15 4 3 2 30 12 14 13 0
  Drugo stablo: 10 15 5 3 4 2 12 14 13 30 0
Izlaz:
  Stabla nisu identicna.
```

[Rešenje 4.20]

*** Zadatak 4.21** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 1 7 8 9 2 2
  Drugo stablo: 3 9 6 11 1
Izlaz:
  Unija:  1 1 2 2 3 6 7 8 9 9 11
  Presek: 1 9
  Razlika: 2 2 7 8
```

[Rešenje 4.21]

Zadatak 4.22 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

Test 1

```
Poziv: ./a.out
Ulaz:
  n: 7
  a: 1 11 8 6 37 25 30
Izlaz:
  1 6 8 11 25 30 37
```

Test 2

```
Poziv: ./a.out
Ulaz:
  n: 55
Izlaz:
  Greska: pogresna dimenzija niza!
```

[Rešenje 4.22]

Zadatak 4.23 Dato je binarno pretraživačko stablo celih brojeva.

(a) Napisati funkciju koja izračunava broj čvorova stabla.

(b) Napisati funkciju koja izračunava broj listova stabla.

(c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.

(d) Napisati funkciju koja izračunava zbir čvorova stabla.

(e) Napisati funkciju koja izračunava najveći element stabla.

(f) Napisati funkciju koja izračunava dubinu stabla.

(g) Napisati funkciju koja izračunava broj čvorova na i-tom nivou stabla.

(h) Napisati funkciju koja ispisuje sve elemente na i-tom nivou stabla.

(i) Napisati funkciju koja izračunava maksimalnu vrednost na i-tom nivou stabla.

(j) Napisati funkciju koja izračunava zbir čvorova na i-tom nivou stabla.

(k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x.

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije.

Test 2

```
Poziv: ./a.out 2 15
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  broj cvorova: 10
  broj listova: 4
  pozitivni listovi: 2 4 13 30
  zbir cvorova: 108
  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 4.23]

Zadatak 4.24 Dato je binarno pretraživačko 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.

(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 gorenavedene funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  Cvor sa maksimalnim desnim proizvodom: 10
  Cvor sa najmanjom levom sumom: 2
  Putanja do najdubljeg cvora: 10 15 12 14 13
  Putanja do najmanjeg cvora: 10 5 3 2
```

Zadatak 4.25 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  0.nivo: 10
  1.nivo: 5 15
  2.nivo: 3 12 30
  3.nivo: 2 4 14
  4.nivo: 13
```

[Rešenje 4.25]

*** Zadatak 4.26** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 14 30 1 0
Izlaz:
  Stabla su slicna kao u ogledalu.
```

Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 20 15 0
Izlaz:
  Stabla nisu slicna kao u ogledalu.
```

Zadatak 4.27 AVL-stablo je binarno stablo pretrage kod koga apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 2 11 16 1 13
Izlaz:
  7
```

Test 2

```
Poziv: ./a.out
Ulaz:
  16 30 40 24 10 18 45 22
Izlaz:
  6
```

[Rešenje 4.27]

Zadatak 4.28 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.

Test 1

```
Poziv: ./a.out
Ulaz:
  100 19 36 17 3 25 1 2 7
Izlaz:
  Stablo je heap.
```

[Rešenje 4.28]

4.3 Rešenja

Rešenje 4.1

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

/* NAPOMENA:
   Jednostruko povezana lista je struktura podataka
   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
   sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
   liste (bez glave) je takodje lista, i naziva se rep liste.
   Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
   baratanja listom mi cuvamo samo pokazivac na glavu liste.
   Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
   sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
   pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
   preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
```

```
16    element u listi nema sledeci element: u tom slucaju se
      njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
18    lista se predstavlja NULL pokazivacem.

20    Prednost koriscenja povezanih lista  u odnosu na dinamicki
      niz je u tome sto se elementi mogu efikasno umetati i brisati
22    sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
      premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
24    ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
      elementi moraju obradjivati redom (iteracijom kroz listu).
26

28    Prilikom promene liste (dodavanje novog elementa, brisanje
       elementa,
      premestanje elemenata, itd.) postoji mogucnost da glava liste bude
30    promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
      U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
32    promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
      pozivajucoj funkciji vrati azurirana informacija o adresi glave
       liste.
34
      Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
36    pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
       cuva
      adresu glave i koju, eventalno, treba azurirati.
38    Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
      promeni promenljivu pozivajuce funkcije direktno. Npr:
40                funkcija_za_promenu(&pok, ...);
*/
42
/* Struktura koja predstavlja cvor liste */
44 typedef struct cvor {
       /* Podatak koji cvor sadrzi */
46     int vrednost;
       /* Pokazivac na sledeci cvor liste */
48     struct cvor *sledeci;
} Cvor;
50


52
/* Pomocna funkcija koja kreira cvor. Funkcija vrednost
54    novog cvora inicijalizuje na broj, dok pokazivac na
      sledeci cvor u novom cvoru postavlja na NULL.
56    Funkcija vraca pokazivac na novokreirani cvor ili NULL
      ako alokacija nije uspesno izvrsena.  */
58 Cvor * napravi_cvor(int broj) {
       Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60     if( novi == NULL )
           return NULL;
62
       /* Inicijalizacija polja u novom cvoru */
64     novi->vrednost = broj;
```

```
    novi->sledeci = NULL;
    return novi;
}



/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;

    /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
     */
    while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
     a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        /* Sledeci element je nova glava liste */
  *adresa_glave = pomocni;
    }
}



/* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
   alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
   za listu ciji pocetni cvor se nalazi na adresi adresa_glave.  */
void proveri_alokaciju(Cvor** adresa_glave, Cvor* novi) {
    /* Ukoliko je novi NULL */
    if ( novi == NULL ) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
  /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
     */

        oslobodi_listu(adresa_glave);
        exit(EXIT_FAILURE);
    }
}



/* Funkcija dodaje novi cvor na pocetak liste.
   Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
    pocetak */
void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
    /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
    alokaciji */
    Cvor *novi = napravi_cvor(broj);
```

```
    proveri_alokaciju(adresa_glave, novi);

    /* Uvezujemo novi cvor na pocetak */
    novi->sledeci = *adresa_glave;
    /* Nov cvor je sada nova glava liste */
    *adresa_glave = novi;
}



/* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL ukoliko je lista prazna */
Cvor* pronadji_poslednji (Cvor* glava) {
    /* ako je lista prazna, nema ni poslednjeg cvor
   i u tom slucaju vracamo NULL.*/
    if( glava == NULL)
        return NULL;

    /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
     pokazivac
       glava na taj sledeci element. Kada izadjemo iz petlje,
       glava ce pokazivati na element liste koji nema sledeceg,
       tj. poslednji element liste je. Zato vracamo vrednost
    pokazivaca glava.

       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 trazi u listi element cija je vrednost jednaka datom
    broju.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj
   ili NULL u slucaju da takav element ne postoji u listi. */
Cvor* pretrazi_listu(Cvor * glava, int broj) {
    for ( ; glava != NULL; glava = glava->sledeci)
      /* Pronasli smo*/
      if (glava->vrednost == broj)
  return glava;

    /* Nema trazenog broja u listi i vracamo NULL*/
    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_element(Cvor ** adresa_glave, int broj) {
    Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;

    /* Brisemo sa pocetka liste sve eventualne cvorove
       koji su jednaki datom broju, i azuriramo pokazivac na glavu */
    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
    ) {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        *adresa_glave = pomocni;
    }

    /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
    if ( *adresa_glave == NULL)
        return;

    /* Od ovog trenutka se u svakom koraku nalazimo
       na tekucem cvoru koji je razlicit od trazenog
       broja (kao i svi levo od njega). Poredimo
       vrednost sledeceg cvora (ako postoji) sa trazenim
       brojem i brisemo ga ako je jednak, a prelazimo na
       sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
       dok ne dodjemo do poslednjeg cvora. */
    tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL)
        if (tekuci->sledeci->vrednost == broj) {
    /* tekuci->sledeci treba obrisati,
       zbog toga sacuvamo njegovu adresu u pomocni */
            pomocni = tekuci->sledeci;
            /* Tekucem preusmerimo pokazivac sledeci
         tako sto preskacemo njegovog trenutnog sledeceg.
         Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
            tekuci->sledeci = tekuci->sledeci->sledeci;
      /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
    */
            free(pomocni);
        } else {
    /* Ne treba brisati sledeceg, prelazimo na sledeci */
            tekuci = tekuci->sledeci;
        }
    return;
}

/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
   Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
   ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
    pokazivac
```

```
208     na glavu liste iz pozivajuce funkcije. */
    void ispisi_listu(Cvor * glava)
210 {
        putchar('[');
212     for ( ; glava != NULL; glava = glava->sledeci)
        {
214         printf("%d", glava->vrednost);
      if( glava->sledeci != NULL )
216     printf(", ");
        }
218
        printf("]\n");
220 }
222
224
    /* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
        Cvor *glava = NULL;  /* na pocetku imamo praznu listu */
228     Cvor *trazeni = NULL;
        int broj;
230
        /* Testiramo dodavanje na pocetak*/
232     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+D
        )\n");
        printf("\n\tLista: ");
234     ispisi_listu(glava);

236     while(scanf("%d",&broj)>0)
        {
238         dodaj_na_pocetak_liste(&glava, broj);
            printf("\n\tLista: ");
240         ispisi_listu(glava);
        }
242
        printf("\nUnesite element koji se trazi u listi: ");
244     scanf("%d", &broj);

246     trazeni=pretrazi_listu(glava, broj);
        if(trazeni==NULL)
248         printf("Element NIJE u listi!\n");
        else
250         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
252
        /* brisemo elemente iz liste cije polje vrednost je jednako
254        broju procitanom sa ulaza */
        printf("\nUnesite element koji se brise iz liste: ");
256     scanf("%d", &broj);

258     obrisi_element(&glava, broj);
```

```
260     printf("Lista nakon brisanja:  ");
        ispisi_listu(glava);
262
        oslobodi_listu(&glava);
264
        return 0;
266 }
```

Rešenje 4.2

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* NAPOMENA:
5     Jednostruko povezana lista je struktura podataka
      koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
7     podatak (odredjenog tipa) i pokazivac na sledeci cvor u
      sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
9     liste (bez glave) je takodje lista, i naziva se rep liste.
      Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
11    baratanja listom mi cuvamo samo pokazivac na glavu liste.
      Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
13    sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
      pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
15    preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
      element u listi nema sledeci element: u tom slucaju se
17    njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
      lista se predstavlja NULL pokazivacem.
19
      Prednost koriscenja povezanih lista  u odnosu na dinamicki
21    niz je u tome sto se elementi mogu efikasno umetati i brisati
      sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
23    premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
      ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
25    elementi moraju obradjivati redom (iteracijom kroz listu).

27
      Prilikom promene liste (dodavanje novog elementa, brisanje
       elementa,
29    premestanje elemenata, itd.) postoji mogucnost da glava liste bude
      promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
31    U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
      promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
33    pozivajucoj funkciji vrati azurirana informacija o adresi glave
       liste.

35    Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
      pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
       cuva
37    adresu glave i koju, eventalno, treba azurirati.
```

```
   Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
   promeni promenljivu pozivajuce funkcije direktno. Npr:
   funkcija_za_promenu(&pok, ...);
*/

/* Struktura koja predstavlja cvor liste */
typedef struct cvor {
    /* Podatak koji cvor sadrzi */
    int vrednost;
    /* Pokazivac na sledeci cvor liste */
    struct cvor *sledeci;
} Cvor;



/* 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;

    /* Inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
    novi->sledeci = NULL;
    return novi;
}



/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;

    /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
     */
    while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
     a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        /* Sledeci element je nova glava liste */
        *adresa_glave = pomocni;
    }
}

```

```
89
/* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko
91  alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
    memorija
   za listu ciji pocetni cvor se nalazi na adresi adresa_glave.  */
93 void proveri_alokaciju(Cvor** adresa_glave, Cvor* novi) {
    /* Ukoliko je novi NULL */
95  if ( novi == NULL ) {
        fprintf(stderr, "Neuspela alokacija za nov cvor\n");
97 /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
    */

99      oslobodi_listu(adresa_glave);
        exit(EXIT_FAILURE);
101 }
}
103

105 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL ukoliko je lista prazna */
107 Cvor* pronadji_poslednji (Cvor* glava) {
    /* ako je lista prazna, nema ni poslednjeg cvor
109 i u tom slucaju vracamo NULL.*/
    if( glava == NULL)
111     return NULL;

113 /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
     pokazivac
   glava na taj sledeci element. Kada izadjemo iz petlje,
115 glava ce pokazivati na element liste
   koji nema sledeceg, tj, poslednji element liste je. Zato vracamo
117 vrednost pokazivaca glava.

119 glava je argument funkcije i njegove promene nece se odraziti na
   vrednost pokazivaca glava  u pozivajucoj funkciji. */
121  while (glava->sledeci != NULL)
        glava = glava->sledeci;
123
    return glava;
125 }

127



129
/* Funkcija dodaje novi cvor na kraj liste. */
131 void dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj) {
    Cvor *novi = napravi_cvor(broj);
133 /* Proveravamo da li je doslo do greske prilikom alokacije
    memorije */
    proveri_alokaciju(adresa_glave, novi);
135
```

```
    /* U slucaju prazne liste  */
    if (*adresa_glave == NULL) {
         /*  Glava nove liste je upravo novi cvor i ujedno i cela lista.
    * Azuriramo vrednost na koju pokazuje adresa_glave i tako azuriramo
    * i pokazivacku promenljivu u pozivajucoj funkciji.  */
        *adresa_glave = novi;
  /* Vracamo se iz funkcije */
        return;
    }

    /* Ako lista nije prazna, pronalazimo poslednji element liste */
    Cvor* poslednji = pronadji_poslednji(*adresa_glave);

    /* Dodajemo novi cvor na kraj preusmeravanjem pokazivaca */
    poslednji->sledeci = novi;
}


/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
   Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
   ova funkcija nece menjati listu, pa nema ni potrebe da azuriza pokazivac
   na glavu liste iz pozivajuce funkcije. */
void ispisi_listu(Cvor * glava)
{
    putchar('[');
    for ( ; glava != NULL; glava = glava->sledeci)
    {
        printf("%d", glava->vrednost);
  if( glava->sledeci != NULL )
    printf(", ");
    }

    printf("]\n");
}




/* Glavni program u kome testiramo sve funkcije za rad sa listama */
int main() {
    Cvor *glava = NULL;  /* na pocetku imamo praznu listu */
    int broj;

    /* Testiramo dodavanje na kraj liste */
    printf("\nUnesite elemente liste! (za kraj unesite EOF tj. CTRL+D)\n");
    printf("\n\tLista: ");
    ispisi_listu(glava);
```

```
185     while(scanf("%d",&broj)>0) {
            dodaj_na_kraj_liste(&glava, broj);
187         printf("\n\tLista: ");
            ispisi_listu(glava);
189     }

191     oslobodi_listu(&glava);

193     return 0;
}
```

Rešenje 4.3

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

4 /* NAPOMENA:
     Jednostruko povezana lista je struktura podataka
6    koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
     podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8    sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
     liste (bez glave) je takodje lista, i naziva se rep liste.
10   Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
     baratanja listom mi cuvamo samo pokazivac na glavu liste.
12   Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
     sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
14   pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
     preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
16   element u listi nema sledeci element: u tom slucaju se
     njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
18   lista se predstavlja NULL pokazivacem.

20   Prednost koriscenja povezanih lista  u odnosu na dinamicki
     niz je u tome sto se elementi mogu efikasno umetati i brisati
22   sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
     premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
24   ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
     elementi moraju obradjivati redom (iteracijom kroz listu).
26

28   Prilikom promene liste (dodavanje novog elementa, brisanje
      elementa,
     premestanje elemenata, itd.) postoji mogucnost da glava liste bude
30   promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
     U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
32   promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
     pozivajucoj funkciji vrati azurirana informacija o adresi glave
      liste.
34

     Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
```

```
   pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
    cuva
   adresu glave i koju, eventalno, treba azurirati.
   Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
   promeni promenljivu pozivajuce funkcije direktno. Npr:
   funkcija_za_promenu(&pok, ...);
*/

/* Struktura koja predstavlja cvor liste */
typedef struct cvor {
    /* Podatak koji cvor sadrzi */
    int vrednost;
    /* Pokazivac na sledeci cvor liste */
    struct cvor *sledeci;
} Cvor;



/* 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;

    /* Inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
    novi->sledeci = NULL;
    return novi;
}



/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;

    /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
     */
    while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
     a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        /* Sledeci element je nova glava liste */
        *adresa_glave = pomocni;
    }
```

```
86 }
88
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
       ukoliko
   alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
       memorija
92  za listu ciji pocetni cvor se nalazi na adresi adresa_glave.  */
   void proveri_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
       if ( novi == NULL ) {
96         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
     /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
         */
98
           oslobodi_listu(adresa_glave);
100        exit(EXIT_FAILURE);
       }
102 }
104
106 /* Funkcija dodaje novi cvor na pocetak liste.
   Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
       pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
       /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
       alokaciji */
110    Cvor *novi = napravi_cvor(broj);
       proveri_alokaciju(adresa_glave, novi);
112
       /* Uvezujemo novi cvor na pocetak */
114    novi->sledeci = *adresa_glave;
       /* Nov cvor je sada nova glava liste */
116    *adresa_glave = novi;
   }
118
120 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov element sa vrednoscu broj.*/
122 Cvor * pronadji_mesto_umetanja(Cvor* glava, int broj ) {
       /*Ako je lista prazna onda nema takvog mesta i vracamo NULL */
124    if(glava == NULL)
           return NULL;
126
       /* Krecemo se kroz listu sve dok se ne dodje do elementa
128    ciji je sledeci element veci ili jednak od novog elementa,
   ili dok se ne dodje do poslednjeg elementa.
130
   Zbog lenjog izracunavanja izraza u C-u prvi deo konjukcije
132 mora biti provera da li smo dosli do poslednjeg elementa liste
```

```
   pre nego sto proverimo vrednost njegovog sledeceg elementa,
134    jer u slucaju poslednjeg, sledeci ne postoji, pa ni vrednost.*/
     while (glava->sledeci != NULL  && glava->sledeci->vrednost < broj
     )
136         glava = glava->sledeci;

138     /* Iz petlje smo mogli izaci jer smo dosli do poslednjeg elementa
           ili smo se zaustavili ranije na elementu ciji sledeci ima
140        vrednost vecu od broj */
     return glava;
142 }


144
void dodaj_iza(Cvor* tekuci, Cvor* novi) {
146     /* Novi element dodajemo iza tekuceg elementa */
     novi->sledeci = tekuci->sledeci;
148     tekuci->sledeci = novi;
}
150
/* Funkcija dodaje novi element u sortiranu listu
152    tako da nova lista ostane sortirana.*/
void dodaj_sortirano(Cvor ** adresa_glave, int broj) {
154     /* U slucaju prazne liste glava nove liste je upravo novi cvor */
     if ( *adresa_glave == NULL ) {
156         Cvor *novi = napravi_cvor(broj);
         /* Proveravamo da li je doslo do greske prilikom alokacije
     memorije */
158         proveri_alokaciju(adresa_glave, novi);
         *adresa_glave = novi;
160         return;
     }
162
     /* Lista nije prazna*/
164     /* Ako je broj manji ili jednak vrednosti u glavi liste,
        onda ga dodajemo na pocetak liste  */
166     if ( (*adresa_glave)->vrednost >= broj ) {
         dodaj_na_pocetak_liste(adresa_glave, broj);
168         return;
     }
170
     /* U slucaju da je glava liste element manji od novog elementa,
172        tada pronalazimo element liste iza koga treba da se umetne nov
      broj */
     Cvor* pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
174
     Cvor *novi = napravi_cvor(broj);
176     /* Proveravamo da li je doslo do greske prilikom alokacije
     memorije */
     proveri_alokaciju(adresa_glave, novi);
178
     /* Uvezujemo novi cvor iza pomocnog */
180     dodaj_iza(pomocni, novi);
```

```
}


/* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju.
   Funkcija se u pretrazi oslanja na cinjenicu da je lista
   koja se pretrazuje rastuce sortirana.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj
   ili NULL u slucaju da takav element ne postoji u listi. */
Cvor* pretrazi_listu(Cvor * glava, int broj) {
    /* U konjukciji koja cini uslov ostanka u petlji, bitan je
    redosled! */
    for ( ; glava != NULL && glava->vrednost <= broj ; glava = glava
    ->sledeci)
      /* Pronasli smo*/
      if (glava->vrednost == broj)
  return glava;

    /* Nema trazenog broja u listi i vracamo NULL*/
    return NULL;
}


/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
    rastuce.
   Funkcija azurira pokazivac na glavu liste, koji moze biti
   promenjen u slucaju da se obrise stara glava liste. */
void obrisi_element(Cvor ** adresa_glave, int broj) {
    Cvor *tekuci = NULL ;
    Cvor *pomocni = NULL ;

    /* Brisemo sa pocetka liste sve eventualne cvorove
       koji su jednaki datom broju, i azuriramo pokazivac na glavu */
    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
    ) {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        *adresa_glave = pomocni;
    }

    /* Ako je nakon toga lista ostala prazna ili glava liste sadrzi
    vrednost
       koja je veca od broja, kako je lista sortirana rastuce nema
    potrebe
       broj traziti u repu liste i zato prekidamo funkciju */
    if ( *adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
```

```
226     /* Od ovog trenutka se u svakom koraku nalazimo u tekucem cvoru
        cija
           vrednost je manja od trazenog broja (kao i svi levo od njega).
228        Poredimo vrednost sledeceg cvora (ako postoji) sa trazenim
        brojem
           i brisemo ga ako je jednak, a prelazimo na sledeci cvor ako je
230        razlicit. Ovaj postupak ponavljamo dok ne dodjemo do
        poslednjeg cvora
           ili prvog cvora cija vrednost je veca od trazenog broja. */
232     tekuci = *adresa_glave;
        while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <=
        broj)
234         if (tekuci->sledeci->vrednost == broj) {
        /* tekuci->sledeci treba obrisati,
236        zbog toga sacuvamo njegovu adresu u pomocni */
               pomocni = tekuci->sledeci;
238            /* Tekucem preusmerimo pokazivac sledeci
             tako sto preskacemo njegovog trenutnog sledeceg.
240           Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
               tekuci->sledeci = tekuci->sledeci->sledeci;
242       /* Sada mozemo
          * slobodno i da oslobodimo cvor sa vrednoscu broj */
244            free(pomocni);
           } else {
246     /* Ne treba brisati sledeceg, jer je manji od trazenog
           i prelazimo na sledeci */
248            tekuci = tekuci->sledeci;
           }
250     return;
}
252
/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
254    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
       ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
        pokazivac
256    na glavu liste iz pozivajuce funkcije. */
void ispisi_listu(Cvor * glava)
258{
    putchar('[');
260 for ( ; glava != NULL; glava = glava->sledeci)
    {
262     printf("%d", glava->vrednost);
  if( glava->sledeci != NULL )
264 printf(", ");
    }
266
    printf("]\n");
268}

270


272
```

```c
/* Glavni program u kome testiramo sve funkcije za rad sa listama */
int main() {
    Cvor *glava = NULL;  /* na pocetku imamo praznu listu */
    Cvor *trazeni = NULL;
    int broj;

    /* Testiramo dodavanje u listu tako da ona bude rastuce sortirana
     */
    printf("\nUnosite elemente liste! (za kraj unesite EOF tj. CTRL+D
    )\n");
    printf("\n\tLista: ");
    ispisi_listu(glava);

    while(scanf("%d",&broj)>0)
    {
        dodaj_sortirano(&glava, broj);
        printf("\n\tLista: ");
        ispisi_listu(glava);
    }

    printf("\nUnesite element koji se trazi u listi: ");
    scanf("%d", &broj);

    trazeni = pretrazi_listu(glava, broj);
    if(trazeni == NULL)
        printf("Element NIJE u listi!\n");
    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);


    /* Brisemo elemente iz liste cije polje vrednost je jednako broju
       procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
    scanf("%d", &broj);

    obrisi_element(&glava, broj);

    printf("Lista nakon brisanja:  ");
    ispisi_listu(glava);

    oslobodi_listu(&glava);

    return 0;
}
```

Rešenje 4.4

Rešenje 4.5

Rešenje 4.6

Rešenje 4.7

Rešenje 4.8

Rešenje 4.9

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

/* NAPOMENA:
   Jednostruko povezana lista je struktura podataka
   koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
   podatak (odredjenog tipa) i pokazivac na sledeci cvor u
   sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
   liste (bez glave) je takodje lista, i naziva se rep liste.
   Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
   baratanja listom mi cuvamo samo pokazivac na glavu liste.
   Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
   sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
   pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
   preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
   element u listi nema sledeci element: u tom slucaju se
   njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
   lista se predstavlja NULL pokazivacem.

   Prednost koriscenja povezanih lista  u odnosu na dinamicki
   niz je u tome sto se elementi mogu efikasno umetati i brisati
   sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
   premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
   ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
   elementi moraju obradjivati redom (iteracijom kroz listu).


   Prilikom promene liste (dodavanje novog elementa, brisanje
    elementa,
   premestanje elemenata, itd.) postoji mogucnost da glava liste bude
   promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
   U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
   promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
   pozivajucoj funkciji vrati azurirana informacija o adresi glave
    liste.

   Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
   pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
    cuva
   adresu glave i koju, eventalno, treba azurirati.
   Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
```

```
   promeni promenljivu pozivajuce funkcije direktno. Npr:
40                 funkcija_za_promenu(&pok, ...);
*/

/* Struktura koja predstavlja cvor liste */
typedef struct cvor {
    /* Podatak koji cvor sadrzi */
    int vrednost;
    /* Pokazivac na sledeci cvor liste */
    struct cvor *sledeci;
} Cvor;
50


52

/* Pomocna funkcija koja kreira cvor. Funkcija vrednost
54    novog cvora inicijalizuje na broj, dok pokazivac na
   sledeci cvor u novom cvoru postavlja na NULL.
56    Funkcija vraca pokazivac na novokreirani cvor ili NULL
   ako alokacija nije uspesno izvrsena.  */
58 Cvor * napravi_cvor(int broj) {
    Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60    if( novi == NULL )
        return NULL;
62
    /* Inicijalizacija polja u novom cvoru */
64    novi->vrednost = broj;
    novi->sledeci = NULL;
66    return novi;
}
68


70


72 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
    Cvor *pomocni = NULL;
76
    /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
     */
78    while (*adresa_glave != NULL) {
        /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
80     a tek onda osloboditi element koji predstavlja glavu liste */
        pomocni = (*adresa_glave)->sledeci;
82        free(*adresa_glave);
        /* Sledeci element je nova glava liste */
84   *adresa_glave = pomocni;
    }
86 }

88
```

```
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
       ukoliko
   alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
       memorija
92    za listu ciji pocetni cvor se nalazi na adresi adresa_glave.  */
   void proveri_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
       if ( novi == NULL ) {
96         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
     /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
         */
98
           oslobodi_listu(adresa_glave);
100        exit(EXIT_FAILURE);
       }
102 }


104


106 /* Funkcija dodaje novi cvor na pocetak liste.
       Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
         pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
       /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
       alokaciji */
110    Cvor *novi = napravi_cvor(broj);
       proveri_alokaciju(adresa_glave, novi);
112
       /* Uvezujemo novi cvor na pocetak */
114    novi->sledeci = *adresa_glave;
       /* Nov cvor je sada nova glava liste */
116    *adresa_glave = novi;
   }
118


120
   /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
122    ili NULL ukoliko je lista prazna */
   Cvor* pronadji_poslednji (Cvor* glava) {
124    /* ako je lista prazna, nema ni poslednjeg cvor
       i u tom slucaju vracamo NULL.*/
126    if( glava == NULL)
           return NULL;
128
       /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
        pokazivac
130      glava na taj sledeci element. Kada izadjemo iz petlje,
         glava ce pokazivati na element liste koji nema sledeceg,
132      tj. poslednji element liste je. Zato vracamo vrednost
       pokazivaca glava.

134      glava je argument funkcije i njegove promene nece se odraziti
```

```
    na
       vrednost pokazivaca glava  u pozivajucoj funkciji. */
136     while (glava->sledeci != NULL)
            glava = glava->sledeci;
138
    return glava;
140 }

142
/* Funkcija trazi u listi element cija je vrednost jednaka datom
    broju.
144    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj
   ili NULL u slucaju da takav element ne postoji u listi. */
146 Cvor* pretrazi_listu(Cvor * glava, int broj) {
    for ( ; glava != NULL; glava = glava->sledeci)
148       /* Pronasli smo*/
      if (glava->vrednost == broj)
150   return glava;

152    /* Nema trazenog broja u listi i vracamo NULL*/
    return NULL;
154 }

156


158
/* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
160    Funkcija azurira pokazivac na glavu liste (koji moze biti
   promenjen u slucaju da se obrise stara glava) */
162 void obrisi_element(Cvor ** adresa_glave, int broj) {
    Cvor *tekuci = NULL;
164    Cvor *pomocni = NULL;

166    /* Brisemo sa pocetka liste sve eventualne cvorove
       koji su jednaki datom broju, i azuriramo pokazivac na glavu */
168    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
    ) {
        /* Sacuvamo adresu repa liste pre oslobadjanja glave */
170        pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
172        *adresa_glave = pomocni;
    }
174
    /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
176    if ( *adresa_glave == NULL)
        return;
178
    /* Od ovog trenutka se u svakom koraku nalazimo
180       na tekucem cvoru koji je razlicit od trazenog
       broja (kao i svi levo od njega). Poredimo
182       vrednost sledeceg cvora (ako postoji) sa trazenim
       brojem i brisemo ga ako je jednak, a prelazimo na
```

```
184         sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
            dok ne dodjemo do poslednjeg cvora. */
186     tekuci = *adresa_glave;
        while (tekuci->sledeci != NULL)
188         if (tekuci->sledeci->vrednost == broj) {
        /* tekuci->sledeci treba obrisati,
190        zbog toga sacuvamo njegovu adresu u pomocni */
                pomocni = tekuci->sledeci;
192             /* Tekucem preusmerimo pokazivac sledeci
              tako sto preskacemo njegovog trenutnog sledeceg.
194           Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
                tekuci->sledeci = tekuci->sledeci->sledeci;
196       /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
        */
                free(pomocni);
198         } else {
        /* Ne treba brisati sledeceg, prelazimo na sledeci */
200             tekuci = tekuci->sledeci;
            }
202     return;
}
204
/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
206    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
       ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
        pokazivac
208    na glavu liste iz pozivajuce funkcije. */
void ispisi_listu(Cvor * glava)
210 {
        putchar('[');
212     for ( ; glava != NULL; glava = glava->sledeci)
        {
214         printf("%d", glava->vrednost);
      if( glava->sledeci != NULL )
216     printf(", ");
        }
218
        printf("]\n");
220 }

222



224
/* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
        Cvor *glava = NULL;  /* na pocetku imamo praznu listu */
228     Cvor *trazeni = NULL;
        int broj;
230
        /* Testiramo dodavanje na pocetak*/
232     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+D
        )\n");
```

```
    printf("\n\tLista: ");
234 ispisi_listu(glava);

236 while(scanf("%d",&broj)>0)
    {
238     dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240     ispisi_listu(glava);
    }
242
    printf("\nUnesite element koji se trazi u listi: ");
244 scanf("%d", &broj);

246 trazeni=pretrazi_listu(glava, broj);
    if(trazeni==NULL)
248     printf("Element NIJE u listi!\n");
    else
250     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);


252
    /* brisemo elemente iz liste cije polje vrednost je jednako
254    broju procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
256 scanf("%d", &broj);

258 obrisi_element(&glava, broj);

260 printf("Lista nakon brisanja:  ");
    ispisi_listu(glava);
262
    oslobodi_listu(&glava);
264
    return 0;
266 }
```

Rešenje 4.10

Rešenje 4.11

Rešenje 4.12

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

4 /* NAPOMENA:
     Jednostruko povezana lista je struktura podataka
6    koja se sastoji od sekvence cvorova. Svaki cvor sadrzi
     podatak (odredjenog tipa) i pokazivac na sledeci cvor u
8    sekvenci. Prvi cvor u sekvenci naziva se glava liste. Ostatak
```

```
   liste (bez glave) je takodje lista, i naziva se rep liste.
   Lista koja ne sadrzi cvorove naziva se prazna lista. Prilikom
   baratanja listom mi cuvamo samo pokazivac na glavu liste.
   Kada pristupimo glavi liste, u njoj imamo zapisanu adresu
   sledeceg elementa, pa mu samim tim mozemo pristupiti. Kada mu
   pristupimo, u njemu je sadrzana adresa sledeceg elementa, pa
   preko tog pokazivaca mozemo da mu pristupimo, itd. Poslednji
   element u listi nema sledeci element: u tom slucaju se
   njegov pokazivac na sledeci postavlja na NULL. Takodje, prazna
   lista se predstavlja NULL pokazivacem.

   Prednost koriscenja povezanih lista  u odnosu na dinamicki
   niz je u tome sto se elementi mogu efikasno umetati i brisati
   sa bilo koje pozicije u nizu, bez potrebe za realokacijom ili
   premestanjem elemenata. Nedostatak ovakvog pristupa je to sto
   ne mozemo nasumicno pristupiti proizvoljnom elementu, vec se
   elementi moraju obradjivati redom (iteracijom kroz listu).


   Prilikom promene liste (dodavanje novog elementa, brisanje
    elementa,
   premestanje elemenata, itd.) postoji mogucnost da glava liste bude
   promenjena, tj. da to postane neki drugi cvor (sa drugom adresom).
   U tom slucaju se pokazivac na glavu liste mora azurirati. Kada
   promenu liste obavljamo u posebnoj funkciji onda je potrebno da se
   pozivajucoj funkciji vrati azurirana informacija o adresi glave
    liste.

   Pozvana funkcija koja vrsi promenu na listi prihvata kao argument
   pokazivac na pokazivacku promenljivu koja u pozivajucoj funkciji
    cuva
   adresu glave i koju, eventalno, treba azurirati.
   Sada pozvana funkcija moze interno da preko dobijenog pokazivaca
   promeni promenljivu pozivajuce funkcije direktno. Npr:
                funkcija_za_promenu(&pok, ...);
*/

/* Struktura koja predstavlja cvor liste */
typedef struct cvor {
    /* Podatak koji cvor sadrzi */
    int vrednost;
    /* Pokazivac na sledeci cvor liste */
    struct cvor *sledeci;
} Cvor;



/* 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.  */
```

```
58 Cvor * napravi_cvor(int broj) {
       Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
60     if( novi == NULL )
           return NULL;
62
       /* Inicijalizacija polja u novom cvoru */
64     novi->vrednost = broj;
       novi->sledeci = NULL;
66     return novi;
   }
68
70
72 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
      ciji se pocetni cvor nalazi na adresi adresa_glave. */
74 void oslobodi_listu(Cvor ** adresa_glave) {
       Cvor *pomocni = NULL;
76
       /* Ako lista nije prazna, onda ima memorije koju treba osloboditi
        */
78     while (*adresa_glave != NULL) {
           /* Potrebno je najpre zapamtiti adresu sledeceg elementa,
80      a tek onda osloboditi element koji predstavlja glavu liste */
           pomocni = (*adresa_glave)->sledeci;
82         free(*adresa_glave);
           /* Sledeci element je nova glava liste */
84    *adresa_glave = pomocni;
       }
86 }
88
90 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
       ukoliko
      alokacija nije bila uspesna, oslobadja se sva prethodno zauzeta
       memorija
92    za listu ciji pocetni cvor se nalazi na adresi adresa_glave.  */
   void proveri_alokaciju(Cvor** adresa_glave, Cvor* novi) {
94     /* Ukoliko je novi NULL */
       if ( novi == NULL ) {
96         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
     /* Oslobadjamo svu dinamicki alociranu memoriju i prekidamo program
        */
98
           oslobodi_listu(adresa_glave);
100        exit(EXIT_FAILURE);
       }
102 }
104
```

```
106 /* Funkcija dodaje novi cvor na pocetak liste.
       Kreira novi cvor koriscenjem funkcije napravi_cvor i uvezuje ga na
         pocetak */
108 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj) {
        /* Kreiramo nov cvor i proveravamo da li je bilo greske pri
        alokaciji */
110     Cvor *novi = napravi_cvor(broj);
        proveri_alokaciju(adresa_glave, novi);
112
        /* Uvezujemo novi cvor na pocetak */
114     novi->sledeci = *adresa_glave;
        /* Nov cvor je sada nova glava liste */
116     *adresa_glave = novi;
    }
118


120
    /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
122    ili NULL ukoliko je lista prazna */
    Cvor* pronadji_poslednji (Cvor* glava) {
124     /* ako je lista prazna, nema ni poslednjeg cvor
       i u tom slucaju vracamo NULL.*/
126     if( glava == NULL)
            return NULL;
128
        /* Sve dok glava ne pokazuje na cvor koji nema sledeceg, pomeramo
         pokazivac
130       glava na taj sledeci element. Kada izadjemo iz petlje,
          glava ce pokazivati na element liste koji nema sledeceg,
132       tj. poslednji element liste je. Zato vracamo vrednost
        pokazivaca glava.

134       glava je argument funkcije i njegove promene nece se odraziti
        na
          vrednost pokazivaca glava  u pozivajucoj funkciji. */
136     while (glava->sledeci != NULL)
            glava = glava->sledeci;
138
        return glava;
140 }


142
    /* Funkcija trazi u listi element cija je vrednost jednaka datom
        broju.
144    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj
       ili NULL u slucaju da takav element ne postoji u listi. */
146 Cvor* pretrazi_listu(Cvor * glava, int broj) {
        for ( ; glava != NULL; glava = glava->sledeci)
148       /* Pronasli smo*/
          if (glava->vrednost == broj)
150   return glava;
```

```
152     /* Nema trazenog broja u listi i vracamo NULL*/
        return NULL;
154 }
156
158
    /* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
160     Funkcija azurira pokazivac na glavu liste (koji moze biti
        promenjen u slucaju da se obrise stara glava) */
162 void obrisi_element(Cvor ** adresa_glave, int broj) {
        Cvor *tekuci = NULL;
164     Cvor *pomocni = NULL;

166     /* Brisemo sa pocetka liste sve eventualne cvorove
           koji su jednaki datom broju, i azuriramo pokazivac na glavu */
168     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj
        ) {
            /* Sacuvamo adresu repa liste pre oslobadjanja glave */
170         pomocni = (*adresa_glave)->sledeci;
            free(*adresa_glave);
172         *adresa_glave = pomocni;
        }
174
        /* Ako je nakon toga lista ostala prazna prekidamo funkciju */
176     if ( *adresa_glave == NULL)
            return;
178
        /* Od ovog trenutka se u svakom koraku nalazimo
180        na tekucem cvoru koji je razlicit od trazenog
           broja (kao i svi levo od njega). Poredimo
182        vrednost sledeceg cvora (ako postoji) sa trazenim
           brojem i brisemo ga ako je jednak, a prelazimo na
184        sledeci cvor ako je razlicit. Ovaj postupak ponavljamo
           dok ne dodjemo do poslednjeg cvora. */
186     tekuci = *adresa_glave;
        while (tekuci->sledeci != NULL)
188         if (tekuci->sledeci->vrednost == broj) {
        /* tekuci->sledeci treba obrisati,
190        zbog toga sacuvamo njegovu adresu u pomocni */
                pomocni = tekuci->sledeci;
192             /* Tekucem preusmerimo pokazivac sledeci
              tako sto preskacemo njegovog trenutnog sledeceg.
194           Njegov novi sledeci ce biti sledeci od ovog koga brisemo. */
                tekuci->sledeci = tekuci->sledeci->sledeci;
196       /* Sada mozemo slobodno i da oslobodimo cvor sa vrednoscu broj
        */
                free(pomocni);
198         } else {
        /* Ne treba brisati sledeceg, prelazimo na sledeci */
200             tekuci = tekuci->sledeci;
            }
```

```
202     return;
}
204
/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
206    Ne saljemo joj adresu promenljive koja cuva glavu liste, jer
    ova funkcija nece menjati listu, pa nema ni potrebe da azuriza
     pokazivac
208    na glavu liste iz pozivajuce funkcije. */
void ispisi_listu(Cvor * glava)
210 {
    putchar('[');
212     for ( ; glava != NULL; glava = glava->sledeci)
    {
214         printf("%d", glava->vrednost);
  if( glava->sledeci != NULL )
216     printf(", ");
    }
218
    printf("]\n");
220 }

222



224
/* Glavni program u kome testiramo sve funkcije za rad sa listama */
226 int main() {
    Cvor *glava = NULL;  /* na pocetku imamo praznu listu */
228     Cvor *trazeni = NULL;
    int broj;
230
    /* Testiramo dodavanje na pocetak*/
232     printf("\nUnesite elemente liste. (za kraj unesite EOF tj. CTRL+D
    )\n");
    printf("\n\tLista: ");
234     ispisi_listu(glava);

236     while(scanf("%d",&broj)>0)
    {
238         dodaj_na_pocetak_liste(&glava, broj);
        printf("\n\tLista: ");
240         ispisi_listu(glava);
    }
242
    printf("\nUnesite element koji se trazi u listi: ");
244     scanf("%d", &broj);

246     trazeni=pretrazi_listu(glava, broj);
    if(trazeni==NULL)
248         printf("Element NIJE u listi!\n");
    else
250         printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
```

```
252
    /* brisemo elemente iz liste cije polje vrednost je jednako
254    broju procitanom sa ulaza */
    printf("\nUnesite element koji se brise iz liste: ");
256 scanf("%d", &broj);

258 obrisi_element(&glava, broj);

260 printf("Lista nakon brisanja:  ");
    ispisi_listu(glava);
262
    oslobodi_listu(&glava);
264
    return 0;
266 }
```

Rešenje 4.13

Rešenje 4.14

Rešenje 4.15

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* a) Struktura kojom se predstavlja cvor binarnog
5  pretrazivackog stabla */
   typedef struct cvor {
7    int broj;
     struct cvor *levo;
9    struct cvor *desno;
   }Cvor;
11
   /* b) Funkcija koja alocira memoriju za novi cvor stabla,
13        inicijalizuje polja strukture i vraca pokazivac na novi
          cvor */
15 Cvor * napravi_cvor(int broj)
   {
17   /* Alociramo memoriju */
     Cvor * novi = (Cvor *) malloc(sizeof(Cvor));
19   if (novi == NULL)
       return NULL;
21   /* Inicijalizujemo polja novog cvora. */
     novi->broj = broj;
23   novi->levo = NULL;
     novi->desno = NULL;
25
     /* Vracamo adresu novog cvora. */
27   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);
  }
}

/* c) Funkcija koja dodaje zadati broj u stablo */
void dodaj_u_stablo(Cvor ** adresa_korena, int broj){

  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {

    /* Kreiramo novi cvor */
    Cvor * novi = napravi_cvor(broj);
    proveri_alokaciju(novi);

    /* I proglasavamo ga korenom stabla */
    *adresa_korena = novi;

    return;
  }

  /* U suprotnom trazimo odgovarajucu poziciju za zadati broj */

  /* Ako je zadata vrednost manja od vrednosti korena */
  if (broj < (*adresa_korena)->broj)
    /* Dodajemo broj u levo podstablo */
    dodaj_u_stablo(&(*adresa_korena)->levo, broj);
  else
    /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa ga
dodajemo u desno podstablo */
    dodaj_u_stablo(&(*adresa_korena)->desno, broj);
}
    /*
 d) Funkcija koja proverava da li se zadati broj nalazi
       u stablu
 */

Cvor * pretrazi_stablo(Cvor * koren, int broj)
{
  /* Ako je stablo prazno, vrednost se sigurno ne nalazi u
   *        njemu */
  if (koren == NULL)
```

```
    return NULL;

  /* Ako je trazena vrednost sadrazana u korenu */
  if (koren->broj == broj) {
    /* Prekidamo pretragu */
    return koren;
  }

  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
  if (broj < koren->broj)
    /* Pretragu nastavljamo u levom podstablu */
    return pretrazi_stablo(koren->levo, broj);
  else
    /* U suprotnom, pretragu nastavljamo u desnom podstablu */
    return pretrazi_stablo(koren->desno, broj);
}

 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost
u stablu */
Cvor * pronadji_najmanji(Cvor * koren)
{
  /* Ako je stablo prazno, prekidamo pretragu */
  if (koren == NULL)
    return NULL;

  /* Vrednosti koje su manje od vrednosti u korenu stabla
        nalaze se levo od njega */

  /* Ako je koren cvor koji nema levo podstablo, onda on
        sadrzi najmanju vrednost */
  if (koren->levo == NULL)
    return koren;

  /* Inace, pretragu treba nastaviti u levom podstablu */
  return pronadji_najmanji(koren->levo);
}

/* f) Funkcija pronalazi cvor koji sadrzi najvecu vrednost u stablu
*/
Cvor * pronadji_najveci(Cvor * koren)
{
  /* Ako je stablo prazno, prekidamo pretragu */
  if (koren == NULL)
    return NULL;

  /* Vrednosti koje su vece od vrednosti u korenu stabla
        nalaze se desno od njega */

  /* Ako je koren cvor koji nema desno podstablo, onda on
        sadrzi najvecu vrednost */
  if (koren->desno == NULL)
    return koren;
```

```
133   /* Inace, pretragu treba nastaviti u desnom podstablu */
      return pronadji_najveci(koren->desno);
135
}
137
/* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
139 void obrisi_element(Cvor ** adresa_korena, int broj)
{
141   Cvor * pomocni_cvor = NULL;

143   /* ako je stablo prazno, brisanje nije primenljivo pa mozemo
prekinuti rad funkcije */
145   if (*adresa_korena == NULL)
        return;
147
      /* Ako je vrednost koju treba obrisati manja od vrednosti
149   u korenu stabla,
      ona se eventualno nalazi u levom podstablu,
151    pa treba rekurzivno primeniti postupak na
              levo
153     podstablo. Koren ovako modifikovanog stabla je nepromenjen. */
      if (broj < (*adresa_korena)->broj) {
155     obrisi_element(&(*adresa_korena)->levo, broj);
        return;
157   }

159   /* Ako je vrednost koju treba obrisati veca od vrednosti u
     korenu stabla,
161   ona se eventualno nalazi u
            desnom
163   podstablu
      pa treba rekurzivno primeniti
165           postupak na
      desno
167   podstablo. Koren ovako
            modifikovanog stabla je
169   nepromenjen. */
       if ((*adresa_korena)->broj < broj) {
171      obrisi_element(&(*adresa_korena)->desno, broj);
         return;
173    }
       /* Slede podslucajevi vezani za slucaj kada je vrednost
175    u
      korenu jednaka broju koji se brise (tj. slucaj
177           kada

179   treba obrisati koren) */

181   /* Ako koren nema sinova, tada se on prosto brise, i
            rezultat je prazno stablo (vracamo NULL) */
183   if ((*adresa_korena)->levo == NULL
```

```
          &&(*adresa_korena)->desno == NULL) {
185   free(*adresa_korena);
      *adresa_korena = NULL;
187   return;
    }
189
    /* Ako koren ima samo levog sina, tada se brisanje vrsi
191        tako sto obrisemo koren, a novi koren postaje levi sin */
    if ((*adresa_korena)->levo != NULL
193         &&(*adresa_korena)->desno == NULL) {
      pomocni_cvor = (*adresa_korena)->levo;
195   free(*adresa_korena);
      *adresa_korena = pomocni_cvor;
197   return;
    }
199
    /* Ako koren ima samo desnog sina, tada se brisanje vrsi
201        tako sto obrisemo koren, a novi koren postaje desni sin
  */
203 if ((*adresa_korena)->desno != NULL
          &&(*adresa_korena)->levo == NULL) {
205   pomocni_cvor = (*adresa_korena)->desno;
      free(*adresa_korena);
207   *adresa_korena = pomocni_cvor;
      return;
209 }

211 /* Slucaj kada koren ima oba sina. Tada se brisanje vrsi
        na sledeci nacin:
213 - najpre se potrazi sledbenik
        korena (u smislu poretka) u stablu. To je upravo po
215     vrednosti najmanji cvor u desnom podstablu.
  On se moze
217 pronaci npr. funkcijom pronadji_najmanji().
  Nakon
219 toga se u koren smesti vrednost tog cvora, a
        u
221 taj cvor
  se smesti vrednost korena (tj. broj koji
223     se
  brise).
225
  - Onda se prosto rekurzivno pozove
227     funkcija
  za brisanje
229 na desno podstablo. S obzirom
        da u njemu
231 treba
  obrisati
233 najmanji element, a on
        zasigurno ima
235 najvise
```

```
 jednog potomka, jasno je da ce
237          njegovo
 brisanje biti
239  obavljeno na
 jedan od
241          jednostavnijih
 nacina koji su
243  gore opisani. */
   pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
245   (*adresa_korena)->broj = pomocni_cvor->broj;
   pomocni_cvor->broj = broj;
247   obrisi_element(&(*adresa_korena)->desno, broj);
}
249
/* h) Funkcija ispisuje stablo u infiksnoj notaciji ( Levo
251        postablo - Koren - Desno podstablo ) */
void ispisi_stablo_infiksno(Cvor * koren)
253 {
   /* Ako stablo nije prazno */
255   if (koren != NULL) {
     /* Prvo ispisujemo sve cvorove levo od korena */
257     ispisi_stablo_infiksno(koren->levo);

259     /* Ispisujemo vrednost u korenu */
     printf("%d ", koren->broj);
261
     /* Na kraju ispisujemo cvorove desno od korena */
263     ispisi_stablo_infiksno(koren->desno);
   }
265
}
267
/* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren - Levo
269 podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren)
271 {
   /* Ako stablo nije prazno */
273   if (koren != NULL) {

275     /* Prvo ispisujemo vrednost u korenu */
     printf("%d ", koren->broj);
277
     /* Ispisujemo sve cvorove levo od korena */
279     ispisi_stablo_prefiksno(koren->levo);

281     /* Na kraju ispisujemo sve cvorove desno od korena */
     ispisi_stablo_prefiksno(koren->desno);
283   }

285 }

287 /* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo
```

```
       podstablo - Desno postablo - Koren) */
289 void ispisi_stablo_postfiksno(Cvor * koren)
    {
291   /* Ako stablo nije prazno */
      if (koren != NULL) {
293
        /* Prvo ispisujemo sve cvorove levo od korena */
295     ispisi_stablo_postfiksno(koren->levo);

297     /* Ispisujemo sve cvorove desno od korena */
        ispisi_stablo_postfiksno(koren->desno);
299
        /* Na kraju ispisujemo vrednost u korenu */
301     printf("%d ", koren->broj);
      }
303
    }
305

307 /*
     k) Funkcija koja oslobadja memoriju zauzetu stablom.
309  */

311 void oslobodi_stablo(Cvor ** adresa_korena)
    {
313   /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
      if (*adresa_korena == NULL)
315     return;

317   /* U suprotnom rekurzivno oslobadjamo memoriju koje zauzima najpre
    levo, a zatim i desno podstablo */
319   oslobodi_stablo(&(*adresa_korena)->levo);
      oslobodi_stablo(&(*adresa_korena)->desno);
321
      /* Oslobadjamo memoriju koju zauzima koren */
323   free(*adresa_korena);

325   /* I proglasavamo stablo praznim */
      *adresa_korena = NULL;
327 }

329 int main()
    {
331   Cvor * koren;
      int n;
333   Cvor * trazeni_cvor;

335   /* Proglasavamo stablo praznim */
      koren = NULL;
337
      /* Dodajemo vrednosti u stablo */
339   printf("Unesite brojeve (CTRL+D za kraj unosa): ");
```

```
  while (scanf("%d", &n) != EOF) {
341   dodaj_u_stablo(&koren, n);
  }
343 /* Generisemo trazene ispise: */
  printf("\nInfiksni ispis: ");
345 ispisi_stablo_infiksno(koren);
  printf("\nPrefiksni ispis: ");
347 ispisi_stablo_prefiksno(koren);
  printf("\nPostfiksni ispis: ");
349 ispisi_stablo_postfiksno(koren);

351

  /* Demonstriramo rad funkcije za pretragu */
353 printf("\nTrazi se broj: ");
  scanf("%d", &n);
355 trazeni_cvor = pretrazi_stablo(koren, n);
  if (trazeni_cvor == NULL)
357   printf("Broj se ne nalazi u stablu!\n");
  else
359   printf("Broj se nalazi u stablu!\n");

361

  /* Demonstriramo rad funkcije za brisanje */
363 printf("Brise se broj: ");
  scanf("%d", &n);
365 obrisi_element(&koren, n);
  printf("Rezultujuce stablo: ");
367 ispisi_stablo_infiksno(koren);
  printf("\n");
369
  /* Oslobadjamo memoriju zauzetu stablom */
371 oslobodi_stablo(&koren);

  /* Prekidamo sa programom */
373 return 0;

375
}
```

Rešenje 4.16

```
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;
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* 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;
65      return;
    }
67
    /* U suprotnom trazimo odgovarajucu poziciju za novu rec */
69
    /* Ako je rec leksikografski manju od reci u korenu ubacujemo
71  je u levo podstablo */
    if (strcmp(rec, (*adresa_korena)->rec) < 0)
73      dodaj_u_stablo(&(*adresa_korena)->levo, rec);
    else
75      /* Ako je rec leksikografski veca od reci u korenu
    ubacujemo je u desno podstablo */
77      if (strcmp(rec, (*adresa_korena)->rec) > 0)
            dodaj_u_stablo(&(*adresa_korena)->desno, rec);
79      else
            /* Ako je rec jednaka reci u korenu, uvecavamo njen
81  broj pojavljivanja */
            (*adresa_korena)->brojac++;
83  }

85
    /* Funkcija koja oslobadja memoriju zauzetu stablom */
87  void oslobodi_stablo(Cvor** adresa_korena)  {

89    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if(*adresa_korena == NULL)
91      return;

93    /* Inace ...*/
    /* Oslobadjamo memoriju zauzetu levim podstablom */
95    oslobodi_stablo(&(*adresa_korena)->levo);
    /* Oslobadjamo memoriju zauzetu desnim podstablom */
97    oslobodi_stablo(&(*adresa_korena)->desno);

99    /* Oslobadjamo memoriju zauzetu korenom */
    free((*adresa_korena)->rec);
101   free(*adresa_korena);

103   /* Proglasavamo stablo praznim */
    *adresa_korena = NULL;
105 }

107

109 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
    sa najvecim brojem pojavljivanja)  */
111 Cvor *nadji_najfrekventniju_rec(Cvor * koren) {

113   Cvor *max, *max_levo, *max_desno;

115   /* Ako je stablo prazno, prekidamo sa pretragom */
```

```
  if (koren == NULL)
117       return NULL;

119   /* Pronalazimo najfrekventniju reci u levom podstablu */
  max_levo = nadji_najfrekventniju_rec(koren->levo);
121
  /* Pronalazimo najfrekventniju reci u desnom podstablu */
123   max_desno = nadji_najfrekventniju_rec(koren->desno);

125   /* Trazimo maksimum vrednosti pojavljivanja reci iz
  levog podstabla, korena i desnog podstabla */
127   max = koren;
  if (max_levo != NULL && max_levo->brojac > max->brojac)
129       max = max_levo;
  if (max_desno != NULL && max_desno->brojac > max->brojac)
131       max = max_desno;

133   /* Vracamo adresu cvora sa najvecim brojacem */
  return max;
135 }


137
/* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
139 pracene brojem pojavljivanja */
void prikazi_stablo(Cvor * koren) {
141
  /* Ako je stablo prazno, zavrsavamo sa ispisom */
143   if (koren == NULL)
      return;
145
  /* Zbog leksikografskog poretka, prvo ispisujemo sve reci iz
147 levog podstabla */
  prikazi_stablo(koren->levo);
149   /* Zatim ispisujemo rec iz korena */
  printf("%s: %d\n", koren->rec, koren->brojac);
151   /* I nastavljamo sa ispisom reci iz desnog podstabla */
  prikazi_stablo(koren->desno);
153 }

155 /* Funkcija ucitava sledecu rec iz zadate datoteke i upisuje je
  u niz rec. Maksimalna duzina reci je odredjena argumentom max.
157   Funkcija vraca EOF ako nema vise reci ili 0 u suprotnom.
  Rec je niz malih ili velikih slova.*/
159 int procitaj_rec(FILE * f, char rec[], int max) {
  /* karakter koji citamo */
161   int c;
  /* indeks pozicije na koju se smesta procitani karakter */
163   int i = 0;

165   /* Sve dok ima mesta za jos jedan karakter u nizu
      i dokle god nismo stigli do kraja datoteke... */
167   while (i < max - 1 && (c = fgetc(f)) != EOF) {
```

```
        /* Proveravamo da li je procitani karakter slovo */
169     if (isalpha(c))
            /* Ako jeste, smestamo ga u niz  - pritom vrsimo
171 konverziju u mala slova jer program treba da bude neosetljiv na
    razliku izmedju malih i velikih slova */
173         rec[i++] = tolower(c);
        else
175         /* Ako nije, proveravamo da li smo procitali barem jedno
    slovo nove rece  */
177         /* Ako jesmo prekidamo sa citanjem */
            if (i > 0)
179             break;

181         /*  U suprotnom idemo na sledecu iteraciju */
    }
183
    /* Dodajemo na rec terminirajucu nulu */
185 rec[i] = '\0';

187 /* Vracamo 0 ako smo procitali rec, EOF u suprotnom */
    return i > 0 ? 0 : EOF;
189 }


191
int main(int argc, char **argv)  {
193   Cvor *koren = NULL, *max;
    FILE *f;
195 char rec[MAX];

197 /* Proveravamo da li je navedeno ime datoteke prilikom
    pokretanja programa */
199 if (argc < 2)  {
        fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
201     exit(EXIT_FAILURE);
    }
203
    /* Otvaramo datoteku iz koje citamo reci */
205 if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "fopen() greska pri otvaranju %s\n", argv[1]);
207     exit(EXIT_FAILURE);
    }
209
    /* Ucitavamo reci iz datoteke i smestamo u binarno stablo pretrage.
211 */
    while (procitaj_rec(f, rec, MAX) != EOF)
213     dodaj_u_stablo(&koren, rec);

215 /* Posto smo zavrsili sa citanjem reci zatvaramo datoteku */
    fclose(f);
217
    /* Prikazujemo sve reci iz teksta i brojeve njihovih
219 pojavljivanja. */
```

```
  prikazi_stablo(koren);

  /* Pronalazimo najfrekventniju rec */
  max = nadji_najfrekventniju_rec(koren);

  /* Ako takve reci nema... */
  if (max== NULL)
      /* Ispisujemo odgovarajuce obavestenje */
      printf("U tekstu nema reci!\n");
  else
      /* Inace, ispisujemo broj pojavljivanja reci */
      printf("Najcesca rec: %s (pojavljuje se %d puta)\n", max->rec,
max->brojac);


  /* Oslobadjamo dinamicki alociran prostor za stablo */
  oslobodi_stablo(&koren);

  /* Zavrsavamo sa programom */
  return 0;
}
```

Rešenje 4.17

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

#define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
#define MAX_IME_I_PREZIME 100

/* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
   broj telefona i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
  char ime_i_prezime[MAX_IME_I_PREZIME];
  char telefon[MAX_CIFARA];
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
  /* Alociramo memoriju za novi cvor */
  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
  if (novi_cvor == NULL)
    return NULL;

  /* Inicijalizujemo polja u novom cvoru */
```

```
28    strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
      strcpy(novi_cvor->telefon, telefon);
30    novi_cvor->levo = NULL;
      novi_cvor->desno = NULL;
32
      /* Vracamo adresu novog cvora */
34    return novi_cvor;
  }
36
  /* Funkcija koja proverava uspesnost kreiranja novog cvora
38    stabla */
  void proveri_alokaciju(Cvor * novi_cvor)
40{
      /* Ukoliko je cvor neuspesno kreiran */
42    if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvrsavanje
44         programa */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
46      exit(EXIT_FAILURE);
      }
48}

50/* Funkcija koja dodaje novu osobu i njen broj telefona u
      stablo. */
52void
  dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
54                 char *telefon)
  {
56    /* Ako je stablo prazno */
      if (*adresa_korena == NULL) {
58      /* Kreiramo novi cvor */
        Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
60      proveri_alokaciju(novi);
        /* i proglasavamo ga korenom stabla */
62      *adresa_korena = novi;
        return;
64    }
      /* U suprotnom trazimo odgovarajucu poziciju za novi unos */
66    /* Kako pretragu treba vrsiti po imenu i prezimenu, stablo
         treba da bude pretrazivacko po ovom polju */
68    /* Ako je zadato ime i prezime leksikografski manje od imena i
         prezimena sadrzanog u korenu, podatke dodajemo u levo
70       podstablo */
      if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72        < 0)
        dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                     telefon);
      else
76      /* Ako je zadato ime i prezime leksikografski vece od imena
           i prezimena sadrzanog u korenu, podatke kodajemo u desno
78         podstablo */
      if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
```

```
80      dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
                       telefon);
82  }

84  /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** adresa_korena)
86  {
      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
88    if (*adresa_korena == NULL)
        return;
90    /* Inace ... */
      /* Oslobadjamo memoriju zauzetu levim podstablom */
92    oslobodi_stablo(&(*adresa_korena)->levo);
      /* Oslobadjamo memoriju zauzetu desnim podstablom */
94    oslobodi_stablo(&(*adresa_korena)->desno);
      /* Oslobadjamo memoriju zauzetu korenom */
96    free(*adresa_korena);
      /* Proglasavamo stablo praznim */
98    *adresa_korena = NULL;
    }
100
    /* Funkcija koja ispisuje imenik u leksikografskom poretku */
102 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
       koristiti za proveru da li je stablo lepo kreirano ili ne */
104 void prikazi_stablo(Cvor * koren)
    {
106   /* Ako je stablo prazno, zavrsavamo sa ispisom */
      if (koren == NULL)
108     return;
      /* Zbog leksikografskog poretka, prvo ispisujemo podatke iz
110      levog podstabla */
      prikazi_stablo(koren->levo);
112   /* Zatim ispisujemo podatke iz korena */
      printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
114   /* I nastavljamo sa ispisom podataka iz desnog podstabla */
      prikazi_stablo(koren->desno);
116 }

118 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
       ime i prezime i broj telefona u odgovarajuce nizove.
120    Maksimalna duzina imena i prezimena odredjena je konstantom
       MAX_IME_PREZIME, a maksimalna duzina broja telefona
122    konstantom MAX_CIFARA. Funkcija vraca EOF ako nema vise
       kontakata ili 0 u suprotnom. */
124 int procitaj_kontakt(FILE * f, char *ime_i_prezime,
                         char *telefon)
126 {
      int c;
128   int i = 0;
      /* Linije datoteke koje obradjujemo su formata Ime Prezime
130      BrojTelefona */
      /* Preskacemo eventualne praznine sa pocetka linije datoteke */
```

```
132   while ((c = fgetc(f)) != EOF && isspace(c));
      /* Prvo procitano slovo upisujemo u ime i prezime */
134   if (!feof(f))
        ime_i_prezime[i++] = c;
136   /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
         cifre, tako da cemo citanje forsirati sve dok ne naidjemo na
138      cifru. Pri tom cemo voditi racuna da li ima dovoljno mesta za
         smestanje procitanog karaktera i da slucajno ne dodjemo do
140      kraja datoteke */
      while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
142     if (!isdigit(c))
          ime_i_prezime[i++] = c;
144     else if (i > 0)
          break;
146   }
      /* Upisujemo terminirajucu nulu na mesto poslednjeg procitanog
148      blanko karaktera */
      ime_i_prezime[--i] = '\0';
150   /* I pocinjemo sa citanjem broja telefona */
      i = 0;
152   /* Upisujemo cifru koju smo vec procitali */
      telefon[i++] = c;
154   /* I citamo peostale cifre - naznaka kraja ce nam biti pojava
         karaktera cije prisustvo nije dozvoljeno u broju telefona */
156   while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
        if (c == '/' || c == '-' || isdigit(c))
158       telefon[i++] = c;
        else
160       break;
      /* Upisujemo terminirajucu nulu */
162   telefon[i] = '\0';
      /* Vracamo 0 ako smo procitali kontakt, EOF u suprotnom */
164   return !feof(f) ? 0 : EOF;
    }
166
    /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
168     prezimenom */
    Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
170 {
      /* Ako je imenik prazan, zavrsavamo sa pretragom */
172   if (koren == NULL)
        return NULL;
174   /* Ako je trazeno ime i prezime sadrzano u korenu, takodje
         zavrsavamo sa pretragom */
176   if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
        return koren;
178   /* Ako je zadato ime i prezime leksikografski manje od
         vrednosti u korenu pretragu nastavljamo levo */
180   if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
        return pretrazi_imenik(koren->levo, ime_i_prezime);
182   else
        /* u suprotnom, pretragu nastavljamo desno */
```

```
184     return pretrazi_imenik(koren->desno, ime_i_prezime);
}
186
int main(int argc, char **argv)
188 {
  char ime_datoteke[MAX_IME_DATOTEKE];
190   Cvor *koren = NULL;
  Cvor *trazeni;
192   FILE *f;
  char ime_i_prezime[MAX_IME_I_PREZIME];
194   char telefon[MAX_CIFARA];
  char c;
196   int i;

198   /* Ucitavamo ime datoteke i pripremamo je za citanje */
  printf("Unesite ime datoteke: ");
200   scanf("%s", ime_datoteke);
  if ((f = fopen(ime_datoteke, "r")) == NULL) {
202     fprintf(stderr, "fopen() greska prilikom otvaranja
%s\n", ime_datoteke);
204     exit(EXIT_FAILURE);
  }
206
  /* Ucitavamo podatke iz datoteke i smestamo kontakte u binarno
208      stablo pretrage. */
  while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
210     dodaj_u_stablo(&koren, ime_i_prezime, telefon);

212   /* Posto smo zavrsili sa citanjem podataka zatvaramo datoteku */
  fclose(f);
214
  /* Omogucavamo pretragu imenika */
216   while (1) {
    /* Ucitavamo ime i prezime */
218     printf("Unesite ime i prezime: ");
    i = 0;
220     while ((c = getchar()) != '\n')
      ime_i_prezime[i++] = c;
222     ime_i_prezime[i] = '\0';
    /* Ako je korisnik uneo naznaku za kraj pretrage,
224        obustavljamo funkcionalnost */
    if (strcmp(ime_i_prezime, "KRAJ") == 0)
226       break;
    /* Inace, ispisujemo rezultat pretrage */
228     trazeni = pretrazi_imenik(koren, ime_i_prezime);
    if (trazeni == NULL)
230       printf("Broj nije u imeniku!\n");
    else
232       printf("Broj je: %s \n", trazeni->telefon);
  }
234
  /* Oslobadjamo memoriju zauzetu imenikom */
```

```
236   oslobodi_stablo(&koren);

238   /* Zavrsavamo sa programom */
      return 0;
240 }
```

Rešenje 4.18

```
#include<stdio.h>
2 #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 drveta */
22 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                     double matematika, double jezik)
24 {

26   /* Alociramo memoriju za novi cvor */
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
28   if (novi == NULL)
       return NULL;
30
     /* Inicijalizujemo polja strukture */
32   strcpy(novi->ime, ime);
     strcpy(novi->prezime, prezime);
34   novi->uspeh = uspeh;
     novi->matematika = matematika;
36   novi->jezik = jezik;
     novi->levo = NULL;
38   novi->desno = NULL;

40   /* Vracamo adresu kreiranog cvora */
     return novi;
42 }
```

```
/* Funkcija kojom se proverava uspesnost alociranja memorije */
void proveri_alokaciju(Cvor * novi_cvor)
{

  /* Ako alokacije nije uspesna */
  if (novi_cvor == NULL) {
    /* Ispisujemo poruku i prekidamo sa izvrsavanjem */
    fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
  }

}

/* Funkcija kojom se oslobadja memorija zauzeta stablom */
void oslobodi_stablo(Cvor ** koren)
{

  /* Ako je stablo prazno, nema potrebe za oslobadjanjem
     memorije */
  if (*koren == NULL)
    return;

  /* oslobadjamo memoriju zauzetu levim podstablom */
  oslobodi_stablo(&(*koren)->levo);

  /* oslobadjamo memoriju zauzetu desnim podstablom */
  oslobodi_stablo(&(*koren)->desno);

  /* oslobadjamo memoriju zauzetu korenom */
  free(*koren);

  /* proglasavamo stablo praznim */
  *koren = NULL;
}

/* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                    double uspeh, double matematika,
                    double jezik)
{

  /* Ako je stablo prazno */
  if (*koren == NULL) {
    /* Kreiramo novi cvor */
    Cvor *novi =
        napravi_cvor(ime, prezime, uspeh, matematika, jezik);
    proveri_alokaciju(novi);

    /* I proglasavamo ga korenom stabla */
    *koren = novi;

    return;
```

```
96    }

98    /* Inace, dodajemo cvor u stablo tako da bude sortiran po
         ukupnom broju poena */
100   if (uspeh + matematika + jezik >
          (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
102     dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                       matematika, jezik);
104   else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
106                    matematika, jezik);
  }
108

110 /* Funkcija ispisuje sadrzaj stabla - ukoliko je vrednost
       argumenta polozili jednaka 0 ispisuju se informacije o
112    ucenicima koji nisu polozili prijemni, a ako je vrednost
       argumenta razlicita od nule, ispisuju se informacije o
114    ucenicima koji su polozili prijemni */
  void stampaj(Cvor * koren, int polozili)
116 {

118   /* Stablo je prazno - prekidamo sa ispisom */
      if (koren == NULL)
120     return;

122   /* Stampamo informacije iz levog podstabla */
      stampaj(koren->levo, polozili);
124
      /* Stampamo informacije iz korenog cvora */
126   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);
      else if (!polozili && koren->matematika + koren->jezik < 10)
132     printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
               koren->prezime, koren->uspeh, koren->matematika,
134            koren->jezik,
               koren->uspeh + koren->matematika + koren->jezik);
136
      /* Stampamo informacije iz desnog podstabla */
138   stampaj(koren->desno, polozili);
  }
140

142 /* Funkcija koja odredjuje koliko studenata nije polozilo
       prijemni ispit */
144 int nisu_polozili(Cvor * koren)
  {
146
      /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
```

```
148   if (koren == NULL)
        return 0;
150
      /* Pretragu vrsimo i u levom i u desnom podstablu - ako uslov
152      za polaganje nije ispunjen za koreni cvor, broj studenata
         uvecavamo za 1 */
154   if (koren->matematika + koren->jezik < 10)
        return 1 + nisu_polozili(koren->levo) +
156         nisu_polozili(koren->desno);

158   return nisu_polozili(koren->levo) +
          nisu_polozili(koren->desno);
160 }

162 int main(int argc, char **argv)
    {
164
      FILE *in;
166   Cvor *koren;
      char ime[MAX], prezime[MAX];
168   double uspeh, matematika, jezik;

170   /* Otvaramo datoteku sa rezultatima sa prijemnog za citanje */
      in = fopen("prijemni.txt", "r");
172   if (in == NULL) {
        fprintf(stderr, "Greska prilikom citanja podataka!\n");
174     exit(EXIT_FAILURE);
      }
176
      /* Citamo podatke i dodajemo ih u stablo */
178   koren = NULL;
      while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
180                 &matematika, &jezik) != EOF) {
        dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika,
182                    jezik);
      }
184
      /* Zatvaramo datoteku */
186   fclose(in);

188   /* Stampamo prvo podatke o ucenicima koji su polozili prijemni
       */
190   stampaj(koren, 1);

192   /* Liniju iscrtavamo samo ako postoje ucenici koji nisu
         polozili prijemni */
194   if (nisu_polozili(koren) != 0)
        printf("-------------------------------------\n");
196
      /* Stampamo podatke o ucenicima koji nisu polozili prijemni */
198   stampaj(koren, 0);
```

```
200   /* Oslobadjamo memoriju zauzetu stablom */
      oslobodi_stablo(&koren);
202
      /* Zavrsavamo sa programom */
204   return 0;

206 }
```

Rešenje 4.19

```
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 */
11 typedef struct cvor_stabla {
     char ime[MAX_NISKA];
13   char prezime[MAX_NISKA];
     int dan;
15   int mesec;
     int godina;
17   struct cvor_stabla *levo;
     struct cvor_stabla *desno;
19 } Cvor;

21 /* Funkcija koja kreira novi cvor */
   Cvor *napravi_cvor(char ime[], char prezime[], int dan,
23                    int mesec, int godina)
   {
25
     /* Alociramo memoriju */
27   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
     if (novi == NULL)
29     return NULL;

31   /* Inicijalizujemo polja strukture */
     strcpy(novi->ime, ime);
33   strcpy(novi->prezime, prezime);
     novi->dan = dan;
35   novi->mesec = mesec;
     novi->godina = godina;
37   novi->levo = NULL;
     novi->desno = NULL;
39
     /* Vracamo adresu novog cvora */
41   return novi;
```

```
}

/* 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)
   */
  if ((*koren)->levo)
    oslobodi_stablo(&(*koren)->levo);

  /* Oslobadjamo memoriju zauzetu desnim podstablom (ako
     postoji) */
  if ((*koren)->desno)
    oslobodi_stablo(&(*koren)->desno);

  /* Oslobadjamo memoriju zauzetu korenom */
  free(*koren);

  /* Proglasavamo stablo praznim */
  *koren = NULL;
}

/* Funkcija koja dodaje novi cvor u stablo - stablo treba da
   bude uredjeno po datumu */
void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                    int dan, int mesec, int godina)
{

  /* Ako je stablo prazno */
  if (*koren == NULL) {

    /* Kreiramo novi cvor */
    Cvor *novi_cvor =
        napravi_cvor(ime, prezime, dan, mesec, godina);
    proveri_alokaciju(novi_cvor);
```

```
95      /* I proglasavamo ga korenom */
        *koren = novi_cvor;
97
        return;
99    }

101   /* Kako se ne unosi godina za pretragu, stablo uredjujemo samo
         po mesecu (i danu u okviru istog meseca) */
103   if (mesec < (*koren)->mesec)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
105                    godina);
      else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
107     dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec,
                       godina);
109   else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec,
111                    godina);
    }
113
    /* Funkcija vrsi pretragu stabla i vraca cvor sa trazenim
115    datumom (null ako takav ne postoji). u promenljivu pom ce
       biti smesten prvi datum (dan i mesec) veci od trazenog
117    datuma (null ako takav ne postoji)

119  */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)
121 {
      /* Stablo je prazno, obustavljamo pretragu */
123   if (koren == NULL)
        return NULL;
125
      /* Nasli smo trazeni datum u stablu */
127   if (koren->dan == dan && koren->mesec == mesec)
        return koren;
129
      /* Ako je mesec trazenog datuma manji od meseca sadrzanog u
131       korenu ili ako su meseci isti ali je dan trazenog datuma
         manji od aktuelnog datuma, pretrazujemo levo podstablo -
133      pre toga svakako proveravamo da li leva grana postoji - ako
         ne postoji treba da vratimo prvi sledeci, a to je bas
135      vrednost uocenog korena */
      if (mesec < koren->mesec
137         || (mesec == koren->mesec && dan < koren->dan)) {
        if (koren->levo == NULL)
139       return koren;
        else
141       return pretrazi(koren->levo, dan, mesec);
      }
143
      /* inace, nastavljamo pretragu u desnom delu */
145   return pretrazi(koren->desno, dan, mesec);
```

```
}

int main(int argc, char **argv)
{
  FILE *in;
  Cvor *koren;
  Cvor *slavljenik;
  char ime[MAX_NISKA], prezime[MAX_NISKA];
  int dan, mesec, godina;

  /* Proveravamo da li je zadato ime ulazne datoteke */
  if (argc < 2) {
    /* Ako nije, ispisujemo poruku i prekidamo sa izvrsavanjem
       programa */
    printf("Nedostaje ime ulazne datoteke!\n");
    return 0;
  }

  /* Inace, pripremamo datoteku za citanje */
  in = fopen(argv[1], "r");
  if (in == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
    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;
    }
```

```
199     /* Slucaj kada smo pronasli prave podatke */
        if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
201       printf("Slavljenik: %s %s\n", slavljenik->ime,
                slavljenik->prezime);
203       continue;
        }
205
        /* Slucaj kada smo pronasli podatke o prvom sledecem
207        rodjendanu */
        printf("Slavljenik: %s %s %d.%d.\n", slavljenik->ime,
209             slavljenik->prezime, slavljenik->dan,
                slavljenik->mesec);
211   }

213   /* Oslobadjamo memoriju zauzetu stablom */
      oslobodi_stablo(&koren);
215
      /* Prekidamo sa izvrsavanjem programa */
217   return 0;
    }
```

Rešenje 4.20

```
    #ifndef __STABLA_H__
2   #define __STABLA_H__ 1

4   /* Struktura kojom se predstavlja cvor binarnog pretrazivackog stabla
    */
6   typedef struct cvor{
      int broj;
8     struct cvor* levo, *desno;
    }Cvor;
10
    /* Funkcija koja alocira memoriju za novi cvor stabla,
12  inicijalizuje polja strukture i vraca pokazivac na novi cvor */
    Cvor* napravi_cvor(int broj);
14
    /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla. */
16  void proveri_alokaciju( Cvor* novi_cvor);

18  /* Funkcija koja dodaje zadati broj u stablo */
    void dodaj_u_stablo(Cvor** adresa_korena, int broj);
20
    /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
22  Cvor* pretrazi_stablo(Cvor * koren, int broj);

24  /* Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost
    u stablu */
26  Cvor * pronadji_najmanji(Cvor * koren);
```

```c
/* 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 */
```

```
37    if (*koren == NULL) {
        /* Kreiramo novi cvor */
39      Cvor *novi = napravi_cvor(broj);
        proveri_alokaciju(novi);
41      /* i proglasavamo ga korenom stabla */
        *koren = novi;
43      return;
      }
45    /* U suprotnom trazimo odgovarajucu poziciju za novi broj */
      /* Ako je broj manji od vrednosti sadrzane u korenu, ubacujemo
47       ga u levo podstablo */
      if (broj < (*koren)->broj)
49      dodaj_u_stablo(&(*koren)->levo, broj);
      else
51      /* Inace, ubacujemo broj u desno podstablo */
        dodaj_u_stablo(&(*koren)->desno, broj);
53  }

55  /* Funkcija koja oslobadja memoriju zauzetu stablom */
    void oslobodi_stablo(Cvor ** koren)
57  {
      /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
59    if (*koren == NULL)
        return;
61    /* Inace ... */
      /* Oslobadjamo memoriju zauzetu levom podstablom */
63    if ((*koren)->levo)
        oslobodi_stablo(&(*koren)->levo);
65    /* Oslobadjamo memoriju zauzetu desnom podstablom */
      if ((*koren)->desno)
67      oslobodi_stablo(&(*koren)->desno);
      /* Oslobadjamo memoriju zauzetu korenom */
69    free(*koren);
      /* Proglasavamo stablo praznim */
71    *koren = NULL;
    }
73
    Cvor *pronadji_najmanji(Cvor * koren)
75  {
      /* ako je stablo prazno, prekidamo pretragu */
77    if (koren == NULL)
        return NULL;
79    /* vrednosti koje su manje od vrednosti u korenu stabla nalaze
         se levo od njega */
81    /* ako je koren cvor koji nema levo podstablo, onda on sadrzi
         najmanju vrednost */
83    if (koren->levo == NULL)
        return koren;
85    /* inace, pretragu treba nastaviti u levom podstablu */
      return pronadji_najmanji(koren->levo);
87  }
```

```
Cvor *pronadji_najveci(Cvor * koren)
{
  /* ako je stablo prazno, prekidamo pretragu */
  if (koren == NULL)
    return NULL;
  /* vrednosti koje su vece od vrednosti u korenu stabla nalaze
     se desno od njega */
  /* ako je koren cvor koji nema desno podstablo, onda on sadrzi
     najvecu vrednost */
  if (koren->desno == NULL)
    return koren;
  /* inace, pretragu treba nastaviti u desnom podstablu */
  return pronadji_najveci(koren->desno);
}

/* Funkcija brise element iz stabla ciji je broj upravo jednak
   broju n. Funkcija azurira koren stabla u pozivajucoj
   funkciji, jer u ovoj funkciji koren moze biti promenjen u
   funkciji. */
void obrisi_element(Cvor ** adresa_korena, int n)
{
  Cvor *pomocni = NULL;
  /* Izlaz iz rekurzije: ako je stablo prazno, nema sta da se
     brise */
  if (*adresa_korena == NULL)
    return;
  /* Ako je vrednost broja veca od vrednosti u korenu stablua,
     tada se broj eventualno nalazi u desnom podstablu, pa treba
     rekurzivno primeniti postupak na desno podstablo. Koren
     ovako modifikovanog stabla je nepromenjen. */
  if ((*adresa_korena)->broj < n) {
    obrisi_element(&(*adresa_korena)->desno, n);
    return;
  }
  /* Ako je vrednost broja manja od vrednosti korena, tada se
     broj eventualno nalazi u levom podstablu, pa treba
     rekurzivno primeniti postupak na levo podstablo. Koren
     ovako modifikovanog stabla je nepromenjen. */
  if ((*adresa_korena)->broj > n) {
    obrisi_element(&(*adresa_korena)->levo, n);
    return;
  }
  /* Slede podslucajevi vezani za slucaj kada je vrednost u
     korenu jednaka broju koji se brise (tj. slucaj kada treba
     obrisati koren) */
  /* Ako koren nema sinova, tada se on prosto brise, i rezultat
     je prazno stablo (vracamo NULL) */
  if ((*adresa_korena)->levo == NULL
      && (*adresa_korena)->desno == NULL) {
    free(*adresa_korena);
    *adresa_korena = NULL;
    return;
```

```
141   }
      /* Ako koren ima samo levog sina, tada se brisanje vrsi tako
143      sto obrisemo koren, a novi koren postaje levo sin */
      if ((*adresa_korena)->levo != NULL
145       && (*adresa_korena)->desno == NULL) {
        pomocni = (*adresa_korena)->levo;
147     free(*adresa_korena);
        *adresa_korena = pomocni;
149     return;
      }
151   /* Ako koren ima samo desnog sina, tada se brisanje vrsi tako
         sto obrisemo koren, a novi koren postaje desno sin */
153   if ((*adresa_korena)->desno != NULL
        && (*adresa_korena)->levo == NULL) {
155     pomocni = (*adresa_korena)->desno;
        free(*adresa_korena);
157     *adresa_korena = pomocni;
        return;
159   }
      /* Slucaj kada koren ima oba sina. Tada se brisanje vrsi na
161      sledeci nacin: - najpre se potrazi sledbenik korena (u
         smislu poretka) u stablu. To je upravo po vrednosti
163      najmanji cvor u desnom podstablu. On se moze pronaci npr.
         funkcijom pronadji_najmanji(). - Nakon toga se u koren
165      smesti vrednost tog cvora, a u taj cvor se smesti vrednost
         korena (tj. broj koji se brise). - Onda se prosto
167      rekurzivno pozove funkcija za brisanje na desno podstablo.
         S obzirom da u njemu treba obrisati najmanji element, a on
169      definitivno ima najvise jednog potomka, jasno je da ce
         njegovo brisanje biti obavljeno na jedan od jednostavnijih
171      nacina koji su gore opisani. */
      pomocni = pronadji_najmanji((*adresa_korena)->desno);
173   (*adresa_korena)->broj = pomocni->broj;
      pomocni->broj = n;
175   obrisi_element(&(*adresa_korena)->desno, n);
    }
177
    /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje
179    elemente u rastucem poretku) */
    void prikazi_stablo(Cvor * koren)
181 {
      /* izlaz iz rekurzije */
183   if (koren == NULL)
        return;
185   prikazi_stablo(koren->levo);
      printf("%d ", koren->broj);
187   prikazi_stablo(koren->desno);
    }
189
    Cvor *pretrazi_stablo(Cvor * koren, int broj)
191 {
      /* ako je stablo prazno, vrednost se sigurno ne nalazi u njemu
```

```
193     */
  if (koren == NULL)
195     return NULL;
  /* ako je trazena vrednost sadrazana u korenu */
197 if (koren->broj == broj) {
    /* prekidamo pretragu */
199   return koren;
  }
201 /* inace, ako je broj manji od vrednosti sadrzane u korenu */
  if (broj < koren->broj)
203   /* pretragu nastavljamo u levom podstablu */
    return pretrazi_stablo(koren->levo, broj);
205 else
    /* u suprotnom, pretragu nastavljamo u desnom podstablu */
207   return pretrazi_stablo(koren->desno, broj);
}
```

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

4 /* Ukljucujemo biblioteku za rad sa stablima - pogledati uvodni
     zadatak ove glave */
6 #include "stabla.h"

8
  /* Funkcija koja proverava da li su dva stabla koja sadrze cele
10    brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
      odnosno 0 ako nisu */
12 int identitet(Cvor * koren1, Cvor * koren2)
  {
14   /* Ako su oba stabla prazna, jednaka su */
    if (koren1 == NULL && koren2 == NULL)
16     return 1;

18   /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
        jednaka */
20   if (koren1 == NULL || koren2 == NULL)
      return 0;
22
    /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
24     vrednosti, mozemo da zakljucimo da se razlikuju */
    if (koren1->broj != koren2->broj)
26     return 0;

28   /* inace, proveravamo da li vazi i jednakost u levih
        podstabala i desnih podstabala */
30   return (identitet(koren1->levo, koren2->levo)
            && identitet(koren1->desno, koren2->desno));
32 }

34 int main()
  {
```

```
36
   int broj;
38 Cvor *koren1, *koren2;

40 koren1 = NULL;
   /* ucitavamo elemente prvog stabla */
42 printf("Prvo stablo: ");
   scanf("%d", &broj);
44 while (broj != 0) {
     dodaj_u_stablo(&koren1, broj);
46   scanf("%d", &broj);
   }
48
   koren2 = NULL;
50 /* ucitavamo elemente drugog stabla */
   printf("Drugo stablo: ");
52 scanf("%d", &broj);
   while (broj != 0) {
54   dodaj_u_stablo(&koren2, broj);
     scanf("%d", &broj);
56 }

58 /* pozivamo funkciju koja ispituje identitet stabala */
   if (identitet(koren1, koren2))
60   printf("Stabla jesu identicna.\n");
   else
62   printf("Stabla nisu identicna.\n");

64 /* oslobadjamo memoriju zauzetu stablima */
   oslobodi_stablo(&koren1);
66 oslobodi_stablo(&koren2);

68 /* zavrsavamo sa radom programa */
   return 0;
70 }
```

Rešenje 4.21

```
#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 kopira
       */
     if (koren == NULL){
12     *duplikat = NULL;
```

```
    return;
  }
  /* Dupliramo koren stabla i postavljamo ga da bude koren novog
stabla  */
  *duplikat =napravi_cvor(koren->broj);
  proveri_alokaciju(*duplikat);

  /* Rekurzivno dupliramo levo podstablo i njegovu adresu cuvamo
  u pokazivacu na levo podstablo korena duplikata. */
  kopiraj_stablo(koren->levo, &(*duplikat)->levo);

  /* Rekurzivno dupliramo desno podstablo i njegovu
  adresu cuvamo u pokazivacu na desno podstablo korena duplikata. */
  kopiraj_stablo(koren->desno, &(*duplikat)->desno);
}

/* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
dobija modifikacijom prvog stabla */
void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2) {
  /* 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);
}

int main() {
  Cvor* koren1;
  Cvor* koren2;
  Cvor *pomocni = NULL;
  int n;


  /* Ucitavamo elemente prvog stabla: */
  koren1=NULL;
  printf("Prvo stablo: ");
  while( scanf("%d",&n) != EOF) {
    dodaj_u_stablo(&koren1,n);
  }

  /* Ucitavamo elemente drugog stabla: */
  koren2=NULL;
  printf("Drugo stablo: ");
  while( scanf("%d",&n) != EOF) {
    dodaj_u_stablo(&koren2,n);
  }

  /* Kreiramo uniju stabala: prvo napravimo kopiju prvog stabla kako bi  mogli da ga iskoristimo i za preostale operacije */
  kopiraj_stablo(koren1,&pomocni);
  kreiraj_uniju(&pomocni, koren2);
  printf("Unija: ");
  prikazi_stablo(pomocni);
  putchar('\n');
  /* Oslobadjamo stablo za rezultatom operacije */
  oslobodi_stablo(&pomocni);


  /* Kreiramo presek stabala: prvo napravimo kopiju prvog stabla kako
  bi mogli da ga iskoristimo i za preostale operacije;
  */
  kopiraj_stablo(koren1,&pomocni);
  kreiraj_presek(&pomocni, koren2);
```

```
  printf("Presek: ");
  prikazi_stablo(pomocni);
  putchar('\n');
  /* Oslobadjamo stablo za rezultatom operacije */
  oslobodi_stablo(&pomocni);


  /* Kreiramo razliku stabala: prvo napravimo
  kopiju prvog stabla kako
  bi mogli da ga iskoristimo i za preostale operacije;
  */
  kopiraj_stablo(koren1,&pomocni);
  kreiraj_razliku(&pomocni, koren2);
  printf("Razlika: ");
  prikazi_stablo(pomocni);
  putchar('\n');
  /* Oslobadjamo stablo za rezultatom operacije */
  oslobodi_stablo(&pomocni);


  /* Oslobadjamo i polazna stabla */
  oslobodi_stablo(&koren2);
  oslobodi_stablo(&koren1);

  /* Zavrsavamo sa programom */
  return 0;
}
```

Rešenje 4.22

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

/* Ukljucujemo biblioteku za rad sa stablima */
#include "stabla.h"

#define MAX 50

/* Funkcija koja obilazi stablo sa leva na desno i smesta
   vrednosti cvorova u niz. Povratna vrednost funkcije je broj
   vrednosti koje su smestene u niz. */

int kreiraj_niz(Cvor * koren, int a[])
{
  int r, s;

  /* Drvo je prazno - u niz je smesteno 0 elemenata */
  if (koren == NULL)
    return 0;

  /* Dodajemo u niz elemente iz levog podstabla */
```

```
  r = kreiraj_niz(koren->levo, a);

  /* Tekuca vrednost promenljive r je broj elemenata koji su
     upisani u niz i na osnovu nje mozemo odrediti indeks novog
     elementa */

  /* Smestamo vrednost iz korena */
  a[r] = koren->broj;

  /* Dodajemo elemente iz desnog podstabla */
  s = kreiraj_niz(koren->desno, a + r + 1);

  /* Racunamo indeks na koji treba smestiti naredni element */
  return r + s + 1;
}


/* Funkcija sortira niz tako sto najpre elemente niza smesti u
   stablo, a zatim kreira novi niz prolazeci kroz stablo sa
   leva 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 4.23

```
#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)
81  return NULL;

83  /* Zbog prirode pretrazivackog stabla, sigurni smo da su
       vrednosti vece od korena u desnom podstablu */
85
  /* Ako desnog podstabla nema */
87  if (koren->desno == NULL)
    /* Najveca vrednost je koren */
89  return koren;

91  /* Inace, najvecu vrednost trazimo jos desno */
  return najveci_element(koren->desno);
93 }

95 /* f) Funckija koja izracunava dubinu stabla */
  int dubina_stabla(Cvor * koren)
97 {

99  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
101  return 0;

103  /* Izracunavamo dubinu levog podstabla */
  int dubina_levo = dubina_stabla(koren->levo);
105
  /* Izracunavamo dubinu desnog podstabla */
107  int dubina_desno = dubina_stabla(koren->desno);

109  /* dubina stabla odgovara vecoj od dubina podstabala - 1
       dodajemo jer racunamo i koren */
111  return dubina_levo >
       dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
113 }

115 /* g) Funckija koja izracunava broj cvorova na i-tom nivou */
  int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
117 {
  /* ideja je da ste spustamo kroz drvo sve dok ne stignemo do
119    trazenog nivoa */

121  /* Ako nema vise cvorova, ne mozemo da se spustamo niz stablo */
  if (koren == NULL)
123  return 0;

125  /* Ako smo stigli do trazenog nivoa, vracamo 1 - to ce kasnije
       zbog rekurzivnih poziva uvecati broj pojavljivanja za 1 */
127  if (i == 0)
    return 1;
129
  /* inace, spustamo se jedan nivo nize i u levom i u desnom
131    postablu */
```

```
  return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
133       + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
}
135

137 /* h) Funckija koja ispisuje sve elemente na i-tom nivou */
void ispis_nivo(Cvor * koren, int i)
139 {
  /* ideja je slicna ideji iz prethodne funkcije */
141
  /* nema vise cvorova, ne mozemo da se spustamo kroz stablo */
143   if (koren == NULL)
    return;
145
  /* ako smo na trazenom nivou - ispisujemo vrednost */
147   if (i == 0) {
    printf("%d ", koren->broj);
149     return;
  }
151   /* inace, spustamo se jedan nivo nize i u levom i u desnom
     podstablu */
153   ispis_nivo(koren->levo, i - 1);
  ispis_nivo(koren->desno, i - 1);
155 }

157 /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom
   nivou stabla */
159 Cvor *max_nivo(Cvor * koren, int i)
{
161
  /* Ako je stablo prazno, obustavljamo pretragu */
163   if (koren == NULL)
    return NULL;
165
  /* Ako smo na trazenom nivou, takodje prekidamo pretragu */
167   if (i == 0)
    return koren;
169
  /* Pronalazimo maksimum sa i-tog nivoa levog podstabla */
171   Cvor *a = max_nivo(koren->levo, i - 1);

173   /* Pronalazimo maksimum sa i-tog nivoa desnog podstabla */
  Cvor *b = max_nivo(koren->desno, i - 1);
175
  /* Trazimo i vracamo maksimum izracunatih vrednosti */
177   if (a == NULL && b == NULL)
    return NULL;
179   if (a == NULL)
    return b;
181   if (b == NULL)
    return a;
183   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,
           max_nivo(koren, i)->broj);

  printf("zbir na %d. nivou: %d\n", i, zbir_nivo(koren, i));
  printf("zbir elemenata manjih ili jednakih od %d: %d\n", x,
         suma(koren, x));

  /* 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;
}

/* Funckija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
{
  /* Ideja je slicna ideji iz prethodne funkcije */

  /* Nema vise cvorova, 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 */
  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;
```

```
53    /* Prvo izracunavamo dubinu stabla */
      int dubina;
55    dubina = dubina_stabla(koren);

57    /* Ispisujemo nivo po nivo stabla */
      for (i = 0; i < dubina; i++) {
59      printf("%d. nivo: ", i);
        ispisi_nivo(koren, i);
61      printf("\n");
      }
63  }

65  int main(int argc, char **argv)
    {
67    Cvor *koren;
      int broj;
69
      /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
71    koren = NULL;
      while (scanf("%d", &broj) != EOF) {
73      dodaj_u_stablo(&koren, broj);
      }
75
      /* Ispisujemo stablo po nivoima */
77    ispisi_stablo_po_nivoima(koren);

79    /* Oslobadjamo memoriju zauzetu stablom */
      oslobodi_stablo(&koren);
81
      /* Prekidamo izvrsavanje programa */
83    return 0;
    }
```

Rešenje 4.26

Rešenje 4.27

```
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   {

11    /* Dubina praznog stabla je 0 */
      if (koren == NULL)
```

```
13      return 0;

15    /* Izracunavamo dubinu levog podstabla */
      int dubina_levo = dubina_stabla(koren->levo);
17
      /* Izracunavamo dubinu desnog podstabla */
19    int dubina_desno = dubina_stabla(koren->desno);

21    /* Dubina stabla odgovara vecoj od dubina podstabala - 1
         dodajemo jer racunamo i koren */
23    return dubina_levo >
          dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
25  }

27  /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za
       AVL stablo */
29  int avl(Cvor * koren)
    {
31    int dubina_levo, dubina_desno;

33    /* Ako je stablo prazno, zaustavljamo brojanje */
      if (koren == NULL) {
35      return 0;
      }
37
      /* Izracunavamo dubinu levog podstabla korena */
39    dubina_levo = dubina_stabla(koren->levo);

41    /* Izracunavamo dubinu desnog podstabla korena */
      dubina_desno = dubina_stabla(koren->desno);
43
      /* Ako je uslov za AVL stablo ispunjen */
45    if (abs(dubina_desno - dubina_levo) <= 1) {
        /* Racunamo broj avl cvorova u levom i desnom podstablu i
47         uvecavamo za jedan iz razloga sto koren ispunjava uslov */
        return 1 + avl(koren->levo) + avl(koren->desno);
49    } else {
        /* Inace, racunamo samo broj avl cvorova u podstablima */
51      return avl(koren->levo) + avl(koren->desno);
      }
53  }

55  int main(int argc, char **argv)
    {
57    Cvor *koren;
      int broj;
59
      /* Citamo vrednosti sa ulaza i dodajemo ih u stablo */
61    koren = NULL;
      while (scanf("%d", &broj) != EOF) {
63      dodaj_u_stablo(&koren, broj);
      }
```

```
65
  /* Racunamo i ispisujemo broj AVL cvorova */
67  printf("%d\n", avl(koren));

69  /* Oslobadjamo memoriju zauzetu stablom */
  oslobodi_stablo(&koren);
71
  /* Prekidamo izvrsavanje programa */
73  return 0;
}
```

Rešenje 4.28

```
1  #include<stdio.h>
  #include<stdlib.h>
3
  /* Ukljucujemo biblioteku za rad sa stablima */
5  #include "stabla.h"

7  /* Funkcija proverava da li je zadato binarno stablo celih
     pozitivnih brojeva heap. Ideja koju cemo implementirati u
9    osnovi ima pronalazenje maksimalne vrednosti levog i
     maksimalne vrednosti desnog podstabla - ako je vrednost u
11   korenu veca od izracunatih vrednosti uoceni fragment stabla
     zadovoljava uslov za heap. Zato ce funkcija vracati
13   maksimalne vrednosti iz uocenog podstabala ili vrednost -1
     ukoliko zakljucimo da stablo nije heap. */
15 int heap(Cvor * koren)
  {
17
  int max_levo, max_desno;
19
  /* Prazno sablo je heap. */
21  if (koren == NULL) {
    /* posto je 0 najmanji pozitivan broj, moze nam posluziti
23     kao indikator */
    return 0;
25
  }
27  /* Ukoliko je stablo list ... */
  if (koren->levo == NULL && koren->desno == NULL) {
29    /* ... vracamo njegovu vrednost */
    return koren->broj;
31  }

33  /* Proveravamo svojstvo za levo podstablo. */
  max_levo = heap(koren->levo);
35
  /* Proveravamo svojstvo za desno podstablo. */
37  max_desno = heap(koren->desno);
  /* Ako levo ili desno podstablo uocenog cvora nije heap, onda
```

```
39      nije ni celo stablo. */
   if (max_levo == -1 || max_desno == -1) {
41    return -1;
   }
43
   /* U suprotonom proveravamo da li svojstvo vazi za uoceni
45    cvor. */
   if (koren->broj > max_levo && koren->broj > max_desno) {
47    /* ako vazi, vracamo vrednost korena */
      return koren->broj;
49 }

51 /* u suprotnom zakljucujemo da stablo nije heap */
   return -1;
53 }

55 int main(int argc, char **argv)
   {
57   Cvor *koren;
     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
   }
```

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 1

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
 5
 Programiranje
 Matematika
 12345
 dInAmiCnArEc
 Ispit
Izlaz:
 Ispit
 Matematika
 Programiranje
```

Test 2

```
Poziv: ./a.out ulaz.txtž
Sadraj datoteke ulaz.txt:
  2
  maksimalano
  poena
Izlaz:
```

Test 3

```
Poziv: ./a.out ulaz.txt
Problem: datoteka
ulaz.txt ne postoji
Izlaz:
  -1
```

Test 4

```
Poziv: ./a.out
Izlaz:
  -1
```

[Rešenje 5.1]

Zadatak 5.2

Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na n-tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj n, a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test 1

```
Ulaz:
 2 8 10 3 6 14 13 7 4 0
Izlaz:
 20
```

Test 2

```
Ulaz:
 0 50 14 5 2 4 56 8 52 7 1 0
Izlaz:
 50
```

[Rešenje 5.2]

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A, a zatim i elementi matrice A. Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od `50`. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

Test 1

```
Ulaz:
  4 5
   1  2  3  4  5
  -1  2 -3  4 -5
  -5 -4 -3 -2  1
  -1  0  0  0  0
 Izlaz:
  0
```

Test 2

```
Ulaz:
 2 3
 0 0 -5
 1 2 -4
 Izlaz:
 2
```

Test 3

```
Ulaz:
 -2
Izlaz (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 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 2

```
Poziv:   ./a.out
Izlaz  (na stderr):  -1
```

Test 3

```
Poziv:   ./a.out ulaz.txt foo
ulaz.txt:   (ne postoji)
Izlaz (na stderr):    -1
```

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 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 2

```
Datoteka:  3
           1.2 3.2 1.1 4.3
Izlaz:     -1
```

Test 3

```
Datoteka:   (nema datoteke)
Izlaz:     -1
```

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 1

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

Test 2

```
Ulaz:
 2 5 3 6 1 0 4 7 9
Izlaz:
 2
```

Test 3

```
Ulaz:
 0 4 2 5
Izlaz:
 0
```

Test 4

```
Ulaz:
  3
Izlaz:
 0
```

Test 5

```
Ulaz:
  -1 4 5 1 7
Izlaz:
 0
```

[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);
            greska();
        }
```

```
    }

    /* Ucitavamo svih n linija iz datoteke. */
    for(i=0; i<n; i++){
        fscanf(ulaz, "%s", linije[i]);
    }

    /* Ispisujemo u odgovarajucem poretku ucitane linije koje
    zadovoljavaju kriterijum. */
    for(i=n-1; i>=0; i--){
        if(isupper(linije[i][0])){
            printf("%s\n", linije[i]);
        }
    }

    /* Oslobadjamo memoriju koju smo dinamicki alocirali. */
    for(i=0; i<n; i++){
        free(linije[i]);
    }

    free(linije);

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

    /* Zavrsavamo sa programom. */
    return 0;

}
```

Rešenje 5.2

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


int sumirajN (Cvor * koren, int n){
    if(koren==NULL){
            return 0;
    }

    if(n==0){
        return koren->broj;
    }

    return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
}


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;
}
```

```
#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;
}


void oslobodi_stablo(Cvor** adresa_korena)  {
```

```
      /* Prazno stablo i nema sta da se oslobadja */
    if( *adresa_korena == NULL)
        return;

    /* Rekurzivno oslobadjamo najpre levo, a onda i desno podstablo*/
    if( (*adresa_korena)->levo )
        oslobodi_stablo(&(*adresa_korena)->levo);
    if( (*adresa_korena)->desno)
        oslobodi_stablo(&(*adresa_korena)->desno);

    free(*adresa_korena);
    *adresa_korena =NULL;
}


void proveri_alokaciju( Cvor* novi) {
    if( novi == NULL) {
        fprintf(stderr, "Malloc greska za nov cvor!\n");
        exit(EXIT_FAILURE);
    }
}

void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
    /* Postojece stablo je prazno*/
    if( *adresa_korena == NULL){
        Cvor* novi = napravi_cvor(broj);
        proveri_alokaciju(novi);
        *adresa_korena = novi;  /*  Kreirani Cvor novi ce biti od
    sada koren stabla*/
        return;
    }

    /* Brojeve smestamo u uredjeno binarno stablo, pa
    ako je broj koji ubacujemo manji od broja koji je u korenu  */
    if( broj < (*adresa_korena)->broj)
    /* Dodajemo u levo podstablo */
        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 */
    else
        dodaj_u_stablo(&(*adresa_korena)->desno, broj);
}
```

```
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura kojom se predstavlja Cvor drveta */
typedef struct dcvor{
    int broj;
    struct dcvor* levo, *desno;
} Cvor;
```

```
/* Funkcija alocira prostor za novi Cvor drveta, 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;

  if (argc != 3) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
```

```
  }

  if ((f = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  while (fgets(linija, MAX, f) != NULL) {
    p = linija;
    while (1) {
      p = strstr(p, argv[2]);
      if (p == NULL)
  break;
      brojac++;
      p = p + strlen(argv[2]);
    }
  }

  fclose(f);

  printf("%d\n", brojac);

  return 0;
}
```

Rešenje 5.5

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

typedef struct _trougao {
  double xa, ya, xb, yb, xc, yc;
} trougao;

double duzina(double x1, double y1, double x2, double y2) {
  return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

double povrsina(trougao t) {
  double a = duzina(t.xb, t.yb, t.xc, t.yc);
  double b = duzina(t.xa, t.ya, t.xc, t.yc);
  double c = duzina(t.xa, t.ya, t.xb, t.yb);
  double s = (a + b + c) / 2;
  return sqrt(s * (s - a) * (s - b) * (s - c));
}

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;
}
```

```c
#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);

}

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;
}
```

```
#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);
```

```
23
/* Funkcija prikazuje stablo s leva u desno (tj.
25    prikazuje elemente u rastucem poretku) */
void prikazi_stablo(Cvor * koren);
27
/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
     vraca
29    pokazican na njegov koren */
Cvor* ucitaj_stablo();
31
/* Funkcija oslobadja prostor koji je alociran za
33    cvorove stabla. */
void oslobodi_stablo(Cvor **koren);
35
#endif
```

Rešenje 5.6

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

4 int f3(Cvor *koren, int n) {
  if (koren == NULL || n < 0)
6   return 0;
  if (n == 0) {
8   if (koren -> levi == NULL && koren -> desni != NULL)
      return 1;
10  if (koren -> levi != NULL && koren -> desni == NULL)
      return 1;
12  return 0;
  }
14 return f3(koren -> levi, n - 1) + f3(koren -> desni, n - 1);
}
16
int main() {
18 Cvor *koren;
  int n;
20
  scanf("%d", &n);
22 koren = ucitaj_stablo();

24 printf("%d\n", f3(koren, n));

26 oslobodi_stablo(&koren);

28 return 0;
}
```