.

Univerzitet u Beogradu
Matematički fakultet

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

Programiranje 2
Zbirka zadataka sa rešenjima

Beograd, 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.

...

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 opisuje kompleksan broj njegovim realnim i imaginarnim delom.

(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 formatu (npr. broj čiji je realan deo `2`, a imaginarni `-3` ispisati kao $(2 - 3i)$ na standardni izlaz).

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

(e) Napisati funkciju `imaginaran_deo` koja vraća vrednost 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. Program najpre za kompleksan broj $z1$ koji se unosi sa standardnog ulaza ispisuje njegov realni deo, imaginarni deo i moduo. Zatim za naredni kompleksan broj $z2$ koji se unosi sa standardnog ulaza ispisuje njegov konjugovano-kompleksan broj i argument. Na kraju program ispisuje zbir, razliku i proizvod brojeva $z1$ i $z2$.

Primer 1

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

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.

Primer 1

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

Zadatak 1.3 Napisati biblioteku za rad sa polinomima.

(a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20. Uputstvo: *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.

(c) Napisati funkciju koja učitava polinom sa standardnog ulaza. Za polinom se najpre unosi stepen pa njegovi koeficijenti.

(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. Program najpre učitava polinom `p` sa standardnog ulaza i ispisuje ga na standardni izlaz u odgovarajućem obliku. Nakon toga program računa i ispisuje vrednost tog polinoma (zaokruženu na dve decimale) u tački koju unosi korisnik. Potom se unosi polinom `q`, i ispisuju se zbir i proizvod polinoma `p` i `q`. Na kraju se sa standardnog ulaza unosi broj `n`, i ispisuje `n`-ti izvod polinoma `p`.

Primer 1

```
INTERAKCIJA PROGRAMA:
 Unesite polinom (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):
 3 1.2 3.5 2.1 4.2
 Unesite tacku u kojoj racunate vrednost polinoma
 5
 Vrednost polinoma u tacki je 252.20
 Unesite polinom (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):
 2 2.1 0 -3.9
 Zbir polinoma je: 1.20x^3+5.60x^2+2.10x+0.30
 Prozvod polinoma je: 2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38
 Unesite izvod polinoma koji zelite:
 2
 2. izvod prvog polinoma je: 7.20x+7.00
```

Zadatak 1.4 Napisati 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 razlomka.

(d) Napisati funkciju koja vraća odgovarajuću realnu vrednost razlomka (tipa `double`).

(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 vrednosti razlomaka koji su dobijeni kao zbir, razlika, proizvod i količnik razlomka `r1` i recipročne vrednosti razlomka `r2`.

Primer 1

```
Interakcija programa:
 Unesite imenilac i brojilac prvog razlomka: 1 2
 Unesite imenilac i brojilac drugog razlomka: 3 1
 Zbir je 5/6
 Razlika je 1/6
 Zbir je 5/6
 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:
 0xABCDE123
Izlaz:
 10101011110011011110000100100011
```

Zadatak 1.6 Napisati funkcije `count_bits1` i `count_bits2` koje broje bitove sa vrednošću `1` u binarnom zapisu celog broja x. Prebrojavanje bitova ostvariti na dva načina:

(a) formiranjem odgovarajuće maske i njenim pomeranjem

(b) formiranjem odgovarajuće maske i pomeranjem promenljive x.

Napisati program koji testira te funkcije za brojeve koji se zadaju u heksadekasnom formatu sa standardnog ulaza.

Test 1

```
ULAZ:
  0x7F
IZLAZ:
  Broj jedinica u zapisu je
  funkcija count_bits1: 7
  funkcija count_bits2: 7
```

Test 2

```
ULAZ:
  0x80
IZLAZ:
  Broj jedinica u zapisu je
  funkcija count_bits1: 1
  funkcija count_bits2: 1
```

Test 3

```
ULAZ:
  0x00FF00FF
IZLAZ:
  Broj jedinica u zapisu je
  funkcija count_bits1: 16
  funkcija count_bits2: 16
```

Test 4

```
ULAZ:
  0xABCDE123
IZLAZ:
  Broj jedinica u zapisu je
  funkcija count_bits1: 17
  funkcija count_bits2: 17
```

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`, odnosno `najmanji` za brojeve koji se zadaju u heksadekasnom formatu sa standardnog ulaza.

Test 1

```
ULAZ:
  0x7F
IZLAZ:
  Najveci:
  11111110000000000000000000000000
  Najmanji:
  00000000000000000000000001111111
```

Test 2

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

Test 3

```
ULAZ:
  0x00FF00FF
IZLAZ:
  Najveci:
  11111111111111110000000000000000
  Najmanji:
  00000000000000001111111111111111
```

Test 4

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

Zadatak 1.8 Napisati funkcije za rad sa bitovima.

(a) Napisati funkciju 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.

Napisati program koji 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 bit najmanje težine na poziciji nula.*

Test 1

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

Zadatak 1.9 Pod rotiranjem ulevo podrazumeva se 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 ulevo datog celog broja `x`.

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

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

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

Test 1

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

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
```

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:
 2147377146
IZLAZ:
 1
```

Test 3

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

Zadatak 1.12 Napisati funkciju koja broji koliko se puta dve uzastopne jedinice pojavljuju u binarnom zapisu celog neoznačenog broja x. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Tri uzastopne jedinice se broje dva puta.*

Test 1

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

Test 2

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

Test 3

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

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

Primer 1

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

Interakcija programa:
 11
 13
```

Primer 2

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

Interakcija programa:
 1024
 1024
```

Primer 2

```
Poziv: ./a.out 12 12

Interakcija programa:
 12345
 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
```

Zadatak 1.15 Napisati funkciju koja za data dva 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. Napomena: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

Test 1

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

Test 2

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

Test 3

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

1.3 Rekurzija

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

Test 1

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

Test 2

```
Ulaz:
  5 3
Izlaz:
  125
```

Test 3

```
Ulaz:
  9 4
Izlaz:
  6561
```

Zadatak 1.18 Koristeći uzajamnu (posrednu) rekurziju napisati:

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

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

Test 1

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

Test 2

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

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

Primer 1

```
Interakcija programa:
 Unesite n (<= 12):  5
 5! = 120
```

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 napisanu funkciju za vrednosti koeficijenata i prirodan broj n koji se unose sa standardnog ulaza.

Primer 1

```
Interakcija programa:
 Unesite koeficijente   2 3
 Unesite koji clan niza se racuna   5
 F(5) = 61
```

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
```

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

Test 1

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

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. Njegovi elementi se unose sve do unosa kraja ulaza (EOF).

Test 1

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

Test 2

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

Test 3

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

Zadatak 1.24 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. Prvo se unosi dimenzija nizova, a zatim i njihovi elementi. Nizovi neće imati više od 256 elemenata.

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
```

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 broj x i niz a koji se unose sa standardnog ulaza. Prvo se unosi x, a zatim elementi niza sve do unosa kraja ulaza. Niz neće imati više od 256 elemenata.

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
```

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
```

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

Test 1

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

Test 2

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

Test 3

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

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
```

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
```

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

Test 1

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

Test 2

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

Test 3

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

Test 4

```
ULAZ:
  programiranje
IZLAZ:
  ne
```

Test 5

```
ULAZ:
  anavolimilovana
IZLAZ:
  da
```

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

Primer 1

```
INTERAKCIJA PROGRAMA:
 Unesite duzinu permutacije:  3
 1 2 3
 1 3 2
 2 1 3
 2 3 1
 3 1 2
 3 2 1
```

*** Zadatak 1.33** Paskalov trougao sadrži brojeve čije se vrednsti računaju tako što svako polje ima vrednost zbira jednog polja levo i jednog polja iznad. Izuzetak su jedinice na krajevima. Vrednosti brojeva Paskalovog trougla odgovaraju binomnim koeficijentima tj. vrednost polja `(n, k)`, gde je n redni broj hipotenuze, a k redni broj elementa u tom redu (na toj hipotenuzi) odgovara binomnom koeficijentu $\binom{n}{k}$, pri čemu brojanje počinje od nule. Na primer, vrednost polja `(4, 2)` je 6.

```
          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 $\binom{n}{k}$ koristeći osobine Paskalovog trougla.

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

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

Test 1

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

 8
```

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 je predstavljan kompleksan broj sadrzi realan i
   imaginaran deo kompleksnog broja */
typedef struct {
  float real;
  float imag;
} KompleksanBroj;

/* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
   kompleksnog broja i smesta ih u strukturu cija 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 + i y) Ovoj funkciji se
   kompleksan broj prenosi po vrednosti (za ispis nije neophodna
   adresa) */
void ispisi_kompleksan_broj(KompleksanBroj z)
{
  printf("(");
  if (z.real != 0) {
    printf("%.2f", z.real);
    if (z.imag > 0)
      printf(" +");

  }
  if (z.imag != 0)
    printf(" %.2f i ", z.imag);

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

  printf(")");
}
```

```
43 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
   float realan_deo(KompleksanBroj z)
45 {
     return z.real;
47 }

49 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
   float imaginaran_deo(KompleksanBroj z)
51 {
     return z.imag;
53 }

55 /* Funkcija vraca vrednost modula kompleksnog broja koji joj se salje
      kao argument */
57 float moduo(KompleksanBroj z)
   {
59   return sqrt(z.real * z.real + z.imag * z.imag);
   }
61
   /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
63    odgovara kompleksnom broju poslatom kao argument */
   KompleksanBroj konjugovan(KompleksanBroj z)
65 {
     KompleksanBroj z1 = z;
67
     z1.imag *= -1;
69
     return z1;
71 }

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

79   z.real += z2.real;
     z.imag += z2.imag;
81
     return z;
83 }

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

91   z.real -= z2.real;
     z.imag -= z2.imag;
93
     return z;
```

```
95 }
97 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
      argumenata funkcije */
99 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
   {
101   KompleksanBroj z;

103   z.real = z1.real * z2.real - z1.imag * z2.imag;
      z.imag = z1.real * z2.imag + z1.imag * z2.real;
105
      return z;
107 }
109 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
      kao argument */
111 float argument(KompleksanBroj z)
   {
113   return atan2(z.imag, z.real);
   }
115
   int main()
117 {
      /* Deklaracija 2 promenljive tipa KompleksanBroj */
119   KompleksanBroj z1, z2;

121   /* Ucitavanje prvog kompleksnog broja u promenljivu z1, a potom
         njegovo ispisivanje na standardni izlaz */
123   ucitaj_kompleksan_broj(&z1);
      ispisi_kompleksan_broj(z1);
125
      /* Ispisuje se na standardni izlaz realan, imaginaran deo i moduo
127      kompleksnog broja z1 */
      printf("\nrealan_deo: %.f\nimaginaran_deo: %f\nmoduo %f\n",
129          realan_deo(z1), imaginaran_deo(z1), moduo(z1));
      printf("\n");
131
      /* Ucitavanje drugog kompleksnog broja u promenljivu z2, a potom
133      njegovo ispisivanje na standardni izlaz */
      ucitaj_kompleksan_broj(&z2);
135   ispisi_kompleksan_broj(z2);

137   /* Racunanje i ispisivanje konjugovano kompleksan broj od z2 */
      printf("\nNjegov konjugovano kompleksan broj: ");
139   ispisi_kompleksan_broj(konjugovan(z2));
      printf("\n");
141
      /* Sabiranje kompleksnih brojeva */
143   printf("\n");
      ispisi_kompleksan_broj(z1);
145   printf(" + ");
      ispisi_kompleksan_broj(z2);
```

```
147   printf("  =  ");
      ispisi_kompleksan_broj(saberi(z1, z2));
149   printf("\n");

151   /* Oduzimanje kompleksnih brojeva */
      printf("\n");
153   ispisi_kompleksan_broj(z1);
      printf(" - ");
155   ispisi_kompleksan_broj(z2);
      printf("  =  ");
157   ispisi_kompleksan_broj(oduzmi(z1, z2));
      printf("\n");
159
      /* Mnozenje kompleksnih brojeva */
161   printf("\n");
      ispisi_kompleksan_broj(z1);
163   printf(" * ");
      ispisi_kompleksan_broj(z2);
165   printf("  =  ");
      ispisi_kompleksan_broj(mnozi(z1, z2));
167
      /* Testiranje funkcije koja racuna argument kompleksnih brojeva */
169   printf("\n");
      ispisi_kompleksan_broj(z2);
171   printf("\nArgument kompleksnog broja %f\n", argument(z2));

173   return 0;
}
```

Rešenje 1.2

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

7  /* Funkcija ucitava sa standardnog ulaza realan i imaginaran deo
      kompleksnog broja i smesta ih u strukturu cija adresa je argument
9     funkcije */
   void ucitaj_kompleksan_broj(KompleksanBroj * z)
11 {
     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
13   scanf("%f", &z->real);
     scanf("%f", &z->imag);
15 }

17 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
      se salje kao argument u obliku (x + y i) */
19 void ispisi_kompleksan_broj(KompleksanBroj z)
   {
```

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

25      if (z.imag > 0)
          printf(" + %.2f i", z.imag);
27      else if (z.imag < 0)
          printf(" - %.2f i", -z.imag);
29    } else
        printf("%.2f i", z.imag);
31
      if (z.imag == 0 && z.real == 0)
33      printf("0");

35    printf(")");
  }
37
  /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
39 float realan_deo(KompleksanBroj z)
  {
41   return z.real;
  }
43
  /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
45 float imaginaran_deo(KompleksanBroj z)
  {
47   return z.imag;
  }
49
  /* Funkcija vraca vrednost modula kompleksnog broja koji joj se salje
51    kao argument */
  float moduo(KompleksanBroj z)
53 {
     return sqrt(z.real * z.real + z.imag * z.imag);
55 }

57 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
      odgovara kompleksnom broju poslatom kao argument */
59 KompleksanBroj konjugovan(KompleksanBroj z)
  {
61   KompleksanBroj z1 = z;
     z1.imag *= -1;
63   return z1;
  }
65
  /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
67    argumenata funkcije */
  KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
69 {
     KompleksanBroj z = z1;
71
     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. Da bi neki program
   mogao da koristi ove brojeve i funkcije iz ove biblioteke,
   neophodno je da ukljuci ovo zaglavlje. */

/* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i 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 navedenim u complex.c */
```

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

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

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

```c
/*************************************************************
Ovaj program koristi 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.

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

Kompilacija se moze 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>
```

```c
/* Ukljucuje aw zaglavlje neophodno za rad sa kompleksnim brojevima
    */
#include "complex.h"

/* U glavnoj funkciji se za uneti kompleksan broj ispisuje 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

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


/* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
   obliku. Kako bi ustedeli kopiranje cele strukture, polinom
   prenosimo po adresi */
void ispisi(const Polinom * p)
{
  int i;
  for (i = p->stepen; i >= 0; i--) {
    if (p->koef[i]) {
      if (p->koef[i] >= 0 && i != p->stepen)
        putchar('+');
      if (i > 1)
        printf("%.2fx^%d", p->koef[i], i);
      else if (i == 1)
        printf("%.2fx", p->koef[i]);
      else
        printf("%.2f", p->koef[i]);
    }
  }
  putchar('\n');
}

/* Funkcija koja ucitava polinom sa tastature */
Polinom ucitaj()
{
  int i;
```

```
31    Polinom p;
33    /* Ucitavamo stepen polinoma */
      scanf("%d", &p.stepen);
35
      /* Ponavljamo ucitavanje stepena sve dok ne unesemo stepen iz
37       dozvoljenog opsega */
      while (p.stepen > MAX_STEPEN || p.stepen < 0) {
39      printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
        scanf("%d", &p.stepen);
41    }
43    /* Unosimo koeficijente polinoma */
      for (i = p.stepen; i >= 0; i--)
45      scanf("%lf", &p.koef[i]);
      return p;
47  }
49  /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
       algoritmom */
51  /* x^4+2x^3+3x^2+2x+1 = ( ( (x+2)*x + 3)*x + 2)*x + 1 */
    double izracunaj(const Polinom * p, double x)
53  {
      double rezultat = 0;
55    int i = p->stepen;
      for (; i >= 0; i--)
57      rezultat = rezultat * x + p->koef[i];
      return rezultat;
59  }
61  /* Funkcija koja sabira dva polinoma */
    Polinom saberi(const Polinom * p, const Polinom * q)
63  {
      Polinom rez;
65    int i;
67    rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
69    for (i = 0; i <= rez.stepen; i++)
        rez.koef[i] =
71          (i > p->stepen ? 0 : p->koef[i]) + (i >
                                                q->
73                                              stepen ? 0 : q->koef[i]);
      return rez;
75
    }
77
    /* Funkcija mnozi dva polinoma p i q */
79  Polinom pomnozi(const Polinom * p, const Polinom * q)
    {
81    int i, j;
      Polinom r;
```

```
83
  r.stepen = p->stepen + q->stepen;
85  if (r.stepen > MAX_STEPEN) {
    fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
87    exit(EXIT_FAILURE);
  }
89
  for (i = 0; i <= r.stepen; i++)
91    r.koef[i] = 0;

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

97  return r;
}
99
/* Funkcija racuna izvod polinoma p */
101 Polinom izvod(const Polinom * p)
{
103   int i;
  Polinom r;
105
  if (p->stepen > 0) {
107   r.stepen = p->stepen - 1;

109   for (i = 0; i <= r.stepen; i++)
      r.koef[i] = (i + 1) * p->koef[i + 1];
111 } else
    r.koef[0] = r.stepen = 0;
113
  return r;
115 }

117 /* Funkcija racuna n-ti izvod polinoma p */
Polinom nIzvod(const Polinom * p, int n)
119 {
  int i;
121   Polinom r;

123   if (n < 0) {
    fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
125   exit(EXIT_FAILURE);
  }
127
  if (n == 0)
129   return *p;

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

```
135   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>
```

```c
#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);
```

```
52    /* Uspesno zavrsavamo program */
      return 0;
54  }
```

Rešenje 1.5

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

10    /* Broj bitova celog broja */
      unsigned velicina = sizeof(unsigned) * 8;
12    /* Maska koja se koristi za "ocitavanje" bitova */
      unsigned maska;
14
      /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
16       na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
         mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
18       se jedini bit jedinica pomera udesno, kako bi se ocitao naredni
         bit broja x koji je argument funkcije. Odgovarajuci karakter,
20       ('0' ili '1'), ispisuje se na standardnom izlazu. Neophodno je
         da promenljiva maska bude deklarisana kao neoznacen ceo broj
22       kako bi se siftovanjem u desno vrsilo logicko sifotvanje
         (popunjavanje nulama) a ne aritmeticko siftovanje (popunjavanje
24       znakom broja). */
      for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
26      putchar(x & maska ? '1' : '0');

28    putchar('\n');
    }
30

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

38    return 0;
    }
```

Rešenje 1.6

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

```c

/* 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');
}

/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   kreiranjem odgovarajuce maske i njenim pomeranjem */
int count_bits1(int x)
{
  int br = 0;
  unsigned wl = sizeof(unsigned) * 8 - 1;

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

  return br;
}

/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   formiranjem odgovarajuce maske i pomeranjem promenljive x */
int count_bits2(int x)
{
  int br = 0;
  unsigned wl = sizeof(int) * 8 - 1;

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

  unsigned maska = 1 << 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\n");
  printf("funkcija count_bits1: %d\n", count_bits1(x));
  printf("funkcija count_bits2: %d\n", count_bits2(x));
  return 0;
}
```

Rešenje 1.7

```
#include <stdio.h>

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

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

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

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

  return rezultat;
}

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

```
37
   /* Promenljiva x se pomera u desno sve dok postoje jedinice u
39    njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
      razlicita od nule). */
41 for (; x != 0; x >>= 1) {
     /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
43      detektuje na poziciji najmanje tezine u binarnoj
        reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
45      sa desna u rezultat */
     if (x & 1) {
47     rezultat <<= 1;
       rezultat |= 1;
49   }
   }
51
   return rezultat;
53 }

55 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
    celog broja u memoriji */
57 void print_bits(int x)
   {
59   unsigned velicina = sizeof(int) * 8;
   unsigned maska;
61
   for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
63   putchar(x & maska ? '1' : '0');

65 putchar('\n');
   }
67
   int main()
69 {
   int broj;
71 scanf("%x", &broj);

73 printf("Najveci:\n");
   print_bits(najveci(broj));
75
   printf("Najmanji:\n");
77 print_bits(najmanji(broj));

79 return 0;
   }
```

Rešenje 1.8

```
   #include <stdio.h>
2


4  /*************************************************************
```

```
   Funckija postavlja na nulu n bitova pocev od pozicije p.
   Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
   se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
   1010 1110 0111 1010 1011 1100 1101 1110 1000 0010 0111 */
unsigned reset(unsigned x, unsigned n, unsigned p)
{
/*****************************************************************
     Cilj je anulirati samo zeljene bitove, a da ostali
     ostanu nepromenjeni. Maska koja ce se koristiti je ona cija
     binarna reprezentacija ima n bitova
     postavljenih na 0 pocev od pozicije p, dok su svi ostali
     postavljeni na 1.

     Na primer, za n=5 i p=10 cilj je maska oblika
     1111 1111 1111 1111 1111 1000 0011 1111
     To se postize na sledeci nacin:
     ~0                  1111 1111 1111 1111 1111 1111 1111 1111
(~0 << n)                1111 1111 1111 1111 1111 1111 1110 0000
~(~0 << n)               0000 0000 0000 0000 0000 0000 0001 1111
(~(~0 << n) << ( p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000
~(~(~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
*****************************************************************/
  unsigned maska = ~(~(~0 << n) << (p - n + 1));

  return x & maska;
}


/*****************************************************************
   Funckija postavlja na 1 n bitova pocev od pozicije p.
   Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
   se broji od nule . Npr, za n=5, p=10
   1010 1011 1100 1101 1110 1010 1110 0111
   1010 1011 1100 1101 1110 1111 1110 0111
*****************************************************************/
unsigned set(unsigned x, unsigned n, unsigned p)
{

/*****************************************************************
     Cilj je samo odredjenih n bitova postaviti na 1, dok
     ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
     formira se maska oblika
     0000 0000 0000 0000 0000 0111 1100 0000
     prateci vrlo slican postupak kao za prethodnu funkciju
*****************************************************************/
  unsigned maska = ~(~0 << n) << (p - n + 1);

  return x | maska;
}


/*****************************************************************
```

```
   Funkcija vraca celobrojno polje bitova, desno poravnato, koje
   predstavlja n bitova pocev od pozicije p u binarnoj
   reprezentaciji broja x, pri cemu se pozicija broji sa desna
   ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
   i broj cija je binarna reprezentacija:
   1010 1011 1100 1101 1110 1010 1110 0111
   trazi se
   0000 0000 0000 0000 0000 0000 0000 1011
*****************************************************************/
unsigned get_bits(unsigned x, unsigned n, unsigned p)
{

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

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


/* Funkcija vraca broj x kome su n bitova pocev od pozicije p
   postavljeni na vrednosti n bitova najnize tezine binarne
   reprezentacije broja y */
unsigned set_n_bits(unsigned x, unsigned n, unsigned p, unsigned y)
{
/*****************************************************************
     Kreira se maska kod kod koje su poslednjih n bitova 1, a
     ostali su 0. Na primer za n=5
     0000 0000 0000 0000 0000 0000 0001 1111
*****************************************************************/
  unsigned last_n_1 = ~(~0 << n);
/*****************************************************************
     Kao sto je i u funkciji reset, i ovde se kreira masku koja ima n
     bitova postavljenih na 0 pocevsi od pozicije p, dok su
     ostali bitovi 1. Na primer za n=5 i p =10
     1111 1111 1111 1111 1111 1000 0011 1111
*****************************************************************/
  unsigned middle_n_0 = ~(~(~0 << n) << (p - n + 1));

  /* U promenljivu x_reset se smesta vrednost dobijena kada se u
     binarnoj reprezentaciji vrednosti promenljive x resetuje n
     bitova na pozicijama pocev od p */
  unsigned x_reset = x & middle_n_0;

  /* U promenljivu y_shift_middle se smesta vrednost dobijena od
     binarne reprezentacije vrednosti promenljive y cijih je n bitova
```

```
     najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
     bitovi su nule. (y & last_n_1) Resetuju se svi bitovi osim
     najnizih n */
  unsigned y_shift_middle = (y & last_n_1) << (p - n + 1);

  return x_reset ^ y_shift_middle;
}


/* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
   njih n */
unsigned invert(unsigned x, unsigned n, unsigned p)
{
/*************************************************************
     Formira se maska sa n jedinica pocev od pozicije p.
     Na primer za n=5 i p=10
     0000 0000 0000 0000 0000 0111 1100 0000
*************************************************************/
  unsigned maska = ~(~0 << n) << (p - n + 1);

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


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

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

  putchar('\n');
}



int main()
{
  unsigned broj, p, n, y;
  scanf("%u%u%u%u", &broj, &n, &p, &y);
  printf("Broj %u %s= ", broj, "");
  print_bits(broj);


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

```

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

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

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

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

  return 0;
}
```

Rešenje 1.9

```
#include <stdio.h>

/*****************************************************************
   Funkcija binarnu reprezentaciju svog argumenta x rotira u
   levo za n mesta i vraca odgovarajuci neoznacen ceo broj cija
   je binarna reprezentacija dobijena nakon rotacije.
   Na primer za n =5 i x cija je interna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
   funkcija vraca neoznacen ceo broj cija je binarna
   reprezentacija:
   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 bi
     pre siftovanja u levo za 1 najvisi bit bio sacuvan. */
  unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
  int i;

  /* n puta se vrsi rotaciju za jedan bit u levo. U svakoj iteraciji
     se odredi prvi bit, a potom se pomera binarna reprezentacija
     trenutne vrednosti promenljive x u levo za 1. Nakon toga, potom
     najnizi bit se postavlja na vrednost koju je imao prvi bit koji
     je istisnut 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 binarna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
   funkcija vraca neoznacen ceo broj cija je binarna
   reprezentacija:
   0001 1101 0101 1110 0110 1111 0000 1001
*************************************************************/
unsigned rotate_right(unsigned x, unsigned n)
{
  unsigned last_bit;
  int i;

  /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
     iteraciji se odredjuje bit najmanje tezine broja x, zatm tako
     odredjeni bit se siftuje u levo tako da najnizi bit dode do
     pozicije najviseg bita. Zatim, nakon siftovanja binarne
     reprezentacije trenutne vrednosti promenljive x za 1 u desno,
     najvisi bit se postaljva na vrednost vec zapamcenog bita koji je
     bio na poziciji najmanje tezine. */
  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
   argument funkcije x oznaceni ceo broj */
int rotate_right_signed(int x, unsigned n)
{
  unsigned last_bit;
  int i;


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

    if (x < 0)
/*************************************************************
         Siftovanjem u desno broja koji je negativan dobija se 1
         kao bit na najvisoj poziciji.  Na primer ako je x
         1010 1011 1100 1101 1110 0001 0010 001b
         (sa b je oznacen ili 1 ili 0 na najnizoj poziciji)
```

```
85          Onda je sadrzaj promenljive last_bit:
            0000 0000 0000 0000 0000 0000 0000 000b
87          Nakon siftovanja sadrzaja promenljive x za 1 u desno
            1101 0101 1110 0110 1111 0000 1001 0001
89          Kako bi umesto 1 na najvisoj poziciji u trenutnoj
            binarnoj reprezentaciji x bilo postavljeno b nije
91          dovoljno da se siftuje na najvisu poziciju jer bi se
            time dobile 0, a u ovom slucaju su potrebne jedinice
93          zbog bitovskog & zato se prvo vrsi komplementiranje, a
            zatim siftovanje
95          ~last_bit << (sizeof(int)*8 -1)
            B000 0000 0000 0000 0000 0000 0000 0000
97          gde B oznacava ~b.
            Potom se ponovo vrsi komplementiranje kako bi se b
99          nalazilo na najvisoj poziciji i sve jedinice na ostalim
            pozicijama
101         ~(~last_bit << (sizeof(int)*8 -1))
            b111 1111 1111 1111 1111 1111 1111 1111
103 ************************************************************/
        x = (x >> 1) & ~(~last_bit << (sizeof(int) * 8 - 1));
105     else
        x = (x >> 1) | last_bit << (sizeof(int) * 8 - 1);
107   }

109   return x;
  }
111

113 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
       celog broja u memoriji */
115 void print_bits(int x)
  {
117   unsigned velicina = sizeof(int) * 8;
    unsigned maska;
119   for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
      putchar(x & maska ? '1' : '0');
121
    putchar('\n');
123 }

125 int main()
  {
127   unsigned x, k;
    scanf("%x%x", &x, &k);
129   printf("x %s = ", "");
    print_bits(x);
131   printf("rotate_left(%u,%u)%s = ", x, k, "");
    print_bits(rotate_left(x, k));
133
    printf("rotate_right(%u,%u)%s = ", x, k, "");
135   print_bits(rotate_right(x, k));
```

```
137   printf("rotate_right_signed(%u,%u) = ", x, k);
      print_bits(rotate_right_signed(x, k));
139
      return 0;
141 }
```

Rešenje 1.10

```
1  #include <stdio.h>

3  /*************************************************************
       Funkcija vraca vrednost cija je binarna reprezentacija slika
5      u ogledalu binarne reprezentacije broja x koji se prosledjuje
       kao argument funkcije. Na primer za x
7      cija binarna reprezentacija izgleda ovako
       10101011110011011110010010003 11
9      funkcija treba da vrati broj cija binarna reprezentacija
       izgleda:
11     11000100100001111011001111010101
   *************************************************************/
13 unsigned mirror(unsigned x)
   {
15   unsigned najnizi_bit;
     unsigned rezultat = 0;
17
     int i;
19   /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuce
        vrednosti broja x se odredjuje i pamti u promenljivoj
21      najnizi_bit, nakon cega se na promenljivu x primeni siftovanje u
        desno. */
23   for (i = 0; i < sizeof(x) * 8; i++) {
       najnizi_bit = x & 1;
25     x >>= 1;
       /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
27        postavljeni bitovi dobijaju vecu poziciju. Novi bit se
          postavlja na najnizu poziciju */
29     rezultat <<= 1;
       rezultat |= najnizi_bit;
31   }
     return rezultat;
33 }


35
   /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
37    celog broja u memoriji */
   void print_bits(int x)
39 {
     unsigned velicina = sizeof(int) * 8;
41   unsigned maska;
     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
43     putchar(x & maska ? '1' : '0');
```

```
45    putchar('\n');
}
47
int main()
49 {
    int broj;
51  scanf("%x", &broj);

53  /* Ispisuje se binarna reprezentaciju unetog broja */
    print_bits(broj);
55
    /* Ispisuje se binarna reprezentaciju broja dobijenog pozivom
57     funkcije mirror */
    print_bits(mirror(broj));
59
    return 0;
61 }
```

Rešenje 1.11

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

8   int broj_nula, broj_jedinica;
    unsigned int maska;
10
    broj_nula = 0;
12  broj_jedinica = 0;

14  /* Maska je inicijalizovana tako da moze da analizira bit najvece
       tezine */
16  maska = 1 << (sizeof(unsigned int) * 4 - 1);

18  /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
       iteraciji pomera u desno pa ce jedini bit koji je postavljen na
20     1 biti na svim pozicijama u binarnoj reprezentaciji maske */
    while (maska != 0) {
22
      /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
24       1 i uveca se odgovarajuci brojac */
      if (n & maska) {
26      broj_jedinica++;
      } else {
28      broj_nula++;
      }
30
```

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

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

}

int main()
{
  unsigned int n;

  scanf("%u", &n);

  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;

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

  /* Postavlja se maska tako da moze da procitamo da li su dva
     najmanja bita u zapisu broja x 11 */
  /* Binarna reprezentacija broja 3 je 000....00011 */
  maska = 3;

  while (x != 0) {

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

    /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
       proveravao sledeci par bitova. Pomeranjem u desno bit najvece
       tezine se popunjava nulom jer je x neoznacen broj. */
```

```
28      x = x >> 1;
    }
30
    return broj_parova;
32
}
34
int main()
36 {
    unsigned int x;
38
    scanf("%u", &x);
40
    printf("%d\n", broj_parova(x));
42
    return 0;
44 }
```

Rešenje 1.14

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

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

10 #define MAX_DUZINA sizeof(unsigned int)*2 +1


12
void prevod(unsigned int x, char s[])
14 {

16   int i;
   unsigned int maska;
18   int vrednost;

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

24   /*************************************************************
        Broj se posmatra od pozicije najmanje tezine ka poziciji
26      najvece tezine. Na primer za broj
        00000000001101000100001111010101
28      u prvom koraku se citaju bitovi izdvojeni sa <...>:
        00000000001101000100001111101<0101>
30      u drugom koraku:
        00000000001101000100001<1101>0101
```

```
     u trecem koraku:
     0000000000011010001000<0011>11010101 i tako redom

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

    /* Primenljiva x se pomera za 4 bita u desnu stranu i time ce u
       narednoj iteraciji biti posmatrane sledece 4 cifre */
    x = x >> 4;
  }

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

int main()
{

  unsigned int x;
  char s[MAX_DUZINA];

  scanf("%u", &x);

  prevod(x, s);

  printf("%s\n", s);

  return 0;
}
```

Rešenje 1.17

```
#include <stdio.h>


/*************************************************************
```

```
  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);
}

/* U prethodnim funkcijama iskomentarisan je poziv funkcije prinf
   koji ispisuje odgovarajucu poruku prilikom svakog ulaska us
   funkciju. Odkomentarisati pozive printf funkcije u obe funkcije da
   uocite razliku u broju rekurzivnih poziva obe verzije. */

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

  printf("%d\n", stepen(x, k));
  // 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 <= 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 <= 9)
    return 1;
  else
    return paran(n / 10);
}

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

Rešenje 1.19

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

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

/* U sledece dve funkcije je prikazan postupak oslobadjanja od repne
   rekurzije koja postoji u funkciji faktorijelRepna, koristeci
   algoritam sa predavanja.

   Najpre, funckija se transformise tako sto rekurzivni poziv zemeni
   sa naredbama kojima se vrednost argumenta funkcije postavlja na
   vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
   goto naredbe za vracanje na pocetak tela funkcije. */

int 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 i prethodna funkcija se koristi samo kao medjukorak. Sledi
   iterativno resenje bez bezuslovnih skokova: */
int faktorijelRepna_v2(int n, int result)
{
  while (n != 0) {
    result = n * result;
    n = n - 1;
  }

  return result;
}

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

```
   zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
   sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
   faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
   funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
   poziv 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 1.21

```
#include <stdio.h>

int zbir_cifara(unsigned int x)
{
  /* Izlazak iz rekurzije: ako je broj jednocifren */
  if (x < 10)
    return x;

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

int main()
{
  unsigned int x;

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

  /* Ispisuje se zbir cifara ucitanog broja */
  printf("%d\n", zbir_cifara(x));

  return 0;
}
```

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)
{
  /* Nije postavljena stroga 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);
}

/*****************************************************************
  Funkcija napisana na drugi nacin:
  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;
}
```

Rešenje 1.23

```c
#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];

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

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

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

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

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

Rešenje 1.24

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

int skalarno(int a[], int b[], int n)
{
```

```
  /* Izlazak iz rekurzije */
  if (n == 0)
    return 0;

  /* Na osnovu resenja problema dimenzije n-1, resava se problem
     dimenzije n */
  else
    return a[n - 1] * b[n - 1] + skalarno(a, b, n - 1);
}

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

  /* Unosi se dimenzija nizova, */
  scanf("%d", &n);

  /* A zatim i elementi nizova. */
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

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

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

  return 0;
}
```

Rešenje 1.25

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

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

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

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

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

```
20
   /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz;
22      Promenljiva i predstavlja indeks tekuceg broja */
   i = 0;
24 while (scanf("%d", &a[i]) != EOF) {
     i++;
26 }
   n = i;
28
   printf("%d\n", br_pojave(x, a, n));
30 return 0;
}
```

Rešenje 1.26

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

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

10  else
     return ((a[n - 3] == x) && (a[n - 2] == y)
12           && (a[n - 1] == z))
        || tri_uzastopna_clana(x, y, z, a, n - 1);
14 }

16 int main()
{
18  int x, y, z, a[MAX_DIM];
   int n;
20
   /* Ucitavaju se tri cela broja za koje se ispituje da li su
22      uzastopni clanovi niza */
   scanf("%d%d%d", &x, &y, &z);
24
   int i = 0;
26 while (scanf("%d", &a[i]) != EOF) {
     i++;
28 }
   n = i;
30
   if (tri_uzastopna_clana(x, y, z, a, n))
32   printf("da\n");
   else
34   printf("ne\n");

36 return 0;
```

```
}
```

Rešenje 1.27

```
#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;

  /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
     postavljen na 1. Koriscenjem odgovarajuce maske proverava se
     vrednost najviseg bita. Rezultat koliko ima jedinica u ostatku
     binarnog zapisa broja x se uvecava za 1. 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.
     Za rekurzicvni poziv se salje vrednost koja se dobija kada se x
     siftuje u levo. Napomena: argument funkcije x je oznacen ceo
     broj, usled cega se ne koristi siftovanje udesno, 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);
  else
    return count(x << 1);

}

/*************************************************************
     Telo prethodne funkcije je moglo biti zapisano i krace:
     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);
```

```
48    printf("%d\n", count(x));

50    return 0;
}
```

Rešenje 1.29

```
#include<stdio.h>
2
/* Rekurzivna funkcija za odredjivanje najvece heksadekadne cifre u
4     broju */
int max_oktalna_cifra(unsigned x)
6 {
    /* Izlazak iz rekurzije */
8   if (x == 0)
      return 0;
10  /* Odredjivanje poslednje heksadekadne cifre u broju */
    int poslednja_cifra = x & 7;
12  /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
       izbrise poslednja oktalna cifra */
14  int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);
    return poslednja_cifra >
16      max_bez_poslednje_cifre ? poslednja_cifra :
        max_bez_poslednje_cifre;
18 }

20 int main()
{
22   unsigned x;
    scanf("%u", &x);
24  printf("%d\n", max_oktalna_cifra(x));
    return 0;
26 }
```

Rešenje 1.30

```
#include<stdio.h>
2


4 /* Rekurzivna funkcija za odredjivanje najvece oktalne cifre u broju
      */
int max_heksadekadna_cifra(unsigned x)
6 {
    /* Izlazak iz rekurzije */
8   if (x == 0)
      return 0;
10  /* Odredjivanje poslednje heksadekadne cifre u broju */
    int poslednja_cifra = x & 15;
12  /* 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

```
#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;

  scanf("%s", s);

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

  /* Ispisuje se poruka 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 se poziva sa argumentom
   m=1 jer formiranje permutacije pocinje od 1. pozicije. Stoga, nece
   se koristi 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 ispisuje se permutacija. */
  if (m > n) {
    ispisiNiz(a, n);
    return;
  }

  /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
     mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
     Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
     ovako postavljenom pocetku permutacije. Kada se to zavrsi, vrsi
     se provera da li postoji jos neki broj koji moze da se stavi na
     m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
     funkcija zavrsava sa radom. Ukoliko takav broj postoji, onda se
     ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
```

```
52      ali sada sa drugacije postavljenim prefiksom. */

54
    for (i = 1; i <= n; i++) {
56      /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
           pozicije, onda se on postavlja na poziciju m i poziva se
58         funkcija da napravi permutaciju za jedan vece duzine, tj. m+1.
           Inace, nastavlja se dalje, trazeci broj koji se nije pojavio
60         do sada u permutaciji. */
        if (!koriscen(a, m - 1, i)) {
62        a[m] = i;
          /* Poziva se ponovo funkcija da dopuni ostatak permutacije
64           posle upisivanja i na poziciju m. */
          permutacija(a, m + 1, n);
66      }
    }
68  }

70  int main(void)
    {
72    int n;
      int a[MAX_DUZINA_NIZA];
74
      printf("Unesite duzinu permutacije: ");
76    scanf("%d", &n);
      if (n < 0 || n >= MAX_DUZINA_NIZA) {
78      fprintf(stderr,
                "Duzina permutacije mora biti broj veci od 0 i manji od %
        d!\n",
80              MAX_DUZINA_NIZA);
        exit(EXIT_FAILURE);
82    }

84    permutacija(a, 1, n);

86    exit(EXIT_SUCCESS);
    }
```

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)
    {
9     return (0 < k
              && k < n) ? binomniKoeficijent(n - 1,
11                                           k - 1) +
```

```
      binomniKoeficijent(n - 1, k) : 1;
}

/*****************************************************************
  Iterativno izracunavanje datog binomnog koeficijenta.

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

/* Prostim opaZanjem se uocava da se svaki element n-te hipotenuze
   (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz
   pomenute dve nove ivicne jedinice lako se zakljucuje da ce suma
   elementa n-te hipotenuze biti tacno 2 puta veca. */
int sumaElemenataHipotenuze(int n)
{
  return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
}

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


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

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

  exit(EXIT_SUCCESS);
}
```

Glava 2

Pokazivači

2.1 Pokazivačka aritmetika

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Primer 1

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

Primer 2

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

Zadatak 2.2 Dat je niz realnih brojeva dimenzije n.

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

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

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

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

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

Primer 1

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

Primer 2

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

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

Primer 1

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

Primer 2

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

Primer 3

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

Primer 4

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

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Primer 1

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

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

Primer 2

```
Poziv: ./a.out

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

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

Primer 1

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

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

Primer 2

```
Poziv: ./a.out

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

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

Primer 1

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

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

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

Primer 2

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

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

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

Primer 3

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

Datoteka ulaz.txt ne postoji

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

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

Primer 1

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

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

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

Primer 2

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

ulaz.txt
 Ovo je sadrzaj datoteke i u njoj ima reci
 koje pocinju sa sa

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

Primer 3

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

Datoteka ulaz.txt ne postoji

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

Primer 4

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

ulaz.txt
  Ovo je sadrzaj ulaza.

Interakcija programa:
  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.

Primer 1

```
Interakcija programa:
  Unesite dimenziju matrice:  3
  Unesite elemente matrice, vrstu po vrstu:
  1 -2 3
  4 -5 6
  7 -8 9
  Trag matrice je 5.
  Euklidska norma matrice je 16.88.
  Vandijagonalna norma matrice je 11.
```

Primer 2

```
Interakcija programa:
  Unesite dimenziju matrice:  0
  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 li su matrice jednake, a zatim ispisati zbir i proizvod učitanih matrica.

Primer 1

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

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.

Primer 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

Interakcija programa:
 Relacija nije refleksivna.
 Relacija nije simetricna.
 Relacija jeste tranzitivna.
 Refleksivno zatvorenje relacije:
 1 0 0 0
 0 1 1 0
 0 0 1 0
 0 0 0 1
 Simetricno zatvorenje relacije:
 1 0 0 0
 0 1 1 0
 0 1 1 0
 0 0 0 0
 Refleksivno-tranzitivno zatvorenje relacije:
 1 0 0 0
 0 1 1 0
 0 0 1 0
 0 0 0 1
```

Zadatak 2.11 Data je kvadratna matrica dimenzije n.

(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 rezultat primene prethodno napisanih funkcija.

Primer 1

```
Poziv: ./a.out 3

Interakcija programa:
 Unesite elemente matrice dimenzije 3:
 1 2 3
 -4 -5 -6
 7 8 9
 Najveci element matrice na sporednoj dijagonali je 7.
 Indeks kolone koja sadrzi najmanji element matrice 2.
 Indeks vrste koja sadrzi najveci element matrice 2.
 Broj negativnih elemenata matrice je 3.
```

Primer 2

```
Poziv: ./a.out 4

Interakcija programa:
 Unesite elemente matrice dimenzije 4:
 -1 -2 -3 -4
 -5 -6 -7 -8
 -9 -10 -11 -12
 -13 -14 -15 -16
 Najveci element matrice na sporednoj dijagonali je -4.
 Indeks kolone koja sadrzi najmanji element matrice 3.
 Indeks vrste koja sadrzi najveci element matrice 0.
 Broj negativnih elemenata matrice je 16.
```

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.

Primer 1

```
Interakcija programa:
 Unesite dimenziju matrice:  4
 Unesite elemente matrice, vrstu po vrstu:
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
 Matrica je ortonormirana.
```

Primer 2

```
Interakcija programa:
 Unesite dimenziju matrice:  3
 Unesite elemente matrice, vrstu po vrstu:
 1 2 3
 5 6 7
 1 4 2
 Matrica nije ortonormirana.
```

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.

Primer 1

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

Primer 2

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

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.*

Primer 1

```
Interakcija programa:
 Unesite dimenziju kvadratne matrice:  3
 Unesite elemente matrice, vrstu po vrstu:
 1 2 3
 4 5 6
 7 8 9
 Unesite stepen koji se racuna:  8
 8. stepen matrice je:
 510008400 626654232 743300064
 1154967822 1419124617 1683281412
 1799927244 2211595002 2623262760
```

2.3 Dinamička alokacija memorije

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

Primer 1

```
Interakcija programa:
 Unesite dimenziju niza:  3
 Unesite elemente niza:
 1 -2 3
 Niz u obrnutom poretku je: 3 -2 1
```

Primer 2

```
Interakcija programa:
 Unesite dimenziju niza:  -1
 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.

Primer 1

```
Interakcija programa:
 Unesite brojeve, nulu za kraj:
 1 -2 3 -4 0
 Unesite elemente niza:
 1 -2 3 -4 0
 Niz u obrnutom poretku je: -4 3 -2 1
```

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.

Primer 1

```
Interakcija programa:
 Unesite dve niske karaktera:
 Jedan Dva
 Nadovezane niske: JedanDva
```

Zadatak 2.18 Napisati program koji sa standardnog ulaza učitava matricu realnih 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.

Primer 1

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

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

(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.

Primer 1

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

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. Ukoliko ima više takvih, ispisati prvu.

Primer 1

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

Primer 2

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

Zadatak 2.21 Data je realna kvadratna matrica dimenzije n.

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

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

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

Primer 1

```
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

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

Primer 1

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

matrice.txt
 3
 1.1 -2.2 3.3
 -4.4 5.5 -6.6
 7.7 -8.8 9.9
 -1.1 2.2 -3.3
 4.4 -5.5 6.6
 -7.7 8.8 -9.9
```

```
Interakcija programa:
 Trazena matrica je:
 1.1 -2.2 3.3
 -1.1 2.2 -3.3
 -4.4 5.5 -6.6
 4.4 -5.5 6.6
 7.7 -8.8 9.9
 -7.7 8.8 -9.9
```

Zadatak 2.23 Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Rezultujuću matricu ispisati na standardni izlaz.

Primer 1

```
Interakcija programa:
 Unesite elemente niza, nulu za kraj:
 1 2 3 0
 Trazena matrica je:
 1 2 3
 2 3 1
 3 1 2
```

Zadatak 2.24 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.*

Primer 1

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

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

**** Zadatak 2.25** U datoteci „temena.txt“ se nalaze tačke koje predstavljaju temena nekog n-tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom n-touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan.

Primer 1

```
TEMENA.TXT
 -1 -1
 1 -1
 1 1
 -1 1

INTERAKCIJA PROGRAMA:
 U datoteci su zadata temena cetvorougla.
 Obim je 8.
 Povrsina je 4.
```

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`).

Primer 1

```
Poziv: ./a.out sin

Interakcija programa:
 Unesite krajeve intervala:
 -0.5 1
 Koliko tacaka ima na ekvidistantnoj
 mrezi (ukljucujuci krajeve intervala)?
 4
  x sin(x)
 ----------------------
 | -0.50000 | -0.47943 |
 | 0.00000 | 0.00000 |
 | 0.50000 | 0.47943 |
 | 1.00000 | 0.84147 |
 ----------------------
```

Primer 2

```
Poziv: ./a.out cos

Interakcija programa:
 Unesite krajeve intervala:
 0 2
 Koliko tacaka ima na ekvidistantnoj
 mrezi (ukljucujuci krajeve intervala)?
 4
  x cos(x)
 ----------------------
 | 0.00000 | 1.00000 |
 | 0.66667 | 0.78589 |
 | 1.33333 | 0.23524 |
 | 2.00000 | -0.41615 |
 ----------------------
```

Zadatak 2.27 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):

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

Primer 1

```
Interakcija programa:
 Unesite ime funkcije, n i a:
 tan 1.570795 10000
 Limes funkcije tan je -10134.5.
```

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.

Primer 1

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

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 i n, a na standardni izlaz ispisuje vrednost odgovarajućeg integrala.

Primer 1

```
Interakcija programa:
 Unesite ime funkcije, n, a i b:
 sin 100 -1.0 3.0
 Vrednost integrala je 1.530295.
```

2.5 Rešenja

Rešenje 2.1

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

#define MAX 100

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

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

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

  /* Vrsi se obrtanje niza */
```

```
25    for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
        int t = *prvi;
27
        /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
29         vrednost koja se nalazi na adresi na koju pokazuje pokazivac
           "poslednji". Nakon toga se pokazivac "prvi" uvecava za jedan
31         sto za posledicu ima da "prvi" pokazuje na sledeci element u
           nizu */
33      *prvi++ = *poslednji;

35      /* Vrednost promenljive "t" se postavlja na adresu na koju
           pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
37         umanjuje za jedan, sto za posledicu ima da pokazivac
           "poslednji" sada pokazuje na element koji mu prethodi u nizu
        */
39      *poslednji-- = t;
    }
41
    /*****************************************************************
43      Drugi nacin za obrtanje niza

45      for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
                                              prvi++, poslednji--) {
47        int t = *prvi;
          *prvi = *poslednji;
49        *poslednji = t;
        }
51     *****************************************************************
        */
}
53
int main()
55 {
    /* Deklarise se niz od najvise MAX elemenata */
57  int a[MAX];

59  /* Broj elemenata niza a */
    int n;
61
    /* Pokazivac na elemente niza */
63  int *p;

65  printf("Unesite dimenziju niza: ");
    scanf("%d", &n);
67
    /* Proverava se da li je doslo do prekoracenja ogranicenja
69     dimenzije */
    if (n <= 0 || n > MAX) {
71    fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
      exit(EXIT_FAILURE);
73  }
```

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

  obrni_niz_v1(a, n);

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

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

  obrni_niz_v2(a, n);

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

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

  return 0;
}
```

Rešenje 2.2

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

#define MAX 100

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

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

  return s;
}

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

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

  return p;
}
```

```
28 /* Funkcija izracunava minimalni element niza */
   double min(double *a, int n)
30 {
     /* Na pocetku, minimalni element je prvi element */
32   double min = *a;
     int i;
34
     /* Ispituje se da li se medju ostalim elementima niza nalazi
36      minimalni */
     for (i = 1; i < n; i++)
38     if (*(a + i) < min)
         min = *(a + i);
40
     return min;
42 }

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

50   /* Ispituje se da li se medju ostalim elementima niza nalazi
        maksimalni */
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
     printf("Unesite dimenziju niza: ");
66   scanf("%d", &n);

68   /* Proverava se da li je doslo do prekoracenja ogranicenja
        dimenzije */
70   if (n <= 0 || n > MAX) {
       fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72     exit(EXIT_FAILURE);
     }
74
     printf("Unesite elemente niza:\n");
76   for (i = 0; i < n; i++)
       scanf("%lf", a + i);
78
```

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

  return 0;
}
```

Rešenje 2.3

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

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

  while (prvi < poslednji) {

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

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

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

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

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

int main()
{
  int a[MAX];
  int n;
```

```
  int *p;

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

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

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

  povecaj_smanji(a, n);

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

  return 0;
}
```

Rešenje 2.4

```
#include <stdio.h>

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

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

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

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

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

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

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

  return 0;
}
```

Rešenje 2.5

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

/* Funkcija ispituje da li je niska palindrom */
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;
```

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

Rešenje 2.6

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

4 #define MAX_KARAKTERA 100

6 /* Implementacija funkcija strlen() iz standardne biblioteke */
  int duzina(char *s)
8 {
    int i;
10  for (i = 0; *(s + i); i++);
    return i;
12 }

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

20  /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
      */
    if (argc < 3) {
22    printf("Greska: ");
      printf("Nedovoljan broj argumenata komandne linije.\n");
24    printf("Program se poziva sa %s ime_dat br_karaktera.\n",
             argv[0]);
26    exit(EXIT_FAILURE);
    }
28
    /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
30     komandne linije. */
    in = fopen(*(argv + 1), "r");
32  if (in == NULL) {
      fprintf(stderr, "Greska: ");
34    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
      exit(EXIT_FAILURE);
36  }
```

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

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

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

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

Rešenje 2.7

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

4 #define MAX_KARAKTERA 100

6 /* Implementacija funkcije strcpy() iz standardne biblioteke */
  void kopiranje_niske(char *dest, char *src)
8 {
    int i;
10  for (i = 0; *(src + i); i++)
      *(dest + i) = *(src + i);
12 }

14 /* Implementacija funkcije strcmp() iz standardne biblioteke */
  int poredjenje_niski(char *s, char *t)
16 {
    int i;
18  for (i = 0; *(s + i) == *(t + i); i++)
      if (*(s + i) == '\0')
20      return 0;
    return *(s + i) - *(t + i);
22 }

24 /* Implementacija funkcije strlen() iz standardne biblioteke */
  int duzina_niske(char *s)
26 {
    int i;
28  for (i = 0; *(s + i); i++);
    return i;
30 }

32 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
      sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
```

```
{
36    if (duzina_niske(sufiks) <= duzina_niske(niska) &&
          poredjenje_niski(niska + duzina_niske(niska) -
38                           duzina_niske(sufiks), sufiks) == 0)
        return 1;
40    return 0;
}
42
/* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
44     prefiks niske zadate prvi argumentom funkcije */
int prefiks_niske(char *niska, char *prefiks)
46 {
    int i;
48    if (duzina_niske(prefiks) <= duzina_niske(niska)) {
      for (i = 0; i < duzina_niske(prefiks); i++)
50      if (*(prefiks + i) != *(niska + i))
          return 0;
52    return 1;
    } else
54    return 0;
}
56
int main(int argc, char **argv)
58 {
    /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
60     greska */
    if (argc < 4) {
62    printf("Greska: ");
      printf("Nedovoljan broj argumenata komandne linije.\n");
64    printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
             argv[0]);
66    exit(EXIT_FAILURE);
    }
68
    FILE *in;
70  int br = 0;
    char rec[MAX_KARAKTERA];
72
    in = fopen(*(argv + 1), "r");
74  if (in == NULL) {
      fprintf(stderr, "Greska: ");
76    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
      exit(EXIT_FAILURE);
78  }

80  /* Provera se opcija kojom je pozvan program a zatim se ucitavaju
      reci iz datoteke i broji se koliko njih zadovoljava trazeni
82    uslov */
    if (!(poredjenje_niski(*(argv + 3), "-s"))) {
84    while (fscanf(in, "%s", rec) != EOF)
        br += sufiks_niske(rec, *(argv + 2));
86    printf("Broj reci koje se zavrsavaju na %s je %d.\n", *(argv + 2)
```

```
    ,
           br);
88  } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
    while (fscanf(in, "%s", rec) != EOF)
90    br += prefiks_niske(rec, *(argv + 2));
    printf("Broj reci koje pocinju na %s je %d.\n", *(argv + 2), br);
92  }

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

Rešenje 2.8

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

5  #define MAX 100

7  /* Deklaracija funkcija koje ce kasnije biti definisane */
   double euklidska_norma(int M[][MAX], int n);
9  int trag(int M[][MAX], int n);
   int gornja_vandijagonalna_norma(int M[][MAX], int n);
11
   int main()
13 {
     int A[MAX][MAX];
15   int i, j, n;

17   printf("Unesite dimenziju matrice: ");
     scanf("%d", &n);
19
     /* Provera prekoracenja dimenzije matrice */
21   if (n > MAX || n <= 0) {
       fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
23     exit(EXIT_FAILURE);
     }
25
     printf("Unesite elemente matrice, vrstu po vrstu:\n ");
27   for (i = 0; i < n; i++)
       for (j = 0; j < n; j++)
29       scanf("%d", &A[i][j]);

31   /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
     for (i = 0; i < n; i++) {
33     /* Ispis elemenata i-te vrste */
       for (j = 0; j < n; j++)
35       printf("%d ", A[i][j]);
       printf("\n");
37   }
```

```
39    /*****************************************************************
       Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
41     Kod ovako definisane matrice, elementi su uzastopno smesteni u
       memoriju, kao na traci. To znaci da su svi elementi prve vrste
43     redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
       prve vrste smesten je prvi element druge vrste za kojim slede
45     svi elementi te vrste i tako dalje redom.

47     for( i = 0; i < n ; i++) {
         for ( j=0 ; j<n ; j++)
49         printf("%d ", *(*(A+i)+j));
         printf("\n");
51     }
       *****************************************************************/
53
    /* Ispisuje se rezultat na standardni izlaz */
55  int tr = trag(A, n);
    printf("Trag matrice je %d.\n", tr);
57
    printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
       ;
59  printf("Vandijagonalna norma matrice je = %d.\n",
           gornja_vandijagonalna_norma(A, n));
61
    return 0;
63 }

65 /* Definicija funkcija koje su ranije bile deklarisane */

67 /* Funkcija izracunava trag matrice */
   int trag(int M[][MAX], int n)
69 {
    int trag = 0, i;
71  for (i = 0; i < n; i++)
      trag += M[i][i];
73  return trag;
   }
75
   /* Funkcija izracunava euklidsku normu matrice */
77 double euklidska_norma(int M[][MAX], int n)
   {
79  double norma = 0.0;
    int i, j;
81
    for (i = 0; i < n; i++)
83    for (j = 0; j < n; j++)
        norma += M[i][j] * M[i][j];
85
    return sqrt(norma);
87 }
```

```
89 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
   int gornja_vandijagonalna_norma(int M[][MAX], int n)
91 {
     int norma = 0;
93   int i, j;

95   for (i = 0; i < n; i++) {
       for (j = i + 1; j < n; j++)
97       norma += abs(M[i][j]);
     }
99
     return norma;
101 }
```

Rešenje 2.9

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

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

23   for (i = 0; i < n; i++) {
       for (j = 0; j < n; j++)
25       printf("%d ", m[i][j]);
       printf("\n");
27   }
   }
29
   /* Funkcija proverava da li su zadate kvadratne matrice a i b
31    dimenzije n jednake */
   int jednake_matrice(int a[][MAX], int b[][MAX], int n)
33 {
     int i, j;
35
```

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

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

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

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

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

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

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

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

  /* Dimenzija matrica */
  int n;

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

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

```
    exit(EXIT_FAILURE);
  }

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

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

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

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

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

  return 0;
}
```

Rešenje 2.10

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

#define MAX 64

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

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

  return 1;
}

/* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono je
   odredjeno matricom koja sadrzi sve elemente polazne matrice
```

```
   dopunjene jedinicama na glavnoj dijagonali */
void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
{
  int i, j;

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

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

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

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

  return 1;
}

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

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

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

```

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

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

92   return 1;
   }
94

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

102  /* Prepisuju se vrednosti elemenata pocetne matrice */
     for (i = 0; i < n; i++)
104    for (j = 0; j < n; j++)
         zatvorenje[i][j] = m[i][j];
106
     /* Odredjuje se reflektivno zatvorenje matrice */
108  for (i = 0; i < n; i++)
       zatvorenje[i][i] = 1;
110
     /* Primenom Varsalovog algoritma odredjuje se tranzitivno
112     zatvorenje matrice */
     for (k = 0; k < n; k++)
114    for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
116        if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
               && (zatvorenje[i][j] == 0))
118          zatvorenje[i][j] = 1;
   }
120
   /* Funkcija ispisuje elemente matrice */
122 void pisi_matricu(int m[][MAX], int n)
   {
124  int i, j;

126  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;

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

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

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

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

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

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

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

```
178   printf("Relacija %s tranzitivna.\n",
          tranzitivnost(m, n) == 1 ? "jeste" : "nije");
180
      printf("Refleksivno zatvorenje relacije:\n");
182   ref_zatvorenje(m, n, pomocna);
      pisi_matricu(pomocna, n);
184
      printf("Simetricno zatvorenje relacije:\n");
186   sim_zatvorenje(m, n, pomocna);
      pisi_matricu(pomocna, n);
188
      printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
190   ref_tran_zatvorenje(m, n, pomocna);
      pisi_matricu(pomocna, n);
192
      /* Zatvara se datoteka */
194   fclose(ulaz);

196   return 0;
  }
```

Rešenje 2.11

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

4 #define MAX 32

6 /* Funkcija izracunava najveci element na sporednoj dijagonali. Za
     elemente sporedne dijagonale vazi da je zbir indeksa vrste i
8    indeksa kolone jednak n-1 */
  int max_sporedna_dijagonala(int m[][MAX], int n)
10 {
    int i;
12  int max_na_sporednoj_dijagonali = m[0][n - 1];

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

18  return max_na_sporednoj_dijagonali;
  }
20
  /* Funkcija izracunava indeks kolone najmanjeg elementa */
22 int indeks_min(int m[][MAX], int n)
  {
24  int i, j;
    int min = m[0][0], indeks_kolone = 0;
26
    for (i = 0; i < n; i++)
28    for (j = 0; j < n; j++)
```

```
      if (m[i][j] < min) {
        min = m[i][j];
        indeks_kolone = j;
      }

  return indeks_kolone;
}

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

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (m[i][j] > max) {
        max = m[i][j];
        indeks_vrste = i;
      }
  return indeks_vrste;
}

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

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

  return broj_negativnih;
}

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

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

  /* Ucitava se vrednost dimenzije i proverava se njena korektnost */
```

```
  n = atoi(argv[1]);

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

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

  printf("Najveci element matrice na sporednoj dijagonali je %d.\n",
         max_sporedna_dijagonala(m, n));

  printf("Indeks kolone koja sadrzi najmanji element matrice %d.\n",
         indeks_min(m, n));

  printf("Indeks vrste koja sadrzi najveci element matrice %d.\n",
         indeks_max(m, n));

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

  return 0;
}
```

Rešenje 2.12

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

#define MAX 32

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

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

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

```
19    int i, j;

21    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
23        printf("%d ", m[i][j]);
        printf("\n");
25    }
  }
27
  /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
29     da li je normirana i ortogonalna. Matrica je normirana ako je
       proizvod svake vrste matrice sa samom sobom jednak jedinici.
31     Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
       vrste matrice jednak nuli */
33 int ortonormirana(int m[][MAX], int n)
  {
35    int i, j, k;
      int proizvod;
37
      /* Ispituje se uslov normiranosti */
39    for (i = 0; i < n; i++) {
        proizvod = 0;
41
        for (j = 0; j < n; j++)
43        proizvod += m[i][j] * m[i][j];

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

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

53        proizvod = 0;

55        for (k = 0; k < n; k++)
            proizvod += m[i][k] * m[j][k];
57
          if (proizvod != 0)
59          return 0;
        }
61    }

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

67 int main()
  {
69    int A[MAX][MAX];
      int n;
```

```
71
   printf("Unesite dimenziju matrice: ");
73   scanf("%d", &n);

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

81   printf("Unesite elemente matrice, vrstu po vrstu:\n");
   ucitaj_matricu(A, n);
83
   printf("Matrica %s ortonormirana.\n",
85        ortonormirana(A, n) ? "je" : "nije");
   return 0;
87 }
```

Rešenje 2.13

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX_V 10
5  #define MAX_K 10

7  /* Funkcija proverava da li su ispisani svi elementi iz matrice,
      odnosno da li se narusio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
   {
11   return !(top <= bottom && left <= right);
   }
13
   /* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
   {
17   int i, j, top, bottom, left, right;

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

23   while (!krajIspisa(top, bottom, left, right)) {

25     for (j = left; j <= right; j++)
         printf("%d ", a[top][j]);
27
       /* Spusta se prvi red */
29     top++;

31     if (krajIspisa(top, bottom, left, right))
```

```
      break;

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

    /* Pomera se desna kolona za naredni krug ispisa blize levom
       kraju */
    right--;

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

    /* Ispisuje se donja vrsta */
    for (j = right; j >= left; j--)
      printf("%d ", a[bottom][j]);

    /* Podize se donja vrsta za naredni krug ispisa */
    bottom--;

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

    /* Ispisuje se prva kolona */
    for (i = bottom; i >= top; i--)
      printf("%d ", a[i][left]);

    /* Priprema se leva kolona za naredni krug ispisa */
    left++;
  }
  putchar('\n');
}

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

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

int main()
{
  int a[MAX_V][MAX_K];
  int m, n;

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

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

```
    exit(EXIT_FAILURE);
  }

  printf("Unesite elemente matrice, vrstu po vrstu:\n");
  ucitaj_matricu(a, n, m);

  printf("Spiralno ispisana matrica: ");
  ispisi_matricu_spiralno(a, n, m);

  return 0;
}
```

Rešenje 2.15

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

int main()
{
  int *p = NULL;
  int i, n;

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

  /* Alocira se 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);
  }

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

  printf("Niz u obrnutom poretku je: ");
  for (i = n - 1; i >= 0; i--)
    printf("%d ", p[i]);
  printf("\n");

  /* Oslobadja se prostor rezervisan funkcijom malloc() */
  free(p);

  return 0;
}
```

Rešenje 2.16

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

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

  /* Velicina alocirane memorije */
  int alocirano;

  /* Broj elemenata niza */
  int n;

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

  /* Inicijalizacija */
  alocirano = n = 0;

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

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

      /* Vrsi se realokacija memorije sa novom velicinom */
      /* Resenje sa funkcijom malloc() */
      b = (int *) malloc(alocirano * sizeof(int));

      if (b == NULL) {
        fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
        free(a);
        exit(EXIT_FAILURE);
      }

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

      free(a);

      /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
         bloka koji je prilikom alokacije zapamcen u promenljivoj b
    */
```

```
      a = b;

      /*******************************************************
        Resenje sa funkcijom realloc()

        Zbog funkcije realloc je neophodno da i u prvoj iteraciji
        "a" bude inicijalizovano na NULL

        a = (int*) realloc(a,alocirano*sizeof(int));
        if(a == NULL) {
          fprintf(stderr, "realloc(): ");
          fprintf(stderr, "greska pri alokaciji memorije.\n");
          exit(EXIT_FAILURE);
        }
      *******************************************************/
    }

    a[n++] = x;

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

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.17

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

#define MAX 1000

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

  /* Proverava se da li je memorija uspesno alocirana */
  if (p == NULL) {
```

```
16      fprintf(stderr, "malloc(): ");
        fprintf(stderr, "greska pri alokaciji memorije.\n");
18      exit(EXIT_FAILURE);
    }
20
    /* Kopiraju se i nadovezuju niske karaktera */
22  strcpy(p, s);
    strcat(p, t);
24
    return p;
26  }

28  int main()
    {
30    char *s = NULL;
    char s1[MAX], s2[MAX];
32
    printf("Unesite dve niske karaktera:\n");
34  scanf("%s", s1);
    scanf("%s", s2);
36
    /* Poziva se funkcija koja nadovezuje niske */
38  s = nadovezi(s1, s2);

40  /* Prikazuje se rezultat */
    printf("Nadovezane niske: %s\n", s);
42
    /* Oslobadja se memorija alocirana u funkciji nadovezi() */
44  free(s);

46  return 0;
    }
```

Rešenje 2.18

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

5   int main()
    {
7     int i, j;

9     /* Pokazivac na dinamicki alociran niz pokazivaca na vrste matrice
        */
      double **A = NULL;
11
      /* Broj vrsta i broj kolona */
13    int n = 0, m = 0;

15    /* Trag matice */
```

```
  double trag = 0;

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

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

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

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

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

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

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

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

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

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

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

  return 0;
```

```
}
```

Rešenje 2.19

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

/* Funkcija ucitava matricu sa ulaza */
void ucitaj_matricu(int **M, int n, int m)
{
  int i, j;

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

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

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

  /* Alocira se 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);
  }

  /* Alocira se 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]);
49      free(matrica);
        exit(EXIT_FAILURE);
51    }
  }
53
  printf("Unesite elemente matrice, vrstu po vrstu:\n");
55  ucitaj_matricu(matrica, n, m);

57  printf("Elementi ispod glavne dijagonale matrice:\n");
  ispisi_elemente_ispod_dijagonale(matrica, n, m);
59
  /* Oslobadja se dinamicki alocirana memorija za matricu. Prvo se
61    oslobadja memorija rezervisana za svaku vrstu */
  for (j = 0; j < n; j++)
63  free(matrica[j]);

65  /* Zatim se oslobadja memorija za niz pokazivaca na vrste matrice
    */
  free(matrica);
67
  return 0;
69}
```

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

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

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

32 return zbir;
}
34
/* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
36    datoteke */
void ucitaj_matricu(FILE * ulaz, float **m, int n)
38 {
   int i, j;
40
   for (i = 0; i < n; i++)
42   for (j = 0; j < n; j++)
       fscanf(ulaz, "%f", &m[i][j]);
44 }

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

52 for (i = 0; i < n; i++) {
     for (j = 0; j < n; j++)
54     printf("%.2f ", m[i][j]);
     printf("\n");
56 }
}
58
/* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
60 float **alociraj_memoriju(int n)
{
62 int i, j;
   float **m;
64
   m = (float **) malloc(n * sizeof(float *));
66 if (m == NULL) {
     fprintf(stderr, "malloc(): Neuspela alokacija\n");
68   exit(EXIT_FAILURE);
   }
70
   for (i = 0; i < n; i++) {
72   m[i] = (float *) malloc(n * sizeof(float));

74   if (m[i] == NULL) {
       printf("malloc(): neuspela alokacija memorije!\n");
76     for (j = 0; j < i; j++)
         free(m[i]);
```

```
78        free(m);
          exit(EXIT_FAILURE);
80      }
    }
82  return m;
}
84
/* Funckija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
86     n */
void oslobodi_memoriju(float **m, int n)
88 {
    int i;
90
    for (i = 0; i < n; i++)
92    free(m[i]);
    free(m);
94 }

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

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

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

118   /* Cita se dimenzija matrice */
    fscanf(ulaz, "%d", &n);
120
    /* Alocira se memorija */
122   a = alociraj_memoriju(n);

124   /* Ucitavaju se elementi matrice */
    ucitaj_matricu(ulaz, a, n);
126
    float zbir = zbir_ispod_sporedne_dijagonale(a, n);
128
```

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

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

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

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

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

  return 0;
}
```

Rešenje 2.26

```

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

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

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

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

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

```
31 {
     double a, b;
33   int n;

35   char ime_fje[6];

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

41   /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
       */
     if (argc < 2) {
43     printf("Greska: ");
       printf("Nedovoljan broj argumenata komandne linije.\n");
45     printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
              argv[0]);
47     exit(EXIT_FAILURE);
     }
49
     /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
51      komandnoj liniji */
     strcpy(ime_fje, argv[1]);
53
     /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
55    */
     if (strcmp(ime_fje, "sin") == 0)
57     fp = &sin;
     else if (strcmp(ime_fje, "cos") == 0)
59     fp = &cos;
     else if (strcmp(ime_fje, "tan") == 0)
61     fp = &tan;
     else if (strcmp(ime_fje, "atan") == 0)
63     fp = &atan;
     else if (strcmp(ime_fje, "acos") == 0)
65     fp = &acos;
     else if (strcmp(ime_fje, "asin") == 0)
67     fp = &asin;
     else if (strcmp(ime_fje, "exp") == 0)
69     fp = &exp;
     else if (strcmp(ime_fje, "log") == 0)
71     fp = &log;
     else if (strcmp(ime_fje, "log10") == 0)
73     fp = &log10;
     else if (strcmp(ime_fje, "sqrt") == 0)
75     fp = &sqrt;
     else if (strcmp(ime_fje, "floor") == 0)
77     fp = &floor;
     else if (strcmp(ime_fje, "ceil") == 0)
79     fp = &ceil;
     else if (strcmp(ime_fje, "sqr") == 0)
81     fp = &sqr;
```

```
  else {
    printf("Program jos uvek ne podrzava trazenu funkciju!\n");
    exit(EXIT_SUCCESS);
  }

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

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

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 broj ili broj -1 ukoliko broj nije pronađen.

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

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

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

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

Test 1

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

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

Test 2

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

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

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

Primer 1

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

Primer 2

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

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 unosi prezime studenta i prikazuju mu se informacije o prvom studentu sa unetim prezimenom. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

Primer 1

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

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

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

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

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

Test 1

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

dat.txt
  12 53
  2.342 34.1
  -0.3 23
  -1 23.1
  123.5 756.12

Izlaz:
  -0.3 23
```

Test 2

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

dat.txt
  12 53
  2.342 34.1
  -0.3 23
  -1 2.1
  123.5 756.12

Izlaz:
  -1 2.1
```

Test 3

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

dat.txt
  12 53
  2.342 34.1
  -0.3 0.23
  -1 2.1
  123.5 756.12

Izlaz:
  -0.3 0.23
```

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

Test 1

```
Izlaz:
 1.57031
```

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

Test 1

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

Izlaz:
 2
```

Test 2

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

Izlaz:
 -1
```

Test 3

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

Izlaz:
 3
```

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

Test 1

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

Izlaz:
 3
```

Test 2

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

Izlaz:
 4
```

Test 3

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

Izlaz:
 -1
```

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

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

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

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

Test 1

```
Ulaz:
 4

Izlaz:
 2 2
```

Test 2

```
Ulaz:
 17

Izlaz:
 4 4
```

Test 3

```
Ulaz:
 1031

Izlaz:
 10 10
```

**** Zadatak 3.10** U prvom kvadrantu dato je $1 \leq \texttt{N} \leq 10000$ duži svojim koordinatama (duži mogu da se seku, preklapaju, itd.). Napisati program koji pronalazi najmanji ugao $0 \leq \alpha \leq 90°$, na dve decimale, takav da je suma dužina duži sa obe strane polupoluprave iz koordinatnog početka pod uglom α jednak (neke duži bivaju presečene, a neke ne). Program prvo učitava broj `N`, a zatim i same koordinate temena duži. Uputstvo: *Vršiti binarnu pretragu intervala* $[0, 90°]$.

Primer 1

```
Interakcija programa:
 Unesi broj tacaka: 2
 Unesi koordinate tacaka:
 2 0 2 1
 1 2 2 2
 26.57
```

Primer 2

```
Interakcija programa:
 Unesi broj tacaka: 2
 Unesi koordinate tacaka:
 1 0 1 1
 0 1 1 1
 45
```

Primer 3

```
Interakcija programa:
 Unesi broj tacaka: 3
 Unesi koordinate tacaka:
 1 0 1 1
 2 0 2 1
 1 2 2 2
 26.57
```

3.2 Sortiranje

Zadatak 3.11 U datom nizu brojeva treba pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, ali neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati razliku pronađena dva broja. Uputstvo: *Prvo sortirati niz.*

Test 1

```
Ulaz:
 23 64 123 76 22 7

Izlaz:
 1
```

Test 2

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

Izlaz:
 0
```

Test 3

```
Ulaz:
 34 30

Izlaz:
 4
```

Zadatak 3.12 Dve niske su anagrami ako se sastoje od istog broja istih karaktera. Napisati program koji proverava da li su dve niske karaktera anagrami. Niske se zadaju sa standardnog ulaza i neće biti duže od 127 karaktera. Uputstvo: *Napisati funkciju koja sortira slova unutar niske karaktera, a zatim za sortirane niske proveriti da li su identične.*

Primer 1

```
Interakcija programa:
 Unesite prvu nisku  anagram
 Unesite drugu nisku  ramgana
 jesu
```

Primer 2

```
Interakcija programa:
 Unesite prvu nisku  anagram
 Unesite drugu nisku  anagrm
 nisu
```

Primer 3

```
Interakcija programa:
 Unesite prvu nisku  test
 Unesite drugu nisku  tset
 jesu
```

Zadatak 3.13 Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza i neće biti duži od 256 i kraći od jednog elemenata. Uputstvo: *Prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.*

Test 1

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

Izlaz:
 4
```

Test 2

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

Izlaz:
 2
```

Test 3

```
Ulaz:
 123

Izlaz:
 123
```

Zadatak 3.14 Napisati funkciju koja proverava da li u datom nizu postoje dva elementa čiji zbir je jednak zadatom celom broju. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz. Elementi niza se unose sve do kraja ulaza. Pretpostaviti da u niz neće biti uneto više od 256 brojeva. Uputstvo: *Prvo sortirati niz.*

Primer 1

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

Primer 2

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

Primer 3

```
Interakcija programa:
 Unesite trazeni zbir:  52
 Unesite elemente niza:
 52
 ne
```

Zadatak 3.15 Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži `selection`, `merge`, `quick`, `bubble`, `insertion` i `shell sort`. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim

nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije `n`.

Test 1

```
Poziv: time a.out 100000 -i -o

Izlaz:
 real 0m17.631s
 user 0m17.604s
 sys 0m0.000s
```

Test 2

```
Poziv: time a.out 100000 -b -r

Izlaz:
 real 0m0.005s
 user 0m0.004s
 sys 0m0.000s
```

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.

Primer 1

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

Primer 2

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

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

prvi-deo.txt
  Andrija Petrovic
  Anja Ilic
  Ivana Markovic
  Lazar Micic
  Nenad Brankovic
  Sofija Filipovic
  Vladimir Savic
  Uros Milic

drugi-deo.txt
  Aleksandra Cvetic
  Bojan Golubovic
  Dragan Markovic
  Filip Dukic
  Ivana Stankovic
  Marija Stankovic
  Ognjen Peric
```

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

Zadatak 3.18 Napisati funkcije koje sortiraju niz struktura tačaka na osnovu sledećih kriterijuma: (i) njihovog rastojanja od koordinatnog početka, (ii) `x` koordinata tačaka, (iii) `y` koordinata tačaka. Napisati program koji učitava niz tačaka iz datoteke čije se ime zadaje kao drugi argument komandne linije, i u zavisnosti od prisutnih opcija (prvi argument) u komandnoj liniji (`-o`, `-x` ili `-y`) sortira tačke po jednom od prethodna tri kriterijuma i rezultat upisuje u datoteku čije se ime zadaje kao treći argument komandne linije. U ulaznoj datoteci nije zadato više od 128 tačaka.

Test 1

```
Poziv: ./a.out -x in.txt out.txt

in.txt
  3 4
  11 6
  7 3
  2 82
  -1 6

out.txt
  -1 6
  2 82
  3 4
  7 3
  11 6
```

Test 2

```
Poziv: ./a.out -o in.txt out.txt

in.txt
  3 4
  11 6
  7 3
  2 82
  -1 6

out.txt
  3 4
  -1 6
  7 3
  11 6
  2 82
```

Zadatak 3.19 Napisati program koji učitava imena i prezimena građana (najviše njih 1000) iz datoteke `biracki-spisak.txt` i kreira biračke spiskove.

Jedan birački spisak je sortiran po imenu građana, a drugi po prezimenu. Program treba da ispisuje koliko građana ima isti redni broj u oba biračka spiska. Pretpostaviti da je za ime, odnosno prezime građana dovoljno 15 karaktera.

Test 1

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

IZLAZ:
  3
```

Test 2

```
BIRACKI-SPISAK.TXT
  Milan Milicevic

IZLAZ:
  1
```

Test 3

```
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI

IZLAZ:
  Problem pri otvaranju
  datoteke.
```

Zadatak 3.20 Definisati strukturu koja čuva imena, prezimena i godišta dece. Pretpostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

Test 1

```
POZIV: ./a.out in.txt out.txt

IN.OUT
  Petar Petrovic 2007
  Milica Antonic 2008
  Ana Petrovic 2007
  Ivana Ivanovic 2009
  Dragana Markovic 2010
  Marija Antic 2007
```

```
OUT.TXT
  Marija Antic 2007
  Ana Petrovic 2007
  Petar Petrovic 2007
  Milica Antonic 2008
  Ivana Ivanovic 2009
  Dragana Markovic 2010
```

Test 2

```
POZIV: ./a.out in.txt out.txt

IN.OUT
  Milijana Maric 2009
```

```
OUT.TXT
  Milijana Maric 2009
```

Zadatak 3.21 Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika tada sortirati ih po dužini ni-

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

Test 1

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

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

Zadatak 3.22 Napisati program koji simulira rad kase u prodavnici. Kupci prilaze kasi, a prodavac unošenjem bar-koda kupljenog proizvoda dodaje njegovu cenu na ukupan račun. Na kraju, program ispisuje ukupnu vrednost svih proizvoda. Sve artikle, zajedno sa bar-kodovima, prozivođačima i cenama učitati iz datoteke `artikli.txt`. Pretraživanje niza artikala vršiti binarnom pretragom.

Primer 1

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

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

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

  UKUPNO: 270.00 dinara.

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

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

  UKUPNO: 230.00 dinara.

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

 Kraj rada kase!
```

Zadatak 3.23 Napisati program koji iz datoteke `aktivnost.txt` čita podatke o aktivnostima studenata na praktikumima i u datoteke `dat1.txt`, `dat2.txt` i `dat3.txt` upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na

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

Test 1

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

DAT1.TXT
 Studenti sortirani po imenu
 leksikografski rastuce:
 Aleksandra Cvetic 4 6
 Andrija Petrovic 2 5
 Anja Ilic 3 1
 Bojan Golubovic 4 3
 Dragan Markovic 3 5
 Ivana Stankovic 3 1
 Lazar Micic 1 3
 Marija Stankovic 1 3
 Nenad Brankovic 2 4
 Ognjen Peric 1 2
 Uros Milic 2 5
```

```
DAT2.TXT
 Studenti sortirani po broju zadataka
 opadajuce, pa po duzini imena rastuce:
 Aleksandra Cvetic 4 6
 Uros Milic 2 5
 Dragan Markovic 3 5
 Andrija Petrovic 2 5
 Nenad Brankovic 2 4
 Lazar Micic 1 3
 Bojan Golubovic 4 3
 Marija Stankovic 1 3
 Ognjen Peric 1 2
 Anja Ilic 3 1
 Ivana Stankovic 3 1

DAT3.TXT
 Studenti sortirani po prisustvu
 opadajuce, pa po broju zadataka,
 pa po prezimenima leksikografski
 opadajuce:
 Aleksandra Cvetic 4 6
 Bojan Golubovic 4 3
 Dragan Markovic 3 5
 Ivana Stankovic 3 1
 Anja Ilic 3 1
 Andrija Petrovic 2 5
 Uros Milic 2 5
 Nenad Brankovic 2 4
 Marija Stankovic 1 3
 Lazar Micic 1 3
 Ognjen Peric 1 2
```

Zadatak 3.24 U datoteci `pesme.txt` nalaze se informacije o gledanosti pesama na Youtube-u. Format datoteke sa informacijama je sledeći:

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

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

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

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

Test 1

```
Poziv: ./a.out

PESME.TXT
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

Izlaz:
  Jelena - Sunce, 92321
  Mika - Kisa, 5341
  Ana - Nebo, 2342
  Pera - Ptice, 327
  Laza - Oblaci, 29
```

Test 2

```
Poziv: ./a.out -i

PESME.TXT
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

Izlaz:
  Ana - Nebo, 2342
  Jelena - Sunce, 92321
  Laza - Oblaci, 29
  Mika - Kisa, 5341
  Pera - Ptice, 327
```

Test 3

```
Poziv: ./a.out -n

PESME.TXT
  5
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
  Mika - Kisa, 5341

Izlaz:
  Mika - Kisa, 5341
  Ana - Nebo, 2342
  Laza - Oblaci, 29
  Pera - Ptice, 327
  Jelena - Sunce, 92321
```

**** Zadatak 3.25** Razmatrajmo dve operacije: operacija `U` je unos novog broja `x`, a operacija `N` određivanje `n`-tog po veličini od unetih brojeva. Implementirati program koji izvršava ove operacije. Može postojati najviše 100000 operacija unosa, a uneti elementi se mogu ponavljati, pri čemu se i ponavljanja računaju prilikom brojanja. Napomena: *Brojeve čuvati u sortiranom nizu i svaki naredni element umetati na svoje mesto.* Optimizovati program, ukoliko se zna da neće biti više od 500 različitih unetih brojeva.

Primer 1

```
Interakcija programa:
  Unesi niz operacija: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
  0 2 8 2 6
```

**** Zadatak 3.26** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

3 5 2 1

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

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

Test 1

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

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

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.27 Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime, a zatim se učitava jedan karakter i pronalazi i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati -1 na standardnom izlazu. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretaživanje niza vršiti bibliotečkom funkcijom `bsearch`. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

Test 1

ULAZ:
R

IZLAZ:
Dusko Radovic

Test 2

ULAZ:
E

IZLAZ:
Dobrica Eric

Test 3

ULAZ:
A

IZLAZ:
Mika Antic

Zadatak 3.28 Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma

sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva, ne veća od 100, a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu. Na standardnom izlazu ispisati odgovarajuću poruku.

Primer 1

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

Primer 2

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

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

Primer 1

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

Zadatak 3.30 Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

(a) leksikografski,

(b) po dužini.

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

Primer 1

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

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

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

Zadatak 3.32 Napisati program koji korišćenjem bibliotečke funkcije `qsort` sortira studente prema broju poena osvojenih na kolokvijumu. Ukoliko više studenata ima isti broj bodova, sortirati ih po prezimenu leksikografski rastuće. Korisnik potom unosi broj bodova i prikazuje mu se jedan od studenata sa tim brojem bodova ili poruka ukoliko nema takvog. Potom, sa standardnog ulaza, unosi se prezime traženog studenta i prikazuje se osoba sa tim prezimenom ili poruka da se nijedan student tako ne preziva. Za pretraživanje koristiti odgovarajuće bibliotečke funkcije. Podaci o studentima čitaju se iz datoteke čije se ime zadaje preko argumenata komandne linije. Za svakog studenta u datoteci postoje ime, prezime i bodovi osvojeni na kolokvijumu. Pretpostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

Primer 1

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

Ulazna datoteka (kolokvijum.txt):
 Aleksandra Cvetic 15
 Bojan Golubovic 30
 Dragan Markovic 25
 Filip Dukic 20
 Ivana Stankovic 25
 Marija Stankovic 15
 Ognjen Peric 20
 Uros Milic 10
 Andrija Petrovic 0
 Anja Ilic 5
 Ivana Markovic 5
 Lazar Micic 20
 Nenad Brankovic 15
```

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

Zadatak 3.33 Uraditi zadatak 3.12, ali korišćenjem bibliotečke `qsort` funkcije.

Zadatak 3.34 Napisati program koji sa standardnog ulaza učitava prvo ceo broj `n` ($n \leq 10$), a zatim niz `S` od `n` niski. Maksimalna dužina svake niske je 31 karakter. Sortirati niz `S` bibliotečkom funkcijom `qsort` i proveriti da li u njemu ima identičnih niski.

Primer 1

```
Interakcija programa:
 Unesite broj niski: 4
 Unesite niske:
 prog search sort search
 ima
```

Primer 2

```
Interakcija programa:
 Unesite broj niski: 3
 Unesite niske:
 test kol ispit
 nema
```

Primer 3

```
Interakcija programa:
 Unesite broj niski: 5
 Unesite niske:
 a ab abc abcd abcde
 nema
```

Zadatak 3.35 Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr15125`, `mm14001`), ime, prezime i broj poena. Ni ime, ni prezime neće biti duže od 20 karaktera. Napisati program koji korišćenjem funkcije `qsort` sortira studente po broju poena opadajuće, ukoliko je prisutna opcija `-p`, ili po nalogu, ukoliko je prisutna opcija `-n`. Studenti se po nalogu sortiraju tako što se sortiraju na osnovu godine, zatim na osnovu smera, i na kraju na osnovu broja indeksa. Sortirane studente upisati u datoteku `izlaz.txt`. Ukoliko je u komandnoj liniji uz opciju `-n` naveden i nalog

nekog studenta, funkcijom `bsearch` potražiti i prijaviti broj poena studenta sa tim nalogom.

Test 1

```
Poziv: ./a.out -n mm13321

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

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

Izlaz:
  mm13321 Marija Radic 12
```

Test 2

```
Poziv: /a.out -p

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

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

Zadatak 3.36 Definisati strukturu `Datum`. Napisati funkciju koja poredi dva datuma hronološki. Potom, napisati i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma), sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza postoje među prethodno unetim datumima. Datumi se učitavaju sve do kraja ulaza.

Primer 1

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

DATOTEKA.TXT
  1.1.2013.
  13.12.2016.
  11.11.2011.
  3.5.2015.
  5.2.2009.
```

```
Interakcija programa:
  Unesi sledeci datum: 13.12.2016.
  postoji
  Unesi sledeci datum: 10.5.2015.
  ne postoji
  Unesi sledeci datum: 5.2.2009.
  postoji
```

Zadatak 3.37 Za zadatu celobrojnu matricu dimenzije $n \times m$ napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrsti. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu.

Test 1

```
Interakcija programa:
 Unesite dimenzije matrice: 3 2
 Unesite elemente matrice po vrstama:
 6 -5
 -4 3
 2 1
 Sortirana matrica je:
 -4 3
 6 -5
 2 1
```

Test 2

```
Interakcija programa:
 Unesite dimenzije matrice: 4 4
 Unesite elemente matrice po vrstama:
 34 12 54 642
 1 2 3 4
 53 2 1 5
 54 23 5 671
 Sortirana matrica je:
 1 2 3 4
 53 2 1 5
 34 12 54 642
 54 23 5 671
```

Zadatak 3.38 Za zadatu kvadratnu matricu dimenzije n napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu.

Primer 1

```
Interakcija programa:
 Unesite dimenziju matrice: 2
 Unesite elemente matrice po vrstama:
 6 -5
 -4 3
 Sortirana matrica je:
 -5 6
 3 -4
```

Primer 2

```
Interakcija programa:
 Unesite dimenziju matrice: 4
 Unesite elemente matrice po vrstama:
 34 12 54 642
 1 2 3 4
 53 2 1 5
 54 23 5 671
 Sortirana matrica je:
 12 34 54 642
 2 1 3 4
 2 53 1 5
 23 54 5 671
```

3.4 Rešenja

Rešenje 3.1

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

/* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
   -lrt zbog funkcije clock_gettime() */

/* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u
```

```
   njemu element x. Pretraga se vrsi prostom iteracijom kroz niz. Ako
   se element pronadje funkcija vraca indeks pozicije na kojoj je
   pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
   pronadjen u nizu, funkcija vraca -1, kao indikator neuspesne
   pretrage. */
int linearna_pretraga(int a[], int n, int x)
{
  int i;
  for (i = 0; i < n; i++)
    if (a[i] == x)
      return i;
  return -1;
}

/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
   pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
int binarna_pretraga(int a[], int n, int x)
{
  int levi = 0;
  int desni = n - 1;
  int srednji;
  /* Dokle god je indeks levi levo od indeksa desni */
  while (levi <= desni) {
    /* Srednji indeks je njihova aritmeticka sredina */
    srednji = (levi + desni) / 2;
    /* Ako je element sa sredisnjim indeksom veci od x, tada se x
       mora nalaziti u levoj polovini niza */
    if (x < a[srednji])
      desni = srednji - 1;
    /* Ako je element sa sredisnjim indeksom manji od x, tada se x
       mora nalaziti u desnoj polovini niza */
    else if (x > a[srednji])
      levi = srednji + 1;
    else
      /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
         x pronadjen na poziciji srednji */
      return srednji;
  }
  /* Ako element x nije pronadjen, vraca se -1 */
  return -1;
}

/* 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;
  int desni = n - 1;
  int srednji;
  /* Dokle god je indeks levi levo od indeksa desni...  */
  while (levi <= desni) {
    /* Ako je trazeni element manji od pocetnog ili veci od
```

```
62          poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
            nije u tom delu niza. Ova provera je neophodna, da se ne bi
64          dogodilo da se prilikom izracunavanja indeksa srednji izadje
            izvan opsega indeksa [levi,desni] */
66        if (x < a[levi] || x > a[desni])
            return -1;
68        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
            a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
70          jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
            desni). Ova provera je neophodna, jer bi se u suprotnom
72          prilikom izracunavanja indeksa srednji pojavilo deljenje
            nulom. */
74        else if (a[levi] == a[desni])
            return levi;
76        /* Racunanje srednjeg indeksa */
          srednji =
78            levi +
              ((double) (x - a[levi]) / (a[desni] - a[levi])) *
80            (desni - levi);
          /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
82          verovatno biti blize trazenoj vrednosti nego da je prosto uvek
            uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
84          porediti sa pretragom recnika: ako neko trazi rec na slovo 'B
        ',
            sigurno nece da otvori recnik na polovini, vec verovatno negde
86          blize pocetku. */
          /* Ako je element sa indeksom srednji veci od trazenog, tada se
88          trazeni element mora nalaziti u levoj polovini niza */
          if (x < a[srednji])
90          desni = srednji - 1;
          /* Ako je element sa indeksom srednji manji od trazenog, tada se
92          trazeni element mora nalaziti u desnoj polovini niza */
          else if (x > a[srednji])
94          levi = srednji + 1;
          else
96          /* Ako je element sa indeksom srednji jednak trazenom, onda se
              pretraga zavrsava na poziciji srednji */
98          return srednji;
        }
100     /* U slucaju neuspesne pretrage vraca se -1 */
        return -1;
102   }

104   int main(int argc, char **argv)
      {
106     int a[MAX];
        int n, i, x;
108     struct timespec time1, time2, time3, time4, time5, time6;
        FILE *f;
110
        /* Provera argumenata komandne linije */
112     if (argc != 3) {
```

```
    fprintf(stderr,
114             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
    ;
    exit(EXIT_FAILURE);
116 }

118 /* Dimenzija niza */
    n = atoi(argv[1]);
120 if (n > MAX || n <= 0) {
    fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
122 exit(EXIT_FAILURE);
    }
124
    /* Broj koji se trazi */
126 x = atoi(argv[2]);

128 /* Elementi niza se generisu slucajno, tako da je svaki sledeci
       veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
130    pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
       izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
132    godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
    srandom(time(NULL));
134 for (i = 0; i < n; i++)
      a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
136
    /* Lineara pretraga */
138 printf("Linearna pretraga\n");
    /* Vreme proteklo od Nove godine 1970 */
140 clock_gettime(CLOCK_REALTIME, &time1);
    i = linearna_pretraga(a, n, x);
142 /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
       linearnu pretragu */
144 clock_gettime(CLOCK_REALTIME, &time2);
    if (i == -1)
146   printf("Element nije u nizu\n");
    else
148   printf("Element je u nizu na poziciji %d\n", i);
    printf("----------------------------------\n");
150
    /* Binarna pretraga */
152 printf("Binarna pretraga\n");
    clock_gettime(CLOCK_REALTIME, &time3);
154 i = binarna_pretraga(a, n, x);
    clock_gettime(CLOCK_REALTIME, &time4);
156 if (i == -1)
      printf("Element nije u nizu\n");
158 else
      printf("Element je u nizu na poziciji %d\n", i);
160 printf("----------------------------------\n");

162 /* Interpolaciona pretraga */
    printf("Interpolaciona pretraga\n");
```

```
164   clock_gettime(CLOCK_REALTIME, &time5);
      i = interpolaciona_pretraga(a, n, x);
166   clock_gettime(CLOCK_REALTIME, &time6);
      if (i == -1)
168     printf("Element nije u nizu\n");
      else
170     printf("Element je u nizu na poziciji %d\n", i);
      printf("---------------------------------\n");
172
      /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
174   if ((f = fopen("vremena.txt", "a")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
176     exit(EXIT_FAILURE);
      }
178
      fprintf(f, "Dimenzija niza: %d\n", n);
180   fprintf(f, "\tLinearna pretraga:%10ld ns\n",
              (time2.tv_sec - time1.tv_sec) * 1000000000 +
182           time2.tv_nsec - time1.tv_nsec);
      fprintf(f, "\tBinarna: %19ld ns\n",
184           (time4.tv_sec - time3.tv_sec) * 1000000000 +
              time4.tv_nsec - time3.tv_nsec);
186   fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
              (time6.tv_sec - time5.tv_sec) * 1000000000 +
188           time6.tv_nsec - time5.tv_nsec);

190   /* Zatvaranje datoteke */
      fclose(f);
192
      return 0;
194 }
```

Rešenje 3.2

```
  #include <stdio.h>
2
  #define MAX 1024
4
  int lin_pretraga_rek_sufiks(int a[], int n, int x)
6 {
    int tmp;
8   /* Izlaz iz rekurzije */
    if (n <= 0)
10    return -1;
    /* Ako je prvi element trazeni */
12  if (a[0] == x)
      return 0;
14  /* Pretraga ostatka niza */
    tmp = lin_pretraga_rek_sufiks(a + 1, n - 1, x);
16  return tmp < 0 ? tmp : tmp + 1;
  }
```

```
18
int lin_pretraga_rek_prefiks(int a[], int n, int x)
20 {
  /* Izlaz iz rekurzije */
22  if (n <= 0)
    return -1;
24  /* Ako je poslednji element trazeni */
  if (a[n - 1] == x)
26    return n - 1;
  /* Pretraga ostatka niza */
28  return lin_pretraga_rek_prefiks(a, n - 1, x);
}
30
int bin_pretraga_rek(int a[], int l, int d, int x)
32 {
  int srednji;
34  if (l > d)
    return -1;
36  /* Sredisnja pozicija na kojoj se trazi vrednost x */
  srednji = (l + d) / 2;
38  /* Ako je element na sredisnjoj poziciji trazeni */
  if (a[srednji] == x)
40    return srednji;
  /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
42     pretrazuje se desna polovina niza */
  if (a[srednji] < x)
44    return bin_pretraga_rek(a, srednji + 1, d, x);
  /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
46     pretrazuje se leva polovina niza */
  else
48    return bin_pretraga_rek(a, l, srednji - 1, x);
}
50

52 int interp_pretraga_rek(int a[], int l, int d, int x)
{
54  int p;
  if (x < a[l] || x > a[d])
56    return -1;
  if (a[d] == a[l])
58    return l;
  /* Pozicija na kojoj se trazi vrednost x */
60  p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
  if (a[p] == x)
62    return p;
  if (a[p] < x)
64    return interp_pretraga_rek(a, p + 1, d, x);
  else
66    return interp_pretraga_rek(a, l, p - 1, x);
}
68
int main()
```

```
{
  int a[MAX];
  int x;
  int i, indeks;

  /* Ucitavanje trazenog broja */
  printf("Unesite trazeni broj: ");
  scanf("%d", &x);

  /* Ucitavanje elemenata niza sve do kraja ulaza - ocekuje se da
     korisnik pritisne CTRL+D za naznaku kraja */
  i = 0;
  printf("Unesite sortiran niz elemenata: ");
  while (scanf("%d", &a[i]) == 1) {
    i++;
  }

  /* Linearna pretraga */
  printf("Linearna pretraga\n");
  indeks = lin_pretraga_rek_sufiks(a, i, x);
  if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
  else
    printf("Pozicija elementa je %d.\n", indeks);

  /* Binarna pretraga */
  printf("Binarna pretraga\n");
  indeks = bin_pretraga_rek(a, 0, i - 1, x);
  if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
  else
    printf("Pozicija elementa je %d.\n", indeks);

  /* Interpolaciona pretraga */
  printf("Interpolaciona pretraga\n");
  indeks = interp_pretraga_rek(a, 0, i - 1, x);
  if (indeks == -1)
    printf("Element se ne nalazi u nizu.\n");
  else
    printf("Pozicija elementa je %d.\n", indeks);

  return 0;
}
```

Rešenje 3.3

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

#define MAX_STUDENATA 128
```

```c
#define MAX_DUZINA 16

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

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

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

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

```
    if (strcmp(a[i].prezime, x) == 0)
59      return i;
  return -1;
61}

63/* Main funkcija mora imati argumente jer se ime datoteke prosledjuje
     kao argument komandne linije */
65int main(int argc, char *argv[])
{
67  Student dosije[MAX_STUDENATA];
  FILE *fin = NULL;
69  int i;
  int br_studenata = 0;
71  long trazen_indeks = 0;
  char trazeno_prezime[MAX_DUZINA];
73
  /* Provera da li je korisnik prilikom poziva programa prosledio ime
75     datoteke sa informacijama o studentima */
  if (argc != 2) {
77    fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke\n",
79          argv[0]);
    exit(EXIT_FAILURE);
81  }

83  /* Otvaranje datoteke */
  fin = fopen(argv[1], "r");
85  if (fin == NULL) {
    fprintf(stderr,
87          "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
    exit(EXIT_FAILURE);
89  }

91  /* Citanje se vrsi sve dok postoji red sa informacijama o studentu
      */
  i = 0;
93  while (1) {
    if (i == MAX_STUDENATA)
95    break;
    if (fscanf
97      (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
        dosije[i].prezime) != 3)
99    break;
    i++;
101 }
  br_studenata = i;
103
  /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
105 fclose(fin);

107 /* Unos indeksa koji se binarno trazi u nizu */
  printf("Unesite indeks studenta cije informacije zelite: ");
```

```
  scanf("%ld", &trazen_indeks);
  i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
  /* Rezultat binarne pretrage */
  if (i == -1)
    printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
  else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

  /* Unos prezimena koje se linearno trazi u nizu */
  printf("Unesite prezime studenta cije informacije zelite: ");
  scanf("%s", trazeno_prezime);
  i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
  /* Rezultat linearne pretrage */
  if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
  else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

  return 0;
}
```

Rešenje 3.4

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

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

typedef struct {
  long indeks;
  char ime[MAX_DUZINA];
  char prezime[MAX_DUZINA];
} Student;

int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
{
  /* Ako je pozicija elementa na levom kraju veca od pozicije
     elementa na desnom kraju dela niza koji se pretrazuje, onda se
     zapravo pretrazuje prazan deo niza. U praznom delu niza nema
     trazenog elementa pa se vraca -1 */
  if (levi > desni)
    return -1;
  /* Racunanje pozicije srednjeg elementa */
  int srednji = (levi + desni) / 2;
  /* Da li je srednji bas onaj trazeni */
  if (a[srednji].indeks == x) {
```

```
    return srednji;
  }
  /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
     poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
     je poznato da je niz sortiran po indeksu u rastucem poretku. */
  if (x < a[srednji].indeks)
    return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
  /* Inace ga treba traziti u desnoj polovini */
  else
    return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
}

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

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

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

  /* Provera da li je korisnik prilikom poziva prosledio ime datoteke
     sa informacijama o studentima */
  if (argc != 2) {
    fprintf(stderr,
```

```
            "Greska: Program se poziva sa %s ime_datoteke\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

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

  /* Citanje se vrsi sve dok postoji sledeci red sa informacijama o
     studentu */
  i = 0;
  while (1) {
    if (i == MAX_STUDENATA)
      break;
    if (fscanf
        (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
         dosije[i].prezime) != 3)
      break;
    i++;
  }
  br_studenata = i;

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

  /* Unos indeksa koji se binarno trazi u nizu */
  printf("Unesite indeks studenta cije informacije zelite: ");
  scanf("%ld", &trazen_indeks);
  i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
                                  trazen_indeks);
  if (i == -1)
    printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
  else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
           dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

  /* Unos prezimena koje se linearno trazi u nizu */
  printf("Unesite prezime studenta cije informacije zelite: ");
  scanf("%s", trazeno_prezime);
  i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
                                      trazeno_prezime);
  if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
  else
    printf
        ("Prvi takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
```

```
132   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;

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

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

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

```

```
  /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
     dodatnih argumenata */
  if (argc == 2)
    /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
    najbliza = najbliza_koordinatnom(tacke, n);
  /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
     pitanju opcija -x */
  else if (strcmp(argv[2], "-x") == 0)
    /* Racuna se rastojanje u odnosu na x osu */
    najbliza = najbliza_x_osi(tacke, n);
  /* Ako je u pitanju opcija -y */
  else if (strcmp(argv[2], "-y") == 0)
    /* Racuna se rastojanje u odnosu na y osu */
    najbliza = najbliza_y_osi(tacke, n);
  else {
    /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
       korisnika i prekida se izvrsavanje programa */
    fprintf(stderr, "Pogresna opcija\n");
    return EXIT_FAILURE;
  }

  /* Stampanje koordinata trazene tacke */
  printf("%g %g\n", najbliza.x, najbliza.y);

  /* Zatvaranje datoteke */
  fclose(ulaz);

  return 0;
}
```

Rešenje 3.6

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

/* Tacnost */
#define EPS 0.001

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

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

  /* Sve dok se ne pronadje trazena vrednost argumenta */
  while (1) {
    /* Polovi se interval */
    s = (l + d) / 2;
```

```
    /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
       tacnosti, prekida se pretraga */
    if (fabs(cos(s)) < EPS) {
      break;
    }
    /* Ako je nula u levom delu intervala, nastavlja se pretraga na
       [l, s] */
    if (cos(l) * cos(s) < 0)
      d = s;
    else
      /* Inace, na intervalu [s, d] */
      l = s;
  }

  /* Stampanje vrednost trazene tacke */
  printf("%g\n", s);

  return 0;
}
```

Rešenje 3.7

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

#define MAX 256

int prvi_veci_od_nule(int niz[], int n)
{
  /* Granice pretrage */
  int l = 0, d = n - 1;
  int s;
  /* Sve dok je leva manja od desne granice */
  while (l <= d) {
    /* Racuna se sredisnja pozicija */
    s = (l + d) / 2;
    /* Ako je broj na toj poziciji veci od nule, a eventualni njegov
       prethodnik manji ili jednak nuli, pretraga je zavrsena */
    if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
      return s;
    /* U slucaju broja manjeg ili jednakog nuli, pretrazuje se desna
       polovina niza */
    if (niz[s] <= 0)
      l = s + 1;
    /* A inace, leva polovina */
    else
      d = s - 1;
  }
  return -1;
}
```

```
30 int main()
   {
32   int niz[MAX];
     int n = 0;
34
     /* Unos niza */
36   while (scanf("%d", &niz[n]) == 1)
       n++;
38
     /* Stampanje rezultata */
40   printf("%d\n", prvi_veci_od_nule(niz, n));

42   return 0;
   }
```

Rešenje 3.8

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAX 256
5
   int prvi_manji_od_nule(int niz[], int n)
7  {
     /* Granice pretrage */
9    int l = 0, d = n - 1;
     int s;
11   /* Sve dok je leva manja od desne granice */
     while (l <= d) {
13     /* Racuna se sredisnja pozicija */
       s = (l + d) / 2;
15     /* Ako je broj na toj poziciji manji od nule, a eventualni njegov
          prethodnik veci ili jednak nuli, pretraga se zavrsava */
17     if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
         return s;
19     /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
          niza */
21     if (niz[s] >= 0)
         l = s + 1;
23     /* A inace leva */
       else
25       d = s - 1;
     }
27   return -1;
   }
29
   int main()
31 {
     int niz[MAX];
33   int n = 0;
```

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

39    /* Stampanje rezultata */
      printf("%d\n", prvi_manji_od_nule(niz, n));
41
      return 0;
43  }
```

Rešenje 3.9

```
1   #include <stdio.h>
    #include <stdlib.h>
3
    unsigned int logaritam_a(unsigned int x)
5   {
      /* Izlaz iz rekurzije */
7     if (x == 1)
        return 0;
9     /* Rekurzivni korak */
      return 1 + logaritam_a(x >> 1);
11  }

13  unsigned int logaritam_b(unsigned int x)
    {
15    /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
         najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
17       do 31 */
      int d = 0, l = sizeof(unsigned int) * 8 - 1;
19    int s;
      /* Sve dok je desna granica pretrage desnije od leve */
21    while (d <= l) {
        /* Racuna se sredisnja pozicija */
23      s = (l + d) / 2;
        /* Proverava se da li je na toj poziciji trazena jedinica */
25      if ((1 << s) <= x && (1 << (s + 1)) > x)
          return s;
27      /* Pretraga desne polovine binarnog zapisa */
        if ((1 << s) > x)
29        l = s - 1;
        /* Pretraga leve polovine binarnog zapisa */
31      else
          d = s + 1;
33    }
      return s;
35  }

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

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

53    return 0;
}
```

Rešenje 3.11

```
1  #include<stdio.h>
   #define MAX 256
3
   /* Iterativna verzija funkcije koja sortira niz celih brojeva,
5     primenom algoritma Selection Sort */
   void selectionSort(int a[], int n)
7  {
     int i, j;
9    int min;
     int pom;
11
     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
13      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
15      min, na kojoj se nalazio najmanji od gore navedenih elemenata.
        */
     for (i = 0; i < n - 1; i++) {
17     /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
          najmanji od elemenata a[i],...,a[n-1]. */
19     min = i;
       for (j = i + 1; j < n; j++)
21       if (a[j] < a[min])
           min = j;
23     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
          su (i) i min razliciti, inace je nepotrebno. */
25     if (min != i) {
         pom = a[i];
27       a[i] = a[min];
         a[min] = pom;
29     }
     }
31 }
```

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


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

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

  /* Sortiranje */
  selectionSort(a, i);

  /* Ispis rezultata */
  printf("%d\n", najmanje_rastojanje(a, i));

  return 0;
}
```

Rešenje 3.12

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

#define MAX_DIM 128

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

```
      s[min] = pom;
    }
  }
}

/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
{
  int i;

  /* Ako dve niske imaju razlicit broj karaktera onda one nisu
     anagrami */
  if (strlen(s) != strlen(t))
    return 0;

  /* Sortiramo niske */
  selectionSort(s);
  selectionSort(t);

  /* Dve sortirane niske su anagrami ako i samo ako su jednake */
  for (i = 0; s[i] != '\0'; i++)
    if (s[i] != t[i])
      return 0;
  return 1;
}

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

  /* Ucitavanje niski sa ulaza */
  printf("Unesite prvu nisku: ");
  scanf("%s", s);
  printf("Unesite drugu nisku: ");
  scanf("%s", t);

  /* Poziv funkcije */
  if (anagrami(s, t))
    printf("jesu\n");
  else
    printf("nisu\n");

  return 0;
}
```

Rešenje 3.13

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

/* Funkcija za sortiranje niza */
```

```c
void selectionSort(int 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 za odredjivanje onog elementa sortiranog niza koji se
   najvise puta pojavio u tom nizu */
int najvise_puta(int a[], int n)
{
  int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
  /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
  for (i = 0; i < n; i = j) {
    br_pojava = 1;
    for (j = i + 1; j < n && a[i] == a[j]; j++)
      br_pojava++;
    /* Ispitivanje da li se do tog trenutka i-ti element pojavio
       najvise puta u nizu */
    if (br_pojava > max_br_pojava) {
      max_br_pojava = br_pojava;
      i_max_pojava = i;
    }
  }
  /* Vraca se element koji se najvise puta pojavio u nizu */
  return a[i_max_pojava];
}

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

  /* Ucitavanje elemenata niza sve do kraja ulaza */
  i = 0;
  while (scanf("%d", &a[i]) != EOF)
    i++;

  /* Niz se sortira */
  selectionSort(a, i);

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

```
57
  return 0;
59 }
```

Rešenje 3.14

```
1  #include<stdio.h>
   #define MAX_DIM 256
3
   /* Funkcija za sortiranje niza */
5  void selectionSort(int a[], int n)
   {
7    int i, j, min, pom;
     for (i = 0; i < n - 1; i++) {
9      min = i;
       for (j = i + 1; j < n; j++)
11       if (a[j] < a[min])
           min = j;
13     if (min != i) {
         pom = a[i];
15       a[i] = a[min];
         a[min] = pom;
17     }
     }
19 }

21 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
      u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
23    poretku */
   int binarna_pretraga(int a[], int n, int x)
25 {
     int levi = 0, desni = n - 1, srednji;
27
     while (levi <= desni) {
29     srednji = (levi + desni) / 2;
       if (a[srednji] == x)
31       return 1;
       else if (a[srednji] > x)
33       desni = srednji - 1;
       else if (a[srednji] < x)
35       levi = srednji + 1;
     }
37   return 0;
   }
39
   int main()
41 {
     int a[MAX_DIM], n = 0, zbir, i;
43
     /* Ucitava se trazeni zbir */
45   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 3.15

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

/* Selection sort */
void selectionsort(int a[], int n);
/* Insertion sort */
void insertionsort(int a[], int n);
/* Bubble sort */
void bubblesort(int a[], int n);
/* Shell sort */
void shellsort(int a[], int n);
/* Merge sort */
void mergesort(int a[], int l, int r);
/* Quick sort */
void quicksort(int a[], int l, int r);

#endif
```

```
/* Datoteka sort.c */

#include "sort.h"

```

```
#define MAX 1000000

/* Funkcija sortira niz celih brojeva metodom sortiranja izborom.
   Ideja algoritma je sledeca: U svakoj iteraciji pronalazi se
   najmanji element i premesta se na pocetak niza. Dakle, u prvoj
   iteraciji, pronalazi se najmanji element, i dovodi na nulto mesto
   u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
   pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
   dovodi na i+1 poziciju. */
void selectionsort(int a[], int n)
{
  int i, j;
  int min;
  int pom;

  /* U svakoj iteraciji ove petlje pronalazi se najmanji element
     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
     poziciju i, dok se element na pozciji i premesta na poziciju
     min, na kojoj se nalazio najmanji od 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 treba element a[i] umetnuti na pravu poziciju medju
   prvih i elemenata tako da se dobije niz duzine i+1 koji je
   sortiran. Ovo se radi tako sto se i-ti element najpre uporedi sa
   njegovim prvim levim susedom (a[i-1]). Ako je a[i] vece, tada je
   on vec na pravom mestu, i niz a[0],a[1],...,a[i] je sortiran, pa
   se moze preci na sledecu iteraciju. Ako je a[i-1] vece, tada se
   zamenjuju a[i] i a[i-1], a zatim se proverava da li je potrebno
   dalje potiskivanje elementa u levo, poredeci ga sa njegovim novim
   levim susedom. Ovim uzastopnim premestanjem se a[i] umece na pravo
   mesto u nizu. */
void insertionsort(int a[], int n)
```

```
{
  int i, j;

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

    /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
       potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
       bude sortiran. Indeks j je trenutna pozicija na kojoj se
       element koji se umece nalazi. Petlja se zavrsava ili kada
       element dodje do levog kraja (j==0) ili kada se naidje na
       element a[j-1] koji je manji od a[j]. */
    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: prolazi se 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 isti 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" potiskuje se najveci element medju elementima
       od a[0] do a[i-1] na poziciju i-1 uporedjujuci susedne
       elemente niza i potiskujuci veci u desno */
    for (j = 0, ind = 0; j < i - 1; j++)
      if (a[j] > a[j + 1]) {
        int temp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = temp;

        /* Promenljiva ind registruje da je bilo premestanja. Samo u
           tom slucaju ima smisla ici na sledecu iteraciju, jer ako
           nije bilo premestanja, znaci da su svi elementi vec u
           dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
           niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
```

```
108             ispravno, ali manje efikasano, jer bi se cesto nepotrebno
                vrsila mnoga uporedjivanja, kada je vec jasno da je
110             sortiranje zavrseno. */
          ind = 1;
112       }
  }
114
  /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
116     dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
     sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
118     prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji
     od n (sto omogucuje razmenu udaljenih elemenata) i tako se dobija
120     h-sortiran niz, tj. niz u kome su elementi na rastojanju h
     sortirani, mada susedni elementi to ne moraju biti. U drugom
122     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
124     dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
     nacina njegovog smanjivanja menja u nekim slucajevima brzinu
126     algoritma, ali bilo koja vrednost ce rezultovati ispravnim
     sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
128     vrednost 1. */
  void shellsort(int a[], int n)
130 {
    int h = n / 2, i, j;
132   while (h > 0) {
      /* Insertion sort sa korakom h */
134     for (i = h; i < n; i++) {
        int temp = a[i];
136       j = i;
        while (j >= h && a[j - h] > temp) {
138       a[j] = a[j - h];
          j -= h;
140     }
        a[j] = temp;
142   }
      h = h / 2;
144 }
  }
146
  /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
148     se vrsi od elementa na poziciji l do onog na poziciji d. Na
     pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a d je
150     jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
     dve polovine, levu i desnu, koje zatim rekurzivno sortira. Od ova
152     dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
     istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
154     elementa koji se smesta u pomocni niz. Na kraju algoritma,
     sortirani elementi su u pomocnom nizu, koji se kopira u originalni
156     niz. */
  void mergesort(int a[], int l, int d)
158 {
    int s;
```

```
160   static int b[MAX];            /* Pomocni niz */
      int i, j, k;
162
      /* Izlaz iz rekurzije */
164   if (l >= d)
        return;
166
      /* Odredjivanje sredisnjeg indeksa */
168   s = (l + d) / 2;

170   /* Rekurzivni pozivi */
      mergesort(a, l, s);
172   mergesort(a, s + 1, d);

174   /* Inicijalizacija indeksa. Indeks i prolazi krozi levu polovinu
         niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
176      prolazi kroz pomocni niz b[] */
      i = l;
178   j = s + 1;
      k = 0;
180
      /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
182   while (i <= s && j <= d) {
        if (a[i] < a[j])
184       b[k++] = a[i++];
        else
186       b[k++] = a[j++];
      }
188
      /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive j
190      iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
         polovine niza */
192   while (i <= s)
        b[k++] = a[i++];
194
      /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive i
196      iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
         polovine niza */
198   while (j <= d)
        b[k++] = a[j++];
200
      /* Prepisuje se "ucesljani" niz u originalni niz */
202   for (k = 0, i = l; i <= d; i++, k++)
        a[i] = b[k];
204 }

206 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
    void swap(int a[], int i, int j)
208 {
      int tmp = a[i];
210   a[i] = a[j];
      a[j] = tmp;
```

```
}


/* Funkcija sortira deo niza brojeva a izmedju pozicija l i r. Njena
   ideja sortiranja je izbor jednog elementa niza, koji se naziva
   pivot, i koji se dovodi na svoje mesto. Posle ovog koraka, svi
   elementi levo od njega bice manji, a svi desno bice veci od njega.
   Kako je pivot doveden na svoje mesto, da bi niz bio kompletno
   sortiran, potrebno je sortirati elemente levo (manje) od njega, i
   elemente desno (vece). Kako su dimenzije ova dva podniza manje od
   dimenzije pocetnog niza koji je trebalo sortirati, ovaj deo moze
   se uraditi rekurzivno. */
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 levo od
     pivot_position (izuzev same pozicije l) su strogo manji od
     pivota. Kada se pronadje neki element manji od pivota, uvecava
     se promenljiva 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);

  /* Postavljanje pivota na svoje mesto */
  swap(a, l, pivot_position);

  /* Rekurzivno sortiranje elementa manjih od pivota */
  quicksort(a, l, pivot_position - 1);
  /* Rekurzivno sortiranje elementa vecih od pivota */
  quicksort(a, pivot_position + 1, r);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sort.h"

/* Maksimalna duzina niza */
#define MAX 1000000

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

```
/*************************************************************
   tip_sortiranja == 0 => selectionsort, (podrazumevano)
   tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
   tip_sortiranja == 2 => bubblesort,    -b opcija komandne linije
   tip_sortiranja == 3 => shellsort,     -s opcija komandne linije
   tip_sortiranja == 4 => mergesort,     -m opcija komandne linije
   tip_sortiranja == 5 => quicksort,     -q opcija komandne linije
*************************************************************/
int tip_sortiranja = 0;
/*************************************************************
   tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
   tip_niza == 1 => rastuce sortirani nizovi,   -r opcija
   tip_niza == 2 => opadajuce soritrani nizovi, -o opcija
*************************************************************/
int tip_niza = 0;

/* Dimenzija niza koji se sortira */
int dimenzija;
int i;
int niz[MAX];

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

/* Ocitavanje opcija i argumenata prilikom poziva programa */
for (i = 1; i < argc; i++) {
  /* Ako je u pitanju opcija... */
  if (argv[i][0] == '-') {
    switch (argv[i][1]) {
    case 'i':
      tip_sortiranja = 1;
      break;
    case 'b':
      tip_sortiranja = 2;
      break;
    case 's':
      tip_sortiranja = 3;
      break;
    case 'm':
      tip_sortiranja = 4;
      break;
    case 'q':
      tip_sortiranja = 5;
      break;
    case 'r':
      tip_niza = 1;
      break;
    case 'o':
```

```
        tip_niza = 2;
64        break;
      default:
66        printf("Pogresna opcija -%c\n", argv[i][1]);
        return 1;
68        break;
      }
70    }
    /* Ako je u pitanju argument, onda je to duzina niza koji treba
72     da se sortira */
    else {
74      dimenzija = atoi(argv[i]);
      if (dimenzija <= 0 || dimenzija > MAX) {
76        fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
        exit(EXIT_FAILURE);
78      }
    }
80  }

82  /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
     niza dobijenom iz komandne linije. srandom() funkcija
84   obezbedjuje novi seed za pozivanje random funkcije, i kako
     generisani niz ne bi uvek bio isti seed je postavljen na tekuce
86   vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
     izmedju 0 i 99 */
88  srandom(time(NULL));
  if (tip_niza == 0)
90  for (i = 0; i < dimenzija; i++)
      niz[i] = random();
92  else if (tip_niza == 1)
    for (i = 0; i < dimenzija; i++)
94    niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
  else
96  for (i = 0; i < dimenzija; i++)
      niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
98
  /* Ispisivanje elemenata niza */
100 /*****************************************************************
    Ovaj deo je iskomentarisan jer sledeci ispis ne treba da se nadje
102 na standardnom izlazu. Njegova svrha je samo bila provera da li
    je niz generisan u skladu sa opcijama komandne linije.
104
    printf("Niz koji sortiramo je:\n");
106 for (i = 0; i < dimenzija; i++)
      printf("%d\n", niz[i]);
108 *****************************************************************/


110
  /* Sortiranje niza na odgovarajuci nacin */
112 if (tip_sortiranja == 0)
    selectionsort(niz, dimenzija);
114 else if (tip_sortiranja == 1)
```

```
    insertionsort(niz, dimenzija);
  else if (tip_sortiranja == 2)
    bubblesort(niz, dimenzija);
  else if (tip_sortiranja == 3)
    shellsort(niz, dimenzija);
  else if (tip_sortiranja == 4)
    mergesort(niz, 0, dimenzija - 1);
  else
    quicksort(niz, 0, dimenzija - 1);

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

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

  return 0;
}
```

Rešenje 3.16

```
#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;
    j++;
  }

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

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

  return 0;
}
```

Rešenje 3.17

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

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

```
  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, "koriscenje programa: %s fajl1 fajl2\n", argv[0])
    ;
16  exit(EXIT_FAILURE);
  }
18
  /* Otvaranje datoteke zadate prvim argumentom komandne linije */
20 fin1 = fopen(argv[1], "r");
  if (fin1 == NULL) {
22  fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
    exit(EXIT_FAILURE);
24 }

26 /* Otvaranje datoteke zadate drugim argumentom komandne linije */
  fin2 = fopen(argv[2], "r");
28 if (fin2 == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
30  exit(EXIT_FAILURE);
  }
32
  /* Otvaranje datoteke za upis rezultata */
34 fout = fopen("ceo-tok.txt", "w");
  if (fout == NULL) {
36  fprintf(stderr,
            "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
38  exit(EXIT_FAILURE);
  }
40
  /* Citanje narednog studenta iz prve datoteke */
42 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
    kraj1 = 1;
44
  /* Citanje narednog studenta iz druge datoteke */
46 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
    kraj2 = 1;
48
  /* Sve dok nije dostignut kraj neke datoteke */
50 while (!kraj1 && !kraj2) {
    if (strcmp(ime1, ime2) < 0) {
52    /* Ime i prezime iz prve datoteke je leksikografski ranije, i
         biva upisano u izlaznu datoteku */
54    fprintf(fout, "%s %s\n", ime1, prezime1);
      /* Citanje narednog studenta iz prve datoteke */
56    if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
        kraj1 = 1;
```

```
    } else {
      /* Ime i prezime iz druge datoteke je leksikografski ranije, i
         biva upisano u izlaznu datoteku */
      fprintf(fout, "%s %s\n", ime2, prezime2);
      /* Citanje narednog studenta iz druge datoteke */
      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;
    }
  }

  /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
     treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
   */
  while (!kraj1) {
    fprintf(fout, "%s %s\n", ime1, prezime1);
    if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
      kraj1 = 1;
  }

  /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
     datoteke, onda ima jos studenata u drugoj datoteci, koje treba
     prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
  while (!kraj2) {
    fprintf(fout, "%s %s\n", ime2, prezime2);
    if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
      kraj2 = 1;
  }

  /* Zatvaranje datoteka */
  fclose(fin1);
  fclose(fin2);
  fclose(fout);

  return 0;
}
```

Rešenje 3.18

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

#define MAX_BR_TACAKA 128

/* Struktura koja reprezentuje koordinate tacke */
typedef struct Tacka {
  int x;
  int y;
} Tacka;
```

```
13
   /* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka
15    (0,0) */
   float rastojanje(Tacka A)
17 {
     return sqrt(A.x * A.x + A.y * A.y);
19 }

21 /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
      pocetka */
23 void sortiraj_po_rastojanju(Tacka t[], int n)
   {
25   int min, i, j;
     Tacka tmp;
27
     for (i = 0; i < n - 1; i++) {
29     min = i;
       for (j = i + 1; j < n; j++) {
31       if (rastojanje(t[j]) < rastojanje(t[min])) {
           min = j;
33       }
       }
35     if (min != i) {
         tmp = t[i];
37       t[i] = t[min];
         t[min] = tmp;
39     }
     }
41 }

43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
   void sortiraj_po_x(Tacka t[], int n)
45 {
     int min, i, j;
47   Tacka tmp;

49   for (i = 0; i < n - 1; i++) {
       min = i;
51     for (j = i + 1; j < n; j++) {
         if (abs(t[j].x) < abs(t[min].x)) {
53         min = j;
         }
55     }
       if (min != i) {
57       tmp = t[i];
         t[i] = t[min];
59       t[min] = tmp;
       }
61   }
   }
63

   /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
```

```
65 void sortiraj_po_y(Tacka t[], int n)
   {
67   int min, i, j;
     Tacka tmp;
69
     for (i = 0; i < n - 1; i++) {
71     min = i;
       for (j = i + 1; j < n; j++) {
73       if (abs(t[j].y) < abs(t[min].y)) {
           min = j;
75       }
       }
77     if (min != i) {
         tmp = t[i];
79       t[i] = t[min];
         t[min] = tmp;
81     }
     }
83 }

85 int main(int argc, char *argv[])
   {
87   FILE *ulaz;
     FILE *izlaz;
89   Tacka tacke[MAX_BR_TACAKA];
     int i, n;
91
     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
93      izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
        datoteke, tj. 4 argumenta */
95   if (argc != 4) {
       fprintf(stderr,
97             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
       return 0;
99   }

101  /* Otvaranje datoteke u kojoj su zadate tacke */
     ulaz = fopen(argv[2], "r");
103  if (ulaz == NULL) {
       fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
105            argv[2]);
       return 0;
107  }

109  /* Otvaranje datoteke u koju treba upisati rezultat */
     izlaz = fopen(argv[3], "w");
111  if (izlaz == NULL) {
       fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
113            argv[3]);
       return 0;
115  }
```

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

125   /* Ukupan broj procitanih tacaka */
      n = i;
127
      /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
129      "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
         (karakter -), a karakter argv[1][1] odredjuje kriterijum
131      sortiranja */
      switch (argv[1][1]) {
133   case 'x':
        /* Sortiranje po vrednosti x koordinate */
135     sortiraj_po_x(tacke, n);
        break;
137   case 'y':
        /* Sortiranje po vrednosti y koordinate */
139     sortiraj_po_y(tacke, n);
        break;
141   case 'o':
        /* Sortiranje po udaljenosti od koorinatnog pocetka */
143     sortiraj_po_rastojanju(tacke, n);
        break;
145   }

147   /* Upisivanje dobijenog niza u izlaznu datoteku */
      for (i = 0; i < n; i++) {
149     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
      }
151
      /* Zatvaranje otvorenih datoteka */
153   fclose(ulaz);
      fclose(izlaz);
155
      return 0;
157 }
```

Rešenje 3.19

```
#include <stdio.h>
2 #include <string.h>
  #include <stdlib.h>
4
  #define MAX 1000
6 #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 samo ako
       su (i) i min razliciti, inace je nepotrebno. */
    if (min != i) {
      pom = a[i];
      a[i] = a[min];
      a[min] = pom;
    }
  }
```

```
60 }

62 /* Pretraga niza Gradjana */
   int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64 {
     int i;
66   for (i = 0; i < n; i++)
       if (strcmp(a[i].ime, x->ime) == 0
68         && strcmp(a[i].prezime, x->prezime) == 0)
         return i;
70   return -1;
   }
72

74 int main()
   {
76   Gradjanin spisak1[MAX], spisak2[MAX];
     int isti_rbr = 0;
78   int i, n;
     FILE *fp = NULL;
80
     /* Otvaranje datoteke */
82   if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
       fprintf(stderr,
84             "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
       exit(EXIT_FAILURE);
86   }

88   /* Citanje sadrzaja */
     for (i = 0;
90        fscanf(fp, "%s %s", spisak1[i].ime,
                 spisak1[i].prezime) != EOF; i++)
92     spisak2[i] = spisak1[i];
     n = i;
94
     /* Zatvaranje datoteke */
96   fclose(fp);

98   sort_ime(spisak1, n);

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

104    printf("Biracki spisak [uredjen prema imenima]:\n");
       for(i=0; i<n; i++)
106      printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
     *****************************************************************/
108
     sort_prezime(spisak2, n);
110
     /*****************************************************************
```

```
    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 3.21

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

#define MAX_BR_RECI 128
#define MAX_DUZINA_RECI 32

/* Funkcija koja izracunava broj suglasnika u reci */
int broj_suglasnika(char s[])
{
  char c;
  int i;
  int suglasnici = 0;
  /* Prolaz karakter po karakter kroz zadatu nisku */
  for (i = 0; s[i]; i++) {
    /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
       pokriven slucaj i malih i velikih suglasnika. */
    if (isalpha(s[i])) {
      c = toupper(s[i]);
      /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
    */
      if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
```

```
        suglasnici++;
    }
  }
  /* Vraca se izracunata vrednost */
  return suglasnici;
}

/* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
   duzini reci se mora proslediti zbog pravilnog upravljanja
   memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECI], int n)
{
  int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
      duzina_j, duzina_min;
  char tmp[MAX_DUZINA_RECI];
  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i; j < n; j++) {
      /* Prvo se uporedjuje broj suglasnika */
      broj_suglasnika_j = broj_suglasnika(reci[j]);
      broj_suglasnika_min = broj_suglasnika(reci[min]);
      if (broj_suglasnika_j < broj_suglasnika_min)
        min = j;
      else if (broj_suglasnika_j == broj_suglasnika_min) {
        /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
           se duzine */
        duzina_j = strlen(reci[j]);
        duzina_min = strlen(reci[min]);

        if (duzina_j < duzina_min)
          min = j;
        else
          /* Ako reci imaju i isti broj suglasnika i iste duzine,
             uporedjuju se leksikografski */
        if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
          min = j;
      }
    }
    if (min != i) {
      strcpy(tmp, reci[min]);
      strcpy(reci[min], reci[i]);
      strcpy(reci[i], tmp);
    }
  }
}

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

  /* Niz u koji ce biti smestane reci. Prvi broj oznacava broj reci,
```

```
    a drugi maksimalnu duzinu pojedinacne reci */
  char reci[MAX_BR_RECI][MAX_DUZINA_RECI];

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

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

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

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

  /* Zatvaranje datoteke */
  fclose(ulaz);

  return 0;
}
```

Rešenje 3.22

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

#define MAX_ARTIKALA 100000

/* Struktura koja predstavlja jedan artikal */
typedef struct art {
  long kod;
  char naziv[20];
```

```
  char proizvodjac[20];
  float cena;
} Artikal;

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

  /* Dokle god je indeks levi levo od indeksa desni */
  while (levi <= desni) {
    /* Racuna se sredisnji indeks */
    int srednji = (levi + desni) / 2;
    /* Ako je sredisnji element veci od trazenog, tada se trazeni
       mora nalaziti u levoj polovini niza */
    if (x < a[srednji].kod)
      desni = srednji - 1;
    /* Ako je sredisnji element manji od trazenog, tada se trazeni
       mora nalaziti u desnoj polovini niza */
    else if (x > a[srednji].kod)
      levi = srednji + 1;
    else
      /* Ako je sredisnji element jednak trazenom, tada je artikal sa
         bar kodom x pronadjen na poziciji srednji */
      return srednji;
  }
  /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
  return -1;
}

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

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

```

```
int main()
{
  Artikal asortiman[MAX_ARTIKALA];
  long kod;
  int i, n;
  float racun;

  FILE *fp = NULL;

  /* Otvaranje datoteke */
  if ((fp = fopen("artikli.txt", "r")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje artikala */
  i = 0;
  while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
                asortiman[i].naziv, asortiman[i].proizvodjac,
                &asortiman[i].cena) == 4)
    i++;

  /* Zatvaranje datoteke */
  fclose(fp);

  n = i;

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

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

```
  kod = 0;
  while (1) {
    printf("----------------------------\n");
    printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
    printf("- Za nov racun unesite kod artikla!\n\n");
    /* Unos bar koda provog artikla sledeceg kupca */
    if (scanf("%ld", &kod) == EOF)
      break;
    /* Trenutni racun novog kupca */
    racun = 0;
    /* Za sve artikle trenutnog kupca */
    while (1) {
      /* Vrsi se njihov pronalazak u nizu */
      if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
        printf("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
      } else {
        printf("\tTrazili ste:\t%s %s %12.2f\n",
               asortiman[i].naziv, asortiman[i].proizvodjac,
               asortiman[i].cena);
        /* I dodavanje na ukupan racun */
        racun += asortiman[i].cena;
      }
      /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
         nema vise artikla */
      printf("Unesite kod artikla [ili 0 za prekid]: \t");
      scanf("%ld", &kod);
      if (kod == 0)
        break;
    }
    /* Stampanje ukupnog racuna trenutnog kupca */
    printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
  }

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.23

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

#define MAX 500

/* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
  char ime[20];
  char prezime[25];
  int prisustvo;
```

```
  int zadaci;
} Student;

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

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

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

/* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
   zadataka opadajuce, a ukoliko neki studenti imaju isti broj
   uradjenih zadataka sortiraju se po duzini imena rastuce. */
void sort_zadatke_pa_imena(Student niz[], int n)
{
  int i, j;
  int max;
  Student pom;
  for (i = 0; i < n - 1; i++) {
    max = i;
    for (j = i + 1; j < n; j++)
      if (niz[j].zadaci > niz[max].zadaci)
        max = j;
      else if (niz[j].zadaci == niz[max].zadaci
               && strlen(niz[j].ime) < strlen(niz[max].ime))
        max = j;
    if (max != i) {
      pom = niz[max];
      niz[max] = niz[i];
      niz[i] = pom;
    }
  }
}

 /* Funkcija za sortiranje niza struktura po broju casova na kojima
    su bili opadajuce. Ukoliko neki studenti imaju isti broj casova,
    sortiraju se opadajuce po broju uradjenih zadataka, a ukoliko se
```

```
    i po broju zadataka poklapaju, njihovo sortiranje ce biti po
    prezimenu opadajuce. */
void sort_prisustvo_pa_zadatke_pa_prezimena(Student niz[], int n)
{
  int i, j;
  int max;
  Student pom;
  for (i = 0; i < n - 1; i++) {
    max = i;
    for (j = i + 1; j < n; j++)
      if (niz[j].prisustvo > niz[max].prisustvo)
        max = j;
      else if (niz[j].prisustvo == niz[max].prisustvo
               && niz[j].zadaci > niz[max].zadaci)
        max = j;
      else if (niz[j].prisustvo == niz[max].prisustvo
               && niz[j].zadaci == niz[max].zadaci
               && strcmp(niz[j].prezime, niz[max].prezime) > 0)
        max = j;
    if (max != i) {
      pom = niz[max];
      niz[max] = niz[i];
      niz[i] = pom;
    }
  }
}

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

  FILE *fp = NULL;

  /* Otvaranje datoteke za citanje */
  if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke aktivnost.txt.\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje sadrzaja */
  for (i = 0;
       fscanf(fp, "%s%s%d%d", praktikum[i].ime,
              praktikum[i].prezime, &praktikum[i].prisustvo,
              &praktikum[i].zadaci) != EOF; i++);
  /* Zatvaranje datoteke */
  fclose(fp);
  br_studenata = i;

  /* Kreiranje prvog spiska studenata po prvom kriterijumu */
  sort_ime_leksikografski(praktikum, br_studenata);
  /* Otvaranje datoteke za pisanje */
```

```
  if ((fp = fopen("dat1.txt", "w")) == NULL) {
117   fprintf(stderr, "Neupesno otvaranje datoteke dat1.txt.\n");
    exit(EXIT_FAILURE);
119 }
  /* Upis niza u datoteku */
121 fprintf
      (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
123 for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
125           praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
127 /* Zatvaranje datoteke */
  fclose(fp);
129
  /* Kreiranje drugog spiska studenata po drugom kriterijumu */
131 sort_zadatke_pa_imena(praktikum, br_studenata);
  /* Otvaranje datoteke za pisanje */
133 if ((fp = fopen("dat2.txt", "w")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
135   exit(EXIT_FAILURE);
  }
137 /* Upis niza u datoteku */
  fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
139 fprintf(fp, "pa po duzini imena rastuce:\n");
  for (i = 0; i < br_studenata; i++)
141   fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
          praktikum[i].prezime, praktikum[i].prisustvo,
143       praktikum[i].zadaci);
  /* Zatvaranje datoteke */
145 fclose(fp);

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

165 return 0;
}
```

Rešenje 3.24

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

#define KORAK 10

/* Struktura koja opisuje jednu pesmu */
typedef struct {
  char *izvodjac;
  char *naslov;
  int broj_gledanja;
} Pesma;

/* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
   rad qsort funkcije) */
int uporedi_gledanost(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return p2->broj_gledanja - p1->broj_gledanja;
}

/* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
   funkcije) */
int uporedi_naslove(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return strcmp(p1->naslov, p2->naslov);
}

/* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
   qsort funkcije) */
int uporedi_izvodjace(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

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

int main(int argc, char *argv[])
{
  FILE *ulaz;
  Pesma *pesme;                 /* Pokazivac na deo memorije za
                                   cuvanje pesama */
  int alocirano_za_pesme;       /* Broj mesta alociranih za pesme */
  int i;                        /* Redni broj pesme cije se
```

```
                                    informacije citaju */
52  int n;                          /* Ukupan broj pesama */
    int j, k;
54  char c;
    int alocirano;                  /* Broj mesta alociranih za propratne
56                                     informacije o pesmama */
    int broj_gledanja;
58
    /* Priprema datoteke za citanje */
60  ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
    if (ulaz == NULL) {
62    printf("Greska pri otvaranju ulazne datoteke!\n");
      return 0;
64  }

66  /* Citanje informacija o pesmama */
    pesme = NULL;
68  alocirano_za_pesme = 0;
    i = 0;
70
    while (1) {
72
      /* Proverava da li je dostignut kraj datoteke */
74    c = fgetc(ulaz);
      if (c == EOF) {
76      /* Nema vise sadrzaja za citanje */
        break;
78    } else {
        /* Inace, vracamo procitani karakter nazad */
80      ungetc(c, ulaz);
      }
82
      /* Provera da li postoji dovoljno memorije za citanje nove pesme
      */
84    if (alocirano_za_pesme == i) {

86      /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
           novih KORAK mesta */
88      alocirano_za_pesme += KORAK;
        pesme =
90          (Pesma *) realloc(pesme,
                              alocirano_za_pesme * sizeof(Pesma));
92
        /* Proverava da li je nova memorija uspesno realocirana */
94      if (pesme == NULL) {
          /* Ako nije ispisuje se obavestenje */
96        printf("Problem sa alokacijom memorije!\n");
          /* I oslobadja sva memorija zauzeta do ovog koraka */
98        for (k = 0; k < i; k++) {
            free(pesme[k].izvodjac);
100         free(pesme[k].naslov);
          }
```

```
102         free(pesme);
            return 0;
104       }
        }
106
        /* Ako jeste, nastavlja se sa citanjem pesama ... */
108     /* Cita se ime izvodjaca */
        j = 0;                          /* Pozicija na koju treba smestiti
110                                        procitani karakter */
        alocirano = 0;                  /* Broj alociranih mesta */
112     pesme[i].izvodjac = NULL;       /* Memorija za smestanje procitanih
                                           karaktera */
114
        /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
116        izvodjaca) citaju se karakteri iz datoteke */
        while ((c = fgetc(ulaz)) != ' ') {
118       /* Proverav da li postoji dovoljno memorije za smestanje
             procitanog karaktera */
120       if (j == alocirano) {

122         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
               se novih KORAK mesta */
124         alocirano += KORAK;
            pesme[i].izvodjac =
126             (char *) realloc(pesme[i].izvodjac,
                                 alocirano * sizeof(char));
128
            /* Provera da li je nova alokacija uspesna */
130         if (pesme[i].izvodjac == NULL) {
              /* Ako nije oslobadja se sva memorija zauzeta do ovog
132               koraka */
              for (k = 0; k < i; k++) {
134             free(pesme[k].izvodjac);
                free(pesme[k].naslov);
136           }
              free(pesme);
138           /* I prekida sa izvrsavanjem programa */
              return 0;
140         }

142       }
          /* Ako postoji dovoljno memorije, smestamo procitani karakter
        */
144       pesme[i].izvodjac[j] = c;
          j++;
146       /* I nastavlja se sa citanjem */
        }
148
        /* Upis terminirajuce nule na kraj reci */
150     pesme[i].izvodjac[j] = '\0';

152     /* Preskace se karakter - */
```

```
    fgetc(ulaz);
154
    /* Preskace se razmak */
156 fgetc(ulaz);

158 /* Cita se naslov pesme */
    j = 0;                      /* Pozicija na koju treba smestiti
160                                procitani karakter */
    alocirano = 0;              /* Broj alociranih mesta */
162 pesme[i].naslov = NULL;      /* Memorija za smestanje procitanih
                                   karaktera */
164
    /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
166    karakteri iz datoteke */
    while ((c = fgetc(ulaz)) != ',') {
168   /* Provera da li postoji dovoljno memorije za smestanje
         procitanog karaktera */
170   if (j == alocirano) {
        /* Ako ne, ako je potrosena sva alocirana memorija, alocira
172          se novih KORAK mesta */
        alocirano += KORAK;
174     pesme[i].naslov =
            (char *) realloc(pesme[i].naslov,
176                          alocirano * sizeof(char));

178     /* Provera da li je nova alokacija uspesna */
        if (pesme[i].naslov == NULL) {
180       /* Ako nije, oslobadja se sva memorija zauzeta do ovog
             koraka */
182       for (k = 0; k < i; k++) {
            free(pesme[k].izvodjac);
184         free(pesme[k].naslov);
          }
186       free(pesme[i].izvodjac);
          free(pesme);
188
          /* I prekida izvrsavanje programa */
190       return 0;
        }
192   }
      /* Ako postoji dovoljno memorije, smesta se procitani karakter
    */
194   pesme[i].naslov[j] = c;
      j++;
196   /* I nastavlja dalje sa citanjem */
    }
198 /* Upisuje se terminirajuca nula na kraj reci */
    pesme[i].naslov[j] = '\0';
200
    /* Preskace se razmak */
202 fgetc(ulaz);
```

```
204     /* Cita se broj gledanja */
        broj_gledanja = 0;
206
        /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
208        datoteke */
        while ((c = fgetc(ulaz)) != '\n') {
210       broj_gledanja = broj_gledanja * 10 + (c - '0');
        }
212     pesme[i].broj_gledanja = broj_gledanja;

214     /* Prelazi se na citanje sledece pesme */
        i++;
216   }

218   /* Informacija o broju procitanih pesama */
      n = i;
220   /* Zatvaranje nepotrebne datoteke */
      fclose(ulaz);
222
      /* Analiza argumenta komandne linije */
224   if (argc == 1) {
        /* Nema dodatnih opcija => sortiranje po broju gledanja */
226     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
      } else {
228     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
          /* Sortiranje po naslovu */
230       qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
        } else {
232       if (argc == 2 && strcmp(argv[1], "-i") == 0) {
            /* Sortirnje po izvodjacu */
234         qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
          } else {
236         printf("Nedozvoljeni argumenti!\n");
            free(pesme);
238         return 0;
          }
240     }
      }
242
      /* Ispis rezultata */
244   for (i = 0; i < n; i++) {
        printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
246            pesme[i].broj_gledanja);
      }
248
      /* Oslobadjanje memorije */
250   for (i = 0; i < n; i++) {
        free(pesme[i].izvodjac);
252     free(pesme[i].naslov);
      }
254   free(pesme);
```

```
256   return 0;
}
```

Rešenje 3.28

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

#define MAX 100

/* Funkcija poredjenja dva cela broja */
int compare_int(const void *a, const void *b)
{
  /* Potrebno je konvertovati void pokazivace u int pokazivace koji
     se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */

  /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
     oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */

  int b1 = *((int *) a);
  int b2 = *((int *) b);

  if (b1 > b2)
    return 1;
  else if (b1 < b2)
    return -1;
  else
    return 0;
}

int compare_int_desc(const void *a, const void *b)
{
  /* Za obrnuti poredak treba samo oduzimati a od b */
  /* return *((int *)b) - *((int *)a); */

  /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
     funkcija */
  return -compare_int(a, b);
}

int main()
{
  size_t n;
  int i, x;
  int a[MAX], *p = NULL;

  /* Unos dimenzije */
  printf("Uneti dimenziju niza: ");
  scanf("%ld", &n);
```

```
  if (n > MAX)
    n = MAX;

  /* Unos elementa niza */
  printf("Uneti elemente niza:\n");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Sortiranje niza celih brojeva */
  qsort(a, n, sizeof(int), &compare_int);

  /* Prikaz sortiranog niz */
  printf("Sortirani niz u rastucem poretku:\n");
  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
  putchar('\n');

  /* Pretrazivanje niza */
  /* Vrednost koja ce biti trazena u nizu */
  printf("Uneti element koji se trazi u nizu: ");
  scanf("%d", &x);

  /* Binarna pretraga */
  printf("Binarna pretraga: \n");
  p = bsearch(&x, a, n, sizeof(int), &compare_int);
  if (p == NULL)
    printf("Elementa nema u nizu!\n");
  else
    printf("Element je nadjen na poziciji %ld\n", p - a);

  /* Linearna pretraga */
  printf("Linearna pretraga (lfind): \n");
  p = lfind(&x, a, &n, sizeof(int), &compare_int);
  if (p == NULL)
    printf("Elementa nema u nizu!\n");
  else
    printf("Element je nadjen na poziciji %ld\n", p - a);

  return 0;
}
```

Rešenje 3.29

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

#define MAX 100

/* Funkcija racuna broj delilaca broja x */
```

```
int no_of_deviders(int x)
{
  int i;
  int br;

  /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
  if (x < 0)
    x = -x;
  if (x == 0)
    return 0;
  if (x == 1)
    return 1;
  /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
  br = 2;
  for (i = 2; i < sqrt(x); i++)
    if (x % i == 0)
      /* Ako i deli x onda su delioci: i, x/i */
      br += 2;
  /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
     promenljiva i bila bas jednaka korenu od x, i tada broj x ima
     jos jednog delioca */
  if (i * i == x)
    br++;

  return br;
}

/* Funkcija poredjenja dva cela broja po broju delilaca */
int compare_no_deviders(const void *a, const void *b)
{
  int ak = *(int *) a;
  int bk = *(int *) b;
  int n_d_a = no_of_deviders(ak);
  int n_d_b = no_of_deviders(bk);

  if (n_d_a > n_d_b)
    return 1;
  else if (n_d_a < n_d_b)
    return -1;
  else
    return 0;
}

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

  /* Unos dimenzije */
  printf("Uneti dimenziju niza: ");
  scanf("%ld", &n);
```

```
  if (n > MAX)
    n = MAX;

  /* Unos elementa niza */
  printf("Uneti elemente niza:\n");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Sortiranje niza celih brojeva prema broju delilaca */
  qsort(a, n, sizeof(int), &compare_no_deviders);

  /* Prikaz sortiranog niza */
  printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
  putchar('\n');

  return 0;
}
```

Rešenje 3.30

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

#define MAX_NISKI 1000
#define MAX_DUZINA 30

/*****************************************************************
  Niz nizova karaktera ovog potpisa
  char niske[3][4];
  se moze graficki predstaviti ovako:
  ---------------------------------------------------
  | a | b | c |\0 || d | e | \0|   || f | g | h | \0||
  ---------------------------------------------------
  Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
  svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
  nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
  adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
  i ona je tipa char*.

  Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
  koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
  reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
  slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
  nad njima.
*****************************************************************/
int poredi_leksikografski(const void *a, const void *b)
{
```

```
  return strcmp((char *) a, (char *) b);
}

/* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
int poredi_duzine(const void *a, const void *b)
{
  return strlen((char *) a) - strlen((char *) b);
}

int main()
{
  int i;
  size_t n;
  FILE *fp = NULL;
  char niske[MAX_NISKI][MAX_DUZINA];
  char *p = NULL;
  char x[MAX_DUZINA];

  /* Otvaranje datoteke */
  if ((fp = fopen("niske.txt", "r")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
    exit(EXIT_FAILURE);
  }

  /* Citanje sadrzaja datoteke */
  for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);

  /* Zatvaranje datoteke */
  fclose(fp);
  n = i;

  /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
     prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
     niske po duzini */
  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

  printf("Leksikografski sortirane niske:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  /* Unos trazene niske */
  printf("Uneti trazenu nisku: ");
  scanf("%s", x);

  /* Binarna pretraga */
  /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
     je niz vec sortiran leksikografski. */
  p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
              &poredi_leksikografski);

```

```
  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
           p, (p - (char *) niske) / MAX_DUZINA);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Linearna pretraga */
  p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
            &poredi_leksikografski);

  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
           p, (p - (char *) niske) / MAX_DUZINA);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Sortiranje po duzini */
  qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

  printf("Niske sortirane po duzini:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.31

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

#define MAX_NISKI 1000
#define MAX_DUZINA 30

/*****************************************************************
  Niz pokazivaca na karaktere ovog potpisa
  char *niske[3];
  posle alokacije u main-u se moze graficki predstaviti ovako:
  -----                  -----------------
  | X | --------------->  | a | b | c | \0|
  -----                  =================
  | Y | --------------->  | d | e | \0|
  -----                  =================
  | Z | --------------->  | f | g | h | \0|
  -----                  -----------------
  Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
  reci. Njegov tip je char*.
```

```
23
   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
25 koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i
   kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
27 moraju i kastovati. Da bi se leksikografski uporedili elementi X i
   Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
29 u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
   kastovani na odgovarajuci tip.
31 ********************************************************************/
int poredi_leksikografski(const void *a, const void *b)
33 {
   return strcmp(*(char **) a, *(char **) b);
35 }

37 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
      leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
{
41   return strlen(*(char **) a) - strlen(*(char **) b);
}
43
/* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
45    element u nizu sa kojim se poredi, pa njega treba kastovati na
      char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
47    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
49    ovde samo treba kastovati i ne dereferencirati. */
int poredi_leksikografski_b(const void *a, const void *b)
51 {
   return strcmp((char *) a, *(char **) b);
53 }

55 int main()
{
57   int i;
   size_t n;
59   FILE *fp = NULL;
   char *niske[MAX_NISKI];
61   char **p = NULL;
   char x[MAX_DUZINA];
63
   /* Otvaranje datoteke */
65   if ((fp = fopen("niske.txt", "r")) == NULL) {
     fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
67   exit(EXIT_FAILURE);
   }
69
   /* Citanje sadrzaja datoteke */
71   i = 0;
   while (fscanf(fp, "%s", x) != EOF) {
73   /* Alociranje dovoljne memorije za i-tu nisku */
     if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
```

```
75        fprintf(stderr, "Greska pri alociranju niske\n");
          exit(EXIT_FAILURE);
77      }
        /* Kopiranje procitane niske na svoje mesto */
79      strcpy(niske[i], x);
        i++;
81    }

83    /* Zatvaranje datoteke */
      fclose(fp);
85    n = i;

87    /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
         prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
89       niske po duzini */
      qsort(niske, n, sizeof(char *), &poredi_leksikografski);
91
      printf("Leksikografski sortirane niske:\n");
93    for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
95    printf("\n");

97    /* Unos trazene niske */
      printf("Uneti trazenu nisku: ");
99    scanf("%s", x);

101   /* Binarna pretraga */
      p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
103   if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
105             *p, p - niske);
      else
107     printf("Niska nije pronadjena u nizu\n");

109   /* Linearna pretraga */
      p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
111   if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
113             *p, p - niske);
      else
115     printf("Niska nije pronadjena u nizu\n");

117   /* Sortiramo po duzini */
      qsort(niske, n, sizeof(char *), &poredi_duzine);
119
      printf("Niske sortirane po duzini:\n");
121   for (i = 0; i < n; i++)
        printf("%s ", niske[i]);
123   printf("\n");

125   /* Oslobadjanje zauzete memorije */
      for (i = 0; i < n; i++)
```

```
127     free(niske[i]);

129   exit(EXIT_SUCCESS);
}
```

Rešenje 3.32

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

6 #define MAX 500

8 /* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
10   char ime[21];
   char prezime[21];
12   int bodovi;
} Student;
14
/* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
16    istim brojem bodova se dodatno sortiraju leksikografski po
    prezimenu */
18 int poredi1(const void *a, const void *b)
{
20   Student *prvi = (Student *) a;
   Student *drugi = (Student *) b;
22
   if (prvi->bodovi > drugi->bodovi)
24     return -1;
   else if (prvi->bodovi < drugi->bodovi)
26     return 1;
   else
28     /* Ako su jednaki po broju bodova, treba ih uporediti po
        prezimenu */
30     return strcmp(prvi->prezime, drugi->prezime);
}
32
/* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
34    Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
    parametar je element niza ciji se bodovi porede. */
36 int poredi2(const void *a, const void *b)
{
38   int bodovi = *(int *) a;
   Student *s = (Student *) b;
40   return s->bodovi - bodovi;
}
42
/* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
44    Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
```

```
   parametar je element niza cije se prezime poredi. */
int poredi3(const void *a, const void *b)
{
  char *prezime = (char *) a;
  Student *s = (Student *) b;
  return strcmp(prezime, s->prezime);
}

int main(int argc, char *argv[])
{
  Student kolokvijum[MAX];
  int i;
  size_t br_studenata = 0;
  Student *nadjen = NULL;
  FILE *fp = NULL;
  int bodovi;
  char prezime[21];

  /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
     informacija korisniku kako se program koristi i prekida se
     izvrsavanje. */
  if (argc < 2) {
    fprintf(stderr,
            "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

  /* Otvaranje datoteke */
  if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje sadrzaja */
  for (i = 0;
       fscanf(fp, "%s%s%d", kolokvijum[i].ime,
              kolokvijum[i].prezime,
              &kolokvijum[i].bodovi) != EOF; i++);

  /* Zatvaranje datoteke */
  fclose(fp);
  br_studenata = i;

  /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
     studenata sa istim brojem bodova sortiranje vrsi po prezimenu */
  qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);

  printf("Studenti sortirani po broju poena opadajuce, ");
  printf("pa po prezimenu rastuce:\n");
  for (i = 0; i < br_studenata; i++)
    printf("%s %s  %d\n", kolokvijum[i].ime,
```

```
            kolokvijum[i].prezime, kolokvijum[i].bodovi);

  /* Pretrazivanje studenata po broju bodova se vrsi binarnom
     pretragom jer je niz sortiran po broju bodova. */
  printf("Unesite broj bodova: ");
  scanf("%d", &bodovi);

  nadjen =
      bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
              &poredi2);

  if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
  else
    printf("Nema studenta sa unetim brojem bodova\n");

  /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
     sortiran po bodovima. */
  printf("Unesite prezime: ");
  scanf("%s", prezime);

  nadjen =
      lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
            &poredi3);

  if (nadjen != NULL)
    printf
        ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
         nadjen->ime, nadjen->prezime, nadjen->bodovi);
  else
    printf("Nema studenta sa unetim prezimenom\n");

  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;
}

```

```c
/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
int anagrami(char s[], char t[])
{
  /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
  if (strlen(s) != strlen(t))
    return 0;

  /* Sortiranje niski */
  qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
  qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

  /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
     suprotnom, nisu */
  return !strcmp(s, t);
}

int main()
{
  char s[MAX], t[MAX];

  /* Unos niski */
  printf("Unesite prvu nisku: ");
  scanf("%s", s);
  printf("Unesite drugu nisku: ");
  scanf("%s", t);

  /* Ispituje se da li su niske anagrami */
  if (anagrami(s, t))
    printf("jesu\n");
  else
    printf("nisu\n");

  return 0;
}
```

Rešenje 3.34

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

#define MAX 10
#define MAX_DUZINA 32

/* Funkcija porenjenja */
int uporedi_niske(const void *pa, const void *pb)
{
  return strcmp((char *) pa, (char *) pb);
}

int main()
```

```
{
  int i, n;
  char S[MAX][MAX_DUZINA];

  /* Unos broja niski */
  printf("Unesite broj niski:");
  scanf("%d", &n);

  /* Unos niza niski */
  printf("Unesite niske:\n");
  for (i = 0; i < n; i++)
    scanf("%s", S[i]);

  /* Sortiranje niza niski */
  qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

  /*****************************************************************
    Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih niski. Koriscen je samo u fazi testiranja programa.

    printf("Sortirane niske su:\n");
    for(i = 0; i < n; i++)
      printf("%s ", S[i]);
  *****************************************************************/

  /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
     niza biti jedna do druge */
  for (i = 0; i < n - 1; i++)
    if (strcmp(S[i], S[i + 1]) == 0) {
      printf("ima\n");
      return 0;
    }

  printf("nema\n");
  return 0;
}
```

Rešenje 3.35

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

#define MAX 21

/* Struktura koja predstavlja jednog studenta */
typedef struct student {
  char nalog[8];
  char ime[MAX];
  char prezime[MAX];
  int poeni;
```

```
13 } Student;

15 /* Funkcija poredi studente prema broju poena, rastuce */
   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;
29   /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
        broj indeksa */
31   int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
33   char smer1 = s.nalog[1];
     char smer2 = t.nalog[1];
35   int indeks1 =
         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
37       s.nalog[6] - '0';
     int indeks2 =
39       (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
         t.nalog[6] - '0';
41   if (godina1 != godina2)
       return godina1 - godina2;
43   else if (smer1 != smer2)
       return smer1 - smer2;
45   else
       return indeks1 - indeks2;
47 }

49 int uporedi_bsearch(const void *a, const void *b)
   {
51   /* Nalog studenta koji se trazi */
     char *nalog = (char *) a;
53   /* Kljuc pretrage */
     Student s = *(Student *) b;
55
     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
57   int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
     char smer1 = nalog[1];
59   char smer2 = s.nalog[1];
     int indeks1 =
61       (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
       ;
     int indeks2 =
63       (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;

  /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
     korisnik nije ispravno pozvao program i prijavljuje se greska.
    */
  if (argc != 2 && argc != 3) {
    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
    );
    exit(EXIT_FAILURE);
  }

  /* Otvaranje datoteke za citanje */
  in = fopen("studenti.txt", "r");
  if (in == NULL) {
    fprintf(stderr,
            "Greska prilikom otvarnja datoteke studenti.txt!\n");
    exit(EXIT_FAILURE);
  }

  /* Otvaranje datoteke za pisanje */
  out = fopen("izlaz.txt", "w");
  if (out == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke izlaz.txt!\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
  while (fscanf
         (in, "%s %s %s %d", niz_studenata[i].nalog,
          niz_studenata[i].ime, niz_studenata[i].prezime,
          &niz_studenata[i].poeni) != EOF)
    i++;

  br_studenata = i;

```

```
  /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
  if (strcmp(argv[1], "-p") == 0)
    qsort(niz_studenata, br_studenata, sizeof(Student),
          &uporedi_poeni);
  /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
  else if (strcmp(argv[1], "-n") == 0)
    qsort(niz_studenata, br_studenata, sizeof(Student),
          &uporedi_nalog);

  /* Sortirani studenti se ispisuju u izlaznu datoteku */
  for (i = 0; i < br_studenata; i++)
    fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
            niz_studenata[i].ime, niz_studenata[i].prezime,
            niz_studenata[i].poeni);

  /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
     studenta... */
  if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
    strcpy(nalog_trazeni, argv[2]);

    /* ... pronalazi se student sa tim nalogom... */
    nadjen =
        (Student *) bsearch(nalog_trazeni, niz_studenata,
                            br_studenata, sizeof(Student),
                            &uporedi_bsearch);

    if (nadjen == NULL)
      printf("Nije nadjen!\n");
    else
      printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
             nadjen->prezime, nadjen->poeni);
  }

  /* Zatvaranje datoteka */
  fclose(in);
  fclose(out);

  return 0;
}
```

Rešenje 3.37

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

/* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
   standardnog ulaza */
void ucitaj_matricu(int **a, int n, int m)
{
  printf("Unesite elemente matrice po vrstama:\n");
  int i, j;
```

```
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 (pokazivace na vrste) na
      osnovu zbira koriscenjem selection sort 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 */
    int **alociraj_memoriju(int n, int m)
68  {
    int i, j;
70  int **a;

72  a = (int **) malloc(n * sizeof(int *));
    if (a == NULL) {
74    fprintf(stderr, "Problem sa alokacijom memorije!\n");
      exit(EXIT_FAILURE);
76  }
    /* Za svaku vrstu ponaosob */
78  for (i = 0; i < n; i++) {
      /* Alocira se memorija */
80    a[i] = (int *) malloc(m * sizeof(int));
      /* Proverava se da li je doslo do greske prilikom alokacije */
82    if (a[i] == NULL) {
        /* Ako jeste, ispisuje se poruka */
84      fprintf(stderr, "Problem sa alokacijom memorije!\n");
        /* I oslobadja memorija zauzeta do ovog koraka */
86      for (j = 0; j < i; j++) {
          free(a[i]);
88      }
        free(a);
90      exit(EXIT_FAILURE);
      }
92  }

94  return a;
    }
96

    /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije nxm
        */
98  void oslobodi_memoriju(int **a, int n, int m)
    {
100 int i;
    for (i = 0; i < n; i++) {
102   free(a[i]);
    }
104 free(a);
    }
106

    int main(int argc, char *argv[])
108 {
    int **a;
110 int n, m;

112 /* Unos dimenzija matrice */
```

```
    printf("Unesite dimenzije matrice: ");
114 scanf("%d %d", &n, &m);

116 /* Alokacija memorije */
    a = alociraj_memoriju(n, m);
118
    /* Ucitavanje elementa matrice */
120 ucitaj_matricu(a, n, m);

122 /* Poziv funkcije koja sortira vrste matrice prema zbiru */
    sortiraj_vrste(a, n, m);
124
    /* Ispis rezultujuce matrice */
126 printf("Sortirana matrica je:\n");
    ispisi_matricu(a, n, m);
128
    /* Oslobadjanje memorije */
130 oslobodi_memoriju(a, n, m);

132 return 0;
}
```

Glava 4

Dinamičke strukture podataka

4.1 Liste

Zadatak 4.1 Napisati biblioteku za rad sa jednostruko povezanom listom čiji čvorovi sadrže cele brojeve.

(a) Definisati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.

(b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje mu polja i vraća njegovu adresu.

(c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.

(d) Napisati funkciju `Cvor *pronadji_poslednji(Cvor * glava)` koja pronalazi poslednji čvor u listi.

(e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.

(f) Napisati funkciju `Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

(g) Napisati funkciju `void dodaj_iza(Cvor * tekuci, Cvor * novi)` koja uvezuje u postojeću listu čvor `novi` iza čvora `tekuci`.

(h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.

(i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradama [,] i međusobno razdvojene zapetama.

(j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.

(k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.

(l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.

(m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.

(n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. Napomena: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

(1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

Primer 1

```
Interakcija programa:
 Unosite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [3, 2]
 14
 Lista: [14, 3, 2]
 5
 Lista: [5, 14, 3, 2]
 3
 Lista: [3, 5, 14, 3, 2]
 17
 Lista: [17, 3, 5, 14, 3, 2]

 Unesite broj koji se trazi:  5
 Trazeni broj 5 je u listi!
```

Primer 2

```
Interakcija programa:
 Unosite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [35, 14, 23]

 Unesite broj koji se trazi:  8
 Broj 8 se ne nalazi u listi!
```

(2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

Primer 1

```
Interakcija programa:
 Unosite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 14, 3]
 3
 Lista: [2, 3, 14, 3, 3]
 17
 Lista: [2, 3, 14, 3, 3, 17]
 3
 Lista: [2, 3, 14, 3, 3, 17, 3]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 14, 17]
```

Primer 2

```
Interakcija programa:
 Unesite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [23, 14]
 35
 Lista: [23, 14, 35]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [23, 14, 35]
```

(3) U glavnom programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. Napomena: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

Primer 1

```
INTERAKCIJA PROGRAMA:
 Unesite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 3, 14]
 3
 Lista: [2, 3, 3, 3, 14]
 5
 Lista: [2, 3, 3, 3, 5, 14]

 Unesite broj koji se trazi:  14
 Trazeni broj 14 je u listi!

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 5, 14]
```

Primer 2

```
INTERAKCIJA PROGRAMA:
 Unesite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [14, 23, 35]

 Unesite broj koji se trazi:  8
 Broj 8 se ne nalazi u listi!

Unesite broj koji se brise:  3
Lista nakon brisanja: [14, 23, 35]
```

Zadatak 4.2 Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: *Koristiti iste* `main` *programe i test primere iz zadatka 4.1.*

Zadatak 4.3 Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti bibilioteku novim funkcijama.

(a) Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)` koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave`.

(b) Napisati funkciju `void ispisi_listu_unazad(Cvor * glava)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. NAPOMENA: *Koristiti iste* `main` *programe i test primere iz zadatka 4.1. Ove programe dopuniti pozivom funkcije koja ispisuje listu unazad.*

Zadatak 4.4 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

```
IZRAZ.TXT
 {[23 + 5344] * (24 - 234)} - 23

IZLAZ:
 Zagrade su ispravno uparene.
```

Test 2

```
IZRAZ.TXT
 {[23 + 5] * (9 * 2)} - {23}

IZLAZ:
 Zagrade su ispravno uparene.
```

Test 3

```
IZRAZ.TXT
 {[2 + 54) / (24 * 87)} + (234 + 23)

IZLAZ:
 Zagrade nisu ispravno uparene.
```

Test 4

```
IZRAZ.TXT
 {(2 - 14) / (23 + 11)}} * (2 + 13)

IZLAZ:
 Zagrade nisu ispravno uparene.
```

Test 5

```
DATOTEKA IZRAZ.TXT JE PRAZNA

IZLAZ:
 Zagrade su ispravno uparene.
```

Test 6

```
DATOTEKA IZRAZ.TXT NE POSTOJI.

IZLAZ:
 Greska prilikom otvaranja
 datoteke izraz.txt!
```

Zadatak 4.5 Napisati program koji proverava ispravnost uparivanja etiketa u `HTML` datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže `HTML` etikete.*

Test 1

```
POZIV: ./a.out datoteka.html

DATOTEKA.HTML
 <html>
  <head>
   <title>Primer</title>
  </head>
  <body>
  </body>

IZLAZ:
 Etikete nisu pravilno uparene
 (etiketa <html> nije zatvorena)
```

Test 2

```
POZIV: ./a.out datoteka.html

DATOTEKA.HTML
   <head>
     <title>Primer</title>
   </head>
   <body>
   </body>
 </html>

IZLAZ:
 Etikete nisu pravilno uparene
 (nadjena je etiketa </html> koja nije otvorena)
```

Test 3

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

DATOTEKA.HTML
 <html>
  <head>
    <title>Primer</title>
  </head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    Sutra ce biti jos lepsi.
    <a link='http://www.math.rs'>Link</a>
  </body>
 </html>

Izlaz:
 Etikete su pravilno uparene!
```

Test 4

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

DATOTEKA.HTML
 <html>
  <head>
    <title>Primer</title>
  </head>
  <body>
 </html>

Izlaz:
 Etikete nisu pravilno uparene
 (nadjena je etiketa </html>, a poslednja
 otvorena je <body>)
```

Test 5

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

Datoteka datoteka.html ne postoji.

Izlaz:
 Greska prilikom otvaranja
 datoteke datoteka.html.
```

Test 6

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

datoteka.html je prazna

Izlaz:
 Etikete su pravilno uparene!
```

Zadatak 4.6 Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike. Službenik evidentira korisničke `JMBG` brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (`EOF`). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Postavlja mu se pitanje `Da li korisnika vracate na kraj reda?` i ukoliko on da odgovor `Da`, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije `Da`, tada službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke `izvestaj.txt`. Ova datoteka, za svaki obrađen zahtev, sadrži `JMBG` i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. Uputstvo: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

Primer 1

```
INTERAKCIJA PROGRAMA:
 Sluzbenik evidentira korisnicke zahteve:
 Novi zahtev [CTRL+D za kraj]
  JMBG:   1234567890123
  Opis problema:   Otvaranje racuna

 Novi zahtev [CTRL+D za kraj]
  JMBG:   2345678901234
  Opis problema:   Podizanje novca

 Novi zahtev [CTRL+D za kraj]
  JMBG:   3456789012345
  Opis problema:   Reklamacija

 Novi zahtev [CTRL+D za kraj]
  JMBG:

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 2345678901234
 i zahtevom: Podizanje novca
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG: 3456789012345
 i zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 3456789012345
 i zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Da li je kraj smene? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

IZVESTAJ.TXT
   JMBG: 2345678901234 Zahtev: Podizanje novca
   JMBG: 3456789012345 Zahtev: Reklamacija
   JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

Zadatak 4.7 Napisati program koji prebrojava pojavljivanja etiketa `HTML` datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
```

```
{
  unsigned broj_pojavljivanja;
  char etiketa[20];
  struct _Element *sledeci;
} Element;
```

Test 1

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

datoteka.html
 <html>
  <head><title>Primer</title></head>
  <body>
    <h1>Naslov</h1>
    Danas je lep i suncan dan. <br>
    A sutra ce biti jos lepsi.
    <a link='http://www.google.com'> Link 1</a>
    <a link='http://www.math.rs'> Link 2</a>
  </body>
 </html>

Izlaz:
 a - 4
 br - 1
 h1 - 2
 body - 2
 title - 2
 head - 2
 html - 2
```

Test 2

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

Datoteka datoteka.html ne postoji.

Izlaz:
 Greska prilikom otvaranja
 datoteke datoteka.html.
```

Zadatak 4.8 U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku `da` ili `ne`, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. Uputstvo: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

Primer 1

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

studenti.txt
  123/2014 Marko Lukic
  3/2014 Ana Sokic
  43/2013 Jelena Ilic
  41/2009 Marija Zaric
  13/2010 Milovan Lazic

Interakcija programa:
  3/2014   da: Ana Sokic
  235/2008   ne
  41/2009   da: Marija Zaric
```

Primer 2

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

Datoteka studenti.txt je prazna

Interakcija programa:
  3/2014   ne
  235/2008   ne
  41/2009   ne
```

Zadatak 4.9 Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. Napomena: *Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

Test 1

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
  2 4 6 10 15

dat2.txt
  5 6 11 12 14 16

Izlaz:
  [2, 4, 5, 6, 6, 10, 11, 12, 14, 15, 16]
```

Test 2

```
Poziv: ./a.out dat1.txt dat2.txt

dat1.txt
  2 4 6 10 15

Datoteka dat2.txt ne postoji.

Izlaz:
  Greska prilikom otvaranja datoteke
  dat2.txt.
```

Test 3

```
Poziv: ./a.out dat1.txt dat2.txt

Datoteka dat1.txt je prazna

dat2.txt
  5 6 11 12 14 16

Izlaz:
  [5, 6, 11, 12, 14, 16]
```

Test 4

```
Poziv: ./a.out dat1.txt

Izlaz:
  Program se poziva sa:
  ./a.out dat1.txt dat2.txt!
```

Zadatak 4.10 Date su dve jednostruko povezane liste `L1` i `L2`. Napisati funkciju koja od ovih listi formira novu listu `L` koja sadrži naizmenično raspoređene čvorove listi `L1` i `L2`: prvi čvor iz `L1`, prvi čvor iz `L2`, drugi čvor `L1`, drugi čvor

`L2`, itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 6 za zadatak 4.9.*

Test 1

```
POZIV: ./a.out dat1.txt dat2.txt

DAT1.TXT
 2 4 6 10 15

DAT2.TXT
 5 6 11 12 14 16

IZLAZ:
 2 5 4 6 6 11 10 12 15 14 16
```

Zadatak 4.11 Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

(a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

(b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

Test 1

```
BROJEVI.TXT
 43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
 12 15 16
```

Test 2

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAZ:
REZULTAT.TXT
 Greska prilikom otvaranja
 datoteke brojevi.txt.
```

Test 3

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
 Rezultat.txt ce biti prazna.
```

Zadatak 4.12 Grupa od n plesača na kostimima ima 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 sada počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n, k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz

ispisati redne brojeve plesača u redosledu napuštanja kruga. Uputstvo: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

Test 1

```
Ulaz:
 5 3

Izlaz:
 3 1 5 2 4
```

Test 2

```
Ulaz:
 8 4

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

Test 3

```
Ulaz:
 3 8

Izlaz:
 n mora biti uvek vece
 od k, a 3 < 8!
```

Zadatak 4.13 Grupa od n plesača na kostimima ima brojeve od 1 do n, 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 sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n, k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. Uputstvo: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.* Napomena: *Iskoristiti test 3 iz 4.12. zadatka.*

Test 1

```
Ulaz:
 5 3

Izlaz:
 3 5 4 2 1
```

Test 2

```
Ulaz:
 8 4

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

4.2 Stabla

Zadatak 4.14 Napisati program za rad sa binarnim pretraživačkim stablima.

(a) Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.

(b) Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.

(c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.

(d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.

(e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.

(f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.

(g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.

(h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.

(i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.

(j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.

(k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

Primer 1

```
Interakcija programa:
 Unesite brojeve (CRL+D za kraj unosa):
 7 2 1 9 32 18
 Infiksni ispis: 1 2 7 9 18 32
 Prefiksni ispis: 7 2 1 9 32 18
 Postfiksni ispis: 1 2 18 32 9 7
 Trazi se broj: 11
 Broj se ne nalazi u stablu!
 Brise se broj: 7
 Rezultujuce stablo: 1 2 9 18 32
```

Primer 2

```
Interakcija programa:
 Unesite brojeve (CRL+D za kraj unosa):
 8 -2 6 13 24 -3
 Infiksni ispis: -3 -2 6 8 13 24
 Prefiksni ispis: 8 -2 -3 6 13 24
 Postfiksni ispis: -3 6 -2 24 13 8
 Trazi se broj: 6
 Broj se nalazi u stablu!
 Brise se broj: 14
 Rezultujuce stablo: -3 -2 6 8 13 24
```

Zadatak 4.15 Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku `Nedostaje ime ulazne datoteke!`. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

Test 1

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

TEST.TXT
 Sunce utorak raCunar SUNCE programiranje
 jabuka PROGramiranje sunCE JABUka

Izlaz:
 jabuka: 2
 programiranje: 2
 racunar: 1
 sunce: 3
 utorak: 1

 Najcesca rec: sunce (pojavljuje se 3 puta)
```

Test 2

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

SUMA.TXT
 lipa zova hrast ZOVA breza LIPA

Izlaz:
 breza: 1
 hrast: 1
 lipa: 2
 zova: 2

 Najcesca rec: lipa
 (pojavljuje se 2 puta)
```

Test 3

```
Poziv: ./a.out

Izlaz:
 Nedostaje ime ulazne datoteke!
```

Zadatak 4.16 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. `Pera Peric 064/123-4567`. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose

sve do unosa reči `KRAJ`, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Primer 1

```
IMENIK.TXT
 Pera Peric 011/3240-987
 Marko Maric 064/1234-987
 Mirko Maric 011/589-333
 Sanja Savkovic 063/321-098
 Zika Zikic 021/759-858

INTERAKCIJA PROGRAMA:
 Unesite ime datoteke: imenik.txt
 Unesite ime i prezime: Pera Peric
 Broj je: 011/3240-987
 Unesite ime i prezime: Marko Markovic
 Broj nije u imeniku!
 Unesite ime i prezime: KRAJ
```

Primer 2

```
DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
 Unesite ime datoteke: imenik1.txt
 Greska prilikom otvaranja datoteke
 imenik1.txt!
```

Zadatak 4.17 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (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

```
PRIJEMNI.TXT
 Marko Markovic 45.4 12.3 11
 Milan Jevremovic 35.2 1.3 9
 Maja Agic 60 19 20
 Nadica Zec 54.2 10 15.8
 Jovana Milic 23.3 2 5.6

IZLAZ:
 1. Maja Agic 60.0 19.0 20.0 99.0
 2. Nadica Zec 54.2 10.0 15.8 80.0
 3. Marko Markovic 45.4 12.3 11.0 68.7
 4. Milan Jevremovic 35.2 1.3 9.0 45.5
 ----------------------------------
 5. Jovana Milic 23.3 2.0 5.6 30.9
```

Test 2

```
PRIJEMNI.TXT
 [Ova datoteka ne postoji]

IZLAZ:
 Greska prilikom otvaranja datoteke!
```

*** Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronala zi 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`.

Primer 1

```
POZIV: ./a.out rodjendani.txt

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.

INTERAKCIJA PROGRAMA:
 Unesite datum: 23.04.
 Slavljenik: Maja Agic
 Unesite datum: 01.01.
 Slavljenik: Nadica Zec
 Unesite datum: 01.05.
 Slavljenik: Jovana Milic 05.05.
 Unesite datum:
```

Primer 2

```
POZIV: ./a.out rodjendani1.txt

INTERAKCIJA PROGRAMA:
 Greska prilikom otvaranja datoteke!
```

Zadatak 4.19 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla.

Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. NAPOMENA: *Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka stabla.h predstavljati popis funkcija biblioteke, a datoteka stabla.c njihove implementacije. Programe koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

Primer 1

```
INTERAKCIJA PROGRAMA:
 Prvo stablo:
 10 5 15 3 2 4 30 12 14 13 0
 Drugo stablo:
 10 15 5 3 4 2 12 14 13 30 0
 Stabla jesu identicna.
```

Primer 2

```
INTERAKCIJA PROGRAMA:
 Prvo stablo:
 10 5 15 4 3 2 30 12 14 13 0
 Drugo stablo:
 10 15 5 3 4 2 12 14 13 30 0
 Stabla nisu identicna.
```

*** Zadatak 4.20** Napisati program 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.

Primer 1

```
INTERAKCIJA PROGRAMA:
 Prvo stablo: 1 7 8 9 2 2
 Drugo stablo: 3 9 6 11 1
 Unija: 1 1 2 2 3 6 7 8 9 9 11
 Presek: 1 9
 Razlika: 2 2 7 8
```

Primer 2

```
INTERAKCIJA PROGRAMA:
 Prvo stablo: 11 2 7 5
 Drugo stablo: 4 3 3 7
 Unija: 2 3 3 4 5 7 7 11
 Presek: 7
 Razlika: 2 5 11
```

Zadatak 4.21 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

Primer 1

```
Interakcija programa:
 n: 7
 a: 1 11 8 6 37 25 30
 1 6 8 11 25 30 37
```

Primer 2

```
Interakcija programa:
 n: 55
 Greska: pogresna dimenzija niza!
```

Zadatak 4.22 Dato je binarno pretraživačko stablo celih brojeva.

(a) Napisati funkciju koja izračunava broj čvorova stabla.

(b) Napisati funkciju koja izračunava broj listova stabla.

(c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.

(d) Napisati funkciju koja izračunava zbir čvorova stabla.

(e) Napisati funkciju koja izračunava najveći element stabla.

(f) Napisati funkciju koja izračunava dubinu stabla.

(g) Napisati funkciju koja izračunava broj čvorova na i-tom nivou stabla.

(h) Napisati funkciju koja ispisuje sve elemente na i-tom nivou stabla.

(i) Napisati funkciju koja izračunava maksimalnu vrednost na i-tom nivou stabla.

(j) Napisati funkciju koja izračunava zbir čvorova na i-tom nivou stabla.

(k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x.

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije.

Test 1

```
Poziv: ./a.out 2 15

Ulaz:
 10 5 15 3 2 4 30 12 14 13

Izlaz:
 Broj cvorova: 10
 Broj listova: 4
 Pozitivni listovi: 2 4 13 30
 Zbir cvorova: 108
 Najveci element: 30
 Dubina stabla: 5
 Broj cvorova na 2. nivou: 3
 Elementi na 2. nivou: 3 12 30
 Maksimalni na 2. nivou: 30
 Zbir na 2. nivou: 45
 Zbir elemenata manjih ili jednakih od 15: 78
```

Test 2

```
Poziv: ./a.out 3 31

Ulaz:
 24 53 61 9 7 55 20 16

Izlaz:
 Broj cvorova: 8
 Broj listova: 3
 Pozitivni listovi: 7 16 55
 Zbir cvorova: 245
 Najveci element: 61
 Dubina stabla: 4
 Broj cvorova na 3. nivou: 2
 Elementi na 3. nivou: 16 55
 Maksimalni na 3. nivou: 55
 Zbir na 3. nivou: 71
 Zbir elemenata manjih ili jednakih od 31: 76
```

Zadatak 4.23 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Ulaz:
 10 5 15 3 2 4 30 12 14 13

Izlaz:
 0.nivo: 10
 1.nivo: 5 15
 2.nivo: 3 12 30
 3.nivo: 2 4 14
 4.nivo: 13
```

Test 2

```
Ulaz:
 6 11 8 3 -2

Izlaz:
 0.nivo: 6
 1.nivo: 3 11
 2.nivo: -2 8
```

Test 3

```
Ulaz:
 24 53 61 9 7 55 20 16

Izlaz:
 0.nivo: 24
 1.nivo: 9 53
 2.nivo: 7 20 61
 3.nivo: 16 55
```

*** Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1

```
Interakcija programa:
 Prvo stablo: 11 20 5 3 0
 Drugo stablo: 8 14 30 1 0
 Stabla su slicna kao u ogledalu.
```

Primer 2

```
Interakcija programa:
 Prvo stablo: 11 20 5 3 0
 Drugo stablo: 8 20 15 0
 Stabla nisu slicna kao u ogledalu.
```

Zadatak 4.25 AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Ulaz:
 10 5 15 2 11 16 1 13

Izlaz:
 7
```

Test 2

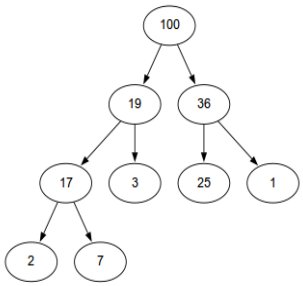
```
Ulaz:
 16 30 40 24 10 18 45 22

Izlaz:
 6
```

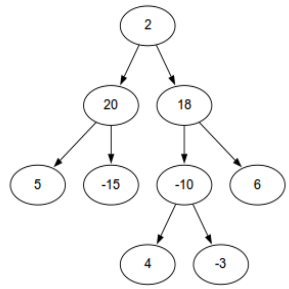
Zadatak 4.26 Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i glavni program koji kreira stablo kao na slici 4.1, poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

Test 1

```
Izlaz:
 Zadato stablo je heap!
```



Slika 4.1: *Zadatak 4.26*



Slika 4.2: *Zadatak 4.27*

Zadatak 4.27 Dato je binarno stablo celih brojeva.

(a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

(b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.

(c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.

(d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardnom izlazu.

Test 1

```
Izlaz:
 Vrednost u cvoru sa maksimalnim desnim zbirom:: 18
 Vrednost u cvoru sa minimalnim levim zbirom: 18
 2 18 -10 4
 2 20 -15
```

4.3 Rešenja

Rešenje 4.1

```
#ifndef _LISTA_H
#define _LISTA_H

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
```

```c
/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

/* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

/* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi);

/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
   neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
   sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
    */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
   neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

#endif
```

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

```
4
Cvor *napravi_cvor(int broj)
6 {
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8   if (novi == NULL)
    return NULL;
10
  novi->vrednost = broj;
12   novi->sledeci = NULL;
  return novi;
14 }

16 void oslobodi_listu(Cvor ** adresa_glave)
{
18   Cvor *pomocni = NULL;

20   /* Ako lista nije prazna, onda treba osloboditi memoriju. */
  while (*adresa_glave != NULL) {
22     /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor koji predstavlja glavu liste */
24     pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
26     /* Sledeci cvor je nova glava liste. */
    *adresa_glave = pomocni;
28   }
}
30
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
32 {
  /* Kreira se nov cvor i proverava se da li je bilo greske pri
34     alokaciji. */
  Cvor *novi = napravi_cvor(broj);
36   if (novi == NULL)
    return 1;
38
  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste. */
40   novi->sledeci = *adresa_glave;
  *adresa_glave = novi;
42
  return 0;
44 }

46 Cvor *pronadji_poslednji(Cvor * glava)
{
48   /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
  if (glava == NULL)
50     return NULL;

52   /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
54     ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
     cvor liste i vraca se vrednost pokazivaca glava.
```

```
56
       Pokazivac glava je argument funkcije i njegove promene nece se
58     odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
   while (glava->sledeci != NULL)
60   glava = glava->sledeci;

62 return glava;
}
64
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
66 {
   Cvor *novi = napravi_cvor(broj);
68 if (novi == NULL)
     return 1;
70
   /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
72    ujedno i cela lista. Azurira se vrednost na koju pokazuje
      adresa_glave i tako se azurira i pokazivacka promenljiva u
74    pozivajucoj funkciji. */
   if (*adresa_glave == NULL) {
76   *adresa_glave = novi;
     return 0;
78 }

80 /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
      se dodaje na kraj liste kao sledbenik poslednjeg. */
82 Cvor *poslednji = pronadji_poslednji(*adresa_glave);
   poslednji->sledeci = novi;
84
   return 0;
86 }

88 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
90 /* U praznoj listi nema takvog mesta i vraca se NULL. */
   if (glava == NULL)
92   return NULL;

94 /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
      pokazivala na cvor ciji je sledeci ili ne postoji ili ima
96    vrednost vecu ili jednaku vrednosti novog cvora.

98    Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
      provera da li se doslo do poslednjeg cvora liste pre nego sto se
100   proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
      sledeci ne postoji, pa ni njegova vrednost. */
102 while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
     glava = glava->sledeci;
104
   /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
106   poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
      vecu od broj. */
```

```
108   return glava;
}
110
void dodaj_iza(Cvor * tekuci, Cvor * novi)
112 {
  /* Novi cvor se dodaje iza tekuceg cvora. */
114   novi->sledeci = tekuci->sledeci;
  tekuci->sledeci = novi;
116 }

118 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
120   /* U slucaju prazne liste glava nove liste je novi cvor. Ukoliko je
      doslo do greske pri alokaciji memorije ćvraa se 1. */
122   if (*adresa_glave == NULL) {
    Cvor *novi = napravi_cvor(broj);
124     if (novi == NULL)
      return 1;
126     *adresa_glave = novi;
    return 0;
128   }

130   /* Lista nije prazna. */
  /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
132     treba dodati na pocetak liste. */
  if ((*adresa_glave)->vrednost >= broj) {
134     return dodaj_na_pocetak_liste(adresa_glave, broj);
  }
136
  /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
138     tada se pronalazi cvor liste iza koga treba uvezati nov cvor. */
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
140   Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
142     return 1;

144   /* Uvezuje se novi cvor iza pomocnog. */
  dodaj_iza(pomocni, novi);
146   return 0;
}
148
Cvor *pretrazi_listu(Cvor * glava, int broj)
150 {
  for (; glava != NULL; glava = glava->sledeci)
152     if (glava->vrednost == broj)
      return glava;
154
  /* Nema trazenog broja u listi i vraca se NULL. */
156   return NULL;
}
158
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
```

```
{
  /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
  for (; glava != NULL && glava->vrednost <= broj;
       glava = glava->sledeci)
    if (glava->vrednost == broj)
      return glava;

  return NULL;
}

void obrisi_cvor(Cvor ** adresa_glave, int broj)
{
  Cvor *tekuci = NULL;
  Cvor *pomocni = NULL;

  /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
     broju, i azurira se pokazivac na glavu liste. */
  while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
    /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
       adresi adresa_glave. */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *adresa_glave = pomocni;
  }

  /* Ako je nakon toga lista ostala prazna, izlazi se iz funkcije. */
  if (*adresa_glave == NULL)
    return;

  /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje na
     cvor cija vrednost je razlicita od trazenog broja. Isto vazi i
     za sve cvorove levo od tekuceg. Poredi se vrednost sledeceg
     cvora (ako postoji) sa trazenim brojem. Cvor se brise ako je
     jednak, ili, ako je razlicit, prelazi se na sledeci cvor. Ovaj
     postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
  tekuci = *adresa_glave;
  while (tekuci->sledeci != NULL)
    if (tekuci->sledeci->vrednost == broj) {
      /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
         prvo cuva u pomocni. */
      pomocni = tekuci->sledeci;
      /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
         njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
         sledeci od cvora koji se brise. */
      tekuci->sledeci = pomocni->sledeci;
      /* Sada treba osloboditi cvor sa vrednoscu broj. */
      free(pomocni);
    } else {
      /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
         pomera na sledeci. */
      tekuci = tekuci->sledeci;
```

```
    }
  return;
}

void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
  Cvor *tekuci = NULL;
  Cvor *pomocni = NULL;

  /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
     broju i azurira se pokazivac na glavu liste. */
  while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
    /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
       adresi adresa_glave. */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *adresa_glave = pomocni;
  }

  /* Ako je nakon toga lista ostala prazna, funkcija se prekida. Isto
     se radi i ukoliko glava liste sadrzi vrednost koja je veca od
     broja, jer kako je lista sortirana rastuce nema potrebe broj
     traziti u repu liste. */
  if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
    return;

  /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
     na cvor cija vrednost je manja od trazenog broja, kao i svim
     cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
     je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
     ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
     cija vrednost je veca od trazenog broja. */
  tekuci = *adresa_glave;
  while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
    )
    if (tekuci->sledeci->vrednost == broj) {
      pomocni = tekuci->sledeci;
      tekuci->sledeci = tekuci->sledeci->sledeci;
      free(pomocni);
    } else {
      /* Ne treba brisati sledeceg od tekuceg jer je manji od
         trazenog i tekuci se pomera na sledeci cvor. */
      tekuci = tekuci->sledeci;
    }
  return;
}

void ispisi_listu(Cvor * glava)
{
  /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
     jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
```

```
     na glavu liste iz pozivajuce funkcije. */
  putchar('[');
  for (; glava != NULL; glava = glava->sledeci) {
    printf("%d", glava->vrednost);
    if (glava->sledeci != NULL)
      printf(", ");
  }

  printf("]\n");
}
```

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

/* 1) Glavni program */
int main()
{
  /* Lista je prazna na pocetku. */
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testiranje dodavanja novog broja na pocetak liste */
  printf("Unesite brojeve: (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  printf("\nUnesite broj koji se trazi: ");
  scanf("%d", &broj);

  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  oslobodi_listu(&glava);

  return 0;
}
```

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

/* 2) Glavni program */
int main()
{
  Cvor *glava = NULL;
  int broj;

  /* Testiranje dodavanja novog broja na kraj liste. */
  printf("Unesite brojeve: (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  oslobodi_listu(&glava);

  return 0;
}
```

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

/* 3) Glavni program */
int main()
{
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

```

```
  /* Testira se dodavanje u listu tako da ona bude neopadajuce
     uredjena */
  printf("Unosite brojeve (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_sortirano(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  printf("\nUnesite broj koji se trazi: ");
  scanf("%d", &broj);

  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  oslobodi_listu(&glava);

  return 0;
}
```

Rešenje 4.2

```
#ifndef _LISTA_H
#define _LISTA_H

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
} Cvor;
```

```
10
   /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
      na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14 Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
      ciji se pokazivac glava nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

20 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
      bilo greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
      pri alokaciji memorije, inace vraca 0. */
26 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

28 /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
      ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
30    memorije, inace vraca 0. */
   int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32
   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
34    Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
      NULL u slucaju da takav cvor ne postoji u listi. */
36 Cvor *pretrazi_listu(Cvor * glava, int broj);

38 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
      U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
40    neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
      sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
       */
42 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

44 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
      pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
46    obrise stara glava liste. */
   void obrisi_cvor(Cvor ** adresa_glave, int broj);
48
   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
50    oslanjajuci se na cinjenicu da je prosledjena lista sortirana
      neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
52    promenjen ukoliko se obrise stara glava liste. */
   void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
54
   /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
56    zapetama. */
   void ispisi_vrednosti(Cvor * glava);
58
   /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
60    liste, razdvojene zapetama i uokvirene zagradama. */
```

```
void ispisi_listu(Cvor * glava);

#endif
```

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

Cvor *napravi_cvor(int broj)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  novi->vrednost = broj;
  novi->sledeci = NULL;
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
  /* Lista je vec prazna */
  if (*adresa_glave == NULL)
    return;

  /* Ako lista nije prazna, treba osloboditi memoriju. Pre
     oslobadjanja memorije za glavu liste, treba osloboditi rep
     liste. */
  oslobodi_listu(&(*adresa_glave)->sledeci);
  /* Nakon oslobodjenog repa, oslobadja se glava liste, i azurira se
     glava u pozivajucoj funkciji tako da odgovara praznoj listi */
  free(*adresa_glave);
  *adresa_glave = NULL;
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se nov cvor i proverava se da li je bilo greske pri
     alokaciji */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;
  return 0;
}

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  if (*adresa_glave == NULL) {
```

```
49      /* Glava liste je upravo novi cvor i ujedno i cela lista. */
        Cvor *novi = napravi_cvor(broj);
51      /* Ukoliko je bilo greske pri alokaciji vraca se 1. */
        if (novi == NULL)
53        return 1;

55      /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
           azurira i pokazivacka promenljiva u pozivajucoj funkciji. */
57      *adresa_glave = novi;
        return 0;
59    }

61    /* Ako lista nije prazna, broj se dodaje u rep liste. */
      /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
63       nov broj se dodaje u rep liste u rekurzivnom pozivu. Informacija
         o uspesnosti alokacije u rekurzivnom pozivu funkcija prosledjuje
65       visem rekurzivnom pozivu koji tu informaciju vraca u rekurzivni
         poziv iznad, sve dok se ne vrati u main. Tek je iz main funkcije
67       moguce pristupiti pravom pocetku liste i osloboditi je celu, ako
         ima potrebe. Ako je funkcija vratila 0, onda nije bilo greske.
        */
69    return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
  }
71
  int dodaj_sortirano(Cvor ** adresa_glave, int broj)
73{
    /* U slucaju prazne liste, glava nove liste je upravo novi cvor. */
75  if (*adresa_glave == NULL) {
      Cvor *novi = napravi_cvor(broj);
77    if (novi == NULL)
        return 1;
79
      *adresa_glave = novi;
81    return 0;
    }
83
    /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
85     glavi liste, onda se dodaje na pocetak liste i vraca se
       informacija o uspesnosti alokacije. */
87  if ((*adresa_glave)->vrednost >= broj)
      return dodaj_na_pocetak_liste(adresa_glave, broj);
89
    /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
91     bude sortirana lista. */
    return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
93}

95Cvor *pretrazi_listu(Cvor * glava, int broj)
  {
97  /* U praznoj listi ga sigurno nema */
    if (glava == NULL)
99    return NULL;
```

```
101   /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
        */
      if (glava->vrednost == broj)
103     return glava;

105   /* Inace, pretraga se nastavlja u repu liste. */
      return pretrazi_listu(glava->sledeci, broj);
107 }

109 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
    {
111   /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
         vrednosti u glavi liste, jer je lista neopadajuce sortirana. */
113   if (glava == NULL || glava->vrednost > broj)
        return NULL;
115
      /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
        */
117   if (glava->vrednost == broj)
        return glava;
119
      /* Inace, pretraga se nastavlja u repu. */
121   return pretrazi_listu(glava->sledeci, broj);
    }
123
    void obrisi_cvor(Cvor ** adresa_glave, int broj)
125 {
      /* U praznoj listi, nema cvorova za brisanje. */
127   if (*adresa_glave == NULL)
        return;
129
      /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj. */
131   obrisi_cvor(&(*adresa_glave)->sledeci, broj);

133   /* Preostaje provera da li glavu liste treba obrisati. */
      if ((*adresa_glave)->vrednost == broj) {
135     /* pomocni pokazuje na cvor koji treba da se obrise. */
        Cvor *pomocni = *adresa_glave;
137     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
           brise se cvor koji je bio glava liste. */
139     *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
141   }
    }
143
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
145 {
      /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
147       od broja, kako je lista sortirana rastuce nema potrebe broj
          traziti u repu liste i zato se funkcija prekida. */
149   if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
```

```
    return;

  /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
  obrisi_cvor(&(*adresa_glave)->sledeci, broj);

  /* Preostaje provera da li glavu liste treba obrisati. */
  if ((*adresa_glave)->vrednost == broj) {
    /* pomocni pokazuje na cvor koji treba da se obrise. */
    Cvor *pomocni = *adresa_glave;
    /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
       brise se cvor koji je bio glava liste. */
    *adresa_glave = (*adresa_glave)->sledeci;
    free(pomocni);
  }
}

void ispisi_vrednosti(Cvor * glava)
{
  /* Prazna lista */
  if (glava == NULL)
    return;

  /* Ispisuje se vrednost u glavi liste. */
  printf("%d", glava->vrednost);

  /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
     se poziva ista funkcija za ispis ostalih. */
  if (glava->sledeci != NULL) {
    printf(", ");
    ispisi_vrednosti(glava->sledeci);
  }
}

void ispisi_listu(Cvor * glava)
{
  /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
     jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
     na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
     zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
     rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
     elemente razdvojene zapetom i razmakom. */
  putchar('[');
  ispisi_vrednosti(glava);
  printf("]\n");
}
```

Rešenje 4.3

```
#ifndef _LISTA_H
#define _LISTA_H
```

```c
/* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   vrednost i pokazivace na sledeci i prethodni cvor liste. */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
  struct cvor *prethodni;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

/* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

/* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi);

/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
   neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
   sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
    */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
```

```
/* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
   pokazivac glava se nalazi na adresi adresa_glave. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
   sortirana. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

/* Funkcija prikazuje cvorove liste pocev od kraja ka glavi liste. */
void ispisi_listu_unazad(Cvor * glava);

#endif
```

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

Cvor *napravi_cvor(int broj)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  novi->vrednost = broj;
  novi->sledeci = NULL;
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  /* Ako lista nije prazna, onda treba osloboditi memoriju. */
  while (*adresa_glave != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi memoriju cvora koji predstavlja glavu liste. */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    /* Sledeci cvor je nova glava liste. */
    *adresa_glave = pomocni;
```

```
28    }
  }
30
  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
32  {
    Cvor *novi = napravi_cvor(broj);
34    if (novi == NULL)
      return 1;
36
    /* Sledbenik novog cvora je glava stare liste */
38    novi->sledeci = *adresa_glave;
    /* Ako stara lista nije bila prazna, onda prethodni od glave treba
40       da bude nov cvor. */
    if (*adresa_glave != NULL)
42      (*adresa_glave)->prethodni = novi;
    /* Novi cvor je nova glava liste. */
44    *adresa_glave = novi;

46    return 0;
  }
48
  Cvor *pronadji_poslednji(Cvor * glava)
50  {
    /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
52    if (glava == NULL)
      return NULL;
54
    /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
56       glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
       ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
58       cvor liste i vraca se vrednost pokazivaca glava.

60       Pokazivac glava je argument funkcije i njegove promene nece se
       odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
62    while (glava->sledeci != NULL)
      glava = glava->sledeci;
64
    return glava;
66  }

68  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
  {
70    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
72      return 1;

74    /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
       ujedno i cela lista. Azurira se vrednost na koju pokazuje
76       adresa_glave i tako se azurira i pokazivacka promenljiva u
       pozivajucoj funkciji. */
78    if (*adresa_glave == NULL) {
      *adresa_glave = novi;
```

```
80      return 0;
    }
82
    /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
84     se dodaje na kraj liste kao sledbenik poslednjeg. */
    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
86  poslednji->sledeci = novi;
    novi->prethodni = poslednji;
88
    return 0;
90  }

92  Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
    {
94    /* U praznoj listi nema takvog mesta i vraca se NULL. */
    if (glava == NULL)
96    return NULL;

98    /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
       pokazivala na cvor ciji je sledeci ili ne postoji ili ima
100    vrednost vecu ili jednaku vrednosti novog cvora.

102    Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
       provera da li se doslo do poslednjeg cvora liste pre nego sto se
104    proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
       sledeci ne postoji, pa ni njegova vrednost. */
106   while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
      glava = glava->sledeci;
108
    /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
110    poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
       vecu od broj. */
112   return glava;
    }
114
    void dodaj_iza(Cvor * tekuci, Cvor * novi)
116 {
    novi->sledeci = tekuci->sledeci;
118   novi->prethodni = tekuci;

120   /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
       a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
122    na ispravne adrese. */
    if (tekuci->sledeci != NULL)
124   tekuci->sledeci->prethodni = novi;
    tekuci->sledeci = novi;
126 }

128 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
    {
130   /* Ako je lista prazna, glava nove liste je novi cvor. */
    if (*adresa_glave == NULL) {
```

```
132     Cvor *novi = napravi_cvor(broj);
        if (novi == NULL)
134       return 1;
        *adresa_glave = novi;
136     return 0;
      }
138
      /* Ukoliko je vrednost glave liste veca ili jednaka od nove
140        vrednosti onda nov cvor treba staviti na pocetak liste. */
      if ((*adresa_glave)->vrednost >= broj) {
142     dodaj_na_pocetak_liste(adresa_glave, broj);
        return 0;
144   }

146   Cvor *novi = napravi_cvor(broj);
      if (novi == NULL)
148     return 1;

150   /* Pronazi se cvor iza koga treba uvezati nov cvor. */
      Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
152   dodaj_iza(pomocni, novi);

154   return 0;
    }
156
    Cvor *pretrazi_listu(Cvor * glava, int broj)
158 {
      for (; glava != NULL; glava = glava->sledeci)
160     if (glava->vrednost == broj)
          return glava;
162
      /* Nema trazenog broja u listi i vraca se NULL. */
164   return NULL;
    }
166
    Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
168 {
      /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
170   for (; glava != NULL && glava->vrednost <= broj;
           glava = glava->sledeci)
172     if (glava->vrednost == broj)
          return glava;
174
      /* Nema trazenog broja u listi i bice vraceno NULL. */
176   return NULL;
    }
178
    /* Kod dvostruko povezane liste brisanje cvora na koji pokazuje
180    tekuci moze se lako uraditi jer sadrzi pokazivace na svog
       sledbenika i prethodnika u listi. */
182 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
    {
```

```
184   /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
      if (tekuci == NULL)
186     return;

188   /* Ako postoji prethodnik od tekuceg, onda se postavlja da njegov
         sledeci bude sledeci od tekuceg. */
190   if (tekuci->prethodni != NULL)
        tekuci->prethodni->sledeci = tekuci->sledeci;
192
      /* Ako postoji sledbenik tekuceg, onda njegov prethodnik treba da
194      bude prethodnik tekuceg. */
      if (tekuci->sledeci != NULL)
196     tekuci->sledeci->prethodni = tekuci->prethodni;

198   /* Ako je glava cvor koji se brise, nova glava liste bice sledbenik
         stare glave. */
200   if (tekuci == *adresa_glave)
        *adresa_glave = tekuci->sledeci;
202
      /* Oslobadja se dinamicki alociran prostor za cvor tekuci. */
204   free(tekuci);
    }
206
    void obrisi_cvor(Cvor ** adresa_glave, int broj)
208 {
      Cvor *tekuci = *adresa_glave;
210
      while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
212     obrisi_tekuci(adresa_glave, tekuci);
    }
214
    void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
216 {
      Cvor *tekuci = *adresa_glave;
218
      while ((tekuci =
220             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, tekuci);
222 }

224 void ispisi_listu(Cvor * glava)
    {
226   /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
         jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
228      na glavu liste iz pozivajuce funkcije. */
      putchar('[');
230   for (; glava != NULL; glava = glava->sledeci) {
        printf("%d", glava->vrednost);
232     if (glava->sledeci != NULL)
          printf(", ");
234   }
```

```
  printf("]\n");
}

void ispisi_listu_unazad(Cvor * glava)
{
  putchar('[');
  if (glava == NULL) {
    printf("]\n");
    return;
  }

  glava = pronadji_poslednji(glava);

  for (; glava != NULL; glava = glava->prethodni) {
    printf("%d", glava->vrednost);
    if (glava->prethodni != NULL)
      printf(", ");
  }
  printf("]\n");
}
```

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

/* 1) Glavni program */
int main()
{
  /* Lista je prazna na pocetku. */
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testiranje dodavanja novog broja na pocetak liste. */
  printf("Unesite brojeve: (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  printf("\nUnesite broj koji se trazi u listi: ");
  scanf("%d", &broj);

  trazeni = pretrazi_listu(glava, broj);
```

```
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(glava);

  oslobodi_listu(&glava);

  return 0;
}
```

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

/* 2) Glavni program */
int main()
{
  Cvor *glava = NULL;
  int broj;

  /* Testiranje dodavanja novog broja na kraj liste. */
  printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  printf("\nUnesite broj koji se brise iz liste: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza. */
  obrisi_cvor(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(glava);

  oslobodi_listu(&glava);
```

```
  return 0;
}
```

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

/* 3) Glavni program */
int main()
{
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se dodavanje u listu tako da ona bude neopadajuce
     uredjena */
  printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_sortirano(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  printf("\nUnesite broj koji se trazi u listi: ");
  scanf("%d", &broj);

  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  printf("\nUnesite broj koji se brise iz liste: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza. */
  obrisi_cvor_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(glava);
```

```
  oslobodi_listu(&glava);

  return 0;
}
```

Rešenje 4.4

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

typedef struct cvor {
  char zagrada;
  struct cvor *sledeci;
} Cvor;

int main()
{
  Cvor *stek = NULL;
  FILE *ulaz = NULL;
  char c;
  Cvor *pomocni = NULL;

  ulaz = fopen("izraz.txt", "r");
  if (ulaz == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke izraz.txt!\n");
    exit(EXIT_FAILURE);
  }

  while ((c = fgetc(ulaz)) != EOF) {
    /* Ako je ucitana otvorena zagrada, stavlja se na stek. */
    if (c == '(' || c == '{' || c == '[') {
      pomocni = (Cvor *) malloc(sizeof(Cvor));
      if (pomocni == NULL) {
        fprintf(stderr, "Greska prilikom alokacije memorije!\n");
        return 1;
      }
      pomocni->zagrada = c;
      pomocni->sledeci = stek;
      stek = pomocni;
    }
    /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
       prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
       otvorena zagrada. */
    else {
      if (c == ')' || c == '}' || c == ']') {
        if (stek != NULL && ((stek->zagrada == '(' && c == ')')
                             || (stek->zagrada == '{' && c == '}')
                             || (stek->zagrada == '[' && c == ']')))
    {
```

```
43            /* Sa vrha steka se uklanja otvorena zagrada */
              pomocni = stek->sledeci;
45            free(stek);
              stek = pomocni;
47          } else {
              /* Zagrade u izrazu nisu ispravno uparene. */
49            break;
            }
51        }
      }
53  }
    /* Procitana je cela datoteka. Zatvaramo je. */
55  fclose(ulaz);

57  /* Ako je stek prazan i procitana je cela datoteka, zagrade su
       ispravno uparene, u suprotnom, nisu. */
59  if (stek == NULL && c == EOF)
      printf("Zagrade su ispravno uparene.\n");
61  else {
      printf("Zagrade nisu ispravno uparene.\n");
63    /* U slucaju neispravnog uparivanja treba osloboditi memoriju
         koja je ostala zauzeta stekom. */
65    while (stek != NULL) {
        pomocni = stek->sledeci;
67      free(stek);
        stek = pomocni;
69    }
  }
71

  return 0;
73 }
```

Rešenje 4.5

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

6 #define MAX 100

8 #define OTVORENA 1
  #define ZATVORENA 2
10
  #define VAN_ETIKETE 0
12 #define PROCITANO_MANJE 1
  #define U_ETIKETI 2
14
  /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
16    pokazivac na sledeci cvor. */
  typedef struct cvor {
```

```
18    char etiketa[MAX];
      struct cvor *sledeci;
20  } Cvor;

22  /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
       adresu ili NULL ako alokacija nije bila uspesna. */
24  Cvor *napravi_cvor(char *etiketa)
    {
26    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
      if (novi == NULL)
28      return NULL;

30    /* Inicijalizacija polja u novom cvoru */
      if (strlen(etiketa) >= MAX) {
32      fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
        etiketa[MAX - 1] = '\0';
34    }
      strcpy(novi->etiketa, etiketa);
36    novi->sledeci = NULL;
      return novi;
38  }

40  /* Funkcija oslobadja memoriju zauzetu stekom. */
    void oslobodi_stek(Cvor ** adresa_vrha)
42  {
      Cvor *pomocni;
44    while (*adresa_vrha != NULL) {
        pomocni = *adresa_vrha;
46      *adresa_vrha = (*adresa_vrha)->sledeci;
        free(pomocni);
48    }
    }
50

    /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
52     ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
       zauzeta memorija za listu ciji se pokazivac vrh nalazi na adresi
54     adresa_vrha. */
    void proveri_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
56  {
      if (novi == NULL) {
58      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_stek(adresa_vrha);
60      exit(EXIT_FAILURE);
      }
62  }

64  /* Funkcija postavlja na vrh steka novu etiketu. */
    void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
66  {
      Cvor *novi = napravi_cvor(etiketa);
68    proveri_alokaciju(adresa_vrha, novi);
      novi->sledeci = *adresa_vrha;
```

```
  *adresa_vrha = novi;
}

/* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
   pokazivac razlicit od NULL, tada u niz karaktera na koji on
   pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
   suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
   samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
   suprotnom. */
int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
{
  Cvor *pomocni;

  /* Pokusaj skidanja vrednost sa vrha praznog steka rezultuje
     greskom i vraca se 0. */
  if (*adresa_vrha == NULL)
    return 0;

  /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
     adresu kopira etiketa sa vrha steka. */
  if (etiketa != NULL)
    strcpy(etiketa, (*adresa_vrha)->etiketa);

  /* Element sa vrha steka se uklanja. */
  pomocni = *adresa_vrha;
  *adresa_vrha = (*adresa_vrha)->sledeci;
  free(pomocni);

  return 1;
}

/* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
   steka. Ukoliko je stek prazan, vraca NULL. */
char *vrh_steka(Cvor * vrh)
{
  if (vrh == NULL)
    return NULL;
  return vrh->etiketa;
}

/* Funkcija prikazuje stek pocev od vrha prema dnu. */
void prikazi_stek(Cvor * vrh)
{
  for (; vrh != NULL; vrh = vrh->sledeci)
    printf("<%s>\n", vrh->etiketa);
}

/* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
   etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
   Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
   procita etiketa. Vraca OTVORENA, ako je procitana otvorena
   etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa. */
```

```
122 int uzmi_etiketu(FILE * f, char *etiketa)
    {
124   int c;
      int i = 0;
126   /* Stanje predstavlja informaciju dokle se stalo sa citanjem
         etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
128      nije zapoceto citanje. */
      /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
130      OTVORENA ili ZATVORENA. */
      int stanje = VAN_ETIKETE;
132   int tip;

134   /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
         to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
136      samo malim slovima. */
      while ((c = fgetc(f)) != EOF) {
138     switch (stanje) {
        case VAN_ETIKETE:
140       if (c == '<')
            stanje = PROCITANO_MANJE;
142       break;
        case PROCITANO_MANJE:
144       if (c == '/') {
            /* Cita se zatvorena etiketa. */
146         tip = ZATVORENA;
          } else {
148         if (isalpha(c)) {
              /* Cita se otvorena etiketa */
150           tip = OTVORENA;
              etiketa[i++] = tolower(c);
152         }
          }
154       /* Od sada se cita etiketa i zato se menja stanje. */
          stanje = U_ETIKETI;
156       break;
        case U_ETIKETI:
158       if (isalpha(c) && i < MAX - 1) {
            /* Ako je procitani karakter slovo i nije premasena
160            dozvoljena duzina etikete, procitani karakter se smanjuje
               i smesta u etiketu. */
162         etiketa[i++] = tolower(c);
          } else {
164         /* Inace, staje se sa citanjem etikete. Korektno se zavrsava
               niska koja sadrzi procitanu etiketu i vraca se njen tip.
        */
166         etiketa[i] = '\0';
            return tip;
168       }
          break;
170     }
      }
172   /* Doslo se do kraja datoteke pre nego sto je procitana naredna
```

```
     etiketa i vraca se EOF. */
174  return EOF;
}
176
int main(int argc, char **argv)
178 {
  /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
180    nije procitana. */
  Cvor *vrh = NULL;
182  char etiketa[MAX];
  int tip;
184  int uparene = 1;
  FILE *f = NULL;
186
  /* Ime datoteke se preuzima iz komandne linije. */
188  if (argc < 2) {
    fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
190    exit(0);
  }
192
  /* Datoteka se otvara za citanje. */
194  if ((f = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
196            argv[1]);
    exit(1);
198  }

200  /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
  while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
202    /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
       etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
204    potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
       koje nemaju sadrzaj (npr link). Zbog jednostavnosti
206    pretpostavlja se da njih nema u HTML dokumentu. */
    if (tip == OTVORENA) {
208      if (strcmp(etiketa, "br") != 0
          && strcmp(etiketa, "hr") != 0
210        && strcmp(etiketa, "meta") != 0)
        potisni_na_stek(&vrh, etiketa);
212    }
    /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
214    u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
       uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
216    je taj uslov ispunjen, skida se sa steka, jer je upravo
       zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
218    nisu pravilno uparene. */
    else if (tip == ZATVORENA) {
220      if (vrh_steka(vrh) != NULL
          && strcmp(vrh_steka(vrh), etiketa) == 0)
222        skini_sa_steka(&vrh, NULL);
      else {
224        printf("Etikete nisu pravilno uparene\n");
```

```
        printf("(nadjena je etiketa </%s>", etiketa);
        if (vrh_steka(vrh) != NULL)
          printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh))
    ;
        else
          printf(" koja nije otvorena)\n");
        uparene = 0;
        break;
      }
    }
  }
  /* Zavrseno je citanje datoteke i zatvara se. */
  fclose(f);

  /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
     da bude prazan. Ukoliko nije, tada postoje etikete koje su
     ostale otvorene. */
  if (uparene) {
    if (vrh_steka(vrh) == NULL)
      printf("Etikete su pravilno uparene!\n");
    else {
      printf("Etikete nisu pravilno uparene\n");
      printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
      /* Oslobadja se memorija zauzeta stekom. */
      oslobodi_stek(&vrh);
    }
  }
  return 0;
}
```

Rešenje 4.6

```
#ifndef _RED_H
#define _RED_H

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

#define MAX 1000
#define JMBG_DUZINA 14

/* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
   opis njegovog zahteva. */
typedef struct {
  char jmbg[JMBG_DUZINA];
  char opis[MAX];
} Zahtev;

/* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
   korisnika i pokazivac na sledeci cvor liste. */
typedef struct cvor {
```

```
  Zahtev nalog;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
   poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
   greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
   treba smestiti u nov cvor zbog smestanja manjeg podatka na
   sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
   bajtovima(B) u odnosu na strukturu Zahtev. */
Cvor *napravi_cvor(Zahtev * zahtev);

/* Funkcija prazni red, oslobadjajuci memoriju koji je red zauzeo. */
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

/* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
   zauzeta memorija za listu ciji se pokazivac pocetak se nalazi na
   adresi adresa_pocetka i prekida program. */
void proveri_alokaciju(Cvor ** adresa_pocetka,
                       Cvor ** adresa_kraja, Cvor * novi);

/* Funkcija dodaje na kraj reda novi zahtev. */
void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                 Zahtev * zahtev);

/* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
   pokazivac razlicit od NULL, tada se u strukturu na koju on
   pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
   suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
   suprotnom. */
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

/* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev korisnika
   na pocetku reda. Ukoliko je red prazan, vraca NULL. */
Zahtev *pocetak_reda(Cvor * pocetak);

/* Funkcija prikazuje sadrzaj reda. */
void prikazi_red(Cvor * pocetak);

#endif
```

```
#include "red.h"

Cvor *napravi_cvor(Zahtev * zahtev)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  novi->nalog = *zahtev;
```

```
  novi->sledeci = NULL;
  return novi;
}

void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
  Cvor *pomocni = NULL;

  while (*pocetak != NULL) {
    pomocni = *pocetak;
    *pocetak = (*pocetak)->sledeci;
    free(pomocni);
  }
  *kraj = NULL;
}

void proveri_alokaciju(Cvor ** adresa_pocetka,
                       Cvor ** adresa_kraja, Cvor * novi)
{
  if (novi == NULL) {
    fprintf(stderr, "Neuspela alokacija za nov cvor\n");
    oslobodi_red(adresa_pocetka, adresa_kraja);
    exit(EXIT_FAILURE);
  }
}

void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                 Zahtev * zahtev)
{
  Cvor *novi = napravi_cvor(zahtev);
  proveri_alokaciju(adresa_pocetka, adresa_kraja, novi);

  /* U red se uvek dodaje na kraj, ali zbog postojanja pokazivaca na
     kraj, dodavanje na kraj je podjednako efikasno kao dodavanje na
     pocetak. */
  if (*adresa_kraja != NULL) {
    (*adresa_kraja)->sledeci = novi;
    *adresa_kraja = novi;
  } else {
    /* Ako je red bio ranije prazan */
    *adresa_pocetka = novi;
    *adresa_kraja = novi;
  }
}

int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev)
{
  Cvor *pomocni = NULL;

  if (*adresa_pocetka == NULL)
    return 0;
```

```
63    if (zahtev != NULL)
        *zahtev = (*adresa_pocetka)->nalog;
65
      pomocni = *adresa_pocetka;
67    *adresa_pocetka = (*adresa_pocetka)->sledeci;
      free(pomocni);
69
      if (*adresa_pocetka == NULL)
71      *adresa_kraja = NULL;

73    return 1;
}
75
Zahtev *pocetak_reda(Cvor * pocetak)
77{
      if (pocetak == NULL)
79      return NULL;

81    return &(pocetak->nalog);
}
83
void prikazi_red(Cvor * pocetak)
85{
      for (; pocetak != NULL; pocetak = pocetak->sledeci)
87      printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

89    printf("\n");
}
```

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

6 #define VREME_ZA_PAUZU 5

8 int main(int argc, char **argv)
{
10    /* Red je prazan. */
      Cvor *pocetak = NULL, *kraj = NULL;
12    Zahtev nov_zahtev;
      Zahtev *sledeci = NULL;
14    char odgovor[3];
      int broj_usluzenih = 0;
16    FILE *izlaz = fopen("izvestaj.txt", "a");

18    if (izlaz == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
20      exit(EXIT_FAILURE);
      }
22
```

```
  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
24       broja i opisa potrebne usluge. */
  printf("Sluzbenik evidentira korisnicke zahteve:\n");
26
  /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
28       JMBG broja procitao i da bi fgets nakon toga procitala ispravan
       red sa opisom zahteva. */
30   printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
  while (scanf("%s", nov_zahtev.jmbg) != EOF) {
32     getchar();
     printf("\tOpis problema: ");
34     fgets(nov_zahtev.opis, MAX - 1, stdin);
     /* Ako je poslednji karakter nov red, eliminise se. */
36     if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
       nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
38     dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
40   }

42   /* Datoteka vise nije potrebna i treba je zatvoriti. */
  fclose(izlaz);
44
  /* Dokle god ima korisnika u redu, treba ih usluziti. */
46   while (1) {
     sledeci = pocetak_reda(pocetak);
48     /* Ako nema nikog vise u redu, prekida se petlja. */
     if (sledeci == NULL)
50       break;

52     printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
     printf("i zahtevom: %s\n", sledeci->opis);
54
     skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
56
     broj_usluzenih++;
58
     printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
60     scanf("%s", odgovor);

62     if (strcmp(odgovor, "Da") == 0)
       dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
64     else
       fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
66               nov_zahtev.opis);

68     if (broj_usluzenih == VREME_ZA_PAUZU) {
       printf("\nDa li je kraj smene? [Da/Ne] ");
70       scanf("%s", odgovor);

72       if (strcmp(odgovor, "Da") == 0)
         break;
74       else
```

```
        broj_usluzenih = 0;
    }
  }

  /*****************************************************************
    Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:

    while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
      printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
            nov_zahtev.jmbg);
      printf("sa zahtevom: %s\n", nov_zahtev.opis);
      broj_usluzenih++;

      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
      scanf("%s", odgovor);
      if (strcmp(odgovor, "Da") == 0)
        dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
      else
        fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
                nov_zahtev.jmbg, nov_zahtev.opis);

      if (broj_usluzenih == VREME_ZA_PAUZU) {
        printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);
        if (strcmp(odgovor, "Da") == 0)
          break;
        else
          broj_usluzenih = 0;
      }
    }
  *****************************************************************/

  /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
     neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
     koju zauzima red sa neobradjenim zahtevima korisnika. */
  oslobodi_red(&pocetak, &kraj);

  return 0;
}
```

Rešenje 4.7

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MAX_DUZINA 20

typedef struct _Cvor {
  unsigned broj_pojavljivanja;
  char etiketa[20];
  struct _Cvor *sledeci;
```

```
} Cvor;

/* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
   ili NULL ako alokacija nije uspesno izvrsena. */
Cvor *napravi_cvor(unsigned br, char *etiketa)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  novi->broj_pojavljivanja = br;
  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;
  return novi;
}

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste. */
void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *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 pokazivac glava se nalazi na adresi
   adresa_glave i prekida program. */
void provera_alokacije(Cvor * novi, Cvor ** adresa_glave)
{
  if (novi == NULL) {
    fprintf(stderr, "malloc() greska u funkciji napravi_cvor()!\n");
    oslobodi_listu(adresa_glave);
    exit(EXIT_FAILURE);
  }
}

/* Funkcija dodaje novi cvor na pocetak liste. */
void dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
                            char *etiketa)
{
  Cvor *novi = napravi_cvor(br, etiketa);
  provera_alokacije(novi, adresa_glave);
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;
}

/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
```

```
   NULL ako takav cvor ne postoji. */
Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
{
  Cvor *tekuci;
  for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
    if (strcmp(tekuci->etiketa, etiketa) == 0)
      return tekuci;
  return NULL;
}

/* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Cvor * glava)
{
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
}

/* Glavni program */
int main(int argc, char **argv)
{
  if (argc != 2) {
    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out datoteka.html!\n")
    ;
    exit(EXIT_FAILURE);
  }

  /* Otvaramo datoteku za citanje */
  FILE *in = NULL;
  in = fopen(argv[1], "r");
  if (in == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  char c;
  int i = 0;
  char procitana[MAX_DUZINA];
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;

  while ((c = fgetc(in)) != EOF) {

    if (c == '<') {
      /* Cita se zatvorena etiketa. */
      if ((c = fgetc(in)) == '/') {
        i = 0;
        while ((c = fgetc(in)) != '>')
          procitana[i++] = c;
      }
      /* Cita se otvorena etiketa. */
```

```
113       else {
            i = 0;
115         procitana[i++] = c;
            while ((c = fgetc(in)) != ' ' && c != '>')
117           procitana[i++] = c;
          }
119       procitana[i] = '\0';

121       /* Trazi se ucitana etiketa medju postojecim cvorovima liste.
             Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
123          sa brojem pojavljivanja 1, inace uvecava se broj
             pojavljivanja etikete. */
125       trazeni = pretrazi_listu(glava, procitana);
          if (trazeni == NULL)
127         dodaj_na_pocetak_liste(&glava, 1, procitana);
          else
129         trazeni->broj_pojavljivanja++;
        }
131   }

133   fclose(in);

135   ispisi_listu(glava);
      oslobodi_listu(&glava);
137
      return 0;
139 }
```

Rešenje 4.8

```
#include<stdio.h>
2 #include<stdlib.h>
#include<string.h>
4
#define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21

8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
     studentu. */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
12  char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
14  struct _Cvor *sledeci;
} Cvor;
16
/* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac na nov
18    cvor ili NULL ukoliko alokacija nije prosla. */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
```

```
22  if (novi == NULL)
      return NULL;
24  /* Inicijalizacija polja novog cvora */
    strcpy(novi->broj_indeksa, broj_indeksa);
26  strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
28  novi->sledeci = NULL;
    return novi;
30 }

32 /* Funkcija oslobadja memoriju zauzetu za cvorove liste. */
   void oslobodi_listu(Cvor ** adresa_glave)
34 {
    if (*adresa_glave == NULL)
36    return;
    /* Rep liste se oslobadja rekurzivnim pozivom. */
38  oslobodi_listu(&(*adresa_glave)->sledeci);
    /* Potom se oslobadja i glava liste. */
40  free(*adresa_glave);
    *adresa_glave = NULL;
42 }

44 /* Funkcija proverava da li je novi NULL pokazivac, i ukoliko jeste
      oslobadja celu listu ciji se pokazivac glava nalazi na adresi
46    adresa_glave i prekida program. */
   void proveri_alokaciju(Cvor ** adresa_glave, Cvor * novi)
48 {
    if (novi == NULL) {
50    fprintf(stderr, "Neuspela alokacija za nov cvor\n");
      oslobodi_listu(adresa_glave);
52    exit(EXIT_FAILURE);
    }
54 }

56 /* Funkcija dodaje novi cvor na pocetak liste. */
   void dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
58                             char *ime, char *prezime)
60 {
    Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
    proveri_alokaciju(adresa_glave, novi);
62  novi->sledeci = *adresa_glave;
    *adresa_glave = novi;
64 }

66 /* Funkcija ispisuje sadrzaj cvorova liste. */
   void ispisi_listu(Cvor * glava)
68 {
    for (; glava != NULL; glava = glava->sledeci)
70    printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
            glava->prezime);
72 }
```

```c
/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu, u
   suprotnom vraca NULL. */
Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
{
  if (glava == NULL)
    return NULL;
  if (!strcmp(glava->broj_indeksa, broj_indeksa))
    return glava;
  return pretrazi_listu(glava->sledeci, broj_indeksa);
}

int main(int argc, char **argv)
{
  /* Argumenti komandne linije su neophodni jer se iz komandne linije
     dobija ime datoteke sa informacijama o studentima. */
  if (argc != 2) {
    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
    exit(EXIT_FAILURE);
  }

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

  /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
     listu */
  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
  char broj_indeksa[MAX_INDEKS];
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;

  /* Ucitavanje vrednosti u listu */
  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
    dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);

  /* Datoteka vise nije potrebna i zatvara se. */
  fclose(in);

  /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
  while (scanf("%s", broj_indeksa) != EOF) {
    trazeni = pretrazi_listu(glava, broj_indeksa);
    if (trazeni == NULL)
      printf("ne\n");
    else
      printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
  }
```

```
126
  /* Oslobadja se memorija zauzeta za cvorove liste. */
128 oslobodi_listu(&glava);

130 return 0;
}
```

Rešenje 4.9

```
1 #include<stdio.h>
#include<stdlib.h>
3 #include "601/lista.h"

5 /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
     na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7    pokazivaca postojecih cvorova listi. */
Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9 {
  /* Pokazivac na glavu rezultujuce liste. */
11 Cvor *rezultujuca = NULL;
  /* Tekuci je pokazivac na pokazivac kome sledecem treba promeniti
13    vrednosti. Inicijalizuje se na adresu pokazivaca rezultujuca jer
      prvo treba odrediti glavu rezultujuce liste. */
15 Cvor **tekuci = &rezultujuca;

17 /* Ako su obe liste prazne, rezultat je isto prazna lista. */
  if (*adresa_glave_1 == NULL && *adresa_glave_2 == NULL)
19   return NULL;

21 /* Ako je prva lista prazna, rezultat je druga lista. */
  if (*adresa_glave_1 == NULL)
23   return *adresa_glave_2;

25 /* Ako je druga lista prazna, rezultat je prva lista. */
  if (*adresa_glave_2 == NULL)
27   return *adresa_glave_1;

29 /* Sve dok u obe liste ima cvorova, azurira se vrednost pokazivaca
     na koji tekuci pokazuje. U prvoj iteraciji tekuci pokazuje na
31    pokazivac rezultujuca i ovako se pokazivac rezultujuca usmerava
      da pokazuje na pocetak nove liste, tj. na cvor sa vrednoscu
33    manjeg od brojeva sadrzanih u cvorovima na koje vode pokazivaci
      na adresama adresa_glave_1 i adresa_glave_2. U svim ostalim
35    iteracijama to isto se dogadja samo pokazivacu na koji tekuci u
      tom trenutku pokazuje. */
37 while (*adresa_glave_1 != NULL && *adresa_glave_2 != NULL) {
    if ((*adresa_glave_1)->vrednost < (*adresa_glave_2)->vrednost) {
39    /* Pokazivac na koji tekuci pokazuje dobija vrednosti
         pokazivaca koji se nalazi na adresa_glave_1. Time sledbenik
41       poslednjeg uvezanog cvora postaje cvor koji je aktuelna
         glava prve liste. */
```

```
43        *tekuci = *adresa_glave_1;
          /* Pomera se glava prve liste na sledeci cvor prve liste.
45
             Ova promena bice vidljiva i van funkcije jer se direktno
47           menja promenljiva koja se nalazi na adresi adresa_glave_1.
       */
          *adresa_glave_1 = (*adresa_glave_1)->sledeci;
49      } else {
          /* Sledbenik poslednjeg uvezanog cvora bice cvor koji je
51           aktuelna glava druge liste. */
          *tekuci = *adresa_glave_2;
53        /* Pomera se glava druge liste na sledeci cvor druge liste */
          *adresa_glave_2 = (*adresa_glave_2)->sledeci;
55      }
        /* Tekuci se pomera na pokazivac sledeci od poslednjeg uvezanog,
57         jer je upravo to pokazivac koji treba da bude azuriran u
           sledecoj iteraciji petlje. */
59      tekuci = &((*tekuci)->sledeci);
    }
61
    /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
63     rezultujucu listu treba nadovezati ostatak druge liste. Tako
       sledbenik poslednjeg uvezanog cvora treba da bude ostatak druge
65     liste. */
    if (*adresa_glave_1 == NULL)
67    *tekuci = *adresa_glave_2;
    else {
69    if (*adresa_glave_2 == NULL)
        *tekuci = *adresa_glave_1;
71  }

73  return rezultujuca;
}
75
/* Druga verzija prethodne funkcije koja ne pristupa pokazivacima
77   preko adresa vec direktno. Ne salju joj se adrese, vec vrednosti
     pokazivaca na glave listi. */
79 Cvor *objedini_v2(Cvor * lista1, Cvor * lista2)
{
81  Cvor *rezultujuca = NULL;
    Cvor *tekuci = NULL;
83
    /* Ako su obe liste prazne i rezultat je prazna lista. */
85  if (lista1 == NULL && lista2 == NULL)
      return NULL;
87
    /* Ako je prva lista prazna, rezultat je druga lista. */
89  if (lista1 == NULL)
      return lista2;
91
    /* Ako je druga lista prazna, rezultat je prva lista. */
93  if (lista2 == NULL)
```

```
    return lista1;

  /* Rezultujuca pokazuje na pocetak nove liste, tj. na cvor sa
     vrednoscu manjeg od brojeva sadrzanih u cvorovima na koje
     pokazuju lista1 i lista2. */
  if (lista1->vrednost < lista2->vrednost) {
    rezultujuca = lista1;
    lista1 = lista1->sledeci;
  } else {
    rezultujuca = lista2;
    lista2 = lista2->sledeci;
  }
  tekuci = rezultujuca;

  /* Kako rezultujuca pokazuje na pocetak nove liste i ne sme joj se
     menjati vrednost, koristi se pokazivac tekuci koji trenutno
     sadrzi adresu promenljive rezultujuca. U svakoj iteraciji
     petlje, dobijace adekvatnog sledbenika tako da i nova lista bude
     uredjena neopadajuce i pomerace se na adresu sledeceg. */
  while (lista1 != NULL && lista2 != NULL) {
    if (lista1->vrednost < lista2->vrednost) {
      tekuci->sledeci = lista1;
      lista1 = lista1->sledeci;
    } else {
      tekuci->sledeci = lista2;
      lista2 = lista2->sledeci;
    }
    tekuci = tekuci->sledeci;
  }

  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
     rezultujucu listu treba nadovezati ostatak druge liste. */
  if (lista1 == NULL)
    tekuci->sledeci = lista2;
  else
    tekuci->sledeci = lista1;

  return rezultujuca;
}

/* Glavni program */
int main(int argc, char **argv)
{
  /* Argumenti komandne linije su neophodni. */
  if (argc != 3) {
    fprintf(stderr,
            "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
    exit(EXIT_FAILURE);
  }

  /* Otvaramo datoteke sa elementima obe liste. */
  FILE *in1 = NULL;
```

```
  in1 = fopen(argv[1], "r");
  if (in1 == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  FILE *in2 = NULL;
  in2 = fopen(argv[2], "r");
  if (in2 == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
    exit(EXIT_FAILURE);
  }

  int broj;
  Cvor *lista1 = NULL;
  Cvor *lista2 = NULL;
  Cvor *rezultat = NULL;

  /* Ucitavanje listi */
  while (fscanf(in1, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&lista1, broj);
  while (fscanf(in2, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&lista2, broj);

  /* Pokazivac rezultat ce pokazivati na glavu liste koja se dobila
     objedinjavanjem listi */
  rezultat = objedini(&lista1, &lista2);

 /*****************************************************************
  Poziv druge verzije prethodne funkcije

  rezultat = objedini_v2(lista1, lista2);
  *****************************************************************/

  /* Ispis rezultujuce liste. */
  ispisi_listu(rezultat);

  /* Kako je lista rezultat dobijena prevezivanjem cvorova polaznih
     listi, njenim oslobadjanjem bice oslobodjena sva zauzeta
     memorija. */
  oslobodi_listu(&rezultat);

  fclose(in1);
  fclose(in2);
  return 0;
}
```

Rešenje 4.14

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

/* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
typedef struct cvor {
  int broj;
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
Cvor *napravi_cvor(int broj)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije. */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizuju se polja novog cvora. */
  novi->broj = broj;
  novi->levo = NULL;
  novi->desno = NULL;

  /* Vraca se adresa 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);
  }
}

/* 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) {

    /* Kreira se novi cvor */
```

```
52      Cvor *novi = napravi_cvor(broj);
        proveri_alokaciju(novi);
54
        /* I proglasava se korenom stabla */
56      *adresa_korena = novi;
        return;
58    }

60    /* U suprotnom trazi se odgovarajuca pozicija za zadati broj: */

62    /* Ako je zadata vrednost manja od vrednosti korena */
      if (broj < (*adresa_korena)->broj)
64
        /* Broj se dodaje u levo podstablo */
66      dodaj_u_stablo(&(*adresa_korena)->levo, broj);

68    else
        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
70         dodaje u desno podstablo */
        dodaj_u_stablo(&(*adresa_korena)->desno, broj);
72  }

74  /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
    Cvor *pretrazi_stablo(Cvor * koren, int broj)
76  {

78    /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
      if (koren == NULL)
80      return NULL;

82    /* Ako je trazena vrednost sadrzana u korenu */
      if (koren->broj == broj) {
84
        /* Prekidamo pretragu */
86      return koren;
      }
88
      /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
90    if (broj < koren->broj)

92      /* Pretraga se nastavlja u levom podstablu */
        return pretrazi_stablo(koren->levo, broj);
94
      else
96      /* U suprotnom, pretraga se nastavlja u desnom podstablu */
        return pretrazi_stablo(koren->desno, broj);
98  }

100 /* e) Funkcija pronalazi cvor koji sadrzi najmanju vrednost u stablu
        */
    Cvor *pronadji_najmanji(Cvor * koren)
102 {
```

```
104   /* Ako je stablo prazno, prekida se pretraga */
      if (koren == NULL)
106     return NULL;

108   /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
         levo od njega */
110
      /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
112      najmanju vrednost */
      if (koren->levo == NULL)
114     return koren;

116   /* Inace, pretragu treba nastaviti u levom podstablu */
      return pronadji_najmanji(koren->levo);
118 }

120 /* f) Funkcija pronalazi cvor koji sadrzi najvecu vrednost u stablu
       */
    Cvor *pronadji_najveci(Cvor * koren)
122 {
      /* Ako je stablo prazno, prekida se pretraga */
124   if (koren == NULL)
        return NULL;
126
      /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
128      desno od njega */

130   /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
         najvecu vrednost */
132   if (koren->desno == NULL)
        return koren;
134
      /* Inace, pretragu treba nastaviti u desnom podstablu */
136   return pronadji_najveci(koren->desno);
    }
138
    /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
140 void obrisi_element(Cvor ** adresa_korena, int broj)
    {
142   Cvor *pomocni_cvor = NULL;

144   /* Ako je stablo prazno, brisanje nije primenljivo */
      if (*adresa_korena == NULL)
146     return;

148   /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
         stabla, ona se eventualno nalazi u levom podstablu, pa treba
150      rekurzivno primeniti postupak na levo podstablo. Koren ovako
         modifikovanog stabla je nepromenjen. */
152   if (broj < (*adresa_korena)->broj) {
        obrisi_element(&(*adresa_korena)->levo, broj);
```

```
154       return;
    }
156
    /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
158       stabla, ona se eventualno nalazi u desnom podstablu pa treba
       rekurzivno primeniti postupak na desno podstablo. Koren ovako
160       modifikovanog stabla je nepromenjen. */
    if ((*adresa_korena)->broj < broj) {
162     obrisi_element(&(*adresa_korena)->desno, broj);
      return;
164   }

166   /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
       jednaka broju koji se brise (tj. slucaj kada treba obrisati
168       koren) */

170   /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
       prazno stablo (vraca se NULL) */
172   if ((*adresa_korena)->levo == NULL
        && (*adresa_korena)->desno == NULL) {
174     free(*adresa_korena);
      *adresa_korena = NULL;
176     return;
    }
178
    /* Ako koren ima samo levog sina, tada se brisanje vrsi tako sto se
180       brise koren, a novi koren postaje levi sin */
    if ((*adresa_korena)->levo != NULL
182       && (*adresa_korena)->desno == NULL) {
      pomocni_cvor = (*adresa_korena)->levo;
184     free(*adresa_korena);
      *adresa_korena = pomocni_cvor;
186     return;
    }
188
    /* Ako koren ima samo desnog sina, tada se brisanje vrsi tako sto
190       se brise koren, a novi koren postaje desni sin */
    if ((*adresa_korena)->desno != NULL
192       && (*adresa_korena)->levo == NULL) {
      pomocni_cvor = (*adresa_korena)->desno;
194     free(*adresa_korena);
      *adresa_korena = pomocni_cvor;
196     return;
    }
198
    /* Slucaj kada koren ima oba sina - najpre se potrazi sledbenik
200       korena (u smislu poretka) u stablu. To je upravo po vrednosti
       najmanji cvor u desnom podstablu. On se moze pronaci npr.
202       funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
       vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
204       broj koji se brise).  Zatim se prosto rekurzivno pozove funkcija
       za brisanje na desno podstablo. S obzirom da u njemu treba
```

```
206       obrisati najmanji element, a on zasigurno ima najvise jednog
          potomka, jasno je da ce njegovo brisanje biti obavljeno na jedan
208       od jednostavnijih nacina koji su gore opisani. */
    pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
210   (*adresa_korena)->broj = pomocni_cvor->broj;
    pomocni_cvor->broj = broj;
212   obrisi_element(&(*adresa_korena)->desno, broj);
}
214
/* h) Funkcija ispisuje stablo u infiksnoj notaciji (Levo postablo -
216    Koren - Desno podstablo ) */
void ispisi_stablo_infiksno(Cvor * koren)
218 {
    /* Ako stablo nije prazno */
220   if (koren != NULL) {

222     /* Prvo se ispisuju svi cvorovi levo od korena */
      ispisi_stablo_infiksno(koren->levo);
224
      /* Zatim se ispisuje vrednost u korenu */
226     printf("%d ", koren->broj);

228     /* Na kraju se ispisuju cvorovi desno od korena */
      ispisi_stablo_infiksno(koren->desno);
230   }
}
232
/* i) Funkcija ispisuje stablo u prefiksnoj notaciji ( Koren - Levo
234    podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren)
236 {
    /* Ako stablo nije prazno */
238   if (koren != NULL) {

240     /* Prvo se ispisuje vrednost u korenu */
      printf("%d ", koren->broj);
242
      /* Zatim se ispisuju svi cvorovi levo od korena */
244     ispisi_stablo_prefiksno(koren->levo);

246     /* Na kraju se ispisuju svi cvorovi desno od korena */
      ispisi_stablo_prefiksno(koren->desno);
248   }
}
250
/* j) Funkcija ispisuje stablo postfiksnoj notaciji ( Levo podstablo
252    - Desno postablo - Koren) */
void ispisi_stablo_postfiksno(Cvor * koren)
254 {

256   /* Ako stablo nije prazno */
    if (koren != NULL) {
```

```
258
    /* Prvo se ispisuju svi cvorovi levo od korena */
260 ispisi_stablo_postfiksno(koren->levo);

262 /* Zatim se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_postfiksno(koren->desno);
264
    /* Na kraju se ispisuje vrednost u korenu */
266 printf("%d ", koren->broj);
  }
268 }

270 /* k) Funkcija koja oslobadja memoriju zauzetu stablom.  */
void oslobodi_stablo(Cvor ** adresa_korena)
272 {
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
274 if (*adresa_korena == NULL)
    return;
276
  /* Inace ... */
278 /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);
280
  /* Oslobadja se memorija zauzetu desnim podstablom */
282 oslobodi_stablo(&(*adresa_korena)->desno);

284 /* Oslobadja se memorija zauzeta korenom */
  free(*adresa_korena);
286
  /* Proglasava se stablo praznim */
288 *adresa_korena = NULL;
}
290
int main()
292 {
  Cvor *koren;
294 int n;
  Cvor *trazeni_cvor;
296
  /* Proglasava se stablo praznim */
298 koren = NULL;

300 /* Dodaju se vrednosti u stablo */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
302 while (scanf("%d", &n) != EOF) {
    dodaj_u_stablo(&koren, n);
304 }

306 /* Generisu se trazeni ispisi: */
  printf("\nInfiksni ispis: ");
308 ispisi_stablo_infiksno(koren);
  printf("\nPrefiksni ispis: ");
```

```
310   ispisi_stablo_prefiksno(koren);
      printf("\nPostfiksni ispis: ");
312   ispisi_stablo_postfiksno(koren);

314   /* Demonstrira se rad funkcije za pretragu */
      printf("\nTrazi se broj: ");
316   scanf("%d", &n);
      trazeni_cvor = pretrazi_stablo(koren, n);
318   if (trazeni_cvor == NULL)
        printf("Broj se ne nalazi u stablu!\n");
320
      else
322     printf("Broj se nalazi u stablu!\n");

324   /* Demonstrira se rad funkcije za brisanje */
      printf("Brise se broj: ");
326   scanf("%d", &n);
      obrisi_element(&koren, n);
328   printf("Rezultujuce stablo: ");
      ispisi_stablo_infiksno(koren);
330   printf("\n");

332   /* Oslobadja se memorija zauzeta stablom */
      oslobodi_stablo(&koren);
334   return 0;
}
```

Rešenje 4.15

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

6 #define MAX 50

8 /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
     pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
    char *rec;
12  int brojac;
    struct cvor *levo;
14  struct cvor *desno;
} Cvor;
16
/* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec)
{
20  /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
22  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
```

```
  if (novi_cvor == NULL)
    return NULL;

  /* Alocira se memorija 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;
  }

  /* Inicijalizuju se polja u novom cvoru */
  strcpy(novi_cvor->rec, rec);
  novi_cvor->brojac = 1;
  novi_cvor->levo = NULL;
  novi_cvor->desno = NULL;

  /* Vraca se adresa 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) {
    /* Kreira se cvor koji sadrzi zadatu rec */
    Cvor *novi = napravi_cvor(rec);
    proveri_alokaciju(novi);

    /* I proglasava se korenom stabla */
    *adresa_korena = novi;
    return;
  }

  /* U suprotnom se trazi odgovarajuca pozicija za novu rec */

  /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
     levo podstablo */
```

```
  if (strcmp(rec, (*adresa_korena)->rec) < 0)
76  dodaj_u_stablo(&(*adresa_korena)->levo, rec);

78  else
    /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
80     desno podstablo */
  if (strcmp(rec, (*adresa_korena)->rec) > 0)
82  dodaj_u_stablo(&(*adresa_korena)->desno, rec);

84  else
    /* Ako je rec jednaka reci u korenu, uvecava se njen broj
86     pojavljivanja */
    (*adresa_korena)->brojac++;
88 }

90 /* Funkcija koja oslobadja memoriju zauzetu stablom */
   void oslobodi_stablo(Cvor ** adresa_korena)
92 {
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
94  if (*adresa_korena == NULL)
    return;
96
  /* Inace ... */
98  /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);
100
  /* Oslobadja se memorija zauzeta desnim podstablom */
102 oslobodi_stablo(&(*adresa_korena)->desno);

104 /* Oslobadja se memorija zauzeta korenom */
  free((*adresa_korena)->rec);
106 free(*adresa_korena);

108 /* Stablo se proglasava praznim */
  *adresa_korena = NULL;
110 }

112 /* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
     sa najvecim brojem pojavljivanja) */
114 Cvor *nadji_najfrekventniju_rec(Cvor * koren)
   {
116 Cvor *max, *max_levo, *max_desno;

118 /* Ako je stablo prazno, prekida se sa pretragom */
  if (koren == NULL)
120   return NULL;

122 /* Pronalazi se najfrekventnija rec u levom podstablu */
  max_levo = nadji_najfrekventniju_rec(koren->levo);
124
  /* Pronalazi se najfrekventnija rec u desnom podstablu */
126 max_desno = nadji_najfrekventniju_rec(koren->desno);
```

```
128   /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
         podstabla, korena i desnog podstabla */
130   max = koren;
      if (max_levo != NULL && max_levo->brojac > max->brojac)
132     max = max_levo;
      if (max_desno != NULL && max_desno->brojac > max->brojac)
134     max = max_desno;

136   /* Vraca se adresa cvora sa najvecim brojem pojavljivanja */
      return max;
138 }

140 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
       pracene brojem pojavljivanja */
142 void prikazi_stablo(Cvor * koren)
    {
144   /* Ako je stablo prazno, zavrsava se sa ispisom */
      if (koren == NULL)
146     return;

148   /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
         podstabla */
150   prikazi_stablo(koren->levo);

152   /* Zatim rec iz korena */
      printf("%s: %d\n", koren->rec, koren->brojac);
154
      /* I nastavlja se sa ispisom reci iz desnog podstabla */
156   prikazi_stablo(koren->desno);
    }
158
    /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
160    niz rec. Maksimalna duzina reci je odredjena argumentom max.
       Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
162    suprotnom. Rec je niz malih ili velikih slova. */
    int procitaj_rec(FILE * f, char rec[], int max)
164 {
      /* Karakter koji se cita */
166   int c;

168   /* Indeks pozicije na koju se smesta procitani karakter */
      int i = 0;
170
      /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
172       nije stiglo do kraja datoteke... */
      while (i < max - 1 && (c = fgetc(f)) != EOF) {
174     /* Proverava se da li je procitani karakter slovo */
        if (isalpha(c))
176       /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
             mala slova jer program treba da bude neosetljiv na razliku
178          izmedju malih i velikih slova */
```

```
      rec[i++] = tolower(c);

    else
      /* Ako nije, proverava se da li je procitano barem jedno slovo
         nove reci */
      /* Ako jeste, prekida se sa citanjem */
    if (i > 0)
      break;

    /* U suprotnom se ide na sledecu iteraciju */
  }

  /* Dodaje se na rec terminirajuca nula */
  rec[i] = '\0';

  /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
  return i > 0 ? 0 : EOF;
}

int main(int argc, char **argv)
{
  Cvor *koren = NULL, *max;
  FILE *f;
  char rec[MAX];

  /* Provera da li je navedeno ime datoteke prilikom pokretanja
     programa */
  if (argc < 2) {
    fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
    exit(EXIT_FAILURE);
  }

  /* Priprema datoteke za citanje */
  if ((f = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "fopen() greska pri otvaranju %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
     pretrage. */
  while (procitaj_rec(f, rec, MAX) != EOF)
    dodaj_u_stablo(&koren, rec);

  /* Posto je citanjem reci zavrseno, zatvara se datoteka */
  fclose(f);

  /* Prikazuju se sve reci iz teksta i brojevi njihovih
     pojavljivanja. */
  prikazi_stablo(koren);

  /* Pronalazi se najfrekventnija rec */
  max = nadji_najfrekventniju_rec(koren);
```

```
232   /* Ako takve reci nema... */
      if (max == NULL)
234
        /* Ispisuje se odgovarajuce obavestenje */
236     printf("U tekstu nema reci!\n");

238   else
        /* Inace, ispisuje se broj pojavljivanja reci */
240     printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
              max->rec, max->brojac);
242
      /* Oslobadja se dinamicki alociran prostor za stablo */
244   oslobodi_stablo(&koren);

246   return 0;
}
```

Rešenje 4.16

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

6 #define MAX_IME_DATOTEKE 50
  #define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime, broj
      telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
     char ime_i_prezime[MAX_IME_I_PREZIME];
14   char telefon[MAX_CIFARA];
     struct cvor *levo;
16   struct cvor *desno;
   } Cvor;
18
   /* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
   {
22   /* Alocira se memorija za novi cvor i proverava se uspesnost
        alokacije. */
24   Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
     if (novi_cvor == NULL)
26     return NULL;

28   /* Inicijalizuju se polja novog cvora */
     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30   strcpy(novi_cvor->telefon, telefon);
     novi_cvor->levo = NULL;
```

```
32    novi_cvor->desno = NULL;

34    /* Vraca se adresa novog cvora */
      return novi_cvor;
36  }

38  /* Funkcija koja proverava uspesnost kreiranja novog cvora 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 programa
44       */
        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. */
    void
52  dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
                   char *telefon)
54  {
      /* Ako je stablo prazno */
56    if (*adresa_korena == NULL) {
        /* Kreira se novi cvor */
58      Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
        proveri_alokaciju(novi);
60
        /* I proglasava se korenom stabla */
62      *adresa_korena = novi;
        return;
64    }

66    /* U suprotnom trazi se odgovarajuca pozicija za novi unos. Kako
         pretragu treba vrsiti po imenu i prezimenu, stablo treba da bude
68       pretrazivacko po ovom polju */

70    /* Ako je zadato ime i prezime leksikografski manje od imena i
         prezimena sadrzanog u korenu, podaci se dodaju u levo podstablo
        */
72    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
          < 0)
74      dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime, telefon);

76    else
        /* Ako je zadato ime i prezime leksikografski vece od imena i
78         prezimena sadrzanog u korenu, podaci se dodaju u desno
           podstablo */
80    if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
        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 ... */
92   /* Oslobadja se memorija zauzeta levim podstablom */
     oslobodi_stablo(&(*adresa_korena)->levo);
94
     /* Oslobadja se memorija zauzeta desnim podstablom */
96   oslobodi_stablo(&(*adresa_korena)->desno);

98   /* Oslobadja se memorija zauzeta korenom */
     free(*adresa_korena);
100
     /* Stablo se proglasava praznim */
102  *adresa_korena = NULL;
   }
104
   /* Funkcija koja ispisuje imenik u leksikografskom poretku */
106 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
       koristiti za proveru da li je stablo lepo kreirano ili ne */
108 void prikazi_stablo(Cvor * koren)
   {
110  /* Ako je stablo prazno, zavrsava se sa ispisom */
     if (koren == NULL)
112    return;

114  /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
        podstabla */
116  prikazi_stablo(koren->levo);

118  /* Zatim se ispisuju podaci iz korena */
     printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
120
     /* I nastavlja se sa ispisom podataka iz desnog podstabla */
122  prikazi_stablo(koren->desno);
   }
124
   /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje ime
126    i prezime i broj telefona u odgovarajuce nizove. Maksimalna duzina
       imena i prezimena odredjena je konstantom MAX_IME_PREZIME, a
128    maksimalna duzina broja telefona konstantom MAX_CIFARA. Funkcija
       vraca EOF ako nema vise kontakata ili 0 u suprotnom. */
130 int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
   {
132  /* Karakter koji se cita */
     int c;
134
```

```
  /* Indeks pozicije na koju se smesta procitani karakter */
136 int i = 0;

138 /* Linije datoteke koje se obradjuju su formata Ime Prezime
       BrojTelefona */
140
    /* Preskacu se eventualne praznine sa pocetka linije datoteke */
142 while ((c = fgetc(f)) != EOF && isspace(c));

144 /* Prvo procitano slovo upisuje se u ime i prezime */
    if (!feof(f))
146   ime_i_prezime[i++] = c;

148 /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
       cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
150    Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
       procitanog karaktera i da se slucajno ne dodje do kraja datoteke
152  */
    while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
154   if (!isdigit(c))
        ime_i_prezime[i++] = c;
156
      else if (i > 0)
158     break;
    }
160
    /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
162     blanko karaktera */
    ime_i_prezime[--i] = '\0';
164
    /* I pocinje se sa citanjem broja telefona */
166 i = 0;

168 /* Upisuje se cifra koja je vec procitana */
    telefon[i++] = c;
170
    /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
172     karaktera cije prisustvo nije dozvoljeno u broju telefona */
    while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
174   if (c == '/' || c == '-' || isdigit(c))
        telefon[i++] = c;
176   else
        break;
178
    /* Upisuje se terminirajuca nula */
180 telefon[i] = '\0';

182 /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
    return !feof(f) ? 0 : EOF;
184 }

186 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
```

```
 */
Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
{
  /* Ako je imenik prazan, zavrsava se sa pretragom */
  if (koren == NULL)
    return NULL;

  /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
     zavrsava sa pretragom */
  if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
    return koren;

  /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
     korenu pretraga se nastavlja levo */
  if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
    return pretrazi_imenik(koren->levo, ime_i_prezime);

  else
    /* u suprotnom, pretraga se nastavlja desno */
    return pretrazi_imenik(koren->desno, ime_i_prezime);
}

int main(int argc, char **argv)
{
  char ime_datoteke[MAX_IME_DATOTEKE];
  Cvor *koren = NULL;
  Cvor *trazeni;
  FILE *f;
  char ime_i_prezime[MAX_IME_I_PREZIME];
  char telefon[MAX_CIFARA];
  char c;
  int i;

  /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
  printf("Unesite ime datoteke: ");
  scanf("%s", ime_datoteke);
  if ((f = fopen(ime_datoteke, "r")) == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke
%s!\n", ime_datoteke);
    exit(EXIT_FAILURE);
  }

  /* Podaci se citaju iz datoteke i smestanju u binarno stablo
     pretrage. */
  while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
    dodaj_u_stablo(&koren, ime_i_prezime, telefon);

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

  /* Omogucava se pretraga imenika */
  while (1) {
```

```
    /* Ucitavaja se ime i prezime */
240     printf("Unesite ime i prezime: ");
    i = 0;
242     while ((c = getchar()) != '\n')
      ime_i_prezime[i++] = c;
244     ime_i_prezime[i] = '\0';

246     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
           funkcionalnost */
248     if (strcmp(ime_i_prezime, "KRAJ") == 0)
      break;
250
    /* Inace se ispisuje rezultat pretrage */
252     trazeni = pretrazi_imenik(koren, ime_i_prezime);
    if (trazeni == NULL)
254       printf("Broj nije u imeniku!\n");
    else
256       printf("Broj je: %s \n", trazeni->telefon);
  }
258
  /* Oslobadja se memorija zauzeta imenikom */
260   oslobodi_stablo(&koren);

262   return 0;
}
```

Rešenje 4.17

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

5 #define MAX 51

7 /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
     studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9    jezika i redom pokazivace na levo i desno podstablo */
  typedef struct cvor_stabla {
11   char ime[MAX];
    char prezime[MAX];
13   double uspeh;
    double matematika;
15   double jezik;
    struct cvor_stabla *levo;
17   struct cvor_stabla *desno;
  } Cvor;
19
  /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                     double matematika, double jezik)
23 {
```

```
  /* Alocira se memorija za novi cvor i proverava se uspesnost
25    alokacije. */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27  if (novi == NULL)
    return NULL;
29
  /* Inicijalizuju se polja strukture */
31  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
33  novi->uspeh = uspeh;
  novi->matematika = matematika;
35  novi->jezik = jezik;
  novi->levo = NULL;
37  novi->desno = NULL;

39  /* Vraca se adresa kreiranog cvora */
  return novi;
41 }

43 /* Funkcija kojom se proverava uspesnost alociranja memorije */
 void proveri_alokaciju(Cvor * novi_cvor)
45 {
  /* Ukoliko je cvor neuspesno kreiran */
47  if (novi_cvor == NULL) {
    /* Ispisuje se odgovarajuca poruka i prekida se izvrsavanje
49     programa */
    fprintf(stderr, "Malloc greska za novi cvor!\n");
51  exit(EXIT_FAILURE);
  }
53 }

55 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
 void oslobodi_stablo(Cvor ** koren)
57 {
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
59  if (*koren == NULL)
    return;
61
  /* Inace ... */
63  /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*koren)->levo);
65
  /* Oslobadja se memorija zauzeta desnim podstablom */
67  oslobodi_stablo(&(*koren)->desno);

69  /* Oslobadja se memorija zauzeta korenom */
  free(*koren);
71
  /* Stablo se proglasava praznim */
73  *koren = NULL;
 }
75
```

```
/* 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) {
    /* Kreira se novi cvor */
    Cvor *novi = napravi_cvor(ime, prezime, uspeh, matematika, jezik)
    ;
    proveri_alokaciju(novi);

    /* I proglasava se korenom stabla */
    *koren = novi;

    return;
  }

  /* Inace, dodaje se cvor u stablo tako da bude sortirano po ukupnom
     broju poena */
  if (uspeh + matematika + jezik >
      (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
    dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                   matematika, jezik);
  else
    dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
                   matematika, jezik);
}

/* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
   polozili jednaka 0 ispisuju se informacije o ucenicima koji nisu
   polozili prijemni, a ako je vrednost argumenta razlicita od nule,
   ispisuju se informacije o ucenicima koji su polozili prijemni */
void stampaj(Cvor * koren, int polozili)
{
  /* Stablo je prazno - prekida se sa ispisom */
  if (koren == NULL)
    return;

  /* Stampaju se informacije iz levog podstabla */
  stampaj(koren->levo, polozili);

  /* Stampaju se informacije iz korenog cvora */
  if (polozili && koren->matematika + koren->jezik >= 10)
    printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
           koren->prezime, koren->uspeh, koren->matematika,
           koren->jezik,
           koren->uspeh + koren->matematika + koren->jezik);
  else if (!polozili && koren->matematika + koren->jezik < 10)
    printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
           koren->prezime, koren->uspeh, koren->matematika,
           koren->jezik,
           koren->uspeh + koren->matematika + koren->jezik);
```

```
127
  /* Stampaju se informacije iz desnog podstabla */
129  stampaj(koren->desno, polozili);
}
131
/* Funkcija koja odredjuje koliko studenata nije polozilo prijemni
133    ispit */
int nisu_polozili(Cvor * koren)
135 {
  /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
137  if (koren == NULL)
    return 0;
139
  /* Pretraga se vrsi i u levom i u desnom podstablu - ako uslov za
141     polaganje nije ispunjen za koreni cvor, broj studenata se
     uvecava za 1 */
143  if (koren->matematika + koren->jezik < 10)
    return 1 + nisu_polozili(koren->levo) +
145         nisu_polozili(koren->desno);

147  return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
}
149
int main(int argc, char **argv)
151 {
  FILE *in;
153  Cvor *koren;
  char ime[MAX], prezime[MAX];
155  double uspeh, matematika, jezik;

157  /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
  in = fopen("prijemni.txt", "r");
159  if (in == NULL) {
    fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
161    exit(EXIT_FAILURE);
  }
163
  /* Citanje podataka i dodavanje u stablo */
165  koren = NULL;
  while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
167                &matematika, &jezik) != EOF) {
    dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik);
169  }

171  /* Zatvaranje datoteke */
  fclose(in);
173
  /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
175  stampaj(koren, 1);

177  /* Linij se iscrtava samo ako postoje ucenici koji nisu polozili
     prijemni */
```

```
179   if (nisu_polozili(koren) != 0)
        printf("-------------------------------------\n");
181
      /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
183   stampaj(koren, 0);

185   /* Oslobadja se memorija zauzeta stablom */
      oslobodi_stablo(&koren);
187
      return 0;
189 }
```

Rešenje 4.18

```
    #include<stdio.h>
2   #include<stdlib.h>
    #include<string.h>
4
    #define MAX_NISKA 51
6
    /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
8      osobe, dan i mesec rodjenja i redom pokazivace na levo i desno
       podstablo */
10  typedef struct cvor_stabla {
      char ime[MAX_NISKA];
12    char prezime[MAX_NISKA];
      int dan;
14    int mesec;
      struct cvor_stabla *levo;
16    struct cvor_stabla *desno;
    } Cvor;
18
    /* Funkcija koja kreira novi cvor */
20  Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
    {
22    /* Alocira se memorija */
      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24    if (novi == NULL)
        return NULL;
26
      /* Inicijalizuju se polja strukture */
28    strcpy(novi->ime, ime);
      strcpy(novi->prezime, prezime);
30    novi->dan = dan;
      novi->mesec = mesec;
32    novi->levo = NULL;
      novi->desno = NULL;
34
      /* Vraca se adresa novog cvora */
36    return novi;
    }
```

```
38
   /* Funkcija koja proverava uspesnost alokacije */
40 void proveri_alokaciju(Cvor * novi_cvor)
   {
42   /* Ako memorija nije uspesno alocirana */
     if (novi_cvor == NULL) {
44     /* Ispisuje se poruka i prekida se sa izvrsavanjem programa */
       fprintf(stderr, "Malloc greska za novi cvor!\n");
46     exit(EXIT_FAILURE);
     }
48 }

50 /* Funkcija koja oslobadja memoriju zauzetu stablom */
   void oslobodi_stablo(Cvor ** koren)
52 {
     /* Stablo je prazno */
54   if (*koren == NULL)
       return;
56
     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
58   if ((*koren)->levo)
       oslobodi_stablo(&(*koren)->levo);
60
     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
62   if ((*koren)->desno)
       oslobodi_stablo(&(*koren)->desno);
64
     /* Oslobadja se memorija zauzeta korenom */
66   free(*koren);

68   /* Proglasava se stablo praznim */
     *koren = NULL;
70 }

72 /* Funkcija koja dodaje novi cvor u stablo - stablo treba da bude
      uredjeno po datumu - prvo po mesecu, a zatim po danu */
74 void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                      int dan, int mesec)
76 {
     /* Ako je stablo prazno */
78   if (*koren == NULL) {

80     /* Kreira se novi cvor */
       Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
82     proveri_alokaciju(novi_cvor);

84     /* I proglasava se korenom */
       *koren = novi_cvor;
86
       return;
88   }
```

```
90    /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
         meseca */
92    if (mesec < (*koren)->mesec)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
94    else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
96    else
        dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
98  }

100 /* Funkcija vrsi pretragu stabla i vraca cvor sa trazenim datumom.
        */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)
102 {
      /* Stablo je prazno, obustavlja se pretraga */
104   if (koren == NULL)
        return NULL;
106
      /* Ako je trazeni datum u korenu */
108   if (koren->dan == dan && koren->mesec == mesec)
        return koren;
110
      /* Ako je mesec trazenog datuma manji od meseca sadrzanog u korenu
112        ili ako su meseci isti ali je dan trazenog datuma manji od
           aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
114        svakako proverava da li leva grana postoji - ako ne postoji
           treba vratiti prvi sledeci, a to je bas vrednost uocenog korena
        */
116   if (mesec < koren->mesec
          || (mesec == koren->mesec && dan < koren->dan)) {
118     if (koren->levo == NULL)
          return koren;
120     else
          return pretrazi(koren->levo, dan, mesec);
122   }

124   /* Inace se nastavlja pretraga u desnom delu */
      return pretrazi(koren->desno, dan, mesec);
126 }

128 /* Funkcija koja pronalazi najmanji datum u stablu */
    Cvor *pronadji_najmanji_datum(Cvor * koren)
130 {
      /* Stablo je prazno, obustavlja se pretraga */
132   if (koren == NULL)
        return NULL;
134
      /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
136        sadrzi najmanji datum */
      if (koren->levo == NULL)
138     return koren;
      else
```

```
140     /* Inace, trazimo manji datum u levom podstablu */
        return pronadji_najmanji_datum(koren->levo);
142 }

144 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
        */
    void datum_u_nisku(int dan, int mesec, char datum[])
146 {
      if (dan < 10) {
148     datum[0] = '0';
        datum[1] = dan + '0';
150   } else {
        datum[0] = dan / 10 + '0';
152     datum[1] = dan % 10 + '0';
      }
154   datum[2] = '.';

156   if (mesec < 10) {
        datum[3] = '0';
158     datum[4] = mesec + '0';
      } else {
160     datum[3] = mesec / 10 + '0';
        datum[4] = mesec % 10 + '0';
162   }
      datum[5] = '.';
164   datum[6] = '\0';
    }
166
    int main(int argc, char **argv)
168 {
      FILE *in;
170   Cvor *koren;
      Cvor *slavljenik;
172   char ime[MAX_NISKA], prezime[MAX_NISKA];
      int dan, mesec;
174   char datum[7];

176   /* Provera da li je zadato ime ulazne datoteke */
      if (argc < 2) {
178     /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
           programa */
180     printf("Nedostaje ime ulazne datoteke!\n");
        return 0;
182   }

184   /* Inace, priprema se datoteka za citanje */
      in = fopen(argv[1], "r");
186   if (in == NULL) {
        fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
188             argv[1]);
        exit(EXIT_FAILURE);
190   }
```

```
192   /* I stablo se popunjava podacima */
      koren = NULL;
194   while (fscanf
             (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
196     dodaj_u_stablo(&koren, ime, prezime, dan, mesec);

198   /* Datoteka se zatvara */
      fclose(in);
200
      /* Omogucuje se pretraga podataka */
202   while (1) {

204     /* Ucitava se novi datum */
        printf("Unesite datum: ");
206     if (scanf("%d.%d.", &dan, &mesec) == EOF)
          break;
208
        /* Pretrazuje se stablo */
210     slavljenik = pretrazi(koren, dan, mesec);

212     /* Ispisuju se pronadjeni podaci */

214     /* Ako slavljenik nije pronadjen, to moze znaci da: */
        /* 1. drvo je prazno */
216     if (slavljenik == NULL && koren == NULL) {
          printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
218       continue;
        }
220     /* 2. posle datuma koji je unesen, nema podataka u stablu - u
           ovom slucaju se pretraga vrsi pocevsi od naredne godine i
222        ispisuje se najmanji datum */
        if (slavljenik == NULL) {
224       slavljenik = pronadji_najmanji_datum(koren);
          datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
226       printf("Slavljenik: %s %s %s\n", slavljenik->ime,
                 slavljenik->prezime, datum);
228       continue;
        }
230
        /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
232     /* 1. Pronadjeni su tacni podaci */
        if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
234       printf("Slavljenik: %s %s\n", slavljenik->ime,
                 slavljenik->prezime);
236       continue;
        }
238
        /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
240     datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
        printf("Slavljenik: %s %s %s\n", slavljenik->ime,
242            slavljenik->prezime, datum);
```

```
  }

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  return 0;
}
```

Rešenje 4.19

```
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura kojom se predstavlja cvor binarnog pretrazivackog stabla
 */
typedef struct cvor {
  int broj;
  struct cvor *levo, *desno;
} Cvor;

/* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
Cvor *napravi_cvor(int broj);

/* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
void proveri_alokaciju(Cvor * novi_cvor);

/* Funkcija koja dodaje zadati broj u stablo */
void dodaj_u_stablo(Cvor ** adresa_korena, int broj);

/* Funkcija koja proverava da li se zadati broj nalazi u stablu */
Cvor *pretrazi_stablo(Cvor * koren, int broj);

/* Funkcija koj pronalazi cvor koji sadrzi najmanju vrednost u stablu
 */
Cvor *pronadji_najmanji(Cvor * koren);

/* Funkcija koja pronalazi cvor koji sadrzi najvecu 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 stablo 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
```

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

Cvor *napravi_cvor(int broj)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije. */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;
  /* Inicijalizuju se polja novog cvora. */
  novi->broj = broj;
  novi->levo = NULL;
  novi->desno = NULL;
  /* Vraca se adresa novog cvora. */
  return novi;
}

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);
  }
}

void dodaj_u_stablo(Cvor ** koren, int broj)
{
  /* Ako je stablo prazno */
  if (*koren == NULL) {
    /* Kreira se novi cvor */
    Cvor *novi = napravi_cvor(broj);
    proveri_alokaciju(novi);
    /* I proglasava se korenom stabla */
    *koren = novi;
    return;
  }
  /* U suprotnom se trazi odgovarajuca pozicija za zadati broj: */

  /* Ako je zadata vrednost manja od vrednosti korena */
  if (broj < (*koren)->broj)
    /* Broj se dodaje u levo podstablo */
    dodaj_u_stablo(&(*koren)->levo, broj);
  else
    /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
       dodaje u desno podstablo */
```

```
    dodaj_u_stablo(&(*koren)->desno, broj);
}

Cvor *pretrazi_stablo(Cvor * koren, int broj)
{
  /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
  if (koren == NULL)
    return NULL;
  /* Ako je trazena vrednost sadrzana u korenu */
  if (koren->broj == broj) {
    /* Prekida se pretraga */
    return koren;
  }
  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
  if (broj < koren->broj)
    /* Pretraga se nastavlja u levom podstablu */
    return pretrazi_stablo(koren->levo, broj);
  else
    /* U suprotnom, pretraga se nastavlja u desnom podstablu */
    return pretrazi_stablo(koren->desno, broj);
}

Cvor *pronadji_najmanji(Cvor * koren)
{
  /* Ako je stablo prazno, prekida se pretraga */
  if (koren == NULL)
    return NULL;

  /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
     levo od njega */

  /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
     najmanju vrednost */
  if (koren->levo == NULL)
    return koren;

  /* Inace, pretragu treba nastaviti u levom podstablu */
  return pronadji_najmanji(koren->levo);
}

Cvor *pronadji_najveci(Cvor * koren)
{
  /* Ako je stablo prazno, prekida se pretraga */
  if (koren == NULL)
    return NULL;

  /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
     desno od njega */

  /* 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);
}

void obrisi_element(Cvor ** adresa_korena, int broj)
{
  Cvor *pomocni_cvor = NULL;

  /* Ako je stablo prazno, brisanje nije primenljivo */
  if (*adresa_korena == NULL)
    return;

  /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
     stabla, ona se eventualno nalazi u levom podstablu, pa treba
     rekurzivno primeniti postupak na levo podstablo. Koren ovako
     modifikovanog stabla je nepromenjen. */
  if (broj < (*adresa_korena)->broj) {
    obrisi_element(&(*adresa_korena)->levo, broj);
    return;
  }

  /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
     stabla, ona se eventualno nalazi u desnom podstablu pa treba
     rekurzivno primeniti postupak na desno podstablo. Koren ovako
     modifikovanog stabla je nepromenjen. */
  if ((*adresa_korena)->broj < broj) {
    obrisi_element(&(*adresa_korena)->desno, broj);
    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 (vraca se NULL) */
  if ((*adresa_korena)->levo == NULL
      && (*adresa_korena)->desno == NULL) {
    free(*adresa_korena);
    *adresa_korena = NULL;
    return;
  }

  /* Ako koren ima samo levog sina, tada se brisanje vrsi tako sto se
     brise koren, a novi koren postaje levi sin */
  if ((*adresa_korena)->levo != NULL
      && (*adresa_korena)->desno == NULL) {
    pomocni_cvor = (*adresa_korena)->levo;
    free(*adresa_korena);
    *adresa_korena = pomocni_cvor;
```

```
    return;
  }

  /* Ako koren ima samo desnog sina, tada se brisanje vrsi tako sto
     se brise koren, a novi koren postaje desni sin */
  if ((*adresa_korena)->desno != NULL
      && (*adresa_korena)->levo == NULL) {
    pomocni_cvor = (*adresa_korena)->desno;
    free(*adresa_korena);
    *adresa_korena = pomocni_cvor;
    return;
  }

  /* Slucaj kada koren ima oba sina - najpre se potrazi sledbenik
     korena (u smislu poretka) u stablu. To je upravo po vrednosti
     najmanji cvor u desnom podstablu. On se moze pronaci npr.
     funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
     vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
     broj koji se brise).  Zatim se prosto rekurzivno pozove funkcija
     za brisanje na desno podstablo. S obzirom da u njemu treba
     obrisati najmanji element, a on zasigurno ima najvise jednog
     potomka, jasno je da ce njegovo brisanje biti obavljeno na jedan
     od jednostavnijih nacina koji su gore opisani. */
  pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
  (*adresa_korena)->broj = pomocni_cvor->broj;
  pomocni_cvor->broj = broj;
  obrisi_element(&(*adresa_korena)->desno, broj);
}

void prikazi_stablo(Cvor * koren)
{
  /* Ako je stablo prazno, prekida se ispis */
  if (koren == NULL)
    return;

  /* Prvo se ispisuju svi cvorovi levo od korena */
  prikazi_stablo(koren->levo);

  /* Zatim se ispisuje vrednost u korenu */
  printf("%d ", koren->broj);

  /* Na kraju se ispisuju svi cvorovi desno od korena */
  prikazi_stablo(koren->desno);
}

void oslobodi_stablo(Cvor ** koren)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
  if (*koren == NULL)
    return;
  /* Inace ... */
  /* Oslobadja se memorija zauzeta levim podstablom */
```

```
  if ((*koren)->levo)
    oslobodi_stablo(&(*koren)->levo);
  /* Oslobadja se memorija zauzetu desnim podstablom */
  if ((*koren)->desno)
    oslobodi_stablo(&(*koren)->desno);
  /* Oslobadja se memorija zauzeta korenom */
  free(*koren);
  /* Proglasava se stablo praznim */
  *koren = NULL;
}
```

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

/* Ukljucuje se biblioteka za rad sa stablima - pogledati uvodni
   zadatak ove glave */
#include "stabla.h"


/* Funkcija koja proverava da li su dva stabla koja sadrze cele
   brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
int identitet(Cvor * koren1, Cvor * koren2)
{
  /* Ako su oba stabla prazna, jednaka su */
  if (koren1 == NULL && koren2 == NULL)
    return 1;

  /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
  if (koren1 == NULL || koren2 == NULL)
    return 0;

  /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
     vrednosti, moze se zakljuciti da se razlikuju */
  if (koren1->broj != koren2->broj)
    return 0;

  /* Inace, proverava se da li vazi i jednakost levih i desnih
     podstabala */
  return (identitet(koren1->levo, koren2->levo)
          && identitet(koren1->desno, koren2->desno));
}

int main()
{
  int broj;
  Cvor *koren1, *koren2;

  /* Ucitavaju se elementi prvog stabla */
  koren1 = NULL;
  printf("Prvo stablo: ");
  scanf("%d", &broj);
```

```
42    while (broj != 0) {
        dodaj_u_stablo(&koren1, broj);
44      scanf("%d", &broj);
      }
46
      /* Ucitavaju se elementi drugog stabla */
48    koren2 = NULL;
      printf("Drugo stablo: ");
50    scanf("%d", &broj);
      while (broj != 0) {
52      dodaj_u_stablo(&koren2, broj);
        scanf("%d", &broj);
54    }

56    /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
         njen rezultat. */
58    if (identitet(koren1, koren2))
        printf("Stabla jesu identicna.\n");
60    else
        printf("Stabla nisu identicna.\n");
62
      /* Oslobadja se memorija zauzeta stablima */
64    oslobodi_stablo(&koren1);
      oslobodi_stablo(&koren2);
66
      return 0;
68  }
```

Rešenje 4.20

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

4 /* Uklucuje se biblioteka za rad sa stablima */
  #include "stabla.h"
6
  /* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
8  */
  void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
10 {
     /* Izlaz iz rekurzije */
12   if (koren == NULL) {
       *duplikat = NULL;
14     return;
     }
16
     /* Duplira se koren stabla i postavlja da bude koren novog stabla
        */
18   *duplikat = napravi_cvor(koren->broj);
     proveri_alokaciju(*duplikat);
20
```

```
  /* Rekurzivno se duplira levo podstablo i njegova adresa se cuva u
     pokazivacu na levo podstablo korena duplikata. */
  kopiraj_stablo(koren->levo, &(*duplikat)->levo);

  /* Rekurzivno se duplira desno podstablo i njegova adresa se cuva 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) {
    /* Dodaje se njegov koren u prvo stablo */
    dodaj_u_stablo(adresa_korena1, koren2->broj);

    /* Rekurzivno se racuna unija 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;

  /* Inace... */
  /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.
     iz levog i desnog podstabla prvog stabla brisu se svi oni
     elementi 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 se on
     uklanja 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;

  /* Inace... */
  /* Kreira se razlika levog i desnog podstabla sa drugim stablom,
     tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
     elementi 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 se isti
     uklanja 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;

  /* Ucitavaju se elementi prvog stabla */
  koren1 = NULL;
  printf("Prvo stablo: ");
  while (scanf("%d", &n) != EOF) {
    dodaj_u_stablo(&koren1, n);
  }

  /* Ucitavaju se elementi drugog stabla */
  koren2 = NULL;
  printf("Drugo stablo: ");
  while (scanf("%d", &n) != EOF) {
    dodaj_u_stablo(&koren2, n);
  }

  /* Kreira se unija stabala: prvo se napravi kopija prvog stabla
     kako bi se isto moglo iskoristiti i za preostale operacije */
  kopiraj_stablo(koren1, &pomocni);
  kreiraj_uniju(&pomocni, koren2);
  printf("Unija: ");
  prikazi_stablo(pomocni);
  putchar('\n');

  /* Oslobadja se stablo sa rezultatom operacije */
  oslobodi_stablo(&pomocni);

  /* Kreira se presek stabala: prvo se napravi kopija prvog stabla
     kako bi se isto moglo iskoristiti i za preostale operacije */
  kopiraj_stablo(koren1, &pomocni);
  kreiraj_presek(&pomocni, koren2);
  printf("Presek: ");
```

```
  prikazi_stablo(pomocni);
  putchar('\n');

  /* Oslobadja se stablo sa rezultatom operacije */
  oslobodi_stablo(&pomocni);

  /* Kreira se razlika stabala: prvo se napravi kopija prvog stabla
     kako bi se isto moglo iskoristiti i za preostale operacije; */
  kopiraj_stablo(koren1, &pomocni);
  kreiraj_razliku(&pomocni, koren2);
  printf("Razlika: ");
  prikazi_stablo(pomocni);
  putchar('\n');

  /* Oslobadja se stablo sa rezultatom operacije */
  oslobodi_stablo(&pomocni);

  /* Oslobadjaju se i polazna stabla */
  oslobodi_stablo(&koren2);
  oslobodi_stablo(&koren1);

  return 0;
}
```

Rešenje 4.21

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

#define MAX 50

/* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
   cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
   su smestene u niz. */
int kreiraj_niz(Cvor * koren, int a[])
{
  int r, s;

  /* Stablo je prazno - u niz je smesteno 0 elemenata */
  if (koren == NULL)
    return 0;

  /* Dodaju se u niz elementi iz levog podstabla */
  r = kreiraj_niz(koren->levo, a);

  /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
     u niz i na osnovu nje se moze odrediti indeks novog elementa */

```

```
  /* Smesta se vrednost iz korena */
  a[r] = koren->broj;

  /* Dodaju se elementi iz desnog podstabla */
  s = kreiraj_niz(koren->desno, a + r + 1);

  /* Racuna se indeks na koji treba smestiti naredni element */
  return r + s + 1;
}

/* Funkcija sortira niz tako sto najpre elemente niza smesti u
   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva u
   desno.

   Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
   sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
   sortiranje vrsi u pomocnoj dinamickoj strukturi, a ne razmenom
   elemenata niza. */
void sortiraj(int a[], int n)
{
  int i;
  Cvor *koren;

  /* Kreira se stablo smestanjem elemenata iz niza u stablo */
  koren = NULL;
  for (i = 0; i < n; i++)
    dodaj_u_stablo(&koren, a[i]);

  /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
     a */
  kreiraj_niz(koren, a);

  /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
     */
  oslobodi_stablo(&koren);
}

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

  /* Ucitavaju se dimenzija i elementi niza */
  printf("n: ");
  scanf("%d", &n);
  if (n < 0 || n > MAX) {
    printf("Greska: pogresna dimenzija niza!\n");
    return 0;
  }

  printf("a: ");
  for (i = 0; i < n; i++)
```

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

79    /* Poziva se funkcija za sortiranje */
      sortiraj(a, n);
81
      /* Ispisuje se rezultat */
83    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
85    printf("\n");

87    return 0;
}
```

Rešenje 4.22

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  /* a) Funkcija koja izracunava broj cvorova stabla */
   int broj_cvorova(Cvor * koren)
9  {
     /* Ako je stablo prazno, broj cvorova je nula */
11   if (koren == NULL)
       return 0;
13
     /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15      levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
        zato sto treba racunati i koren */
17   return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
   }
19
   /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
   {
23   /* Ako je stablo prazno, broj listova je nula */
     if (koren == NULL)
25     return 0;

27   /* Proverava se da li je tekuci cvor list */
     if (koren->levo == NULL && koren->desno == NULL)
29     /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
          uvecati broj listova za 1 */
31     return 1;

33   /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
      */
35   return broj_listova(koren->levo) + broj_listova(koren->desno);
   }
```

```
37
/* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39 void pozitivni_listovi(Cvor * koren)
{
41   /* Slucaj kada je stablo prazno */
   if (koren == NULL)
43     return;

45   /* Ako je cvor list i sadrzi pozitivnu vrednost */
   if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47     /* Stampa se */
     printf("%d ", koren->broj);
49
   /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
51   pozitivni_listovi(koren->levo);
   pozitivni_listovi(koren->desno);
53 }

55 /* d) Funkcija koja izracunava zbir cvorova stabla */
int zbir_svih_cvorova(Cvor * koren)
57 {
   /* Ako je stablo prazno, zbir cvorova je 0 */
59   if (koren == NULL)
     return 0;
61
   /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
63      elemenata u podstablima */
   return koren->broj + zbir_svih_cvorova(koren->levo) +
65       zbir_svih_cvorova(koren->desno);
}
67
/* e) Funkcija koja izracunava najveci element stabla */
69 Cvor *najveci_element(Cvor * koren)
{
71   /* Ako je stablo prazno, obustavlja se pretraga */
   if (koren == NULL)
73     return NULL;

75   /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena se
      nalaze u desnom podstablu */
77
   /* Ako desnog podstabla nema */
79   if (koren->desno == NULL)
     /* Najveca vrednost je koren */
81     return koren;

83   /* Inace, najveca vrednost se trazi desno */
   return najveci_element(koren->desno);
85 }

87 /* f) Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
```

```
89 {
     /* Dubina praznog stabla je 0 */
91   if (koren == NULL)
       return 0;
93
     /* Izracunava se dubina levog podstabla */
95   int dubina_levo = dubina_stabla(koren->levo);

97   /* Izracunava se dubina desnog podstabla */
     int dubina_desno = dubina_stabla(koren->desno);
99
     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
101     jer se racuna i koren */
     return dubina_levo >
103      dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
   }
105
   /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
107 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
   {
109   /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
       nivoa */
111
     /* Ako nema vise cvorova, nema spustanja niz stablo */
113  if (koren == NULL)
       return 0;
115
     /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
117     rekurzivnih poziva uvecati broj cvorova za 1 */
     if (i == 0)
119    return 1;

121  /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
       */
     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
123      + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
   }
125
   /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
127 void ispis_nivo(Cvor * koren, int i)
   {
129   /* Ideja je slicna ideji iz prethodne funkcije */

131   /* Nema vise cvorova, nema spustanja kroz stablo */
     if (koren == NULL)
133    return;

135  /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
     if (i == 0) {
137    printf("%d ", koren->broj);
       return;
139  }
```

```
  /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
141     desnom podstablu */
  ispis_nivo(koren->levo, i - 1);
143 ispis_nivo(koren->desno, i - 1);
}
145
/* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
147    stabla */
Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
149 {
  /* Ako je stablo prazno, obustavlja se pretraga */
151 if (koren == NULL)
    return NULL;
153
  /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
155 if (i == 0)
    return koren;
157
  /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
159 Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);

161 /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
  Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
163
  /* Trazi se i vraca maksimum izracunatih vrednosti */
165 if (a == NULL && b == NULL)
    return NULL;
167 if (a == NULL)
    return b;
169 if (b == NULL)
    return a;
171 return a->broj > b->broj ? a : b;
}
173
/* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
175 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
{
177 /* Ako je stablo prazno, zbir je nula */
  if (koren == NULL)
179   return 0;

181 /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
  if (i == 0)
183   return koren->broj;

185 /* Inace, spustanje se nastavlja za jedan nivo nize i traze se sume
      iz levog i desnog podstabla */
187 return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
       + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
189 }

191
```

```
/* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
193    manje ili jednake od date vrednosti x */
int zbir_manjih_od_x(Cvor * koren, int x)
195 {
  /* Ako je stablo prazno, zbir je nula */
197   if (koren == NULL)
    return 0;
199
  /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
201     prirode pretrazivackog stabla treba obici i levo i desno
    podstablo */
203   if (koren->broj <= x)
    return koren->broj + zbir_manjih_od_x(koren->levo, x) +
205         zbir_manjih_od_x(koren->desno, x);

207   /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
     medju njima jedino moze biti onih koje zadovoljavaju uslov */
209   return zbir_manjih_od_x(koren->levo, x);
}
211
int main(int argc, char **argv)
213 {
  /* Analiza argumenata komandne linije */
215   if (argc != 3) {
    fprintf(stderr,
217             "Greska! Program se poziva sa: ./a.out nivo
    broj_za_pretragu\n");
    exit(EXIT_FAILURE);
219   }
   int i = atoi(argv[1]);
221   int x = atoi(argv[2]);

223   /* Kreira se stablo */
   Cvor *koren = NULL;
225   int broj;
   while (scanf("%d", &broj) != EOF)
227     dodaj_u_stablo(&koren, broj);

229   /* ispisuju se rezultati rada funkcija */
   printf("Broj cvorova: %d\n", broj_cvorova(koren));
231   printf("Broj listova: %d\n", broj_listova(koren));
   printf("Pozitivni listovi: ");
233   pozitivni_listovi(koren);
   printf("\n");
235   printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
   if (najveci_element(koren) == NULL)
237     printf("Najveci element: ne postoji\n");
   else
239     printf("Najveci element: %d\n", najveci_element(koren)->broj);

241   printf("Dubina stabla: %d\n", dubina_stabla(koren));
```

```
243   printf("Broj cvorova na %d. nivou: %d\n", i,
          broj_cvorova_na_itom_nivou(koren, i));
245   printf("Elementi na %d. nivou: ", i);
      ispis_nivo(koren, i);
247   printf("\n");
      if (najveci_element_na_itom_nivou(koren, i) == NULL)
249     printf("Nema elemenata na %d. nivou!\n", i);
      else
251     printf("Maksimalni element na %d. nivou: %d\n", i,
            najveci_element_na_itom_nivou(koren, i)->broj);
253
      printf("Zbir elemenata na %d. nivou: %d\n", i,
255         zbir_cvorova_na_itom_nivou(koren, i));
      printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
257         zbir_manjih_od_x(koren, x));

259   /* Oslobadja se memorija zauzeta stablom */
      oslobodi_stablo(&koren);
261
      return 0;
263 }
```

Rešenje 4.23

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

4 /* Ukljucuje se biblioteka za rad sa stablima */
  #include "stabla.h"
6
  /* Funkcija koja izracunava dubinu stabla */
8 int dubina_stabla(Cvor * koren)
  {
10   /* Dubina praznog stabla je 0 */
     if (koren == NULL)
12     return 0;

14   /* Izracunava se dubina levog podstabla */
     int dubina_levo = dubina_stabla(koren->levo);
16
     /* Izracunava se dubina desnog podstabla */
18   int dubina_desno = dubina_stabla(koren->desno);

20   /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
        jer se racuna i koren */
22   return dubina_levo >
         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }

26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */
   void ispisi_nivo(Cvor * koren, int i)
```

```
{
  /* Ideja je slicna ideji iz prethodne funkcije */
  /* Nema vise cvorova, nema spustanja niz stablo */
  if (koren == NULL)
    return;

  /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
  if (i == 0) {
    printf("%d ", koren->broj);
    return;
  }
  /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
     podstablu */
  ispisi_nivo(koren->levo, i - 1);
  ispisi_nivo(koren->desno, i - 1);
}

/* Funkcija koja ispisuje stablo po nivoima */
void ispisi_stablo_po_nivoima(Cvor * koren)
{
  int i;

  /* Prvo se izracunava dubina stabla */
  int dubina;
  dubina = dubina_stabla(koren);

  /* Ispisuje se nivo po nivo stabla */
  for (i = 0; i < dubina; i++) {
    printf("%d. nivo: ", i);
    ispisi_nivo(koren, i);
    printf("\n");
  }
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int broj;

  /* Citaju se vrednosti sa ulaza i dodaju se u stablo */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    dodaj_u_stablo(&koren, broj);
  }

  /* Ispisuje se stablo po nivoima */
  ispisi_stablo_po_nivoima(koren);

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  return 0;
```

```
}
```

Rešenje 4.25

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
    return 0;

  /* Izracunava se dubina levog podstabla */
  int dubina_levo = dubina_stabla(koren->levo);

  /* Izracunava se dubina desnog podstabla */
  int dubina_desno = dubina_stabla(koren->desno);

  /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
     jer se racuna i koren */
  return dubina_levo >
      dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
   stablo */
int avl(Cvor * koren)
{
  int dubina_levo, dubina_desno;

  /* Ako je stablo prazno, zaustavlja se brojanje */
  if (koren == NULL) {
    return 0;
  }

  /* Izracunava se dubina levog podstabla korena */
  dubina_levo = dubina_stabla(koren->levo);

  /* Izracunava se dubina desnog podstabla korena */
  dubina_desno = dubina_stabla(koren->desno);

  /* Ako je uslov za AVL stablo ispunjen */
  if (abs(dubina_desno - dubina_levo) <= 1) {
    /* Racuna se broj AVL cvorova u levom i desnom podstablu i
       uvecava za jedan iz razloga sto koren ispunjava uslov */
    return 1 + avl(koren->levo) + avl(koren->desno);
```

```
  } else {
    /* Inace, racuna se samo broj AVL cvorova u podstablima */
    return avl(koren->levo) + avl(koren->desno);
  }
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int broj;

  /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    dodaj_u_stablo(&koren, broj);
  }

  /* Racuna se i ispisuje broj AVL cvorova */
  printf("%d\n", avl(koren));

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  return 0;
}
```

Rešenje 4.26

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
   brojeva heap. Ideja koja ce biti implementirana u osnovi ima
   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
   desnog podstabla - ako je vrednost u korenu veca od izracunatih
   vrednosti uoceni fragment stabla zadovoljava uslov za heap. Zato
   ce funkcija vracati maksimalne vrednosti iz uocenog podstabala ili
   vrednost -1 ukoliko se zakljuci da stablo nije heap. */
int heap(Cvor * koren)
{
  int max_levo, max_desno;

  /* Prazno sablo je heap - kao rezultat se vraca 0 kao najmanji
     pozitivan broj */
  if (koren == NULL) {
    return 0;
  }
  /* Ukoliko je stablo list... */
```

```
24    if (koren->levo == NULL && koren->desno == NULL) {
        /* Vraca se njegova vrednost */
26      return koren->broj;
      }
28    /* Inace... */

30    /* Proverava se svojstvo za levo podstablo. */
      max_levo = heap(koren->levo);
32
      /* Proverava se svojstvo za desno podstablo. */
34    max_desno = heap(koren->desno);

36    /* Ako levo ili desno podstablo uocenog cvora nije heap, onda nije
         ni celo stablo. */
38    if (max_levo == -1 || max_desno == -1) {
        return -1;
40    }

42    /* U suprotonom proverava se da li svojstvo vazi za uoceni cvor. */
      if (koren->broj > max_levo && koren->broj > max_desno) {
44      /* Ako vazi, vraca se vrednost korena */
        return koren->broj;
46    }

48    /* U suprotnom zakljucuje se da stablo nije heap */
      return -1;
50  }

52  int main(int argc, char **argv)
    {
54    Cvor *koren;
      int heap_indikator;
56
      /* Kreira se stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2 7 */
58    koren = NULL;
      koren = napravi_cvor(100);
60    koren->levo = napravi_cvor(19);
      koren->levo->levo = napravi_cvor(17);
62    koren->levo->levo->levo = napravi_cvor(2);
      koren->levo->levo->desno = napravi_cvor(7);
64    koren->levo->desno = napravi_cvor(3);
      koren->desno = napravi_cvor(36);
66    koren->desno->levo = napravi_cvor(25);
      koren->desno->desno = napravi_cvor(1);
68
      /* Poziva se funkcija kojom se proverava da li je stablo heap */
70    heap_indikator = heap(koren);

72    /* Ispisuje se rezultat */
      if (heap_indikator == -1) {
74      printf("Zadato tablo nije heap\n");
      } else {
```

```
76      printf("Zadato stablo je heap!\n");
    }
78
    /* Oslobadja se memorija zauzeta stablom. */
80  oslobodi_stablo(&koren);

82  return 0;
}
```

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.

Primer 1

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

ulaz.txt
  5
  Programiranje
  Matematika
  12345
  dInAmiCnArEc
  Ispit

Interakcija programa:
  Ispit
  Matematika
  Programiranje
```

Primer 2

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

ulaz.txt
  2
  maksimalano
  poena

Interakcija programa:
```

Primer 3

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

Datoteka ulaz.txt ne postoji

Interakcija programa:
  -1
```

Primer 4

```
Poziv: ./a.out

Interakcija programa:
 -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
```

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:
 -1
```

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.

Primer 1

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

ulaz.txt
   Ovo je test primer.
   U njemu se rec test javlja
  vise puta. testtesttest

Interakcija programa:
  5
```

Primer 2

```
Poziv: ./a.out

Interakcija programa:
  (na stderr) -1
```

Primer 3

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

Datoteka ulaz.txt ne postoji

Interakcija programa:
  (na stderr) -1
```

Primer 4

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

Datoteka ulaz.txt je prazna

Interakcija programa:
  0
```

Zadatak 5.5 Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac: $P = \sqrt{s * (s-a) * (s-b) * (s-c)}$, gde je s poluobim trougla). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Primer 1

```
TROUGLOVI.TXT
 4
  0 0 0 1.2 1 0
  0.3 0.3 0.5 0.5 0.9 1
 -2 0 0 0 0 1
 -2 0 0 0 0 1

INTERAKCIJA PROGRAMA:
 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
```

Primer 2

```
TROUGLOVI.TXT
 3
  1.2 3.2 1.1 4.3

INTERAKCIJA PROGRAMA:
 -1
```

Primer 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
 -1
```

Primer 4

```
TROUGLOVI.TXT
 0

INTERAKCIJA PROGRAMA:
```

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
```

5.3 Programiranje 2, praktični deo ispita, septembar 2015.

Zadatak 5.7 Sa standardnog ulaza se učitavaju neoznačeni celi brojevi `x` i `n`. Na standardni izlaz ispisati neoznačen ceo broj koji se dobija od broja `x` kada se njegov binarni zapis rotira za `n` mesta udesno (na primer, ako je binarni zapis broja `x` jednak `00000000000000000000000000001111`, i ako je `n=1` tada na standardni izlaz treba ispisati neožnačen broj čiji je binarni zapis jednak `10000000000000000000000000000111`).

Test 1

```
Ulaz:
 6 1
Izlaz:
 3
```

Test 2

```
Ulaz:
 15 3
Izlaz:
 3758096385
```

Test 3

```
Ulaz:
 31 100
Izlaz:
 4026531841
```

Test 4

```
Ulaz:
 4 0
Izlaz:
 4
```

Test 5

```
Ulaz:
 0 5
Izlaz:
 0
```

Zadatak 5.8 Napisati funkciju `void dopuni_listu(Cvor** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

Test 1

```
Ulaz:
 1 2 3 4 5
Izlaz:
 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00
```

Test 2

```
Ulaz:
 12
Izlaz:
 12.00
```

Test 3

```
Ulaz:
 prazna lista
Izlaz:
```

Test 4

```
Ulaz:
 13.3 15.8
Izlaz:
 13.30 14.55
```

Zadatak 5.9 Sa standardnog ulaza se učitava dimenzija `n` kvadratne celobrojne matrice `A` ($n > 0$), a zatim i elementi matrice `A`. Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje je suma brojeva u svakom redu i svakoj koloni jed-

naka). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati -". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati `-1`.

Test 1

```
ULAZ:
 4
 1 2 3 4
 2 1 4 3
 3 4 2 1
 4 3 1 2
IZLAZ:
 10
```

Test 2

```
ULAZ:
 3
 1 1 1
 1 1 1
 1 1 1
IZLAZ:
 3
```

Test 3

```
ULAZ:
 2
 1 1
 2 2
IZLAZ:
 -
```

Test 4

```
ULAZ:
 2
 1 2
 1 2
IZLAZ:
 -
```

Test 5

```
ULAZ:
 1
 5
IZLAZ:
 5
```

Test 6

```
ULAZ:
 0
IZLAZ:
 -1
```

5.4 Rešenja

Rešenje 5.1

```
#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;

  /* Proverava argumenata komandne linije. */
  if (argc != 2) {
```

```
    greska();
22  }

24  /* Otvaranje datoteke cije ime je navedeno kao argument komandne
       linije neposredno nakon imena programa koji se poziva. */
26  ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
28    greska();
    }
30
    /* Ucitavanje broja linija. */
32  fscanf(ulaz, "%d", &n);

34  /* Alociranje memorije na osnovu ucitanog broja linija. */
    linije = (char **) malloc(n * sizeof(char *));
36  if (linije == NULL) {
      greska();
38  }
    for (i = 0; i < n; i++) {
40    linije[i] = malloc(MAX * sizeof(char));
      if (linije[i] == NULL) {
42      for (j = 0; j < i; j++) {
          free(linije[j]);
44      }
        free(linije);
46      greska();
      }
48  }

50  /* Ucitavanje svih n linija iz datoteke. */
    for (i = 0; i < n; i++) {
52    fscanf(ulaz, "%s", linije[i]);
    }
54
    /* Ispisivanje u odgovarajucem poretku ucitane linije koje
56     zadovoljavaju kriterijum. */
    for (i = n - 1; i >= 0; i--) {
58    if (isupper(linije[i][0])) {
        printf("%s\n", linije[i]);
60    }
    }
62
    /* Oslobadjanje memorije koja je dinamicki alocirana. */
64  for (i = 0; i < n; i++) {
      free(linije[i]);
66  }

68  free(linije);

70  /* Zatvaranje datoteku. */
    fclose(ulaz);
72
```

```
  return 0;
}
```

Rešenje 5.2

```
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura kojom se predstavlja Cvor stabla */
typedef struct dcvor {
  int broj;
  struct dcvor *levo, *desno;
} Cvor;

/* Funkcija alocira prostor za novi Cvor stabla, inicijalizuje polja
   strukture i vraca pokazivac na nov Cvor */
Cvor *napravi_cvor(int b);

/* Funkcija oslobadja 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
```

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

Cvor *napravi_cvor(int b)
{
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja novog Cvora */
  novi->broj = b;
  novi->levo = NULL;
  novi->desno = NULL;

  return novi;
}
```

```
19 void oslobodi_stablo(Cvor ** adresa_korena)
   {
21   /* Prazno stablo i nema sta da se oslobadja */
     if (*adresa_korena == NULL)
23     return;

25   /* Rekurzivno se oslobadja najpre levo, a onda i desno podstablo */
     if ((*adresa_korena)->levo)
27     oslobodi_stablo(&(*adresa_korena)->levo);
     if ((*adresa_korena)->desno)
29     oslobodi_stablo(&(*adresa_korena)->desno);

31   free(*adresa_korena);
     *adresa_korena = NULL;
33 }

35 void proveri_alokaciju(Cvor * novi)
   {
37   if (novi == NULL) {
       fprintf(stderr, "Malloc greska za nov cvor!\n");
39     exit(EXIT_FAILURE);
     }
41 }

43 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
   {
45   /* Postojece stablo je prazno */
     if (*adresa_korena == NULL) {
47     Cvor *novi = napravi_cvor(broj);
       proveri_alokaciju(novi);
49     /* Kreirani Cvor novi ce biti od sada koren stabla */
       *adresa_korena = novi;
51     return;
     }
53
     /* Brojevi se smestaju u uredjeno binarno stablo, pa ako je broj
55      koji se ubacuje manji od broja koji je u korenu onda se dodaje u
        levo podstablo. */
57   if (broj < (*adresa_korena)->broj)
       dodaj_u_stablo(&(*adresa_korena)->levo, broj);
59   /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
        dodaje se nov Cvor desno od korena. */
61   else
       dodaj_u_stablo(&(*adresa_korena)->desno, broj);
63 }
```

```
1  #include <stdio.h>
   #include "stabla.h"
3
   int sumirajN(Cvor * koren, int n)
5  {
```

```
  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;

  scanf("%d", &nivo);

  while (1) {
    scanf("%d", &n);
    /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
    if (n == 0)
      break;

    /* Ako nije, dodaje se procitani broj u stablo. */
    dodaj_u_stablo(&koren, n);
  }

  /* Ispisuje se rezultat rada trazene funkcije */
  printf("%d\n", sumirajN(koren, nivo));

  /* Oslobadja se memorija */
  oslobodi_stablo(&koren);

  return 0;
}
```

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;

  /* Ucitavanje 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;
  }

  /* Ucitavanje elemenata matrice */
  for (i = 0; i < v; i++) {
    for (j = 0; j < k; j++) {
      scanf("%d", &m[i][j]);
    }
  }

  /* Pronalazenje kolone 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;
      }
    }

  }

  /* Ispisivanje trazenog rezultata */
  printf("%d\n", max_indeks_kolone);

  return 0;
}
```

Rešenje 5.4

```c
#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);
  }

  /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
     linije */
  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

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

/* Struktura trougao */
typedef struct _trougao {
  double xa, ya, xb, yb, xc, yc;
```

```
} trougao;

/* Funkcija racuna duzinu duzi */
double duzina(double x1, double y1, double x2, double y2)
{
  return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

/* Funkcija racuna povrsinu trougla */
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));
}

/* Funkcija racuna poredi dva trougla, napisana tako da se moze
   proslediti funkciji qsort */
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;
}
```

Rešenje 5.6

```
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura koja predstavlja cvor stabla, sadrzi vrednost koja se
   cuva i pokazivace na levo i desno podstablo. */
typedef struct cvor {
  int vrednost;
  struct cvor *levi;
  struct cvor *desni;
} Cvor;

/* Pomocna funkcija za kreiranje cvora. Cvor se kreira dinamicki,
   funkcijom malloc(). U slucaju greske program se prekida i ispisuje
   se poruka o gresci. U slucaju uspeha inicijalizuje se vrednost
   datim brojem, a pokazivaci na podstabla se inicijalizuju na NULL.
   Funkcija vraca adresu novokreiranog cvora */
Cvor *napravi_cvor(int broj);

/* Funkcija dodaje novi cvor u stablo sa datim korenom. Ukoliko broj
   vec postoji u stablu, ne radi nista. Cvor se kreira funkcijom
   napravi_cvor(). */
void dodaj_u_stablo(Cvor ** koren, int broj);

/* Funkcija prikazuje stablo s leva u desno (tj. prikazuje elemente u
```

```
   rastucem poretku) */
void prikazi_stablo(Cvor * koren);

/* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
   vraca pokazican na njegov koren */
Cvor *ucitaj_stablo();

/* Funkcija oslobadja prostor koji je alociran za cvorove stabla. */
void oslobodi_stablo(Cvor ** koren);

#endif
```

```
#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)
```

```
41      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;
}
```

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

int f3(Cvor * koren, int n)
{
  if (koren == NULL || n < 0)
    return 0;
  if (n == 0) {
    if (koren->levi == NULL && koren->desni != NULL)
      return 1;
    if (koren->levi != NULL && koren->desni == NULL)
      return 1;
    return 0;
  }
  return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
}
```

```
17
int main()
19 {
   Cvor *koren;
21 int n;

23 scanf("%d", &n);
   koren = ucitaj_stablo();
25
   printf("%d\n", f3(koren, n));
27
   oslobodi_stablo(&koren);
29
   return 0;
31 }
```

Rešenje 5.7

```
#include <stdio.h>
2
unsigned int Rotiraj(unsigned int x, unsigned int n)
4 {
   int i;
6  unsigned int maska = 1;

8  /* Formiranje maske sa n jedinica na kraju 000...00001111 */
   for (i = 1; i < n; i++)
10   maska = (maska << 1) | 1;

12 return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
}
14
int main()
16 {
   unsigned int x, n;
18
   scanf("%u%u", &x, &n);
20
   printf("%u\n", Rotiraj(x, n));
22
   return 0;
24 }
```

Rešenje 5.8

```
#ifndef __LISTE_H__
2 #define __LISTE_H__ 1

4 /* Struktura koja predstavlja cvor liste */
```

```c
typedef struct cvor {
  double vrednost;
  struct cvor *sledeci;
} Cvor;

/* Pomocna funkcija koja kreira cvor. */
Cvor *napravi_cvor(double broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija proverava uspesnost alokacije memorije za cvor novi i
   ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
   zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
   adresa_glave.  */
void proveri_alokaciju(Cvor ** adresa_glave, Cvor * novi);

/* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
Cvor *pronadji_poslednji(Cvor * glava);

/* Funkcija dodaje novi cvor na kraj liste. */
void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
void ispisi_listu(Cvor * glava);

/* Funkcija koja dopunjuje listu na nacin opisan u zadatku */
void dopuni_listu(Cvor ** adresa_glave);

#endif
```

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

/* Pomocna funkcija koja kreira cvor. */
Cvor *napravi_cvor(double 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;

  while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *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 cija 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");
    oslobodi_listu(adresa_glave);
    exit(EXIT_FAILURE);
  }
}

/* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
Cvor *pronadji_poslednji(Cvor * glava)
{
  /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
     funkcija vraca NULL. */
  if (glava == NULL)
    return NULL;

  while (glava->sledeci != NULL)
    glava = glava->sledeci;

  return glava;
}

/* Funkcija dodaje novi cvor na kraj liste. */
void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
{
  Cvor *novi = napravi_cvor(broj);
  proveri_alokaciju(adresa_glave, novi);

  if (*adresa_glave == NULL) {
    *adresa_glave = novi;
    return;
  }
```

```
  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
  poslednji->sledeci = novi;
}

/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
void ispisi_listu(Cvor * glava)
{
  for (; glava != NULL; glava = glava->sledeci)
    printf("%.2lf ", glava->vrednost);

  putchar('\n');
}

/* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka */
void dopuni_listu(Cvor ** adresa_glave)
{
  Cvor *tekuci;
  Cvor *novi;
  double aritmeticka_sredina;
  if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
    return;

  tekuci = *adresa_glave;
  while (tekuci->sledeci != NULL) {
    aritmeticka_sredina =
        ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
    novi = napravi_cvor(aritmeticka_sredina);
    proveri_alokaciju(adresa_glave, novi);

    novi->sledeci = tekuci->sledeci;
    tekuci->sledeci = novi;
    tekuci = tekuci->sledeci;
    tekuci = tekuci->sledeci;
  }

  return;
}
```

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

int main()
{
  Cvor *glava = NULL;
  double broj;

  /* Ucitavanje se vrsi do kraja ulaza. Elementi se dodaju na kraj
     liste! */
  while (scanf("%lf", &broj) > 0)
    dodaj_na_kraj_liste(&glava, broj);
```

```
14    dopuni_listu(&glava);

16    ispisi_listu(glava);

18    oslobodi_listu(&glava);

20    return 0;
}
```

Rešenje 5.9

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

4 /* Funkcija proverava da li je magican kvadrat koji joj se
     prosledjuje kao argument. Ukoliko jeste magican funkcija vraca 1,
6    inace 0. */
int magicni_kvadrat(int **M, int n)
8 {
    int i, j;
10  int zbir = 0, zbir_pom;

12  for (j = 0; j < n; j++)
      zbir += M[0][j];
14
    for (i = 1; i < n; i++) {
16    zbir_pom = 0;
      for (j = 0; j < n; j++)
18      zbir_pom += M[i][j];
      if (zbir_pom != zbir)
20      return 0;
    }
22
    for (j = 0; j < n; j++) {
24    zbir_pom = 0;
      for (i = 0; i < n; i++)
26      zbir_pom += M[i][j];
      if (zbir_pom != zbir)
28      return 0;
    }
30  return 1;
}
32
int main()
34 {
    int n, i, j;
36  int **matrica = NULL;
    int zbir = 0;
38
    scanf("%d", &n);
```

```
40
   if (n <= 0) {
42   printf("-1\n");
     exit(EXIT_FAILURE);
44 }

46 matrica = (int **) malloc(n * sizeof(int *));
   if (matrica == NULL) {
48   printf("-1\n");
     exit(EXIT_FAILURE);
50 }

52 for (i = 0; i < n; i++) {
     matrica[i] = (int *) malloc(n * sizeof(int));
54
     if (matrica[i] == NULL) {
56     fprintf(stderr, "-1\n");
       for (j = 0; j < i; j++)
58       free(matrica[j]);

60     free(matrica);
       exit(EXIT_FAILURE);
62   }
   }
64
   for (i = 0; i < n; i++)
66   for (j = 0; j < n; j++)
       scanf("%d", &matrica[i][j]);
68
   if (magicni_kvadrat(matrica, n)) {
70   for (i = 0; i < n; i++)
       zbir += matrica[0][i];
72   printf("%d\n", zbir);
   } else
74   printf("-\n");

76 for (j = 0; j < n; j++)
     free(matrica[j]);
78
   free(matrica);
80
   return 0;
82 }
```