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.

Autori:
dr Milena Vujošević Janičić, docent na Matematičkom fakultetu u Beogradu
dr Jelena Graovac, docent na Matematičkom fakultetu u Beogradu
Ana Spasić, asistent na Matematičkom fakultetu u Beogradu
Mirko Spasić, asistent na Matematičkom fakultetu u Beogradu
Anđelka Zečević, asistent na Matematičkom fakultetu u Beogradu
Nina Radojičić, asistent na Matematičkom fakultetu u Beogradu

PROGRAMIRANJE 2
Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu
Studentski trg 16, 11000 Beograd
Za izdavača: *prof. dr Zoran Rakić*, dekan

Recenzenti:
dr Gordana Pavlović-Lažetić, redovni profesor na Matematičkom fakultetu u Beogradu
dr Dragan Urošević, naučni savetnik na Matematičkom institutu SANU

Obrada teksta, crteži i korice: *autori*
Štampa:
Tiraž:

CIP Каталогизација у публикацији
Народна библиотека Србије, Београд

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

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

(e) Napisati funkciju `float imaginaran_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja.

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

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

(h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja sabira dva kompleksna broja `z1` i `z2`.

(i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja oduzima dva kompleksna broja `z1` i `z2`.

(j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja množi dva kompleksna broja `z1` i `z2`.

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

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 SA PROGRAMOM:
 Unesite realan i imaginaran deo kompleksnog broja:  1 -3
 (1.00 - 3.00 i)
 Unesite realan i imaginaran deo kompleksnog broja:  -1 4
 (-1.00 + 4.00 i)
 Unesite znak (+,-,*):  -
 (1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)
 realan_deo: 2
 imaginaran_deo: -7.000000
 moduo: 7.280110
 Njegov konjugovano kompleksan broj: (2.00 + 7.00 i)
 Argument kompleksnog broja: - 1.292497
```

[Rešenje 1.1]

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

Primer 1

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

[Rešenje 1.2]

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 `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz. Ispisivanje polinoma počinje od najvišeg stepena ka najnižem. Ipisisuju se samo oni koeficijenti koji su različiti od nule.

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

(d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` za izračunavanje vrednosti polinoma `p` u datoj tački `x` koristeći Hornerov algoritam.

(e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja sabira dva polinoma `p` i `q`.

(f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja množi dva polinoma `p` i `q`.

(g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja računa izvod polinoma `p`.

(h) Napisati funkciju `Polinom nIzvod(const Polinom * p, int n)` koja računa `n`-ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Najpre se polinomi `p` i `q` unose sa standardnog ulaza i ispisuju na standardni izlaz u odgovarajućem obliku. Zatim se računa i ispisuje zbir i proizvod polinoma `p` i `q`. Označimo izračunati proizvod sa `r`. Nakon toga program računa i ispisuje vrednost polinoma `r` (zaokruženu na dve decimale) u tački koju unosi korisnik. Na kraju se sa standardnog ulaza unosi broj `n`, i ispisuje `n`-ti izvod polinoma `r`.

Primer 1

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

[Rešenje 1.3]

Zadatak 1.4 Napisati biblioteku za rad sa razlomcima.

(a) Definisati strukturu `Razlomak` za reprezentovanje razlomaka.

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

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

(d) Napisati funkciju `int brojilac(const Razlomak * r)` koje vraćaju brojilac razlomka `r`.

(e) Napisati funkciju `int imenilac(const Razlomak * r)` koje vraćaju imenilac razlomka `r`.

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

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

(h) Napisati funkciju `Razlomak skrati(const Razlomak * r)` koja skraćuje dati razlomak `r`.

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

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

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

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

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

1.2 Algoritmi za rad sa bitovima

Zadatak 1.5 Napisati biblioteku `stampanje_bitova` za rad sa bitovima koja sadrži 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
```

[Rešenje 1.5]

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 za broj koji unosi u heksadekasnom formatu sa standardnog ulaza računa broj bitova sa vrednošću `1` korišćenjem funkcije `count_bits1` ili funkcije `count_bits2`. Od korisnika sa standardnog ulaza tražiti da izabere koju od ove funkcije treba koristiit u zavisnosti da li unese `1` ili `2` .

Primer 1

```
Interakcija sa programom:
 Unesite broj: 0x7F
 Unesite redni broj funkcije: 1
 Broj jedinica u zapisu je
 funkcija count_bits1: 7
```

Primer 2

```
Interakcija sa programom:
 Unesite broj: 0x80
 Unesite redni broj funkcije: 2
 Broj jedinica u zapisu je
 funkcija count_bits2: 1
```

Primer 3

```
Interakcija sa programom:
 Unesite broj: 0x00FF00FF
 Unesite redni broj funkcije: 2
 Broj jedinica u zapisu je
 funkcija count_bits2: 16
```

Primer 4

```
Interakcija sa programom:
 Unesite broj: 0xABCDE123
 Unesite redni broj funkcije: 1
 Broj jedinica u zapisu je
 funkcija count_bits1: 17
```

[Rešenje 1.6]

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

Napisati program koji testira prethodno napisane funkcije tako što prikazuje binarnu reprezentaciju brojeva koji se dobijaju nakon poziva funkcije `najveci`, 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
```

[Rešenje 1.7]

Zadatak 1.8 Napisati funkcije za rad sa bitovima. Napomena: *Pozicije se broje počev od pozicije bita najmanje težine, pri čemu je bit najmanje težine na poziciji nula.*

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

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

(c) Napisati funkciju `get_bits` koja određuje broj u kome se n bitova najmanje težine poklapa sa n bitova broja x počevši od pozicije p.

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

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

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve x, n, p, y koji se unose sa standardnog ulaza. Program treba nakon učitavanja odgovarajućih vrednosti ispiše najpre binarne reprezenatacije brojeva x i y, a potom i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija.

Primer 1

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

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

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

 x =    235                                  = 00000000000000000000000011101011
 y =    127                                  = 00000000000000000000000001111111
 set_n_bits(   235,     9,    24,   127)     = 00000000011111110000000011101011

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

Primer 2

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

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

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

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

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

[Rešenje 1.8]

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite neoznacen ceo broj x: B10011A7
 Unesite neoznacen ceo broj k: 5
 x = 10110001000000000001000110100111
 rotate_left(2969571751, 5)         = 00100000000000100011010011110110
 rotate_right(2969571751, 5)        = 00111101100010000000000010001101
 rotate_right_signed(2969571751, 5) = 00111101100010000000000010001101
```

[Rešenje 1.9]

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

Test 1

```
ULAZ:
 255
IZLAZ:
 00000000000000000000001001010101
 10101010010000000000000000000000
```

Test 2

```
ULAZ:
 -15
IZLAZ:
 11111111111111111111111111101011
 11010111111111111111111111111111
```

[Rešenje 1.10]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.11]

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 sadrže dve uzastopne jedinice dva puta.*

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.12]

Zadatak 1.13 Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama i 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 sa programom:
Ulaz:
  11
Izlaz:
  13
```

Primer 2

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

Interakcija sa programom:
Ulaz:
  1024
Izlaz:
  1024
```

Primer 2

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

Interakcija sa programom:
Ulaz:
  12345
Izlaz:
  12345
```

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.14]

Zadatak 1.15 Napisati funkciju koja za 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

(a) tako da rešenje bude linearne složenosti,

(b) tako da rešenje bude logaritamske složenosti.

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza najpre unosi redni broj funckije koja se primenjuje, a zatim i ceo broj x i prirodan

broj k, a potom se na standarni izlaz ispisuje rezultat primene odgovarajuće funkcije na unete brojeve.

Primer 1

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

Primer 2

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

[Rešenje 1.17]

Zadatak 1.18 Koristeći uzajamnu (posrednu) rekurziju napisati:

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

(b) i funkciju `neparan` koja vraća 1, ukoliko je broj cifara nekog broja neparan, a 0 inače.

Napisati program koji testira napisane funkcije tako što za heksadekadni broj koji se unosi sa standardnog ulaza ispisuje da li je broj njegovih cifara paran ili neparan.

Test 1

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

Test 2

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

[Rešenje 1.18]

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

Primer 1

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

Primer 2

```
Interakcija sa programom:
 Unesite n (<= 12):   0
 0! = 1
```

[Rešenje 1.19]

Zadatak 1.20 Elementi niza 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 funkciju koja računa n-ti element u nizu F

(a) iterativno,

(b) tako da funkcija bude rekurzivna i da koristi navedene rekurentne relacije,

(c) tako da funkcija bude rekurzivna ali da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkciju tako što se sa standardnog ulaza najpre unosi redni broj funckije koja se primenjuje, a zatim i vrednosti koeficijenata a i b i prirodan broj n. Na standardni izlaz se ispisuje rezultat primene odabrane funkcije na unete vrednosti koeficijenata a i b i prirodan broj n. NAPOMENA: *Niz F definisan na ovaj način predstavlja uopštenje Fibonačijevih brojeva.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite redni broj funkcije koju zelite:
 1 - iterativna
 2 - rekurzivna
 3 - rekurzivna napredna
 1
 Unesite koeficijente:   2 3
 Unesite koji clan niza se racuna:   5
 F(5) = 61
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite redni broj funkcije koju zelite:
 1 - iterativna
 2 - rekurzivna
 3 - rekurzivna napredna
 3
 Unesite koeficijente:  4 2
 Unesite koji clan niza se racuna:  8
 F(8) = 31360
```

[Rešenje 1.20]

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

Test 1

```
ULAZ:
 123
IZLAZ:
 6
```

Test 2

```
ULAZ:
 23156
IZLAZ:
 17
```

Test 3

```
ULAZ:
 1432
IZLAZ:
 10
```

Test 4

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

Test 5

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

[Rešenje 1.21]

Zadatak 1.22 Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva

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

(b) sabirajući elemente počev od kraja niza ka početku niza.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije, zatim dimenziju n ($0 < n \leq 100$) celobrojnog niza, a potom i elemente niza. Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim nizom.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite redni broj funkcije (1 ili 2):  1
 Unesite dimenziju niza:  5
 Unesite elemente niza:
 1 2 3 4 5
 Suma elemenata je 15
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite redni broj funkcije (1 ili 2):  2
 Unesite dimenziju niza:  4
 Unesite elemente niza:
 -5 2 -3 6
 Suma elemenata je 0
```

[Rešenje 1.22]

Zadatak 1.23 Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. Niz neće imati više od 256 elemenata. 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
```

[Rešenje 1.23]

Zadatak 1.24 Napisati rekurzivnu funkciju 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.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju nizova:  3
 Unesite elemente prvog niza:
 1 2 3
 Unesite elemente drugog niza:
 1 2 3
 Skalarni proizvod je 14
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju nizova:  2
 Unesite elemente prvog niza:
 3 5
 Unesite elemente drugog niza:
 2 6
 Skalarni proizvod je 36
```

[Rešenje 1.24]

Zadatak 1.25 Napisati rekurzivnu funkciju 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.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite ceo broj:
 4
 Unesite elemente niza:
 1 2 3 4
 Broj pojavljivanja je 1
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite ceo broj:
 11
 Unesite elemente niza:
 3 2 11 14 11 43 1
 Broj pojavljivanja je 2
```

Primer 3

```
INTERAKCIJA SA PROGRAMOM:
 Unesite ceo broj:
 1
 Unesite elemente niza:
 3 21 5 6
 Broj pojavljivanja je 0
```

[Rešenje 1.25]

Zadatak 1.26 Napisati rekurzivnu funkciju 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.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite tri cela broja:
 1 2 3
 Unesite elemente niza:
 4 1 2 3 4 5
 Uneti brojevi jesu uzastopni clanovi niza.
```

Primer 2

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

[Rešenje 1.26]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 1.27]

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

Test 1

```
ULAZ:
 10
IZLAZ:
 00000000000000000000000000001010
```

Test 2

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

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
```

[Rešenje 1.29]

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

Test 1

```
ULAZ:
 5
IZLAZ:
 5
```

Test 2

```
ULAZ:
 16
IZLAZ:
 1
```

Test 3

```
ULAZ:
 18
IZLAZ:
 2
```

[Rešenje 1.30]

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. Pretpostaviti 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
```

[Rešenje 1.31]

*** 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 ($n \leq 15$) unet sa standardnog ulaza.

Test 1

```
Ulaz:
 2
Izlaz:
 1 2
 2 1
```

Test 2

```
Ulaz:
 3
 1 2 3
 1 3 2
 2 1 3
 2 3 1
 3 1 2
 3 2 1
```

Test 3

```
Ulaz:
 -5
 Duzina
 permutacije
 mora biti
 broj iz
 intervala
 [0, 50]!
```

[Rešenje 1.32]

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

```
          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 redni broj hipotenuze najpre iscrtava Paskalov trougao, a zatim štampa sumu elemenata hipotenuze.

Test 1

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

 8
```

Test 2

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

 32
```

[Rešenje 1.33]

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

Test 1

```
Ulaz:
 2
Izlaz:
 a a
 a b
 b a
 b b
```

Test 2

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

Zadatak 1.35 *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 <stdlib.h>
#include <math.h>

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

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

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

  /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
     razlicit od nule: tada se realni deo ispisuje na standardni
     izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
     imaginarni deo pozitivan ili negativan, a potom i apsolutna
     vrednost imaginarnog dela kompleksnog broja 2) realni deo
     kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
     s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
     decimalnih mesta */
```

```
40    if (z.real != 0) {
        printf("%.2f", z.real);
42
        if (z.imag > 0)
44        printf(" + %.2f i", z.imag);
        else if (z.imag < 0)
46        printf(" - %.2f i", -z.imag);
    } else {
48      if (z.imag == 0)
          printf("0");
50      else
          printf("%.2f i", z.imag);
52  }

54  /* Zavrsava se sa ispisom */
    printf(")");
56
}
58
/* Funkcija vraca vrednosti realnog dela kompleksnog broja */
60 float realan_deo(KompleksanBroj z)
{
62  return z.real;
}
64
/* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
66 float imaginaran_deo(KompleksanBroj z)
{
68  return z.imag;
}
70
/* Funkcija vraca vrednost modula zadatog kompleksnog broja */
72 float moduo(KompleksanBroj z)
{
74  return sqrt(z.real * z.real + z.imag * z.imag);
}
76
/* Funkcija vraca vrednost konjugovano kompleksnog broja koji
78    odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z)
80 {
    /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
82     imaginarnom delu kompleksnog broja */

84  KompleksanBroj z1 = z;

86  z1.imag *= -1;

88  return z1;
}
90
/* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
```

```
92    argumenata funkcije */
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
94 {
  /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
96    broj ciji je realan deo zbir realnih delova kompleksnih brojeva
      z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
98    brojeva z1 i z2 */

100 KompleksanBroj z = z1;

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

108 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
    argumenata funkcije */
110 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
{
112 /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
      broj ciji je realan deo razlika realnih delova kompleksnih
114   brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
      kompleksnih brojeva z1 i z2 */
116
  KompleksanBroj z = z1;
118
  z.real -= z2.real;
120 z.imag -= z2.imag;

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

138 return z;
}
140
/* Funkcija vraca argument zadatog kompleksnog broja */
142 float argument(KompleksanBroj z)
{
```

```
144   /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
         iz biblioteke math.h */
146
      return atan2(z.imag, z.real);
148 }

150 int main()
    {
152   char c;

154   /* Deklaracija 3 promenljive tipa KompleksanBroj */
      KompleksanBroj z1, z2, z;
156
      /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
158      ispisivanje na standardni izlaz */
      ucitaj_kompleksan_broj(&z1);
160   ispisi_kompleksan_broj(z1);
      printf("\n");
162
      /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
164      na standardni izlaz */
      ucitaj_kompleksan_broj(&z2);
166   ispisi_kompleksan_broj(z2);
      printf("\n");
168
      /* Ucitavanje i provera znaka na osnovu koga korisnik bira
170      aritmeticku operaciju koja ce se izvrsiti nad kompleksnim
         brojevima */
172   getchar();
      printf("Unesite znak (+,-,*): ");
174   scanf("%c", &c);
      if (c != '+' && c != '-' && c != '*') {
176     printf("Greska: nedozvoljena vrednost operatora!\n");
        exit(EXIT_FAILURE);
178   }

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

192   /* Ispisuje se rezultat */
      printf("\n");
194   ispisi_kompleksan_broj(z1);
      printf(" %c ", c);
```

```
196   ispisi_kompleksan_broj(z2);
      printf("  =  ");
198   ispisi_kompleksan_broj(z);
      printf("\n");
200
      /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
202      broja */
      printf("\nrealan_deo: %.f\nimaginaran_deo: %f\nmoduo: %f\n",
204          realan_deo(z), imaginaran_deo(z), moduo(z));

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

212   /* Testira se funkcija koja racuna argument kompleksnog broja */
      printf("Argument kompleksnog broja: %f\n", argument(z));
214
      return 0;
216 }
```

Rešenje 1.2

Datoteka 1.1: *complex.h*

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

7 /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i
     onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
9    posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
     narednoj pretrocesorskoj define direktivi. */
11 #ifndef  _COMPLEX_H
   #define _COMPLEX_H
13
   /* Zaglavlja standardne biblioteke koje sadrze deklaracije funkcija
15    koje se koriste u definicijama funkcija navedenim u complex.c */
   #include <stdio.h>
17 #include <math.h>

19 /* Struktura KompleksanBroj */
   typedef struct {
21   float real;
     float imag;
23 } KompleksanBroj;
```

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

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

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

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

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

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

/* Funkcija vraca vrednost konjugovano kompleksnog broja koji
   odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z);

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

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

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

/* Funkcija vraca argument zadatog kompleksnog broja */
float argument(KompleksanBroj z);

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

Datoteka 1.2: *complex.c*

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

```

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

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

  /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
     razlicit od nule: tada se realni deo ispisuje na standardni
     izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
     imaginarni deo pozitivan ili negativan, a potom i apsolutna
     vrednost imaginarnog dela kompleksnog broja 2) realni deo
     kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
     s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
     decimalnih mesta */

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

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

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

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

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

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

66 KompleksanBroj konjugovan(KompleksanBroj z)
{
68   /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
        se promeni znak imaginarnom delu kompleksnog broja */
70   KompleksanBroj z1 = z;
   z1.imag *= -1;
72   return z1;
}
74
   KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76 {
   /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
78      broj ciji je realan deo zbir realnih delova kompleksnih brojeva
        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
80      brojeva z1 i z2 */
   KompleksanBroj z = z1;
82
   z.real += z2.real;
84   z.imag += z2.imag;

86   return z;
}
88
   KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
90 {
   /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
92      broj ciji je realan deo razlika realnih delova kompleksnih
        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
94      kompleksnih brojeva z1 i z2 */
   KompleksanBroj z = z1;
96   z.real -= z2.real;
   z.imag -= z2.imag;
98   return z;
}
100
   KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
102 {
   /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
104     broj ciji se realan i imaginaran deo racunaju po formuli za
        mnozenje kompleksnih brojeva z1 i z2 */
106  KompleksanBroj z;

108  z.real = z1.real * z2.real - z1.imag * z2.imag;
   z.imag = z1.real * z2.imag + z1.imag * z2.real;
110
```

```
  return z;
}

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

Datoteka 1.3: *main.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 complex 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

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


#include <stdio.h>
/* Ukljucuje se 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);

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

```
39
  return 0;
41 }
```

Rešenje 1.3

Datoteka 1.4: *polinom.h*

```
#ifndef  _POLINOM_H
2 #define _POLINOM_H

4 #include <stdio.h>
#include <stdlib.h>
6
/* Maksimalni stepen polinoma */
8 #define MAX_STEPEN 20


10
/* Polinomi se predstavljaju strukturom koja cuva koeficijente
12    (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
typedef struct {
14   double koef[MAX_STEPEN + 1];
  int stepen;
16 } Polinom;

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

24 /* Funkcija koja ucitava polinom sa tastature */
Polinom ucitaj();
26
/* Funkcija racuna i vraca vrednost polinoma p u tacki x Hornerovim
28    algoritmom */
double izracunaj(const Polinom * p, double x);
30
/* Funkcija koja sabira dva polinoma */
32 Polinom saberi(const Polinom * p, const Polinom * q);

34 /* Funkcija koja mnozi dva polinoma p i q */
Polinom pomnozi(const Polinom * p, const Polinom * q);
36
/* Funkcija koja racuna izvod polinoma p */
38 Polinom izvod(const Polinom * p);

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

Datoteka 1.5: *polinom.c*

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

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

    if (p->koef[i]) {
    /* Polinom nije nula polinom, cim je neki od koeficijenata
         razlicit od nule */
    nulaPolinom = 0;
      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]);
    }
  }
  /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada se
     ispisuje nula. */
  if(nulaPolinom)
    printf("0");
  putchar('\n');
}

Polinom ucitaj()
{
  int i;
  Polinom p;

  /* Ucitava se stepena polinoma */
  scanf("%d", &p.stepen);

  /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
     dozvoljenog opsega */
  while (p.stepen > MAX_STEPEN || p.stepen < 0) {
    printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
    scanf("%d", &p.stepen);
```

```
50    }
52    /* Unose se koeficijenti polinoma */
      for (i = p.stepen; i >= 0; i--)
54      scanf("%lf", &p.koef[i]);
56    /* Vraca se procitani polinom */
      return p;
58  }
60  double izracunaj(const Polinom * p, double x)
    {
62    /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
         svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
64       vrednost odgovarajuceg koeficijenta */
66    /* Primer: Hornerov algoritam za polinom x^4+2x^3+3x^2+2x+1:
         x^4+2x^3+3x^2+2x+1 = (((x+2)*x + 3)*x + 2)*x + 1 */
68
      double rezultat = 0;
70    int i = p->stepen;
      for (; i >= 0; i--)
72      rezultat = rezultat * x + p->koef[i];
      return rezultat;
74  }
76  Polinom saberi(const Polinom * p, const Polinom * q)
    {
78    Polinom rez;
      int i;
80
      /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
82       stepenom */
      rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
84
      /* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
86       sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
         sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
88       od stepena nekog od polaznih polinoma podrazumeva se da je
         koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
90       polinoma */
      for (i = 0; i <= rez.stepen; i++)
92      rez.koef[i] =
            (i > p->stepen ? 0 : p->koef[i]) +
94          (i > q->stepen ? 0 : q->koef[i]);
96    /* Vraca se dobijeni polinom */
      return rez;
98
  }
100
  Polinom pomnozi(const Polinom * p, const Polinom * q)
```

```
102 {
      int i, j;
104   Polinom r;

106   /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
        */
      r.stepen = p->stepen + q->stepen;
108   if (r.stepen > MAX_STEPEN) {
        fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
110     exit(EXIT_FAILURE);
      }
112
      /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na nulu
        */
114   for (i = 0; i <= r.stepen; i++)
        r.koef[i] = 0;
116
      /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
118        za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
      for (i = 0; i <= p->stepen; i++)
120     for (j = 0; j <= q->stepen; j++)
          r.koef[i + j] += p->koef[i] * q->koef[j];
122
      /* Vraca se dobijeni polinom */
124   return r;
    }
126
    Polinom izvod(const Polinom * p)
128 {
      int i;
130   Polinom r;

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

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

```
152   Polinom r;

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

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

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

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

Datoteka 1.6: *main.c*

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

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

10  /* Unos polinoma p */
    printf
12      ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
      najveceg stepena do nultog):\n");
    p = ucitaj();
14
    /* Ispis polinoma p */
16  ispisi(&p);

18  /* Unos polinoma q */
    printf
20      ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
      od najveceg stepena do nultog):\n");
    q = ucitaj();
22
    /* Polinomi se sabiraju i ispisuje se izracunati zbir */
24  r = saberi(&p, &q);
    printf("Zbir polinoma je: ");
```

```
26    ispisi(&r);

28    /* Polinomi se mnoze i ispisuje se izracunati prozivod */
      r = pomnozi(&p, &q);
30    printf("Prozvod polinoma je polinom r:\n");
      ispisi(&r);
32
      /* Ispisuje se vrednost polinoma u unetoj tacki */
34    printf("Unesite tacku u kojoj racunate vrednost polinoma r\n");
      scanf("%lf", &x);
36    printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&r, x));

38
      /* Racuna se n-ti izvoda polinoma i ispisuje se dobijeni polinoma
        */
40    printf("Unesite izvod polinoma koji zelite:\n");
      scanf("%d", &n);
42    r = nIzvod(&r, n);
      printf("%d. izvod polinoma r je: ", n);
44    ispisi(&r);

46    return 0;
  }
```

Rešenje 1.5

Datoteka 1.7: *stampanje_bitova.h*

```
  #ifndef  _STAMPANJE_BITOVA_H
2 #define _STAMPANJE_BITOVA_H

4 #include <stdio.h>

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

12 #endif
```

Datoteka 1.8: *stampanje_bitova.c*

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include "stampanje_bitova.h"
4
  void print_bits(unsigned x)
```

```
{
  /* Broj bitova celog broja */
  unsigned velicina = sizeof(unsigned) * 8;

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

  /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
     na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
     mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
     se jedini bit jedinica pomera udesno, kako bi se ocitao naredni
     bit broja x koji je argument funkcije. Odgovarajuci karakter,
     ('0' ili '1'), ispisuje se na standardnom izlazu. Neophodno je
     da promenljiva maska bude deklarisana kao neoznacen ceo broj
     kako bi se pomeranjem u desno vrsilo logicko pomeranje
     (popunjavanje nulama) a ne aritmeticko pomeranje (popunjavanje
     znakom broja). */
  for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
    putchar(x & maska ? '1' : '0');

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

Datoteka 1.9: *main.c*

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

int main()
{
  int broj;

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

  /* I ispisuje se njegova binarna reprezentacija */
  print_bits(broj);

  return 0;
}
```

Rešenje 1.6

```
#include <stdio.h>

/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   kreiranjem odgovarajuce maske i njenim pomeranjem */
int count_bits1(int x)
{
```

```
7   int br = 0;
    unsigned wl = sizeof(unsigned) * 8 - 1;
9
    /* Formiranje se maska cija binarna reprezentacija izgleda
11     100000...0000000, koja sluzi za ocitavanje bita najvece tezine.
       U svakoj iteraciji maska se pomera u desno za 1 mesto, i
13     ocitavamo sledeci bit. Petlja se zavrsava kada vise nema
       jedinica tj. kada maska postane nula. */
15  unsigned maska = 1 << wl;
    for (; maska != 0; maska >>= 1)
17    x & maska ? br++ : 1;

19  return br;
}
21
/* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
23    formiranjem odgovarajuce maske i pomeranjem promenljive x */
int count_bits2(int x)
25 {
    int br = 0;
27  unsigned wl = sizeof(int) * 8 - 1;

29  /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1 bi
       vrsila aritmeticko pomeranje u desno, tj. popunjavanje bita
31     najvece tezine bitom znaka. U tom slucaju nikad ne bi bio
       ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
33     petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
       koja ocitava bit najvece tezine. */
35
    unsigned maska = 1 << wl;
37  for (; x != 0; x <<= 1)
      x & maska ? br++ : 1;
39
    return br;
41 }

43 int main()
{
45  int x, i;

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

51  /* Dozvoljava se korisniku da bira na koji nacin ce biti izracunat
       broj jedinica u zapisu broja */
53  printf("Unesite redni broj funkcije:\n");
    scanf("%d", &i);
55
    /* Ispisuje se rezultat */
57  printf("Broj jedinica u zapisu je\n");
    if (i == 1)
```

```
59      printf("funkcija count_bits1: %d\n", count_bits1(x));
    else
61      printf("funkcija count_bits2: %d\n", count_bits2(x));

63  return 0;
}
```

Rešenje 1.7 NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

```
1  #include <stdio.h>
   #include "stampanje_bitova.h"
3
   /* Funkcija vraca najveci neoznaceni broj sastavljen od istih bitova
5     koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
   unsigned najveci(unsigned x)
7  {
     unsigned velicina = sizeof(unsigned) * 8;
9
     /* Formira se maska 100000...0000000 */
11   unsigned maska = 1 << (velicina - 1);

13   /* Rezultat se inicijalizuje vrednoscu 0 */
     unsigned rezultat = 0;
15
     /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj
17      binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
        od nule). */
19   for (; x != 0; x <<= 1) {
       /* Za svaku jedinicu koja se koriscenjem maske detektuje na
21        poziciji najvece tezine u binarnoj reprezentaciji promenjive
          x, potiskuje se jedna nova jedinicu sa leva u rezultat */
23     if (x & maska) {
         rezultat >>= 1;
25       rezultat |= maska;
       }
27   }

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

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

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

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

int main()
{
  int broj;

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

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

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

  return 0;
}
```

Rešenje 1.8 NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

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

/* Funckija postavlja na nulu n bitova pocev od pozicije p. */
unsigned reset(unsigned x, unsigned n, unsigned p)
{
/*************************************************************
     Formira se maska cija binarna reprezentacija ima n bitova
     postavljenih na 0 pocev od pozicije p, dok su svi ostali
     postavljeni na 1. Na primer, za n=5 i p=10 formira se
     maska oblika
     1111 1111 1111 1111 1111 1000 0011 1111
```

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

  return x & maska;
}


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

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

  return x | maska;
}

/* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
   predstavlja n bitova pocev od pozicije p u binarnoj
   reprezentaciji broja x. */
unsigned 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 najmanje 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. */
  unsigned last_n_1 = ~(~0 << n);

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

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

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

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

```

```
   /* Ispisuju se binarne reprezentacije broja x i broja koji se
      dobije kada se primeni funkcija reset za x, n i p*/
   printf("x = %6u %28s = ", x, "");
   print_bits(x);
   printf("reset(%10u,%6u,%6u)%12s = ", x, n, p, "");
   print_bits( reset(x, n, p));
   printf("\n");

   /* Ispisuju se binarne reprezentacije broja x i broja koji se
      dobije kada se primeni funkcija set za x, n i p*/
   printf("x = %10u %28s = ", x, "");
   print_bits(x);
   printf("set(%10u,%6u,%6u)%14s = ", x, n, p, "");
   print_bits( set(x, n, p));
   printf("\n");

   /* Ispisuju se binarne reprezentacije broja x i broja koji se
      dobije kada se primeni funkcija get_bits za x, n i p*/
   printf("x = %10u %28s = ", x, "");
   print_bits(x);
   printf("get_bits(%10u,%6u,%6u)%9s = ", x, n, p, "");
   print_bits( get_bits(x, n, p));
   printf("\n");

   /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
      dobije kada se primeni funkcija set_n_bits za x, n i p*/
   printf("x = %10u %28s = ", x, "");
   print_bits(x);
   printf("y = %10u %29s= ", y, "");
   print_bits(y);
   printf("set_n_bits(%10u,%4u,%4u,%10u) = ", x, n, p, y);
   print_bits( set_n_bits(x, n, p, y));
   printf("\n");

   /* Ispisuju se binarne reprezentacije broja x i broja koji se
      dobije kada se primeni funkcija invert za x, n i p*/
   printf("x = %10u %28s = ", x, "");
   print_bits(x);
   printf("invert(%10u,%6u,%6u)%11s = ", x, n, p, "");
   print_bits( invert(x, n, p));

   return 0;
}
```

Rešenje 1.9 NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

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

```

```c
/* Funkcija ceo broj x rotira u levo za n mesta. */
unsigned rotate_left(int x, unsigned n)
{
  unsigned most_significant_bit;

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

  /* n puta se vrsi rotaciju za jedan bit u levo. U svakoj iteraciji
     se odredi bit na poziciji najvece tezine, a potom se pomera
   binarna reprezentacija trenutne vrednosti promenljive x u levo
   za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
   na vrednost koju je imao bit na poziciji najvece tezine koji je
   istisnut pomeranjem */
  for (i = 0; i < n; i++) {
    most_significant_bit = x & most_significant_bit_mask;
    x = x << 1 | (most_significant_bit ? 1 : 0);
  }

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

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

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

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

/* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
   argument funkcije x oznaceni ceo broj */
```

```
55 int rotate_right_signed(int x, unsigned n)
   {
57   unsigned least_significant_bit;
     int i;
59
     /* U svakoj iteraciji se odredjuje bit na poziciji najmanje tezine
61      i smesta u promenljivu least_significant_bit. Kako je x oznacen
      ceo broj, tada se prilikom pomeranja u desno vrsi aritmeticko
63    pomeranje i cuva se znak broja. Dakle, razlikuju se dva slucaja
      u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
65    izvrsi pre petlje, s obzirom da rotiranjem u desno na poziciju
      nejvece tezine moze doci i 0 i 1, nezavisno od pocetnog znaka
67    broja smestenog u promenljivu x. */
     for (i = 0; i < n; i++) {
69     least_significant_bit = x & 1;

71     if (x < 0)
     /*****************************************************************
73     Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
     na poziciji najvece tezine. Na primer ako je x
75     1010 1011 1100 1101 1110 0001 0010 001b
         (sa b je oznacen ili 1 ili 0 na poziciji najmanje tezine)
77     Onda je sadrzaj promenljive least_significant_bit:
       0000 0000 0000 0000 0000 0000 0000 000b
79     Nakon siftovanja sadrzaja promenljive x za 1 u desno
       1101 0101 1110 0110 1111 0000 1001 0001
81     Kako bi umesto 1 na poziciji najvece tezine u trenutnoj binarnoj
       reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
83     poziciju najvece tezine jer bi se time dobile 0, a u ovom slucaju
     su potrebne jedinice zbog bitovskog & zato se prvo vrsi
85     komplementiranje, a zatim pomeranje
       ~least_significant_bit << (sizeof(int)*8 -1)
87     B000 0000 0000 0000 0000 0000 0000 0000
       gde B oznacava ~b.
89     Potom se ponovo vrsi komplementiranje kako bi se b nalazilo na
       poziciji najvece tezine i sve jedinice na ostalim pozicijama
91     ~(~least_significant_bit << (sizeof(int)*8 -1))
       b111 1111 1111 1111 1111 1111 1111 1111
93   *****************************************************************/
         x = (x >> 1) & ~(~least_significant_bit << (sizeof(int) * 8 -
       1));
95     else
         x = (x >> 1) | least_significant_bit << (sizeof(int) * 8 - 1);
97   }

99   /* Vraca se dobijena vrednost */
     return x;
101 }

103 int main()
    {
105   unsigned x, k;
```

```
107   /* Ucitavaju se vrednosti sa standardnog ulaza */
      printf("Unesite neoznacen ceo broj x:");
109   scanf("%x", &x);
      printf("Unesite neoznacen ceo broj k:");
111   scanf("%x", &k);

113   /* Ispisuje se binarna reprezentacija broja x */
      printf("x = ", "");
115   print_bits(x);

117   /* Testira se rad napisanih funkcija */
      printf("rotate_left(%10u,%10u) %8s= ", x, k, "");
119   print_bits(rotate_left(x, k));

121   printf("rotate_right(%10u,%10u) %7s= ", x, k, "");
      print_bits(rotate_right(x, k));
123
      printf("rotate_right_signed(%10u,%10u) = ", x, k);
125   print_bits(rotate_right_signed(x, k));

127   return 0;
}
```

Rešenje 1.10 NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.*

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

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

```
  }

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

int main()
{
  int broj;

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

  /* Ispisuje se njegova binarna reprezentacija */
  print_bits(broj);

  /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
     funkcije mirror */
  print_bits(mirror(broj));

  return 0;
}
```

Rešenje 1.11

```
#include <stdio.h>

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

  int broj_nula, broj_jedinica;
  unsigned int maska;

  broj_nula = 0;
  broj_jedinica = 0;

  /* Maska je inicijalizovana tako da moze da analizira bit najvece
     tezine */
  maska = 1 << (sizeof(unsigned int) * 8 - 1);

  /* 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
     1 biti na svim pozicijama u binarnoj reprezentaciji maske */
  while (maska != 0) {

    /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
       1 i uveca se odgovarajuci brojac */
    if (n & maska) {
      broj_jedinica++;
```

```
    } else {
      broj_nula++;
    }

    /* 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;

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

  /* Ispisuje se rezultat */
  printf("%d\n", Broj01(n));

  return 0;
}
```

Rešenje 1.12

```
#include <stdio.h>

/* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju u
   binarnom zapisu celog čneoznaenog broja x */
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) {
```

```
22      /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
        */
        if ((x & maska) == maska) {
24        broj_parova++;
        }
26
        /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
28         proveravao sledeci par bitova. Pomeranjem u desno bit najvece
           tezine se popunjava nulom jer je x neoznacen broj */
30      x = x >> 1;
    }
32
    /* Vraca se dobijena vrednost */
34  return broj_parova;

36}

38int main()
{
40  unsigned int x;

42  /* Ucitava se broj sa ulaza */
    scanf("%u", &x);
44
    /* Ispisuje se rezultat */
46  printf("%d\n", broj_parova(x));

48  return 0;
}
```

Rešenje 1.14

```
#include <stdio.h>
2
/* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
4     su za svaku heksadekadnu cifru potrebne 4 binarne cifre i jedna
      dodatna pozicija za terminirajucu nulu.
6     Prethodni izraz se moze zapisati kao sizeof(unsigned int)*2+1. */
#define MAX_DUZINA sizeof(unsigned int)*2 +1
8
/* Funkcija za neoznacen broj x formira nisku s koja sadrzi njegov
10     heksadekadni zapis */
void prevod(unsigned int x, char s[])
12{

14  int i;
    unsigned int maska;
16  int vrednost;

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

```
20    maska = 15;
22    /*****************************************************************
        Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
24      tezine. Na primer za broj cija je binarna reprezentacija
        00000000001101000100001111010101
26      u prvom koraku se citaju bitovi izdvojeni sa <...>:
        00000000001101000100000111101<0101>
28      u drugom koraku:
        00000000001101000100000011<1101>0101
30      u trecem koraku:
        00000000001101000100<0011>11010101 i tako redom...
32
        Indeks i oznacava poziciju na koju se smesta vrednost.
34    *****************************************************************/
      for (i = MAX_DUZINA - 2; i >= 0; i--) {
36      /* Vrednost izdvojene cifre */
        vrednost = x & maska;
38
        /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
40         dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
           10 do 15 odgovarajuci karakter se dobija tako sto se prvo
42         oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
           dobijenu vrednost doda ASCII kod 'A' (time se dobija
44         odgovarajuce slovo 'A', 'B', ... 'F') */
        if (vrednost < 10) {
46        s[i] = vrednost + '0';
        } else {
48        s[i] = vrednost - 10 + 'A';
        }
50
        /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u
52         narednoj iteraciji posmatraju sledeca 4 bita */
        x = x >> 4;
54    }
56    /* Upisuje se terminirajuca nula */
      s[MAX_DUZINA - 1] = '\0';
58  }
60  int main()
    {
62    unsigned int x;
      char s[MAX_DUZINA];
64
      /* Ucitava se broj sa ulaza */
66    scanf("%u", &x);
68    /* Poziva se funkcija za prevodjenje */
      prevod(x, s);
70
      /* I stampa se dobijena niska */
```

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

74    return 0;
}
```

Rešenje 1.17

```
#include <stdio.h>

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

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

/*********************************************************************
   Resenje logaritamske slozenosti:
   x^0 =1;
   x^k = x * (x^2 )^(k/2) , za neparno k
   x^k = (x^2)^(k/2) , za parno k
   Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
   se doslo do rezultata, i stoga je efikasnije.
*********************************************************************/
int stepen2(int x, int k)
{
  if (k == 0)
    return 1;

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

  /* Inace (ukoliko je stepen neparan) */
  return x * stepen2(x * x, k / 2);
}

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

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

```
46    /* Ucitavaju se vrednosti za x i k */
      printf("Unesite broj x:\n");
48    scanf("%d%d", &x);
      printf("Unesite broj k:\n");
50    scanf("%d%d", &k);

52    /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
      if (x == 1)
54      printf("%d\n", stepen(x, k));
      else
56      printf("%d\n", stepen2(x, k));

58    return 0;
  }
```

Rešenje 1.18

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

4 #define MAX 100

6 /* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
     (posrednu) rekurziju */
8
  /* Deklaracija funkcije neparan mora da bude navedena jer se ta
10    funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
      definicije. Funkcija je mogla biti deklarisana i u telu funkcije
12    paran. */
  unsigned neparan(unsigned n);
14
   /* Funkcija vraca 1 ako broj n ima paran broj cifara, inace vraca 0
      */
16 unsigned paran(unsigned n)
  {
18   if (n <= 9)
       return 0;
20   else
       return neparan(n / 10);
22 }

24  /* Funkcija vraca 1 ako broj n ima neparan broj cifara, inace vraca
       0 */
26 unsigned neparan(unsigned n)
  {
28   if (n <= 9)
       return 1;
30   else
       return paran(n / 10);
32 }
```

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

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

46 return 0;
}
```

Rešenje 1.19

```
1 #include <stdio.h>
  #include <stdlib.h>
3
/* Pomocna funkcija koja izracunava n! * result. Koristi repnu
5    rekurziju. Result je argument u kome se akumulira do tada
     izracunatu vrednost faktorijela. Kada dodje do izlaza iz rekurzije
7    iz rekurzije potrebno je da vratimo result. */
int faktorijelRepna(int n, int result)
9 {
   if (n == 0)
11   return result;

13 return faktorijelRepna(n - 1, n * result);
}
15
/* U sledecim funkcijama je prikazan postupak oslobadjanja od
17   repne rekurzije koja postoji u funkciji faktorijelRepna. */

19 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
     naredbama kojima se vrednost argumenta funkcije postavlja na
21   vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
     goto naredbe za vracanje na pocetak tela funkcije. */
23 int faktorijelRepna_v1(int n, int result)
{
25 pocetak:
   if (n == 0)
27   return result;

29 result = n * result;
   n = n - 1;
31 goto pocetak;
}
33
```

```
/* 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);
}

int main()
{
  int n;

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 1.20

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

```

```c
/* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
int F_iterativna(int n, int a, int b)
{
  int i;
  int F_0 = 0;
  int F_1 = 1;
  int tmp;

  if (n == 0)
    return 0;

  for (i = 2; i <= n; i++) {
    tmp = a * F_1 + b * F_0;
    F_0 = F_1;
    F_1 = tmp;
  }

  return F_1;
}

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

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

}

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

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

/* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
   verzija */
```

```c
int F_napredna(int n, int a, int b)
{
  /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
     staticke promenljive na podrazumevane vrednosti. Stoga, elemente
     celobrojnog niza inicijalizuje na 0 */
  static int f[20];

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

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

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

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

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

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

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

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

  return 0;
}
```

Rešenje 1.21

```c
#include <stdio.h>

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

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

int main()
{
  unsigned int x;

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

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

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

/* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
   jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
   elementa niza i sume preostalih n-1 elementa. */
int sumaNiza2(int *a, int n)
{
```

```
20    if (n <= 0)
        return 0;
22
    return a[0] + sumaNiza2(a + 1, n - 1);
24  }

26  int main()
    {
28    int a[MAX_DIM];
      int n, i = 0, ind;
30
      /* Ucitava se redni broj funkcije */
32    printf("Unesite redni broj funkcije (1 ili 2):\n");
      scanf("%d", &ind);
34
      /* Ucitava se broj elemenata niza */
36    printf("Unesite dimenziju niza:");
      scanf("%d", &n);
38
      /* Ucitava se n elemenata niza. */
40    printf("Unesite elemente niza:");
      for (i = 0; i < n; i++)
42      scanf("%d", &a[i]);

44    /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
         funkcije sumaNiza, ondosno sumaNiza2 */
46    if (ind == 1)
        printf("Suma elemenata je %d\n", sumaNiza(a, n));
48    else
        printf("Suma elemenata je %d\n", sumaNiza2(a, n));
50
      return 0;
52  }
```

Rešenje 1.23

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

4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
     dimenzije n */
6 int maksimum_niza(int niz[], int n)
  {
8   /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveci je
       ujedno i jedini element niza */
10  if (n == 1)
      return niz[0];
12
    /* Resavanje problema manje dimenzije */
14  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. U niz se ne moze
     ucitati vise od MAX_DIM brojeva, pa se u slucaju da promenljiva
     i dostigne vrednost MAX_DIM prekida unos novih brojeva. */
  int i = 0;
  while (scanf("%d", &brojevi[i]) != EOF) {
    i++;
    if (i == MAX_DIM)
      break;
  }
  n = i;

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


  return 0;
}
```

Rešenje 1.24

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

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

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

```c

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

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

  /* Provera da li je dimenzija niza odgovarajuca */
  if (n < 0 || n > MAX_DIM) {
    printf("Dimenzija mora biti prirodan broj <= %d!\n", MAX_DIM);
    return 0;
  }

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

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

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

  return 0;
}
```

Rešenje 1.25

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

/* Funkcija koja racuna broj pojavljivanja elementa x u nizu a duzine
   n */
int br_pojave(int x, int a[], int n)
{
  /* Izlazak iz rekurzije: za niz duzine jedan broj pojava broja x u
     nizu je 1 ukoliko je jedini element a[0] bas x ili 0 inace */
  if (n == 1)
    return a[0] == x ? 1 : 0;

  /* U promenljivu bp se smesta broj pojave broja x u prvih n-1
     elemenata niza a. Ukupan broj pojavljivanja broja x u celom nizu
     a je jednak bp uvecanom za jedan ukoliko je se na poziciji n-1 u
     nizu a nalazi broj x */
  int bp = br_pojave(x, a, n - 1);
  return a[n - 1] == x ? 1 + bp : bp;
}
```

```c

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

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

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

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

  return 0;
}
```

Rešenje 1.26

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

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

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

```c
int main()
{
  int x, y, z, a[MAX_DIM];
  int n;

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

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

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

  return 0;
}
```

Rešenje 1.27

```c
#include <stdio.h>

/* Funkcija koja broji bitove postavljene na 1. */
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 bita na poziciji najvece tezine i na osnovu toga se
   razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
   broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
   x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
   na poziciji najvece tezine jedinica, rezultat dobijen
```

```
   rekurzivnim pozivom funkcije za x<<1 treba uvecati za jedan.
     Za rekurzivni poziv se salje vrednost koja se dobija kada se x
     pomeri u levo. Napomena: argument funkcije x je oznacen ceo
     broj, usled cega se ne koristi pomeranje udesno, jer funkciji
     moze biti prosledjen i negativan broj. Iz tog razloga,
     odlucujemo se da proveramo najvisi, umesto najnizeg bita */
  if (x & (1 << (sizeof(x) * 8 - 1)))
    return 1 + count(x << 1);
  else
    return count(x << 1);
  /*****************************************************************
    Krace zapisano
    return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
  *****************************************************************/
}

int main()
{
  int x;

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

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

  return 0;
}
```

Rešenje 1.29

```
#include <stdio.h>

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

  /* Odredjivanje poslednje oktalne cifre u broju */
  int poslednja_cifra = x & 7;

  /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
     izbrise poslednja oktalna cifra */
  int max_bez_poslednje_cifre = max_oktalna_cifra(x >> 3);

  return poslednja_cifra > max_bez_poslednje_cifre ? poslednja_cifra
      : max_bez_poslednje_cifre;
```

```
}

int main()
{
  unsigned x;

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

  /* Ispisuje se vrednost najvece oktalne cifre unetog broja */
  printf("%d\n", max_oktalna_cifra(x));

  return 0;
}
```

Rešenje 1.30

```
#include <stdio.h>


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

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

  /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
     njega izbrise poslednja heksadekadna cifra */
  int 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;

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

  /* Ispisuje se vrednost najvece heksadekadne cifre unetog broja */
  printf("%d\n", max_heksadekadna_cifra(x));

  return 0;
```

```
}
```

Rešenje 1.31

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

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

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

  /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
     poslednji karakter i da je palindrom niska koja nastaje kada se
     polaznoj nisci otklone prvi i poslednji karakter */
  return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
}

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

  /* Ucitavanje niske sa standardnog ulaza */
  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

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

```
4
  /* Funkcija koja ispisuje elemente niza a duzine n */
6 void ispisiNiz(int a[], int n)
  {
8   int i;

10  for (i = 1; i <= n; i++)
      printf("%d ", a[i]);
12  printf("\n");
  }
14
  /* Funkcija koja proverava da li se broj x nalazi u nizu a duzine n
       */
16 int koriscen(int a[], int n, int x)
  {
18  int i;

20  /* Obilaze se svi elementi niza */
    for (i = 1; i <= n; i++)
22    /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
      if (a[i] == x)
24      return 1;

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

30 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija
     kao argument niz a[] u koji se smesta permutacija, broj m oznacava
32   da se u okviru tog poziva funkcije na m-tu poziciju u permutaciji
     smesta jedan od preostalih celih brojeva, n je velicina skupa koji
34   se permutuje. Funkciju se inicijalno poziva sa argumentom m = 1,
     jer formiranje permutacije pocinje od pozicije broj 1. Stoga, a[0]
36   se ne koristi. */
  void permutacija(int a[], int m, int n)
38 {
    int i;
40
    /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
42     premasila velicinu skupa, onda se svi brojevi vec nalaze u
       permutaciji i ispisuje se permutacija. */
44  if (m > n) {
      ispisiNiz(a, n);
46    return;
    }
48
    /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
50     mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
       Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
52     ovako postavljenom pocetku permutacije. Kada se to zavrsi, vrsi
       se provera da li postoji jos neki broj koji moze da se stavi na
54     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,
     ali sada sa drugacije postavljenim prefiksom. */
  for (i = 1; i <= n; i++) {
    /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
       pozicije, onda se on postavlja na poziciju m i poziva se
       ponovo funkcija da dopuni ostatak permutacije posle upisivanja
       i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
       se nije pojavio do sada u permutaciji. */
    if (!koriscen(a, m - 1, i)) {
      a[m] = i;
      permutacija(a, m + 1, n);
    }
  }
}

int main(void)
{
  int n;
  int a[MAX_DUZINA_PERMUTACIJE+1];

  /* Ucitava se broja n i provera se da li je u odgovarajucem opsegu
     */
  scanf("%d", &n);
  if (n < 0 || n > MAX_DUZINA_PERMUTACIJE) {
    fprintf(stderr,
            "Duzina permutacije mora biti broj iz intervala [0, %d]!\
    n",
            MAX_DUZINA_PERMUTACIJE);
    exit(EXIT_FAILURE);
  }

  /* Ispisuju se permutacije duzine n */
  permutacija(a, 1, n);

  exit(EXIT_SUCCESS);
}
```

Rešenje 1.33

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

/* Rekurzivna funkcija za racunanje binomnog koeficijenta */
int binomniKoeficijent(int n, int k)
{
  /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0.
     Ukoliko je k strogo izmedju 0 i n, onda se koristi formula
     bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
     koja se moze izvesti iz definicije binomnog koeficijenata */
  return (0 < k && k < n) ?
```

```
12        binomniKoeficijent(n - 1, k - 1) + binomniKoeficijent(n - 1,
                                                                k) : 1;
14  }

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

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

24      return b;
      }
26
      Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
28    resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
  *****************************************************************/
30
  /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
32     zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
       ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
34     veca od sume elemenata prethodne hipotenuze. */
  int sumaElemenataHipotenuze(int n)
36{
    return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
38}

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

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

```
64    /* Ispisuje se suma elemenata hipotenuze */
      printf("%d\n", sumaElemenataHipotenuze(r));
66
      exit(EXIT_SUCCESS);
68  }
```

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

Primer 2

```
Interakcija sa programom:
 Unesite dimenziju niza:  0
 Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.1]

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

(a) funkciju `zbir` koja izračunava zbir elemenata niza,

(b) funkciju `proizvod` koja izračunava proizvod elemenata niza,

(c) funkciju `min_element` koja izračunava najmanji elemenat niza,

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

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

Primer 1

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

Primer 2

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

[Rešenje 2.2]

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju niza:  5
 Unesite elemente niza:
 1 2 3 4 5
 Transformisan niz je:
 2 3 3 3 4
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju niza:  4
 Unesite elemente niza:
 4 -3 2 -1
 Transformisan niz je:
 5 -2 1 -2
```

Primer 3

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju niza:  0
 Greska: neodgovarajuca dimenzija niza.
```

Primer 4

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju niza:  101
 Greska: neodgovarajuca dimenzija niza.
```

[Rešenje 2.3]

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Primer 1

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

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

Primer 2

```
Poziv: ./a.out

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

[Rešenje 2.4]

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

Primer 1

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

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

Primer 2

```
Poziv: ./a.out

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

[Rešenje 2.5]

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

Primer 1

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

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

INTERAKCIJA SA PROGRAMOM:
 Broj reci ciji je broj karaktera 1 je 3.
```

Primer 2

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

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

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

[Rešenje 2.6]

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

Primer 1

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

ULAZ.TXT
 Ovo je sadrzaj datoteke i u njoj ima reci
 koje se zavrsavaju na ke

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

Primer 2

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

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

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

Primer 3

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

Datoteka ulaz.txt ne postoji

Interakcija sa programom:
 Greska: Neuspesno otvaranje
 datoteke ulaz.txt.
```

Primer 4

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

ulaz.txt
 Ovo je sadrzaj ulaza.

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

[Rešenje 2.7]

2.2 Višedimenzioni nizovi

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

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

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

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

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

Primer 1

```
Interakcija sa programom:
 Unesite broj vrsta matrice:  3
 Unesite elemente matrice, 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 sa programom:
 Unesite dimenziju matrice:  0
 Greska: neodgovarajuca dimenzija matrice.
```

[Rešenje 2.8]

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

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

(b) Napisati funkciju koja izračunava zbir matrica.

(c) Napisati funkciju koja izračunava proizvod matrica.

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrica:  3
Unesite elemente prve matrice, 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
```

[Rešenje 2.9]

Zadatak 2.10 Relacija se može predstaviti kvadratnom matricom nula i jedinica na sledeći način: element i je u relaciji sa elementom j ukoliko se u preseku i-te vrste i j-te kolone matrice nalazi broj 1, a nije u relaciji ukoliko se tu nalazi broj 0.

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

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

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

(d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja je nadskup date).

(e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja je nadskup date).

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

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

Primer 1

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

[Rešenje 2.10]

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

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

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

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice čiji se broj vrsta n ($0 < n \leq 32$) zadaje kao argument komandne linije. Na standardni izlaz ispisati rezultat primene prethodno napisanih funkcija.

Primer 1

```
Poziv: ./a.out 3

Interakcija sa programom:
 Unesite elemente matrice dimenzije 3x3:
 1 2 3
 -4 -5 -6
 7 8 9
 Najveci element sporedne dijagonale je 7.
 Indeks kolone sa najmanjim elementom je 2.
 Indeks vrste sa najvecim elementom je 2.
 Broj negativnih elemenata matrice je 3.
```

Primer 2

```
Poziv: ./a.out 4

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

[Rešenje 2.11]

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

Primer 1

```
Interakcija sa programom:
 Unesite broj vrsta matrice:  4
 Unesite elemente matrice, 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 sa programom:
 Unesite broj vrsta matrice:  3
 Unesite elemente matrice, vrstu po vrstu:
 1 2 3
 5 6 7
 1 4 2
 Matrica nije ortonormirana.
```

[Rešenje 2.12]

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

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

(b) Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice, u smeru kretanja kazaljke na satu.

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite broj vrsta i broj kolona matrice:
 3 3
 Unesite elemente matrice, 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 SA PROGRAMOM:
 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
```

[Rešenje 2.13]

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite broj vrsta kvadratne matrice:  3
 Unesite elemente matrice, 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 SA PROGRAMOM:
 Unesite dimenziju niza:  3
 Unesite elemente niza:
 1 -2 3
 Niz u obrnutom poretku je: 3 -2 1
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju niza:  -1
 malloc(): neuspela alokacija memorije.
```

[Rešenje 2.15]

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

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

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

Od korisnika sa ulaza tražiti da izabere način realokacije memorije.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite zeljeni nacin realokacije (M ili R):
 M
 Unesite brojeve, nulu za kraj:
 1 -2 3 -4 0
 Niz u obrnutom poretku je: -4 3 -2 1
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite zeljeni nacin realokacije (M ili R):
 R
 Unesite brojeve, nulu za kraj:
 6 -1 5 -2 4 -3 0
 Niz u obrnutom poretku je: -3 4 -2 5 -1 6
```

[Rešenje 2.16]

Zadatak 2.17 Napisati funkciju koja kao rezultat vraća nisku koja se dobija nadovezivanjem dve niske, bez promene njihovog sadržaja. Napisati program koji testira rad napisane funkcije. Sa standardnog ulaza učitati dve niske karaktera (pretpostaviti da niske nisu duže od 50 karaktera i da ne sadrže praznine). Na standardni izlaz ispisati nisku koja se dobija njihovim nadovezivanjem. Za rezultujuću nisku dinamički alocirati memoriju.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dve niske karaktera:
 Jedan Dva
 Nadovezane niske: JedanDva
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dve niske karaktera:
 Ana Marija
 Nadovezane niske: AnaMarija
```

[Rešenje 2.17]

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

Primer 1

```
Interakcija sa programom:
 Unesite broj vrsta i broj kolona 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.
```

Primer 2

```
Interakcija sa programom:
 Unesite broj vrsta i broj kolona matrice:
 2 2
 Unesite elemente matrice, vrstu po vrstu:
 -0.1 -0.2
 -0.3 -0.4
 Trag unete matrice je -0.50.
```

[Rešenje 2.18]

Zadatak 2.19 Napisati biblioteku za rad sa celobrojnim matricama.

(a) Napisati funkciju `int **alociraj_matricu(int n, int m)` koja dinamički alocira memoriju potrebnu za matricu dimenzija $n \times m$.

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

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

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

(e) Napisati funkciju `void ucitaj_kvadratnu_matricu(int **A, int n)` koja učitava već alociranu kvadratnu matricu dimenzije $n \times n$ sa standardnog ulaza.

(f) Napisati funkciju `void ispisi_matricu(int **A, int n, int m)` koja ispisuje matricu dimenzija $n \times m$ na standardnom izlazu.

(g) Napisati funkciju `void ispisi_kvadratnu_matricu(int **A, int n)` koja ispisuje kvadratnu matricu dimenzije $n \times n$ na standardnom izlazu.

(h) Napisati funkciju `int ucitaj_matricu_iz_datoteke(int **A, int n, int m, FILE * f)` koja učitava već alociranu matricu dimenzija $n \times m$ iz već otvorene datoteke f. U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.

(i) Napisati funkciju `int ucitaj_kvadratnu_matricu_iz_datoteke(int **A, int n, FILE * f)` koja učitava već alociranu kvadratnu matricu dimenzije $n \times n$ iz već otvorene datoteke f. U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.

(j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzija $n \times m$ u već otvorenu datoteku f. U slučaju neuspešnog upisivanja vratiti vrednost različitu od 0.

(k) Napisati funkciju `int upisi_kvadratnu_matricu_u_datoteku(int **A, int n, FILE * f)` koja upisuje kvadratnu matricu dimenzije $n \times n$ u već otvorenu datoteku f. U slučaju neuspešnog upisivanja vratiti vrednost različitu od 0.

Napisati programe koji testiraju napisanu biblioteku.

(1) Program učitava dimenziju nekvadratne matrice sa standardnog ulaza, a zatim i samu matricu. Potom, matricu upisati u datoteku *matrica.txt*.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesi broj vrsta matrice: 3
 Unesi broj kolona matrice: 4
 Unesi elemente matrice po vrstama:
 1 2 3 4
 5 6 7 8
 9 10 11 12

MATRICA.TXT
 1 2 3 4
 5 6 7 8
 9 10 11 12
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesi broj vrsta matrice: 5
 Unesi broj kolona matrice: 0
 Neodgovarajce dimenzije matrice
```

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

Test 1

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

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

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

Test 2

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

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

Izlaz:
  Neispravan pocetak fajla
```

Test 3

```
Poziv: ./a.out

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

[Rešenje 2.19]

Zadatak 2.20 Data je celobrojna matrica dimenzije $n \times m$. Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu). Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati n i m (ne praviti nikakve pretpostavke o njihovoj veličini), zatim učitati elemente matrice i na standardni izlaz ispisati elemente ispod glavne dijagonale matrice. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

Primer 1

```
Interakcija sa programom:
 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
```

[Rešenje 2.20]

Zadatak 2.21 Za zadatu matricu dimenzije $n \times m$ napisati funkciju koja izračunava redni broj kolone matrice čiji je zbir maksimalan. Napisati program koji testira ovu funkciju. Sa standardnog ulaza učitati dimenzije matrice n i m (ne praviti nikakve pretpostavke o njihovoj veličini), a zatim elemente matrice. Na standardni izlaz ispisati redni broj kolone matrice sa maksimalnim zbirom. Ukoliko ima više takvih, ispisati prvu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

Primer 1

```
Interakcija sa programom:
 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 sa programom:
 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 2.22 Data je realna kvadratna matrica dimenzije $n \times n$.

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

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

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

Primer 1

```
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 sa programom:
 Zbir apsolutnih vrednosti ispod
 sporedne dijagonale je 25.30.
 Transformisana matrica je:
 1.10 -1.10 1.65
 -8.80 5.50 -3.30
 15.40 -17.60 9.90
```

[Rešenje 2.22]

Zadatak 2.23 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 sa programom:
 1.1 -2.2 3.3
 -1.1 2.2 -3.3
 -4.4 5.5 -6.6
 4.4 -5.5 6.6
 7.7 -8.8 9.9
 -7.7 8.8 -9.9
```

Zadatak 2.24 Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Rezultujuću matricu ispisati na standardni izlaz. Napomena: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

Primer 1

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

Primer 2

```
Interakcija sa programom:
 Unesite elemente niza, nulu za kraj:
 -5 -2 -4 -1 0
 Trazena matrica je:
 -5 -2 -4 -1
 -2 -4 -1 -5
 -4 -1 -5 -2
 -1 -5 -2 -4
```

Zadatak 2.25 Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci `slicice.txt` se nalaze informacije o sličicama koje mu nedostaju u formatu:

`redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`

Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronađe ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. Napomena: *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 sa programom:
 Petru ukupno nedostaje 7 slicica.
 Reprezentacija za koju je sakupio
 najmanji broj slicica je Brazil.
```

**** Zadatak 2.26** U datoteci `temena.txt` se nalaze tačke koje predstavljaju temena nekog n-tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom n-touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan.

Primer 1

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

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

Primer 2

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

```
INTERAKCIJA SA PROGRAMOM:
 U datoteci su zadata temena petougla.
 Obim je 18.80.
 Povrsina je 22.59.
```

2.4 Pokazivači na funkcije

Zadatak 2.27 Napisati program koji tabelarno štampa vrednosti proizvoljne realne funkcije sa jednim realnim argumentomo, dnosno izračunava i ispisuje vrednosti date funkcije u n ekvidistantnih tačaka na intervalu $[a, b]$. 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 SA PROGRAMOM:
 Unesite krajeve intervala:
 -0.5 1
 Koliko tacaka ima na ekvidistantnoj
 mrezi (ukljucujuci krajeve intervala)?
 4
  x sin(x)
 ----------------------
 | -0.50000 | -0.47943 |
 | 0.00000 | 0.00000 |
 | 0.50000 | 0.47943 |
 | 1.00000 | 0.84147 |
 ----------------------
```

Primer 2

```
POZIV: ./a.out cos

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

[Rešenje 2.27]

Zadatak 2.28 Napisati funkciju koja izračunava limes funkcije `f(x)` u tački a. Adresa funkcije `f` čiji se limes računa se prenosi kao parametar funkciji za računanje limesa. Limes se računa sledećom aproksimacijom (vrednosti n i a uneti sa standardnog ulaza kao i ime funkcije):

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite ime funkcije, n i a:
 tan 10000 1.570795
 Limes funkcije tan je -10134.46.
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite ime funkcije, n i a:
 cos 5000 0.25
 Limes funkcije cos je 0.97.
```

Zadatak 2.29 Napisati funkciju koja određuje integral funkcije `f(x)` na intervalu $[a, b]$. Adresa funkcije `f` se prenosi kao parametar. Integral se računa prema formuli:

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

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite ime funkcije, n, a i b:
 sin 10000 -5.2 2.1
 Vrednost integrala je 0.973993.
```

Zadatak 2.30 Napisati funkciju koja približno izračunava integral funkcije `f(x)` na intervalu $[a, b]$. Funkcija `f` se prosleđuje kao parametar, a integral se procenjuje po Simpsonovoj formuli:

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

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

Primer 1

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

Primer 2

```
Interakcija sa programom:
 Unesite ime funkcije, n, a i b:
 tan 5000 -4.1 -2.3
 Vrednost integrala je -0.147640.
```

2.5 Rešenja

Rešenje 2.1

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

#define MAX 100

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

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

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

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

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

    /* Vrednost promenljive "t" se postavlja na adresu na koju
       pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
       umanjuje za jedan, sto za posledicu ima da pokazivac
```

```
        "poslednji" sada pokazuje na element koji mu prethodi u nizu
     */
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 }

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

79 obrni_niz_v1(a, n);

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

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

87 obrni_niz_v2(a, n);
```

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

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

95    exit(EXIT_SUCCESS);
}
```

Rešenje 2.2

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

4 #define MAX 100

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

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

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

22  for (; n; n--)
      p *= (*(a + n - 1));
24
    return p;
26 }

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

```
40
  return min;
42 }

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

50   /* Ispituje se da li se medju ostalim elementima niza nalazi
       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 */
80   printf("Zbir elemenata niza je %5.3f.\n", zbir(a, n));
  printf("Proizvod elemenata niza je %5.3f.\n", proizvod(a, n));
82   printf("Minimalni element niza je %5.3f.\n", min(a, n));
  printf("Maksimalni element niza je %5.3f.\n", max(a, n));
84

  exit(EXIT_SUCCESS);
86 }
```

Rešenje 2.3

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

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

  while (prvi < poslednji) {

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

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

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

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

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

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

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

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

```
  }

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

  povecaj_smanji(a, n);

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.4

```
#include <stdio.h>

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

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

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

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

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

```
    argc = i;
  }

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

  return 0;
}
```

Rešenje 2.5

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

/* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
   cita spreda i odpozadi */
int palindrom(char *niska)
{
  int i, j;
  for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
    if (*(niska + i) != *(niska + j))
      return 0;
  return 1;
}

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

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

  printf
      ("Broj argumenata komandne linije koji su palindromi je %d.\n",
       n);
  return 0;
}
```

Rešenje 2.6

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

#define MAX_KARAKTERA 100

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

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

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

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

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

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

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

  /* Zatvara se datoteka */
  fclose(in);
```

```
50
  exit(EXIT_SUCCESS);
52 }
```

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  int duzina_sufiksa = duzina_niske(sufiks);
  int duzina_niske_pom = duzina_niske(niska);
38  if (duzina_sufiksa <= duzina_niske_pom &&
      poredjenje_niski(niska + duzina_niske_pom -
40                       duzina_sufiksa, sufiks) == 0)
    return 1;
42  return 0;
}
44

/* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
```

```
46    prefiks niske zadate prvi argumentom funkcije */
int prefiks_niske(char *niska, char *prefiks)
48 {
  int i;
50   int duzina_prefiksa = duzina_niske(prefiks);
  int duzina_niske_pom = duzina_niske(niska);
52   if (duzina_prefiksa <= duzina_niske_pom) {
    for (i = 0; i < duzina_prefiksa; i++)
54       if (*(prefiks + i) != *(niska + i))
        return 0;
56     return 1;
  } else
58     return 0;
}
60
int main(int argc, char **argv)
62 {
  /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
64     greska */
  if (argc < 4) {
66     printf("Greska: ");
    printf("Nedovoljan broj argumenata komandne linije.\n");
68     printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
           argv[0]);
70     exit(EXIT_FAILURE);
  }
72
  FILE *in;
74   int br = 0;
  char rec[MAX_KARAKTERA+1];
76
  in = fopen(*(argv + 1), "r");
78   if (in == NULL) {
    fprintf(stderr, "Greska: ");
80     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
    exit(EXIT_FAILURE);
82   }

84   /* Provera se opcija kojom je pozvan program a zatim se ucitavaju
     reci iz datoteke i broji se koliko njih zadovoljava trazeni
86     uslov */
  if (!(poredjenje_niski(*(argv + 3), "-s"))) {
88     while (fscanf(in, "%s", rec) != EOF)
      br += sufiks_niske(rec, *(argv + 2));
90     printf("Broj reci koje se zavrsavaju na %s je %d.\n", *(argv + 2)
    ,
           br);
92   } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
    while (fscanf(in, "%s", rec) != EOF)
94       br += prefiks_niske(rec, *(argv + 2));
    printf("Broj reci koje pocinju na %s je %d.\n", *(argv + 2), br);
96   }
```

```
98    fclose(in);

100   exit(EXIT_SUCCESS);
}
```

Rešenje 2.8

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

#define MAX 100

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

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

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

  return sqrt(norma);
}

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

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

  return norma;
}

int main()
```

```
{
  int A[MAX][MAX];
  int i, j, n;

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

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

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

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

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.9

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

#define MAX 100

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

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

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

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

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

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

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

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

```
51    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
53        c[i][j] = a[i][j] + b[i][j];
  }
55
  /* Funkcija izracunava proizvod dve kvadratne matice */
57 void pomnozi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
  {
59    int i, j, k;

61    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
63        /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
          c[i][j] = 0;
65        for (k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
67      }
  }
69
  int main()
71 {
    /* Matrice ciji se elementi zadaju sa ulaza */
73  int a[MAX][MAX], b[MAX][MAX];

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

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

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

```
103   else
        printf("Matrice nisu jednake.\n");
105
      printf("Zbir matrica je:\n");
107   ispisi_matricu(zbir, n);

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

Rešenje 2.10

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

4 #define MAX 64

6 /* Funkcija proverava da li je relacija refleksivna. Relacija je
     refleksivna ako je svaki element u relaciji sa sobom, odnosno ako
8    se u matrici relacije na glavnoj dijagonali nalaze jedinice */
  int refleksivnost(int m[][MAX], int n)
10 {
    int i;
12
    for (i = 0; i < n; i++) {
14    if (m[i][i] != 1)
        return 0;
16  }

18  return 1;
  }
20
  /* Funkcija odredjuje refleksivno zatvorenje zadate relacije. Ono je
22   odredjeno matricom koja sadrzi sve elemente polazne matrice
     dopunjene jedinicama na glavnoj dijagonali */
24 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
  {
26  int i, j;

28  /* Prepisuju se vrednosti elemenata pocetne matrice */
    for (i = 0; i < n; i++)
30    for (j = 0; j < n; j++)
        zatvorenje[i][j] = m[i][j];
32
    /* Na glavnoj dijagonali se postavljaju jedinice */
34  for (i = 0; i < n; i++)
      zatvorenje[i][i] = 1;
36 }
```

```
38 /* Funkcija proverava da li je relacija simetricna. Relacija je
      simetricna ako za svaki par elemenata vazi: ako je element "i" u
40    relaciji sa elementom "j", onda je i element "j" u relaciji sa
      elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
42    dijagonalu */
   int simetricnost(int m[][MAX], int n)
44 {
     int i, j;
46
     /* Obilaze se elementi ispod glavne dijagonale matrice i uporedjuju
48      se sa njima simetricnim elementima */
     for (i = 0; i < n; i++)
50     for (j = 0; j < i; j++)
         if (m[i][j] != m[j][i])
52         return 0;

54   return 1;
   }
56
   /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono je
58    odredjeno matricom koja sadrzi sve elemente polazne matrice
      dopunjene tako da matrica postane simetricna u odnosu na glavnu
60    dijagonalu */
   void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
62 {
     int i, j;
64
     for (i = 0; i < n; i++)
66     for (j = 0; j < n; j++)
         zatvorenje[i][j] = m[i][j];
68
     for (i = 0; i < n; i++)
70     for (j = 0; j < n; j++)
         if (zatvorenje[i][j] == 1)
72         zatvorenje[j][i] = 1;
   }
74


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++)
128       printf("%d ", m[i][j]);
        printf("\n");
130   }
}
132
  int main(int argc, char *argv[])
134 {
      FILE *ulaz;
136   int m[MAX][MAX];
      int pomocna[MAX][MAX];
138   int n, i, j;

140   /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
        */
```

```
  if (argc < 2) {
142   printf("Greska: ");
    printf("Nedovoljan broj argumenata komandne linije.\n");
144   printf("Program se poziva sa %s ime_dat.\n", argv[0]);
    exit(EXIT_FAILURE);
146 }

148 /* Otvara se datoteka za citanje */
  ulaz = fopen(argv[1], "r");
150 if (ulaz == NULL) {
    fprintf(stderr, "Greska: ");
152   fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
    exit(EXIT_FAILURE);
154 }

156 /* Ucitava se dimenzija matrice */
  fscanf(ulaz, "%d", &n);
158
  /* Proverava se da li je doslo do prekoracenja dimenzije */
160 if (n > MAX || n <= 0) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
162   fprintf(stderr, "matrice.\n");
    exit(EXIT_FAILURE);
164 }

166 /* Ucitava se element po element matrice */
  for (i = 0; i < n; i++)
168   for (j = 0; j < n; j++)
      fscanf(ulaz, "%d", &m[i][j]);
170
  /* Ispisuje se rezultat */
172 printf("Relacija %s refleksivna.\n",
         refleksivnost(m, n) == 1 ? "jeste" : "nije");
174
  printf("Relacija %s simetricna.\n",
176        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 */
  fclose(ulaz);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.11

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

#define MAX 32

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

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

  return max_na_sporednoj_dijagonali;
}

/* Funkcija izracunava indeks kolone najmanjeg elementa */
int indeks_min(int m[][MAX], int n)
{
  int i, j;
  int min = m[0][0], indeks_kolone = 0;

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

```
44      for (j = 0; j < n; j++)
          if (m[i][j] > max) {
46          max = m[i][j];
            indeks_vrste = i;
48        }
    return indeks_vrste;
50  }

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

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

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

72  /* Proverava se broj argumenata komandne linije */
    if (argc < 2) {
74    printf("Greska: ");
      printf("Nedovoljan broj argumenata komandne linije.\n");
76    printf("Program se poziva sa %s br_vrsta_mat.\n", argv[0]);
      exit(EXIT_FAILURE);
78  }

80  /* Ucitava se broj vrsta matrice */
    n = atoi(argv[1]);
82
    if (n > MAX || n <= 0) {
84    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
      fprintf(stderr, "matrice.\n");
86    exit(EXIT_FAILURE);
    }
88
    /* Ucitava se matrica */
90  printf("Unesite elemente matrice dimenzije %dx%d:\n", n, n);
    for (i = 0; i < n; i++)
92    for (j = 0; j < n; j++)
        scanf("%d", &m[i][j]);
94
    printf("Najveci element sporedne dijagonale je %d.\n",
```

```
96          max_sporedna_dijagonala(m, n));

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

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

Rešenje 2.12

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

11    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
13        scanf("%d", &m[i][j]);
    }
15
    /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
        */
17  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)
```

```
{
  int i, j, k;
  int proizvod;

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

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

    if (proizvod != 1)
      return 0;
  }

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

      proizvod = 0;

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

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

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

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

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

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

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

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

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

Rešenje 2.13

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

#define MAX_V 10
#define MAX_K 10

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

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

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

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

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

    /* Spusta se prvi red */
    top++;

    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');
}

/* Funkcija ucitava matricu */
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:\n");
  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);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.15

```c
#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);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.16

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

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

  /* Velicina alocirane memorije */
  int alocirano;

  /* Broj elemenata niza */
```

```
14  int n;

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

22  /* Inicijalizacija */
    alocirano = n = 0;
24
    printf("Unesite zeljeni nacin realokacije (M ili R):\n");
26  scanf("%c", &realokacija);

28  printf("Unesite brojeve, nulu za kraj:\n");
    scanf("%d", &x);
30
    while (x != 0) {
32    if (n == alocirano) {
        alocirano = alocirano + KORAK;
34
        if (realokacija == 'M') {
36        /* Vrsi se realokacija memorije sa novom velicinom */
          b = (int *) malloc(alocirano * sizeof(int));
38
          if (b == NULL) {
40          fprintf(stderr, "malloc(): ");
            fprintf(stderr, "greska pri alokaciji memorije.\n");
42          free(a);
            exit(EXIT_FAILURE);
44        }

46        /* Svih n elemenata koji pocinju na adresi a prepisujemo na
             novu aderesu b */
48        for (i = 0; i < n; i++)
            b[i] = a[i];
50
          free(a);
52
          /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
54           bloka koji je prilikom alokacije zapamcen u promenljivoj b
           */
56        a = b;
        } else if (realokacija == 'R') {
58
          /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
60           "a" bude inicijalizovano na NULL */

62        a = (int *) realloc(a, alocirano * sizeof(int));
          if (a == NULL) {
64          fprintf(stderr, "realloc(): ");
            fprintf(stderr, "greska pri alokaciji memorije.\n");
```

```
66            exit(EXIT_FAILURE);
            }
68        }
      }

      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) {
    fprintf(stderr, "malloc(): ");
    fprintf(stderr, "greska pri alokaciji memorije.\n");
    exit(EXIT_FAILURE);
  }

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

  return p;
}
```

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

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.18

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

int main()
{
  int i, j;

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

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

  /* Trag matice */
  double trag = 0;

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

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

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

```
29    }
31    /* Dinamicki se alocira prostor za elemente u vrstama */
      for (i = 0; i < n; i++) {
33      A[i] = (double **)malloc(sizeof(double) * m);
35      /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
           potrebno je osloboditi svih i-1 prethodno alociranih vrsta, i
37         alociran niz pokazivaca */
        if (A[i] == NULL) {
39        for (j = 0; j < i; j++)
            free(A[j]);
41        free(A);
          exit(EXIT_FAILURE);
43      }
      }
45
      printf("Unesite elemente matrice, vrstu po vrstu:\n");
47    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
49        scanf("%lf", &A[i][j]);
51    /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
         dijagonali */
53    trag = 0.0;
55    for (i = 0; i < n; i++)
        trag += A[i][i];
57
      printf("Trag unete matrice je %.2f.\n", trag);
59
      /* Oslobadja se prostor rezervisan za svaku vrstu */
61    for (j = 0; j < n; j++)
        free(A[j]);
63
      /* Oslobadja se memorija za niz pokazivaca na vrste */
65    free(A);
67    exit(EXIT_SUCCESS);
  }
```

Rešenje 2.19

Datoteka 2.1: *matrica.h*

```
1  #ifndef _MATRICA_H_
   #define _MATRICA_H_ 1
3
   /* Funkcija dinamicki alocira memoriju za matricu dimenzija n x m */
5  int **alociraj_matricu(int n, int m);
```

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

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

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

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

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

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

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

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

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

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

#endif
```

Datoteka 2.2: *matrica.c*

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

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

```
  int i, j;

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

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

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

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

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

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

```

```
void ucitaj_kvadratnu_matricu(int **matrica, int n)
{
  /* Ucitavanje matrice n x n */
  ucitaj_matricu(matrica, n, n);
}

void ispisi_matricu(int **matrica, int n, int m)
{
  int i, j;
  /* Ispis po vrstama */
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
      printf("%d ", matrica[i][j]);
    printf("\n");
  }
}

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

int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
{
  int i, j;
  /* Elementi matrice se ucitacaju po vrstama */
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      /* Ako je nemoguce ucitati sledeci element, povratna vrednost
         funkcije je 1, kao indikator neuspesnog ucitavanja */
      if (fscanf(f, "%d", &matrica[i][j]) != 1)
        return 1;

  /* Uspesno ucitana matrica */
  return 0;
}

int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
                                         FILE * f)
{
  /* Ucitavanje matrice n x n iz datoteke */
  return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
}

int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
{
  int i, j;
  /* Ispis po vrstama */
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
      /* Ako je nemoguce ispisati sledeci element, povratna vrednost
```

```
            funkcije je 1, kao indikator neuspesnog ispisa */
        if (fprintf(f, "%d ", matrica[i][j]) <= 0)
          return 1;
      fprintf(f, "\n");
    }

    /* Uspesno upisana matrica */
    return 0;
}

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

Datoteka 2.3: *main_a.c*

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

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

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

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

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

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

```
34    /* Otvaranje fajla za upis matrice */
      if ((f = fopen("matrica.txt", "w")) == NULL) {
36      fprintf(stderr, "fopen() error\n");
        matrica = dealociraj_matricu(matrica, n);
38      exit(EXIT_FAILURE);
      }
40
      /* Upis matrice u fajl */
42    if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
        fprintf(stderr, "Neuspesno upisivanje matrice u datoteku\n");
44      matrica = dealociraj_matricu(matrica, n);
        exit(EXIT_FAILURE);
46    }

48    /* Zatvaranje fajla */
      fclose(f);
50
      /* Dealokacija matrice */
52    matrica = dealociraj_matricu(matrica, n);

54    exit(EXIT_SUCCESS);
  }
```

Datoteka 2.4: *main_b.c*

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

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

11    /* Provera argumenata komandne linije */
      if (argc != 2) {
13      fprintf(stderr, "Koriscenje programa: %s datoteka\n", argv[0]);
        exit(EXIT_FAILURE);
15    }

17    /* Otvaranje fajla za citanje */
      if ((f = fopen(argv[1], "r")) == NULL) {
19      fprintf(stderr, "fopen() error\n");
        exit(EXIT_FAILURE);
21    }

23    /* Ucitavanje dimenzije matrice */
      if (fscanf(f, "%d", &n) != 1) {
25      fprintf(stderr, "Neispravan pocetak fajla\n");
```

```
    exit(EXIT_FAILURE);
  }

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

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

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

  /* Zatvaranje fajla */
  fclose(f);

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

  /* Dealokacija matrice */
  matrica = dealociraj_matricu(matrica, n);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.20

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

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

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

```
  }
}

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

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

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

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

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

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.22

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

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

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

/* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
```

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

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

  return zbir;
}

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

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

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

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

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

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

```

```
  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}

96int 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 */
130   izmeni(a, n);

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

136   printf("Transformisana matrica je:\n");
      ispisi_matricu(a, n);
138
      /* Oslobadja se memorija */
140   oslobodi_memoriju(a, n);

142   /* Zatvara se datoteka */
      fclose(ulaz);
144
      exit(EXIT_SUCCESS);
146 }
```

Rešenje 2.27

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

7 /* Funkcija tabela() prihvata granice intervala a i b, broj
     ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
9    funkciju koja prihvata double argument, i vraca double vrednost.
     Za tako datu funkciju ispisuju se njene vrednosti u intervalu
11   [a,b] u n ekvidistantnih tacaka intervala */
  void tabela(double a, double b, int n, double (*fp) (double))
13 {
    int i;
15  double x;

17  printf("-----------------------\n");
    for (i = 0; i < n; i++) {
19    x = a + i * (b - a) / (n - 1);
      printf("| %8.5f | %8.5f |\n", x, (*fp) (x));
21  }
    printf("-----------------------\n");
23 }
```

```
25 double sqr(double a)
   {
27   return a * a;
   }
29
   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 {
83      printf("Program jos uvek ne podrzava trazenu funkciju!\n");
        exit(EXIT_SUCCESS);
85  }

87  printf("Unesite krajeve intervala:\n");
    scanf("%lf %lf", &a, &b);
89
    printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
91  printf("(ukljucujuci krajeve intervala)?\n");
    scanf("%d", &n);
93
    /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
95     uneti broj veci od 2 */
    if (n < 2) {
97      fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
        exit(EXIT_FAILURE);
99  }

101 /* Ispisuje se ime funkcije */
    printf("     x %10s(x)\n", ime_fje);
103
    /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
105    komandne linije */
    tabela(a, b, n, fp);
107
    exit(EXIT_SUCCESS);
109 }
```

Glava 3

Algoritmi pretrage i sortiranja

3.1 Algoritmi pretrage

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 dopisati u datoteku `vremena.txt`.

Test 1

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

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

VREMENA.TXT

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

Test 2

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

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

VREMENA.TXT

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

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

[Rešenje 3.1]

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

Primer 1

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

Primer 2

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

[Rešenje 3.2]

Zadatak 3.3 Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće.

Za svakog studenta u jednom redu stoje informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti `-indeks` ili `-prezime`. U slučaju neuspešnih pretragi, štampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složenosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

Primer 1

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

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

```
Interakcija sa programom:
  Unesite indeks studenta
  cije informacije zelite:
  20140076
  Indeks: 20140076,
  Ime i prezime: Sonja Stevanovic
```

Primer 2

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

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

```
Interakcija sa programom:
  Unesite prezime studenta
  cije informacije zelite:
  Popovic
  Indeks: 20140032,
  Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

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

[Rešenje 3.4]

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

Test 1

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

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

Izlaz:
 -0.3 23
```

Test 2

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

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

Izlaz:
 -1 2.1
```

Test 3

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

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

Izlaz:
 -0.3 0.23
```

[Rešenje 3.5]

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

Test 1

```
Izlaz:
 1.57031
```

[Rešenje 3.6]

Zadatak 3.7 Napisati funkciju koja u rastuće sortiranom nizu celih brojeva binarnom pretragom pronalazi indeks prvog elementa većeg od nule. Ukoliko nema elemenata većih od nule, funkcija kao rezultat vraća `-1`. Napisati program koji testira ovu funkciju za rastući niz 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
```

[Rešenje 3.7]

Zadatak 3.8 Napisati funkciju koja u opadajuće sortiranom nizu celih brojeva binarnom pretragom pronalazi indeks prvog elementa manjeg od nule. Ukoliko nema elemenata manjih od nule, funkcija kao rezultat vraća `-1`. Napisati program koji testira ovu funkciju za opadajući niz 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
```

[Rešenje 3.8]

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

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

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

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

Test 1

```
Ulaz:
 4

Izlaz:
 2 2
```

Test 2

```
Ulaz:
 17

Izlaz:
 4 4
```

Test 3

```
Ulaz:
 1031

Izlaz:
 10 10
```

[Rešenje 3.9]

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesi broj tacaka: 2
 Unesi koordinate tacaka:
 2 0 2 1
 1 2 2 2
 26.57
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesi broj tacaka: 2
 Unesi koordinate tacaka:
 1 0 1 1
 0 1 1 1
 45
```

Primer 3

```
INTERAKCIJA SA PROGRAMOM:
 Unesi broj tacaka: 3
 Unesi koordinate tacaka:
 1 0 1 1
 2 0 2 1
 1 2 2 2
 26.57
```

3.2 Algoritmi sortiranja

Zadatak 3.11 Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. `selection sort`), sortiranja spajanjem (engl. `merge sort`), brzog sortiranja (engl. `quick sort`), mehurastog sortiranja (engl. `bubble sort`), sortiranja direktnim umetanjem (engl. `insertion sort`) i sortiranja umetanjem sa inkrementom (engl. `shell sort`). Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: `-m` za sortiranje spajanjem, `-q` za brzo sortiranje, `-b` za mehurasto, `-i` za sortiranje direktnim umetanjem ili `-s` za sortiranje umetanjem sa inkrementom. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati algoritmom sortiranja izborom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija `-r`, nerastuće ako je prisutna opcija `-o` ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije `n`.

Test 1

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

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

Test 2

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

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

Test 3

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

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

Test 4

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

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

Test 5

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

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

Test 6

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

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

[Rešenje 3.11]

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 sa programom:
  Unesite prvu nisku  anagram
  Unesite drugu nisku  ramgana
  jesu
```

Primer 2

```
Interakcija sa programom:
  Unesite prvu nisku  anagram
  Unesite drugu nisku  anagrm
  nisu
```

Primer 3

```
Interakcija sa programom:
  Unesite prvu nisku  test
  Unesite drugu nisku  tset
  jesu
```

[Rešenje 3.12]

Zadatak 3.13 U datom nizu brojeva treba pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, ali neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati razliku pronađena dva broja. Uputstvo: *Prvo sortirati niz.* Napomena: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11.*

Test 1

```
Ulaz:
  23 64 123 76 22 7

Izlaz:
  1
```

Test 2

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

Izlaz:
  0
```

Test 3

```
Ulaz:
  34 30

Izlaz:
  4
```

[Rešenje 3.13]

Zadatak 3.14 Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza i neće biti duži od 256 i kraći od jednog elemenata. Uputstvo: *Prvo sortirati*

niz, a zatim naći najdužu sekvencu jednakih elemenata. NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11.*

Test 1

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

IZLAZ:
 4
```

Test 2

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

IZLAZ:
 2
```

Test 3

```
ULAZ:
 123

IZLAZ:
 123
```

[Rešenje 3.14]

Zadatak 3.15 Napisati funkciju koja proverava da li u datom nizu postoje dva elementa čiji zbir je jednak zadatom celom broju. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz. Elementi niza se unose sve do kraja ulaza. Pretpostaviti da u niz neće biti uneto više od 256 brojeva. UPUTSTVO: *Prvo sortirati niz.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.11.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite trazeni zbir: 34
 Unesite elemente niza:
 134 4 1 6 30 23
 da
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite trazeni zbir: 12
 Unesite elemente niza:
 53 1 43 3 56 13
 ne
```

Primer 3

```
INTERAKCIJA SA PROGRAMOM:
 Unesite trazeni zbir: 52
 Unesite elemente niza:
 52
 ne
```

[Rešenje 3.15]

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

Primer 1

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

Primer 2

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

[Rešenje 3.16]

Zadatak 3.17 Napisati program koji čita sadržaj dveju datoteka od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima, a u slučaju istog imena po prezimenima, i kreira jedinstven spisak studenata sortiranih takođe po istom kriterijumu. Program dobija nazive datoteka iz komandne linije i jedinstveni spisak upisuje u datoteku `ceo-tok.txt`. Pretpostaviti da 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
  Uros Milic
  Vladimir Savic

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

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

[Rešenje 3.17]

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
```

[Rešenje 3.18]

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, i da se nijedno ime i prezime ne pojavljuje više od jednom.

Test 1

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

IZLAZ:
 3
```

Test 2

```
BIRACKI-SPISAK.TXT
 Milan Milicevic

IZLAZ:
 1
```

Test 3

```
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI

IZLAZ:
 Problem pri otvaranju
 datoteke.
```

[Rešenje 3.19]

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

Test 1

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

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

[Rešenje 3.21]

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

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

  UKUPNO: 270.00 dinara.

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

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

  UKUPNO: 230.00 dinara.

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

 Kraj rada kase!
```

[Rešenje 3.22]

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
```

[Rešenje 3.23]

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
```

[Rešenje 3.24]

**** 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 sa programom:
 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
```

Zadatak 3.27 Za zadatu celobrojnu matricu dimenzije $n \times m$ napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrsti. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. Napomena: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

Test 1

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

Test 2

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

[Rešenje 3.27]

Zadatak 3.28 Za zadatu kvadratnu matricu dimenzije n napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka 2.19.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite dimenziju matrice: 2
 Unesite elemente matrice po vrstama:
 6 -5
 -4 3
 Sortirana matrica je:
 -5 6
 3 -4
```

Primer 2

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

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.29 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:
 X

IZLAZ:
 -1
```

Zadatak 3.30 Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva, ne veća od 100, a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu. Na standardnom izlazu ispisati odgovarajuću poruku.

Primer 1

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

[Rešenje 3.30]

Zadatak 3.31 Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva (ne veću od 100), a potom i same elemente niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku prema broju delilaca i tako dobijeni niz odštampati na standardnom izlazu.

Primer 1

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

Primer 2

```
Interakcija sa programom:
 Uneti dimenziju niza: 1
 Uneti elemente niza:
 234
 Sortirani niz u rastucem
 poretku prema broju delilaca
 234
```

Primer 3

```
Interakcija sa programom:
 Uneti dimenziju niza: 0
 Uneti elemente niza:
 Sortirani niz u rastucem
 poretku prema broju
 delilaca:
```

[Rešenje 3.31]

Zadatak 3.32 Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

(a) leksikografski,

(b) po dužini.

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

Primer 1

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

INTERAKCIJA SA PROGRAMOM:
 Leksikografski sortirane niske:
 aleksandar ana andjela ljubica matej milica milos nikola petar
 Uneti trazenu nisku: matej
 Niska "matej"je pronadjena u nizu na poziciji 4
 Niske sortirane po duzini:
 ana matej milos petar milica nikola andjela ljubica aleksandar
 Niska "matej"je pronadjena u nizu na poziciji 1
```

[Rešenje 3.32]

Zadatak 3.33 Uraditi prethodni zadatak 3.32 sa dinamički alociranim niskama i sortiranjem niza pokazivača, umesto niza niski.

[Rešenje 3.33]

Zadatak 3.34 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 sa programom:
Studenti sortirani po broju poena
opadajuce, pa po prezimenu rastuce:
Bojan Golubovic 30
Dragan Markovic 25
Ivana Stankovic 25
Filip Dukic 20
Lazar Micic 20
Ognjen Peric 20
Nenad Brankovic 15
Aleksandra Cvetic 15
Marija Stankovic 15
Uros Milic 10
Anja Ilic 5
Ivana Markovic 5
Andrija Petrovic 0
Unesite broj bodova: 20
Pronadjen je student sa unetim
brojem bodova: Filip Dukic 20
Unesite prezime: Markovic
Pronadjen je student sa unetim
prezimenom: Dragan Markovic 25
```

[Rešenje 3.34]

Zadatak 3.35 Uraditi zadatak 3.12, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.35]

Zadatak 3.36 Napisati program koji sa standardnog ulaza učitava prvo ceo broj `n` ($n \leq 10$), a zatim niz `S` od `n` niski. Maksimalna dužina svake niske je 31 karakter. Sortirati niz `S` bibliotečkom funkcijom `qsort` i proveriti da li u njemu ima identičnih niski.

Primer 1

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

Primer 2

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

Primer 3

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

[Rešenje 3.36]

Zadatak 3.37 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
```

[Rešenje 3.37]

Zadatak 3.38 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 sa programom:
  Unesi sledeci datum: 13.12.2016.
  postoji
  Unesi sledeci datum: 10.5.2015.
  ne postoji
  Unesi sledeci datum: 5.2.2009.
  postoji
```

3.4 Rešenja

Rešenje 3.1

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

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

/* 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 levom delu niza */
    if (x < a[srednji])
      desni = srednji - 1;
    /* Ako je element sa sredisnjim indeksom manji od x, tada se x
       mora nalaziti u desnom delu niza */
    else if (x > a[srednji])
      levi = srednji + 1;
    else
      /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
         x pronadjen na poziciji srednji */
      return srednji;
```

```
47    }
      /* Ako element x nije pronadjen, vraca se -1 */
49    return -1;
}
51
/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
53     pozicije nadjenog elementa ili -1, ako element nije pronadjen */
int interpolaciona_pretraga(int a[], int n, int x)
55 {
    int levi = 0;
57  int desni = n - 1;
    int srednji;
59  /* Dokle god je indeks levi levo od indeksa desni...  */
    while (levi <= desni) {
61    /* Ako je trazeni element manji od pocetnog ili veci od
         poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
63       nije u tom delu niza. Ova provera je neophodna, da se ne bi
         dogodilo da se prilikom izracunavanja indeksa srednji izadje
65       izvan opsega indeksa [levi,desni] */
      if (x < a[levi] || x > a[desni])
67      return -1;
      /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
69       a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
         jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
71       desni). Ova provera je neophodna, jer bi se u suprotnom
         prilikom izracunavanja indeksa srednji pojavilo deljenje
73       nulom. */
      else if (a[levi] == a[desni])
75      return levi;
      /* Racunanje srednjeg indeksa */
77    srednji =
          levi +
79        (int) ((double) (x - a[levi]) / (a[desni] - a[levi]) *
                 (desni - levi));
81    /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
         verovatno biti blize trazenoj vrednosti nego da je prosto uvek
83       uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
         porediti sa pretragom recnika: ako neko trazi rec na slovo
85       'B', sigurno nece da otvori recnik na polovini, vec verovatno
         negde blize pocetku. */
87    /* Ako je element sa indeksom srednji veci od trazenog, tada se
         trazeni element mora nalaziti u levoj polovini niza */
89    if (x < a[srednji])
        desni = srednji - 1;
91    /* Ako je element sa indeksom srednji manji od trazenog, tada se
         trazeni element mora nalaziti u desnoj polovini niza */
93    else if (x > a[srednji])
        levi = srednji + 1;
95    else
        /* Ako je element sa indeksom srednji jednak trazenom, onda se
97         pretraga zavrsava na poziciji srednji */
        return srednji;
```

```
99    }
      /* U slucaju neuspesne pretrage vraca se -1 */
101   return -1;
}
103
int main(int argc, char **argv)
105 {
      int a[MAX];
107   int n, i, x;
      struct timespec time1, time2, time3, time4, time5, time6;
109   FILE *f;
      /* Provera argumenata komandne linije */
111   if (argc != 3) {
        fprintf(stderr,
113             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
        ;
        exit(EXIT_FAILURE);
115   }

117   /* Dimenzija niza */
      n = atoi(argv[1]);
119   if (n > MAX || n <= 0) {
        fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
121     exit(EXIT_FAILURE);
      }
123
      /* Broj koji se trazi */
125   x = atoi(argv[2]);
      /* Elementi niza se generisu slucajno, tako da je svaki sledeci
127      veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
         pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
129      izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
         godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
131   srandom(time(NULL));
      for (i = 0; i < n; i++)
133     a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
      /* Lineara pretraga */
135   printf("Linearna pretraga:\n");
      /* Vreme proteklo od Nove godine 1970 */
137   clock_gettime(CLOCK_REALTIME, &time1);
      i = linearna_pretraga(a, n, x);
139   /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
         linearnu pretragu */
141   clock_gettime(CLOCK_REALTIME, &time2);
      if (i == -1)
143     printf("Element nije u nizu\n");
      else
145     printf("Element je u nizu na poziciji %d\n", i);
      /* Binarna pretraga */
147   printf("Binarna pretraga:\n");
      clock_gettime(CLOCK_REALTIME, &time3);
149   i = binarna_pretraga(a, n, x);
```

```
  clock_gettime(CLOCK_REALTIME, &time4);
  if (i == -1)
    printf("Element nije u nizu\n");
  else
    printf("Element je u nizu na poziciji %d\n", i);
  /* Interpolaciona pretraga */
  printf("Interpolaciona pretraga:\n");
  clock_gettime(CLOCK_REALTIME, &time5);
  i = interpolaciona_pretraga(a, n, x);
  clock_gettime(CLOCK_REALTIME, &time6);
  if (i == -1)
    printf("Element nije u nizu\n");
  else
    printf("Element je u nizu na poziciji %d\n", i);
  /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
  if ((f = fopen("vremena.txt", "a")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
    exit(EXIT_FAILURE);
  }

  fprintf(f, "Dimenzija niza: %d\n", n);
  fprintf(f, "\tLinearna:%10ld ns\n",
          (time2.tv_sec - time1.tv_sec) * 1000000000 +
          time2.tv_nsec - time1.tv_nsec);
  fprintf(f, "\tBinarna: %19ld ns\n",
          (time4.tv_sec - time3.tv_sec) * 1000000000 +
          time4.tv_nsec - time3.tv_nsec);
  fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
          (time6.tv_sec - time5.tv_sec) * 1000000000 +
          time6.tv_nsec - time5.tv_nsec);
  /* Zatvaranje datoteke */
  fclose(f);
  exit(EXIT_SUCCESS);
}
```

Rešenje 3.2

```
#include <stdio.h>

#define MAX 1024

int lin_pretraga_rek_sufiks(int a[], int n, int x)
{
  int tmp;
  /* Izlaz iz rekurzije */
  if (n <= 0)
    return -1;
  /* Ako je prvi element trazeni */
  if (a[0] == x)
    return 0;
  /* Pretraga ostatka niza */
```

```
  tmp = lin_pretraga_rek_sufiks(a + 1, n - 1, x);
  return tmp < 0 ? tmp : tmp + 1;
}

int lin_pretraga_rek_prefiks(int a[], int n, int x)
{
  /* Izlaz iz rekurzije */
  if (n <= 0)
    return -1;
  /* Ako je poslednji element trazeni */
  if (a[n - 1] == x)
    return n - 1;
  /* Pretraga ostatka niza */
  return lin_pretraga_rek_prefiks(a, n - 1, x);
}

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


int interp_pretraga_rek(int a[], int l, int d, int x)
{
  int p;
  if (x < a[l] || x > a[d])
    return -1;
  if (a[d] == a[l])
    return l;
  /* Pozicija na kojoj se trazi vrednost x */
  p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
  if (a[p] == x)
    return p;
  if (a[p] < x)
    return interp_pretraga_rek(a, p + 1, d, x);
  else
    return interp_pretraga_rek(a, l, p - 1, x);
```

```
}

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>
```

```c
#include <string.h>

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

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

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

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

/* Main funkcija mora imati argumente jer se ime datoteke i opcija
   prosledjuju kao argumenti 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];
  int bin_pretraga;

  /* Provera da li je korisnik prilikom poziva programa prosledio ime
     datoteke sa informacijama o studentima i opciju pretrage */
  if (argc != 3) {
    fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke opcija\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

  /* Provera prosledjene opcije */
  if (strcmp(argv[2], "-indeks") == 0)
    bin_pretraga = 1;
  else if (strcmp(argv[2], "-prezime") == 0)
    bin_pretraga = 0;
  else {
    fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
    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 red sa informacijama o studentu
     */
  i = 0;
  while (1) {
    if (i == MAX_STUDENATA)
```

```
106       break;
      if (fscanf
108         (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
            dosije[i].prezime) != 3)
110       break;
      i++;
112   }
    br_studenata = i;
114
    /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
116 fclose(fin);

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

Rešenje 3.4

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

#define MAX_STUDENATA 128
6 #define MAX_DUZINA 16
```

```
8  typedef struct {
     long indeks;
10   char ime[MAX_DUZINA];
     char prezime[MAX_DUZINA];
12 } Student;

14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                   long x)
16 {
     /* Ako je pozicija elementa na levom kraju veca od pozicije
18      elementa na desnom kraju dela niza koji se pretrazuje, onda se
        zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20      trazenog elementa pa se vraca -1 */
     if (levi > desni)
22     return -1;
     /* Racunanje pozicije srednjeg elementa */
24   int srednji = (levi + desni) / 2;
     /* Da li je srednji bas onaj trazeni */
26   if (a[srednji].indeks == x) {
       return srednji;
28   }
     /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30      poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
        je poznato da je niz sortiran po indeksu u rastucem poretku. */
32   if (x < a[srednji].indeks)
       return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34   /* Inace ga treba traziti u desnoj polovini */
     else
36     return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
   }
38
   int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
     /* Ako je niz prazan, vraca se -1 */
42   if (n == 0)
       return -1;
44   /* Kako se trazi prvi student sa trazenim prezimenom, pocinje se sa
        prvim studentom u nizu. */
46   if (strcmp(a[0].prezime, x) == 0)
       return 0;
48   int i = linearna_pretraga_rekurzivna_v2(a + 1, n - 1, x);
     return i >= 0 ? 1 + i : -1;
50 }

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

```
60      return n - 1;
    return linearna_pretraga_rekurzivna(a, n - 1, x);
62  }

64  /* Main funkcija mora imati argumente jer se ime datoteke i opcija
       prosledjuju kao argumenti komandne linije */
66  int main(int argc, char *argv[])
    {
68    Student dosije[MAX_STUDENATA];
      FILE *fin = NULL;
70    int i;
      int br_studenata = 0;
72    long trazen_indeks = 0;
      char trazeno_prezime[MAX_DUZINA];
74    int bin_pretraga;

76    /* Provera da li je korisnik prilikom poziva programa prosledio ime
         datoteke sa informacijama o studentima i opciju pretrage */
78    if (argc != 3) {
        fprintf(stderr,
80              "Greska: Program se poziva sa %s ime_datoteke opcija\n",
                argv[0]);
82      exit(EXIT_FAILURE);
      }
84
      /* Provera prosledjene opcije */
86    if (strcmp(argv[2], "-indeks") == 0)
        bin_pretraga = 1;
88    else if (strcmp(argv[2], "-prezime") == 0)
        bin_pretraga = 0;
90    else {
        fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
92      exit(EXIT_FAILURE);
      }
94
      /* Otvaranje datoteke */
96    fin = fopen(argv[1], "r");
      if (fin == NULL) {
98      fprintf(stderr,
                "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
100     exit(EXIT_FAILURE);
      }
102
      /* Citanje se vrsi sve dok postoji red sa informacijama o studentu
         */
104   i = 0;
      while (1) {
106     if (i == MAX_STUDENATA)
          break;
108     if (fscanf
            (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
110          dosije[i].prezime) != 3)
```

```
      break;
    i++;
  }
  br_studenata = i;

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

  /* Pretraga po indeksu */
  if (bin_pretraga) {
    /* 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,
                                    trazen_indeks);
    /* Rezultat binarne pretrage */
    if (i == -1)
      printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
      printf("Indeks: %ld, Ime i prezime: %s %s\n",
             dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
  }
  /* Pretraga po prezimenu */
  else {
    /* Unos prezimena koje se linearno trazi u nizu */
    printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
    i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
                                        trazeno_prezime);
    /* Rezultat linearne pretrage */
    if (i == -1)
      printf("Ne postoji student sa prezimenom %s\n",
             trazeno_prezime);
    else
      printf("Indeks: %ld, Ime i prezime: %s %s\n",
             dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
  }
  exit(EXIT_SUCCESS);
}
```

Rešenje 3.5

```
#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]);
    exit(EXIT_FAILURE);
  }

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

  /* Stampanje koordinata trazene tacke */
  printf("%g %g\n", najbliza.x, najbliza.y);

  /* Zatvaranje datoteke */
  fclose(ulaz);

  exit(EXIT_SUCCESS);
}
```

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;
}

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

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

  /* Stampanje rezultata */
  printf("%d\n", prvi_veci_od_nule(niz, n));

  return 0;
}
```

Rešenje 3.8

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

#define MAX 256

int prvi_manji_od_nule(int niz[], int n)
{
  /* Granice pretrage */
  int l = 0, d = n - 1;
  int s;
  /* Sve dok je leva manja od desne granice */
  while (l <= d) {
    /* Racuna se sredisnja pozicija */
    s = (l + d) / 2;
    /* Ako je broj na toj poziciji manji od nule, a eventualni njegov
       prethodnik veci ili jednak nuli, pretraga se zavrsava */
    if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
      return s;
    /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
       niza */
    if (niz[s] >= 0)
      l = s + 1;
    /* A inace leva */
    else
      d = s - 1;
  }
  return -1;
}

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

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

  /* Stampanje rezultata */
  printf("%d\n", prvi_manji_od_nule(niz, n));

  return 0;
}
```

Rešenje 3.9

```c
#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   exit(EXIT_SUCCESS);
}
```

Rešenje 3.11

Datoteka 3.1: *sort.h*

```
#ifndef _SORT_H_
#define _SORT_H_ 1

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

/* Insertion sort: Funkcija sortira niz celih brojeva metodom
   sortiranja umetanjem.  Ideja algoritma je sledeca: neka je na
   pocetku i-te iteracije niz prvih i elemenata
   (a[0],a[1],...,a[i-1]) sortirano. U i-toj iteraciji treba element
   a[i] umetnuti na pravu poziciju medju prvih i elemenata tako da se
   dobije niz duzine i+1 koji je sortiran. Ovo se radi tako sto se
   i-ti element najpre uporedi sa njegovim prvim levim susedom
   (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i niz
   a[0],a[1],...,a[i] je sortiran, pa se moze preci na sledecu
   iteraciju. Ako je a[i-1] vece, tada se zamenjuju a[i] i a[i-1], a
   zatim se proverava da li je potrebno dalje potiskivanje elementa u
   levo, poredeci ga sa njegovim novim levim susedom. Ovim uzastopnim
   premestanjem se a[i] umece na pravo mesto u nizu. */
void insertion_sort(int a[], int n);

/* Bubble sort: 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 bubble_sort(int a[], int n);

/* Selsort: Ovaj algoritam je jednostavno prosirenje sortiranja
   umetanjem koje dopusta direktnu razmenu udaljenih elemenata.
   Prosirenje se sastoji u tome da se kroz algoritam umetanja prolazi
   vise puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
   koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
   tako se dobija h-sortiran niz, tj. niz u kome su elementi na
   rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
```

```
48    drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
      za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
50    kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
      h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
52    algoritma, ali bilo koja vrednost ce rezultovati ispravnim
      sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
54    vrednost 1. */
void shell_sort(int a[], int n);
56
/* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
58    Sortiranje se vrsi od elementa na poziciji l do onog na poziciji
      d. Na pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a
60    d je jednako poslednjem validnom indeksu u nizu. Funkcija niz
      podeli na dve polovine, levu i desnu, koje zatim rekurzivno
62    sortira. Od ova dva sortirana podniza, sortiran niz se dobija
      ucesljavanjem, tj.  istovremenim prolaskom kroz oba niza i izborom
64    trenutnog manjeg elementa koji se smesta u pomocni niz. Na kraju
      algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u
66    originalni niz. */
void merge_sort(int a[], int l, int d);
68
/* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
70    i d. Njena ideja sortiranja je izbor jednog elementa niza, koji se
      naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
72    svi elementi levo od njega bice manji, a svi desno bice veci od
      njega. Kako je pivot doveden na svoje mesto, da bi niz bio
74    kompletno sortiran, potrebno je sortirati elemente levo (manje) od
      njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
76    manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
      deo moze se uraditi rekurzivno. */
78 void quick_sort(int a[], int l, int d);

80 #endif
```

Datoteka 3.2: *sort.c*

```
  #include "sort.h"
2
  #define MAX 1000000
4
  void selection_sort(int a[], int n)
6 {
    int i, j;
8   int min;
    int pom;
10
    /* U svakoj iteraciji ove petlje pronalazi se najmanji element
12     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
       poziciju i, dok se element na pozciji i premesta na poziciju
14     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;
    }
  }
}

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

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

    /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
       potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
       bude sortiran. Indeks j je trenutna pozicija na kojoj se
       element koji se umece nalazi. Petlja se zavrsava ili kada
       element dodje do levog kraja (j==0) ili kada se naidje na
       element a[j-1] koji je manji od a[j]. */
    int temp = a[i];
    for (j = i; j > 0 && temp < a[j - 1]; j--)
      a[j] = a[j - 1];
    a[j] = temp;
  }
}

void bubble_sort(int a[], int n)
{
  int i, j;
  int ind;

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

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

```
        a[j] = a[j + 1];
68      a[j + 1] = temp;

70      /* Promenljiva ind registruje da je bilo premestanja. Samo u
           tom slucaju ima smisla ici na sledecu iteraciju, jer ako
72         nije bilo premestanja, znaci da su svi elementi vec u
           dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
74         niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
           ispravno, ali manje efikasano, jer bi se cesto nepotrebno
76         vrsila mnoga uporedjivanja, kada je vec jasno da je
           sortiranje zavrseno. */
78      ind = 1;
      }
80 }

82 void shell_sort(int a[], int n)
{
84   int h = n / 2, i, j;
   while (h > 0) {
86     /* Insertion sort sa korakom h */
     for (i = h; i < n; i++) {
88       int temp = a[i];
       j = i;
90       while (j >= h && a[j - h] > temp) {
         a[j] = a[j - h];
92       j -= h;
       }
94     a[j] = temp;
     }
96   h = h / 2;
   }
98 }

100 void merge_sort(int a[], int l, int d)
{
102   int s;
   static int b[MAX];            /* Pomocni niz */
104   int i, j, k;

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

110   /* Odredjivanje sredisnjeg indeksa */
   s = (l + d) / 2;
112
   /* Rekurzivni pozivi */
114   merge_sort(a, l, s);
   merge_sort(a, s + 1, d);
116
   /* Inicijalizacija indeksa. Indeks i prolazi krozi levu polovinu
118      niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
```

```
     prolazi kroz pomocni niz b[] */
  i = l;
  j = s + 1;
  k = 0;

  /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
  while (i <= s && j <= d) {
    if (a[i] < a[j])
      b[k++] = a[i++];
    else
      b[k++] = a[j++];
  }

  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive j
     iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
     polovine niza */
  while (i <= s)
    b[k++] = a[i++];

  /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive i
     iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
     polovine niza */
  while (j <= d)
    b[k++] = a[j++];

  /* Prepisuje se "ucesljani" niz u originalni niz */
  for (k = 0, i = l; i <= d; i++, k++)
    a[i] = b[k];
}

/* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
{
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}

void quick_sort(int a[], int l, int d)
{
  int i, pivot_position;

  /* Izlaz iz rekurzije -- prazan niz */
  if (l >= d)
    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 <= d; 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 */
  quick_sort(a, l, pivot_position - 1);
  /* Rekurzivno sortiranje elementa vecih od pivota */
  quick_sort(a, pivot_position + 1, d);
}
```

Datoteka 3.3: *main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sort.h"

/* Maksimalna duzina niza */
#define MAX 1000000

int main(int argc, char *argv[])
{
  /*****************************************************************
     tip_sortiranja == 0 => selectionsort, (podrazumevano)
     tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
     tip_sortiranja == 2 => bubblesort,    -b opcija komandne linije
     tip_sortiranja == 3 => shellsort,     -s opcija komandne linije
     tip_sortiranja == 4 => mergesort,     -m opcija komandne linije
     tip_sortiranja == 5 => quicksort,     -q opcija komandne linije
  *****************************************************************/
  int tip_sortiranja = 0;
  /*****************************************************************
     tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)
     tip_niza == 1 => rastuce sortirani nizovi,   -r opcija
     tip_niza == 2 => opadajuce soritrani nizovi, -o opcija
  *****************************************************************/
  int tip_niza = 0;

  /* Dimenzija niza koji se sortira */
  int dimenzija;
  int i;
  int niz[MAX];

  /* Provera argumenata komandne linije */
  if (argc < 2) {
```

```
    fprintf(stderr,
35              "Program zahteva bar 2 argumenta komandne linije!\n");
    exit(EXIT_FAILURE);
37  }

39  /* Ocitavanje opcija i argumenata prilikom poziva programa */
    for (i = 1; i < argc; i++) {
41    /* Ako je u pitanju opcija... */
      if (argv[i][0] == '-') {
43      switch (argv[i][1]) {
        case 'i':
45        tip_sortiranja = 1;
          break;
47      case 'b':
          tip_sortiranja = 2;
49        break;
        case 's':
51        tip_sortiranja = 3;
          break;
53      case 'm':
          tip_sortiranja = 4;
55        break;
        case 'q':
57        tip_sortiranja = 5;
          break;
59      case 'r':
          tip_niza = 1;
61        break;
        case 'o':
63        tip_niza = 2;
          break;
65      default:
          printf("Pogresna opcija -%c\n", argv[i][1]);
67        return 1;
          break;
69      }
      }
71    /* Ako je u pitanju argument, onda je to duzina niza koji treba
         da se sortira */
73    else {
        dimenzija = atoi(argv[i]);
75      if (dimenzija <= 0 || dimenzija > MAX) {
          fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
77        exit(EXIT_FAILURE);
        }
79    }
    }
81

    /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
83     niza dobijenom iz komandne linije. srandom() funkcija
       obezbedjuje novi seed za pozivanje random funkcije, i kako
85     generisani niz ne bi uvek bio isti seed je postavljen na tekuce
```

```
    vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
87    izmedju 0 i 99 */
  srandom(time(NULL));
89  if (tip_niza == 0)
    for (i = 0; i < dimenzija; i++)
91    niz[i] = random();
  else if (tip_niza == 1)
93  for (i = 0; i < dimenzija; i++)
      niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
95  else
    for (i = 0; i < dimenzija; i++)
97    niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;

99  /* Ispisivanje elemenata niza */
  /*****************************************************************
101   Ovaj deo je iskomentarisan jer sledeci ispis ne treba da se nadje
      na standardnom izlazu. Njegova svrha je samo bila provera da li
103   je niz generisan u skladu sa opcijama komandne linije.

105   printf("Niz koji sortiramo je:\n");
      for (i = 0; i < dimenzija; i++)
107     printf("%d\n", niz[i]);
  *****************************************************************/
109

111 /* Sortiranje niza na odgovarajuci nacin */
  if (tip_sortiranja == 0)
113   selection_sort(niz, dimenzija);
  else if (tip_sortiranja == 1)
115   insertion_sort(niz, dimenzija);
  else if (tip_sortiranja == 2)
117   bubble_sort(niz, dimenzija);
  else if (tip_sortiranja == 3)
119   shell_sort(niz, dimenzija);
  else if (tip_sortiranja == 4)
121   merge_sort(niz, 0, dimenzija - 1);
  else
123   quick_sort(niz, 0, dimenzija - 1);

125 /* Ispis elemenata niza */
  /*****************************************************************
127   Ovaj deo je iskomentarisan jer vreme potrebno za njegovo
      izvrsavanje ne bi trebalo da bude ukljuceno u vreme izmereno
129   programom time. Takodje, kako je svrha ovog programa da prikaze
      vremena razlicitih algoritama sortiranja, dimenzije nizova ce
131   biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
      od toliko elemenata. Ovaj deo je koriscen u razvoju programa
133   zarad testiranja korektnosti.

135   printf("Sortiran niz je:\n");
      for (i = 0; i < dimenzija; i++)
137     printf("%d\n", niz[i]);
```

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

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.12

```
#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>
#include "sort.h"
#define MAX 256

/* 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++;

  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
     se selection sort. */
  selection_sort(a, i);
```

```
31
  /* Ispis rezultata */
33  printf("%d\n", najmanje_rastojanje(a, i));

35  return 0;
}
```

Rešenje 3.14

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

/* Funkcija za odredjivanje onog elementa sortiranog niza koji se
   najvise puta pojavio u tom nizu */
int najvise_puta(int a[], int n)
{
  int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
  /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
  for (i = 0; i < n; i = j) {
    br_pojava = 1;
    for (j = i + 1; j < n && a[i] == a[j]; j++)
      br_pojava++;
    /* Ispitivanje da li se do tog trenutka i-ti element pojavio
       najvise puta u nizu */
    if (br_pojava > max_br_pojava) {
      max_br_pojava = br_pojava;
      i_max_pojava = i;
    }
  }
  /* Vraca se element koji se najvise puta pojavio u nizu */
  return a[i_max_pojava];
}

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

  /* Ucitavanje elemenata niza sve do kraja ulaza */
  i = 0;
  while (scanf("%d", &a[i]) != EOF)
    i++;

  /* Za sortiranje niza moze se koristiti bilo koja od funkcija
     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
     se merge sort. */
  merge_sort(a, 0, i - 1);

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

```
43    return 0;
}
```

Rešenje 3.15

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

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

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

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

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

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

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

  for (i = 0; i < n; i++)
    /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
       niza nalazi element koji sabran sa njim ima ucitanu vrednost
```

```
47          zbira */
      if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
49        printf("da\n");
          return 0;
51      }
    printf("ne\n");
53
    return 0;
55  }
```

Rešenje 3.16

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

4  int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
              int dim3)
6  {
     int i = 0, j = 0, k = 0;
8    /* U slucaju da je dimenzija treceg niza manja od neophodne,
        funkcija vraca -1 */
10   if (dim3 < dim1 + dim2)
       return -1;
12
     /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja jednog
14      od njih */
     while (i < dim1 && j < dim2) {
16     if (niz1[i] < niz2[j])
         niz3[k++] = niz1[i++];
18     else
         niz3[k++] = niz2[j++];
20   }
     /* Ostatak prvog niza prepisujemo u treci */
22   while (i < dim1)
       niz3[k++] = niz1[i++];
24
     /* Ostatak drugog niza prepisujemo u treci */
26   while (j < dim2)
       niz3[k++] = niz2[j++];
28   return dim1 + dim2;
  }
30
  int main()
32 {
     int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
34   int i = 0, j = 0, k, dim3;

36   /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
        Pretpostavka je da na ulazu nece biti vise od MAX_DIM elemenata
       */
38   printf("Unesite elemente prvog niza: ");
```

```
  while (1) {
40  scanf("%d", &niz1[i]);
    if (niz1[i] == 0)
42    break;
    i++;
44  }
  printf("Unesite elemente drugog niza: ");
46  while (1) {
    scanf("%d", &niz2[j]);
48  if (niz2[j] == 0)
      break;
50  j++;
  }
52
  /* Poziv trazene funkcije */
54  dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

56  /* Ispis niza */
  for (k = 0; k < dim3; k++)
58  printf("%d ", niz3[k]);
  printf("\n");
60
  return 0;
62}
```

Rešenje 3.17

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
int main(int argc, char *argv[])
6 {
  FILE *fin1 = NULL, *fin2 = NULL;
8 FILE *fout = NULL;
  char ime1[11], ime2[11];
10 char prezime1[16], prezime2[16];
  int kraj1 = 0, kraj2 = 0;
12
  /* Ako nema dovoljno arguemenata komandne linije */
14 if (argc < 3) {
    fprintf(stderr, "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) {
      int tmp = strcmp(ime1, ime2);
52    if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
        /* Ime i prezime iz prve datoteke je leksikografski ranije, i
54         biva upisano u izlaznu datoteku */
        fprintf(fout, "%s %s\n", ime1, prezime1);
56      /* Citanje narednog studenta iz prve datoteke */
        if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
58        kraj1 = 1;
      } else {
60      /* Ime i prezime iz druge datoteke je leksikografski ranije, i
           biva upisano u izlaznu datoteku */
62      fprintf(fout, "%s %s\n", ime2, prezime2);
        /* Citanje narednog studenta iz druge datoteke */
64      if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
          kraj2 = 1;
66    }
    }
68
    /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
70     druge datoteke, onda ima jos studenata u prvoj datoteci, koje
       treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
72   */
    while (!kraj1) {
74    fprintf(fout, "%s %s\n", ime1, prezime1);
      if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
```

```
76        kraj1 = 1;
  }
78
  /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
80     datoteke, onda ima jos studenata u drugoj datoteci, koje treba
       prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
82  while (!kraj2) {
      fprintf(fout, "%s %s\n", ime2, prezime2);
84    if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
        kraj2 = 1;
86  }

88  /* Zatvaranje datoteka */
  fclose(fin1);
90  fclose(fin2);
  fclose(fout);
92
  exit(EXIT_SUCCESS);
94 }
```

Rešenje 3.18

```
1  #include <stdio.h>
  #include <string.h>
3  #include <math.h>
  #include <stdlib.h>
5
  #define MAX_BR_TACAKA 128
7
  /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
    int x;
11  int y;
  } Tacka;
13
  /* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka
15    (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 */
65void 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  }
```

```
  }
}

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

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

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

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

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

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

  /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
     "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica
     (karakter -), a karakter argv[1][1] odredjuje kriterijum
     sortiranja */
  switch (argv[1][1]) {
  case 'x':
```

```
    /* Sortiranje po vrednosti x koordinate */
    sortiraj_po_x(tacke, n);
    break;
  case 'y':
    /* Sortiranje po vrednosti y koordinate */
    sortiraj_po_y(tacke, n);
    break;
  case 'o':
    /* Sortiranje po udaljenosti od koorinatnog pocetka */
    sortiraj_po_rastojanju(tacke, n);
    break;
  }

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

  /* Zatvaranje otvorenih datoteka */
  fclose(ulaz);
  fclose(izlaz);

  return 0;
}
```

Rešenje 3.19

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

#define MAX 1000
#define MAX_DUZINA 16

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

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

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

```
    for (j = i + 1; j < n; j++)
26      if (strcmp(a[j].ime, a[min].ime) < 0)
          min = j;
28    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
         su (i) i min razliciti, inace je nepotrebno. */
30    if (min != i) {
        pom = a[i];
32      a[i] = a[min];
        a[min] = pom;
34    }
  }
36}

38/* Funkcija sortira niz gradjana rastuce po prezimenima */
void sort_prezime(Gradjanin a[], int n)
40{
  int i, j;
42  int min;
  Gradjanin pom;
44
  for (i = 0; i < n - 1; i++) {
46  /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
       najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
48  min = i;
    for (j = i + 1; j < n; j++)
50    if (strcmp(a[j].prezime, a[min].prezime) < 0)
        min = j;
52  /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
       su (i) i min razliciti, inace je nepotrebno. */
54  if (min != i) {
      pom = a[i];
56    a[i] = a[min];
      a[min] = pom;
58  }
  }
60}

62/* Pretraga niza Gradjana */
int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64{
  int i;
66  for (i = 0; i < n; i++)
    if (strcmp(a[i].ime, x->ime) == 0
68      && strcmp(a[i].prezime, x->prezime) == 0)
      return i;
70  return -1;
}
72

74int 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
  /*****************************************************************
112   Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
    sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
    printf("Biracki spisak [uredjen prema prezimenima]:\n");
116 for(i=0; i<n; i++)
      printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118 *****************************************************************/

120 /* Linearno pretrazivanje nizova */
  for (i = 0; i < n; i++)
122   if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
      isti_rbr++;
124
  /* Alternativno (efikasnije) resenje */
126 /*****************************************************************
    for(i=0; i<n ;i++)
128   if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
```

```
            strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130     isti_rbr++;
    ****************************************************************/
132
    /* Ispis rezultata */
134   printf("%d\n", isti_rbr);

136   exit(EXIT_SUCCESS);
}
```

Rešenje 3.21

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

5  #define MAX_BR_RECI 128
   #define MAX_DUZINA_RECI 32
7
   /* Funkcija koja izracunava broj suglasnika u reci */
9  int broj_suglasnika(char s[])
   {
11   char c;
     int i;
13   int suglasnici = 0;
     /* Prolaz karakter po karakter kroz zadatu nisku */
15   for (i = 0; s[i]; i++) {
       /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17        pokriven slucaj i malih i velikih suglasnika. */
       if (isalpha(s[i])) {
19       c = toupper(s[i]);
         /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
       */
21       if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
           suglasnici++;
23     }
     }
25   /* Vraca se izracunata vrednost */
     return suglasnici;
27 }

29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
      duzini reci se mora proslediti zbog pravilnog upravljanja
31    memorijom */
   void sortiraj_reci(char reci[][MAX_DUZINA_RECI], int n)
33 {
     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35       duzina_j, duzina_min;
     char tmp[MAX_DUZINA_RECI];
37   for (i = 0; i < n - 1; i++) {
       min = i;
```

```
39      for (j = i; j < n; j++) {
          /* Prvo se uporedjuje broj suglasnika */
41        broj_suglasnika_j = broj_suglasnika(reci[j]);
          broj_suglasnika_min = broj_suglasnika(reci[min]);
43        if (broj_suglasnika_j < broj_suglasnika_min)
            min = j;
45        else if (broj_suglasnika_j == broj_suglasnika_min) {
            /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
47             se duzine */
            duzina_j = strlen(reci[j]);
49          duzina_min = strlen(reci[min]);

51          if (duzina_j < duzina_min)
              min = j;
53          else
              /* Ako reci imaju i isti broj suglasnika i iste duzine,
55               uporedjuju se leksikografski */
            if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
57            min = j;
          }
59      }
        if (min != i) {
61        strcpy(tmp, reci[min]);
          strcpy(reci[min], reci[i]);
63        strcpy(reci[i], tmp);
        }
65    }
  }
67
  int main()
69{
    FILE *ulaz;
71  int i = 0, n;

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

77  /* Otvaranje datoteke niske.txt za citanje */
    ulaz = fopen("niske.txt", "r");
79  if (ulaz == NULL) {
      fprintf(stderr,
81            "Greska prilikom otvaranja datoteke niske.txt!\n");
      return 0;
83  }

85  /* Sve dok se moze procitati sledeca rec */
    while (fscanf(ulaz, "%s", reci[i]) != EOF) {
87    /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
         prekida se ucitavanje */
89    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 */
```

```
28      if (x < a[srednji].kod)
          desni = srednji - 1;
30      /* Ako je sredisnji element manji od trazenog, tada se trazeni
           mora nalaziti u desnoj polovini niza */
32      else if (x > a[srednji].kod)
          levi = srednji + 1;
34      else
          /* Ako je sredisnji element jednak trazenom, tada je artikal sa
36           bar kodom x pronadjen na poziciji srednji */
          return srednji;
38    }
    /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
40  return -1;
}
42
/* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44 void selection_sort(Artikal a[], int n)
{
46  int i, j;
  int min;
48  Artikal pom;

50  for (i = 0; i < n - 1; i++) {
      min = i;
52    for (j = i + 1; j < n; j++)
        if (a[j].kod < a[min].kod)
54        min = j;
      if (min != i) {
56      pom = a[i];
        a[i] = a[min];
58      a[min] = pom;
      }
60  }
}
62
int main()
64 {
  Artikal asortiman[MAX_ARTIKALA];
66  long kod;
  int i, n;
68  float racun;

70  FILE *fp = NULL;

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

78  /* Ucitavanje artikala */
  i = 0;
```

```
80  while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
                asortiman[i].naziv, asortiman[i].proizvodjac,
82              &asortiman[i].cena) == 4)
      i++;
84
    /* Zatvaranje datoteke */
86  fclose(fp);

88  n = i;

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

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

114 kod = 0;
    while (1) {
116   printf("---------------------------\n");
      printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
118   printf("- Za nov racun unesite kod artikla!\n\n");
      /* Unos bar koda provog artikla sledeceg kupca */
120   if (scanf("%ld", &kod) == EOF)
        break;
122   /* Trenutni racun novog kupca */
      racun = 0;
124   /* Za sve artikle trenutnog kupca */
      while (1) {
126     /* Vrsi se njihov pronalazak u nizu */
        if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
128       printf("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
        } else {
130       printf("\tTrazili ste:\t%s %s %12.2f\n",
```

```
                asortiman[i].naziv, asortiman[i].proizvodjac,
132             asortiman[i].cena);
        /* I dodavanje na ukupan racun */
134     racun += asortiman[i].cena;
      }
136   /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
         nema vise artikla */
      printf("Unesite kod artikla [ili 0 za prekid]: \t");
138   scanf("%ld", &kod);
      if (kod == 0)
140     break;
    }
142 /* Stampanje ukupnog racuna trenutnog kupca */
    printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
144 }

146
  printf("Kraj rada kase!\n");
148

  exit(EXIT_SUCCESS);
150 }
```

Rešenje 3.23

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

5  #define MAX 500

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

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

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

```
29      if (min != i) {
          pom = niz[min];
31        niz[min] = niz[i];
          niz[i] = pom;
33      }
    }
35  }

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

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

```
81                && strcmp(niz[j].prezime, niz[max].prezime) > 0)
          max = j;
83      if (max != i) {
          pom = niz[max];
85        niz[max] = niz[i];
          niz[i] = pom;
87      }
    }
89  }

91  int main(int argc, char *argv[])
    {
93    Student praktikum[MAX];
      int i, br_studenata = 0;
95
      FILE *fp = NULL;
97
      /* Otvaranje datoteke za citanje */
99    if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
        fprintf(stderr, "Neupesno otvaranje datoteke aktivnost.txt.\n");
101     exit(EXIT_FAILURE);
      }
103
      /* Ucitavanje sadrzaja */
105   for (i = 0;
           fscanf(fp, "%s%s%d%d", praktikum[i].ime,
107               praktikum[i].prezime, &praktikum[i].prisustvo,
                  &praktikum[i].zadaci) != EOF; i++);
109   /* Zatvaranje datoteke */
      fclose(fp);
111   br_studenata = i;

113   /* Kreiranje prvog spiska studenata po prvom kriterijumu */
      sort_ime_leksikografski(praktikum, br_studenata);
115   /* Otvaranje datoteke za pisanje */
      if ((fp = fopen("dat1.txt", "w")) == NULL) {
117     fprintf(stderr, "Neupesno otvaranje datoteke dat1.txt.\n");
        exit(EXIT_FAILURE);
119   }
      /* Upis niza u datoteku */
121   fprintf
          (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
123   for (i = 0; i < br_studenata; i++)
        fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
125               praktikum[i].prezime, praktikum[i].prisustvo,
                  praktikum[i].zadaci);
127   /* Zatvaranje datoteke */
      fclose(fp);
129
      /* Kreiranje drugog spiska studenata po drugom kriterijumu */
131   sort_zadatke_pa_imena(praktikum, br_studenata);
      /* Otvaranje datoteke za pisanje */
```

```
  if ((fp = fopen("dat2.txt", "w")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
    exit(EXIT_FAILURE);
  }
  /* Upis niza u datoteku */
  fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
  fprintf(fp, "pa po duzini imena rastuce:\n");
  for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
  /* Zatvaranje datoteke */
  fclose(fp);

  /* Kreiranje treceg spiska studenata po trecem kriterijumu */
  sort_prisustvo_pa_zadatke_pa_prezimena(praktikum, br_studenata);
  /* Otvaranje datoteke za pisanje */
  if ((fp = fopen("dat3.txt", "w")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke dat3.txt.\n");
    exit(EXIT_FAILURE);
  }
  /* Upis niza u datoteku */
  fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
  fprintf(fp, "pa po broju zadataka,\n");
  fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
  for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
  /* Zatvaranje datoteke */
  fclose(fp);

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.24

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

#define KORAK 10

/* Struktura koja opisuje jednu pesmu */
typedef struct {
  char *izvodjac;
  char *naslov;
  int broj_gledanja;
} Pesma;

/* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
```

```
   rad qsort funkcije) */
int uporedi_gledanost(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return p2->broj_gledanja - p1->broj_gledanja;
}

/* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
   funkcije) */
int uporedi_naslove(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

  return strcmp(p1->naslov, p2->naslov);
}

/* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
   qsort funkcije) */
int uporedi_izvodjace(const void *pp1, const void *pp2)
{
  Pesma *p1 = (Pesma *) pp1;
  Pesma *p2 = (Pesma *) pp2;

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

int main(int argc, char *argv[])
{
  FILE *ulaz;
  Pesma *pesme;                 /* Pokazivac na deo memorije za
                                   cuvanje pesama */
  int alocirano_za_pesme;       /* Broj mesta alociranih za pesme */
  int i;                        /* Redni broj pesme cije se
                                   informacije citaju */
  int n;                        /* Ukupan broj pesama */
  int j, k;
  char c;
  int alocirano;                /* Broj mesta alociranih za propratne
                                   informacije o pesmama */
  int broj_gledanja;

  /* Priprema datoteke za citanje */
  ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
  if (ulaz == NULL) {
    printf("Greska pri otvaranju ulazne datoteke!\n");
    return 0;
  }

  /* Citanje informacija o pesmama */
```

```
  pesme = NULL;
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 */
      if (j == alocirano) {
        /* Ako ne, ako je potrosena sva alocirana memorija, alocira
           se novih KORAK mesta */
        alocirano += KORAK;
        pesme[i].naslov =
            (char *) realloc(pesme[i].naslov,
                             alocirano * sizeof(char));

        /* Provera da li je nova alokacija uspesna */
        if (pesme[i].naslov == NULL) {
          /* Ako nije, oslobadja se sva memorija zauzeta do ovog
             koraka */
          for (k = 0; k < i; k++) {
            free(pesme[k].izvodjac);
            free(pesme[k].naslov);
          }
          free(pesme[i].izvodjac);
          free(pesme);

          /* I prekida izvrsavanje programa */
          return 0;
        }
      }
      /* Ako postoji dovoljno memorije, smesta se procitani karakter
    */
      pesme[i].naslov[j] = c;
      j++;
      /* I nastavlja dalje sa citanjem */
    }
    /* Upisuje se terminirajuca nula na kraj reci */
    pesme[i].naslov[j] = '\0';

    /* Preskace se razmak */
    fgetc(ulaz);

    /* Cita se broj gledanja */
    broj_gledanja = 0;

    /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
       datoteke */
    while ((c = fgetc(ulaz)) != '\n') {
      broj_gledanja = broj_gledanja * 10 + (c - '0');
    }
    pesme[i].broj_gledanja = broj_gledanja;

    /* Prelazi se na citanje sledece pesme */
    i++;
  }

  /* Informacija o broju procitanih pesama */
  n = i;
```

```
  /* Zatvaranje nepotrebne datoteke */
  fclose(ulaz);

  /* Analiza argumenta komandne linije */
  if (argc == 1) {
    /* Nema dodatnih opcija => sortiranje po broju gledanja */
    qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
  } else {
    if (argc == 2 && strcmp(argv[1], "-n") == 0) {
      /* Sortiranje po naslovu */
      qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
    } else {
      if (argc == 2 && strcmp(argv[1], "-i") == 0) {
        /* Sortirnje po izvodjacu */
        qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
      } else {
        printf("Nedozvoljeni argumenti!\n");
        free(pesme);
        return 0;
      }
    }
  }

  /* Ispis rezultata */
  for (i = 0; i < n; i++) {
    printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
           pesme[i].broj_gledanja);
  }

  /* Oslobadjanje memorije */
  for (i = 0; i < n; i++) {
    free(pesme[i].izvodjac);
    free(pesme[i].naslov);
  }
  free(pesme);

  return 0;
}
```

Rešenje 3.27

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

/* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
int zbir_vrste(int **a, int v, int m)
{
  int i, zbir = 0;

```

```
  for (i = 0; i < m; i++) {
    zbir += a[v][i];
  }
  return zbir;
}

/* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
   osnovu zbira koriscenjem selection sort algoritma */
void sortiraj_vrste(int **a, int n, int m)
{
  int i, j, min;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++) {
      if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
        min = j;
      }
    }
    if (min != i) {
      int *tmp;
      tmp = a[i];
      a[i] = a[min];
      a[min] = tmp;
    }
  }
}

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

  /* Unos dimenzija matrice */
  printf("Unesite dimenzije matrice: ");
  scanf("%d %d", &n, &m);

  /* Alokacija memorije */
  a = alociraj_matricu(n, m);

  /* Ucitavanje elementa matrice */
  printf("Unesite elemente matrice po vrstama:\n");
  ucitaj_matricu(a, n, m);

  /* Poziv funkcije koja sortira vrste matrice prema zbiru */
  sortiraj_vrste(a, n, m);

  /* Ispis rezultujuce matrice */
  printf("Sortirana matrica je:\n");
  ispisi_matricu(a, n, m);

  /* Oslobadjanje memorije */
```

```
  a = dealociraj_matricu(a, n);

  return 0;
}
```

Rešenje 3.30

```
#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);

  /* return b1 - b2; */
  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;
```

```
45    /* Unos dimenzije */
      printf("Uneti dimenziju niza: ");
47    scanf("%ld", &n);
      if (n > MAX)
49      n = MAX;

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

59    /* Prikaz sortiranog niz */
      printf("Sortirani niz u rastucem poretku:\n");
61    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
63    putchar('\n');

65    /* Pretrazivanje niza */
      /* Vrednost koja ce biti trazena u nizu */
67    printf("Uneti element koji se trazi u nizu: ");
      scanf("%d", &x);
69
      /* Binarna pretraga */
71    printf("Binarna pretraga: \n");
      p = bsearch(&x, a, n, sizeof(int), &compare_int);
73    if (p == NULL)
        printf("Elementa nema u nizu!\n");
75    else
        printf("Element je nadjen na poziciji %ld\n", p - a);
77
      /* Linearna pretraga */
79    printf("Linearna pretraga (lfind): \n");
      p = lfind(&x, a, &n, sizeof(int), &compare_int);
81    if (p == NULL)
        printf("Elementa nema u nizu!\n");
83    else
        printf("Element je nadjen na poziciji %ld\n", p - a);
85
      return 0;
87  }
```

Rešenje 3.31

```
1   #include <stdio.h>
    #include <stdlib.h>
3   #include <math.h>
    #include <search.h>
5
```

```c
#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);

  return n_d_a - n_d_b;
}

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

  /* Unos dimenzije */
  printf("Uneti dimenziju niza: ");
  scanf("%ld", &n);
  if (n > MAX)
    n = MAX;
```

```
59    /* Unos elementa niza */
      printf("Uneti elemente niza:\n");
61    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
63
      /* Sortiranje niza celih brojeva prema broju delilaca */
65    qsort(a, n, sizeof(int), &compare_no_deviders);

67    /* Prikaz sortiranog niza */
      printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
69    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
71    putchar('\n');

73    return 0;
  }
```

Rešenje 3.32

```
1   #include <stdio.h>
    #include <stdlib.h>
3   #include <string.h>
    #include <search.h>
5
    #define MAX_NISKI 1000
7   #define MAX_DUZINA 31

9   /*****************************************************************
      Niz nizova karaktera ovog potpisa
11    char niske[3][4];
      se moze graficki predstaviti ovako:
13    ----------------------------------------------------
      | a | b | c |\0 || d | e | \0|   || f | g | h | \0||
15    ----------------------------------------------------
      Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
17    svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
      nalazi na adresi koja je za 4 veca od prve reci, a za 4 manja od
19    adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
      i ona je tipa char*.
21
      Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23    koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
      reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25    slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
      nad njima.
27  *****************************************************************/
    int poredi_leksikografski(const void *a, const void *b)
29  {
      return strcmp((char *) a, (char *) b);
31  }
```

```
33 /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
      leksikografski, vec po duzini */
35 int poredi_duzine(const void *a, const void *b)
   {
37   return strlen((char *) a) - strlen((char *) b);
   }
39
   int main()
41 {
     int i;
43   size_t n;
     FILE *fp = NULL;
45   char niske[MAX_NISKI][MAX_DUZINA];
     char *p = NULL;
47   char x[MAX_DUZINA];

49   /* Otvaranje datoteke */
     if ((fp = fopen("niske.txt", "r")) == NULL) {
51     fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
       exit(EXIT_FAILURE);
53   }

55   /* Citanje sadrzaja datoteke */
     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
57
     /* Zatvaranje datoteke */
59   fclose(fp);
     n = i;
61
     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
63      prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
        niske po duzini */
65   qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

67   printf("Leksikografski sortirane niske:\n");
     for (i = 0; i < n; i++)
69     printf("%s ", niske[i]);
     printf("\n");
71
     /* Unos trazene niske */
73   printf("Uneti trazenu nisku: ");
     scanf("%s", x);
75
     /* Binarna pretraga */
77   /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
        je niz vec sortiran leksikografski. */
79   p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
                 &poredi_leksikografski);
81
     if (p != NULL)
83     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
```

```
            p, (p - (char *) niske) / MAX_DUZINA);
  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");

  /* 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");

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.33

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

#define MAX_NISKI 1000
#define MAX_DUZINA 31

/*****************************************************************
  Niz pokazivaca na karaktere ovog potpisa
  char *niske[3];
  posle alokacije u main-u se moze graficki predstaviti ovako:
  -----                    -----------------
  | X | --------------->   | a | b | c | \0|
  -----                    =================
  | Y | --------------->   | d | e | \0|
  -----                    =================
  | Z | --------------->   | f | g | h | \0|
  -----                    -----------------
  Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
  njegov element pokazivac koji pokazuje na alocirane karaktere i-te
  reci. Njegov tip je char*.

  Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
```

```
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) + 1) * sizeof(char))) == NULL)
       {
75       fprintf(stderr, "Greska pri alociranju niske\n");
```

```
      exit(EXIT_FAILURE);
    }
    /* Kopiranje procitane niske na svoje mesto */
    strcpy(niske[i], x);
    i++;
  }

  /* Zatvaranje datoteke */
  fclose(fp);
  n = i;

  /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
     prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
     niske po duzini */
  qsort(niske, n, sizeof(char *), &poredi_leksikografski);

  printf("Leksikografski sortirane niske:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  /* Unos trazene niske */
  printf("Uneti trazenu nisku: ");
  scanf("%s", x);

  /* Binarna pretraga */
  p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
           *p, p - niske);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Linearna pretraga */
  p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
           *p, p - niske);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Sortiramo po duzini */
  qsort(niske, n, sizeof(char *), &poredi_duzine);

  printf("Niske sortirane po duzini:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  /* Oslobadjanje zauzete memorije */
  for (i = 0; i < n; i++)
    free(niske[i]);
```

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

Rešenje 3.34

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

#define MAX 500

/* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
  char ime[21];
  char prezime[21];
  int bodovi;
} Student;

/* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
   istim brojem bodova se dodatno sortiraju leksikografski po
   prezimenu */
int poredi1(const void *a, const void *b)
{
  Student *prvi = (Student *) a;
  Student *drugi = (Student *) b;

  if (prvi->bodovi > drugi->bodovi)
    return -1;
  else if (prvi->bodovi < drugi->bodovi)
    return 1;
  else
    /* Ako su jednaki po broju bodova, treba ih uporediti po
       prezimenu */
    return strcmp(prvi->prezime, drugi->prezime);
}

/* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
   Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
   parametar je element niza ciji se bodovi porede. */
int poredi2(const void *a, const void *b)
{
  int bodovi = *(int *) a;
  Student *s = (Student *) b;
  return s->bodovi - bodovi;
}

/* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
   Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
   parametar je element niza cije se prezime poredi. */
```

```
int poredi3(const void *a, const void *b)
{
  char *prezime = (char *) a;
  Student *s = (Student *) b;
  return strcmp(prezime, s->prezime);
}

int main(int argc, char *argv[])
{
  Student kolokvijum[MAX];
  int i;
  size_t br_studenata = 0;
  Student *nadjen = NULL;
  FILE *fp = NULL;
  int bodovi;
  char prezime[21];

  /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
     informacija korisniku kako se program koristi i prekida se
     izvrsavanje. */
  if (argc < 2) {
    fprintf(stderr,
            "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

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

  /* Ucitavanje sadrzaja */
  for (i = 0;
       fscanf(fp, "%s%s%d", kolokvijum[i].ime,
              kolokvijum[i].prezime,
              &kolokvijum[i].bodovi) != EOF; i++);

  /* Zatvaranje datoteke */
  fclose(fp);
  br_studenata = i;

  /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
     studenata sa istim brojem bodova sortiranje vrsi po prezimenu */
  qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);

  printf("Studenti sortirani po broju poena opadajuce, ");
  printf("pa po prezimenu rastuce:\n");
  for (i = 0; i < br_studenata; i++)
    printf("%s %s  %d\n", kolokvijum[i].ime,
           kolokvijum[i].prezime, kolokvijum[i].bodovi);
```

```
98
   /* Pretrazivanje studenata po broju bodova se vrsi binarnom
100      pretragom jer je niz sortiran po broju bodova. */
   printf("Unesite broj bodova: ");
102 scanf("%d", &bodovi);

104 nadjen =
        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
106             &poredi2);

108 if (nadjen != NULL)
      printf
110       ("Pronadjen je student sa unetim brojem bodova: %s %s %d\n",
          nadjen->ime, nadjen->prezime, nadjen->bodovi);
112 else
      printf("Nema studenta sa unetim brojem bodova\n");
114
   /* Pretraga po prezimenu se mora vrsiti linearno jer je niz
116      sortiran po bodovima. */
   printf("Unesite prezime: ");
118 scanf("%s", prezime);

120 nadjen =
        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
122           &poredi3);

124 if (nadjen != NULL)
      printf
126       ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
          nadjen->ime, nadjen->prezime, nadjen->bodovi);
128 else
      printf("Nema studenta sa unetim prezimenom\n");
130
   exit(EXIT_SUCCESS);
132 }
```

Rešenje 3.35

```
   #include <stdio.h>
2  #include <string.h>
   #include <stdlib.h>
4
   #define MAX 128
6
   /* Funkcija poredi dva karaktera */
8  int uporedi_char(const void *pa, const void *pb)
   {
10   return *(char *) pa - *(char *) pb;
   }
12
   /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
```

```c
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 3.36

```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 3.37

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

#define MAX 21

/* Struktura koja predstavlja jednog studenta */
typedef struct student {
  char nalog[8];
  char ime[MAX];
  char prezime[MAX];
  int poeni;
} Student;
```

```
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';
```

```
65    if (godina1 != godina2)
        return godina1 - godina2;
67    else if (smer1 != smer2)
        return smer1 - smer2;
69    else
        return indeks1 - indeks2;
71  }

73  int main(int argc, char **argv)
    {
75    Student *nadjen = NULL;
      char nalog_trazeni[8];
77    Student niz_studenata[100];
      int i = 0, br_studenata = 0;
79    FILE *in = NULL, *out = NULL;

81    /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
         korisnik nije ispravno pozvao program i prijavljuje se greska.
       */
83    if (argc != 2 && argc != 3) {
        fprintf(stderr,
85              "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
        );
        exit(EXIT_FAILURE);
87    }

89    /* Otvaranje datoteke za citanje */
      in = fopen("studenti.txt", "r");
91    if (in == NULL) {
        fprintf(stderr,
93              "Greska prilikom otvarnja datoteke studenti.txt!\n");
        exit(EXIT_FAILURE);
95    }

97    /* Otvaranje datoteke za pisanje */
      out = fopen("izlaz.txt", "w");
99    if (out == NULL) {
        fprintf(stderr,
101             "Greska prilikom otvaranja datoteke izlaz.txt!\n");
        exit(EXIT_FAILURE);
103   }

105   /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
      while (fscanf
107           (in, "%s %s %s %d", niz_studenata[i].nalog,
              niz_studenata[i].ime, niz_studenata[i].prezime,
109           &niz_studenata[i].poeni) != EOF)
        i++;
111
      br_studenata = i;
113
      /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
```

```
115   if (strcmp(argv[1], "-p") == 0)
        qsort(niz_studenata, br_studenata, sizeof(Student),
117           &uporedi_poeni);
      /* A ako je prisutna opcija -n, vrsi se sortiranje po nalogu */
119   else if (strcmp(argv[1], "-n") == 0)
        qsort(niz_studenata, br_studenata, sizeof(Student),
121           &uporedi_nalog);

123   /* Sortirani studenti se ispisuju u izlaznu datoteku */
      for (i = 0; i < br_studenata; i++)
125     fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
              niz_studenata[i].ime, niz_studenata[i].prezime,
127           niz_studenata[i].poeni);

129   /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
         studenta... */
131   if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
        strcpy(nalog_trazeni, argv[2]);
133
        /* ... pronalazi se student sa tim nalogom... */
135     nadjen =
            (Student *) bsearch(nalog_trazeni, niz_studenata,
137                             br_studenata, sizeof(Student),
                                &uporedi_bsearch);
139
        if (nadjen == NULL)
141       printf("Nije nadjen!\n");
        else
143       printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
                nadjen->prezime, nadjen->poeni);
145   }

147   /* Zatvaranje datoteka */
      fclose(in);
149   fclose(out);

151   exit(EXIT_SUCCESS);
}
```

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 `int dodaj_iza(Cvor * tekuci, int broj)` koja iza čvora `tekuci`. dodaje novi čvor sa vrednošću `broj`.

(h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.

(i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradama [,] i međusobno razdvojene zapetama.

(j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.

(k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.

(l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.

(m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.

(n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (`EOF`). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

(1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

Primer 1

```
Interakcija sa programom:
 Unosite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [3, 2]
 14
 Lista: [14, 3, 2]
 5
 Lista: [5, 14, 3, 2]
 3
 Lista: [3, 5, 14, 3, 2]
 17
 Lista: [17, 3, 5, 14, 3, 2]

 Unesite broj koji se trazi:  5
 Trazeni broj 5 je u listi!
```

Primer 2

```
Interakcija sa programom:
 Unosite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [35, 14, 23]

 Unesite broj koji se trazi:  8
 Broj 8 se ne nalazi u listi!
```

(2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

Primer 1

```
Interakcija sa programom:
 Unosite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 14, 3]
 3
 Lista: [2, 3, 14, 3, 3]
 17
 Lista: [2, 3, 14, 3, 3, 17]
 3
 Lista: [2, 3, 14, 3, 3, 17, 3]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 14, 17]
```

Primer 2

```
Interakcija sa programom:
 Unesite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [23, 14]
 35
 Lista: [23, 14, 35]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [23, 14, 35]
```

(3) U programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. Napomena: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Unesite brojeve: (za kraj CTRL+D)
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 3, 14]
 3
 Lista: [2, 3, 3, 3, 14]
 5
 Lista: [2, 3, 3, 3, 5, 14]

 Unesite broj koji se trazi:  14
 Trazeni broj 14 je u listi!

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 5, 14]
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Unesite brojeve: (za kraj CTRL+D)
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [14, 23, 35]

 Unesite broj koji se trazi:  8
 Broj 8 se ne nalazi u listi!

Unesite broj koji se brise:  3
Lista nakon brisanja: [14, 23, 35]
```

[Rešenje 4.1]

Zadatak 4.2 Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: *Koristiti iste* `main` *programe i test primere iz zadatka 4.1.*

[Rešenje 4.2]

Zadatak 4.3 Napisati program koji prebrojava pojavljivanja etiketa `HTML` datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
  unsigned broj_pojavljivanja;
  char etiketa[20];
  struct _Element *sledeci;
} Element;
```

Test 1

```
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 za grešku:
 Greska prilikom otvaranja
 datoteke datoteka.html.
```

[Rešenje 4.3]

Zadatak 4.4 U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku `da` ili `ne`, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (`EOF`). Poruke o greškama ispisivati na standardni izlaz za greške. Uputstvo: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

Primer 1

```
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 sa programom:
  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 sa programom:
  3/2014   ne
  235/2008   ne
  41/2009   ne
```

[Rešenje 4.4]

Zadatak 4.5 Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

(a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

(b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

Test 1

```
brojevi.txt
  43 12 15 16 4 2 8

Izlaz:
rezultat.txt
  12 15 16
```

Test 2

```
Datoteka brojevi.txt
ne postoji.

Izlaz za grešku:
  Greska prilikom otvaranja
  datoteke brojevi.txt.
```

Test 3

```
Datoteka brojevi.txt je prazna

Izlaz:
rezultat.txt
  Rezultat.txt ce biti prazna.
```

Zadatak 4.6 Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. 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 za grešku:
 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 za grešku:
 Program se poziva sa:
 ./a.out dat1.txt dat2.txt!
```

[Rešenje 4.6]

Zadatak 4.7 Date su dve jednostruko povezane liste `L1` i `L2`. Napisati funkciju koja od ovih listi formira novu listu `L` koja sadrži naizmenično raspoređene čvorove listi `L1` i `L2`: prvi čvor iz `L1`, prvi čvor iz `L2`, drugi čvor `L1`, drugi čvor `L2`, itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

Napomena: *Iskoristiti testove 2 - 6 za zadatak 4.6.*

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 4.8 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem

steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

```
IZRAZ.TXT
 {[23 + 5344] * (24 - 234)} - 23

IZLAZ:
 Zagrade su ispravno uparene.
```

Test 2

```
IZRAZ.TXT
 {[23 + 5] * (9 * 2)} - {23}

IZLAZ:
 Zagrade su ispravno uparene.
```

Test 3

```
IZRAZ.TXT
 {[2 + 54) / (24 * 87)} + (234 + 23)

IZLAZ:
 Zagrade nisu ispravno uparene.
```

Test 4

```
IZRAZ.TXT
 {(2 - 14) / (23 + 11)}} * (2 + 13)

IZLAZ:
 Zagrade nisu ispravno uparene.
```

Test 5

```
DATOTEKA IZRAZ.TXT JE PRAZNA

IZLAZ:
 Zagrade su ispravno uparene.
```

Test 6

```
DATOTEKA IZRAZ.TXT NE POSTOJI.

IZLAZ ZA GREŠKU:
 Greska prilikom otvaranja
 datoteke izraz.txt!
```

[Rešenje 4.8]

Zadatak 4.9 Napisati program koji proverava ispravnost uparivanja etiketa u `HTML` datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže `HTML` etikete.*

Test 1

```
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 za grešku:
 Greska prilikom otvaranja
 datoteke datoteka.html.
```

Test 6

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

DATOTEKA.HTML JE PRAZNA

Izlaz:
 Etikete su pravilno uparene!
```

[Rešenje 4.9]

Zadatak 4.10 Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve korisničke `JMBG` brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (`EOF`). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Program mu postavlja pitanje `Da li korisnika vracate na kraj reda?` i ukoliko on da odgovor `Da`, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije `Da`, službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke `izvestaj.txt`. Ova datoteka, za svaki obrađen zahtev, sadrži `JMBG` i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. Uputstvo: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Sluzbenik evidentira korisnicke zahteve:
 Novi zahtev [CTRL+D za kraj]
  JMBG:   1234567890123
  Opis problema:   Otvaranje racuna

 Novi zahtev [CTRL+D za kraj]
  JMBG:   2345678901234
  Opis problema:   Podizanje novca

 Novi zahtev [CTRL+D za kraj]
  JMBG:   3456789012345
  Opis problema:   Reklamacija

 Novi zahtev [CTRL+D za kraj]
  JMBG:

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 2345678901234
 i zahtevom: Podizanje novca
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG: 3456789012345
 i zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 3456789012345
 i zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Da li je kraj smene? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

IZVESTAJ.TXT
  JMBG: 2345678901234 Zahtev: Podizanje novca
  JMBG: 3456789012345 Zahtev: Reklamacija
  JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

[Rešenje 4.10]

Zadatak 4.11 Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti bibilioteku novim funkcijama.

(a) Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor **`

`adresa_kraja, Cvor * tekuci)` koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave` i poslednji čvor liste na adresi `adresa_kraja`.

(b) Napisati funkciju `void ispisi_listu_unazad(Cvor * kraj)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Zbog efikasnog izvršavanja operacija dodavanja na kraj liste i ispisivanja liste unazad treba, pored pokazivača na glavu liste, čuvati i pokazivač na poslednji čvor liste. NAPOMENA: *Funkcije testirati koristeći test primere iz zadatka 4.1*

[Rešenje 4.11]

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 ZA GREŠKU:
 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.*

Test 1

```
Ulaz:
 5 3

Izlaz:
 3 5 4 2 1
```

Test 2

```
Ulaz:
 8 4

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

Test 3

```
Ulaz:
 5 8

Izlaz za grešku:
 n mora biti uvek vece
 od k, a 5 < 8!
```

4.2 Stabla

Zadatak 4.14 Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

(a) Definisati strukturu `Cvor` kojom se opisuje čvor stabla, a koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.

(b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.

(c) Napisati funkciju `int dodaj_u_stablo(Cvor ** adresa_korena, int broj)` koja u stablo na koje pokazuje argument `adresa_korena` dodaje ceo broj `broj`. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.

(d) Napisati funkciju `Cvor *pretrazi_stablo(Cvor * koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. 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 ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.

(h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor * koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.

(i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor * koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.

(j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor * koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.

(k) Napisati funkciju `void oslobodi_stablo(Cvor ** adresa_korena)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `adresa_korena`.

Korišćenjem kreirane biblioteke, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuce stablo: 1 2 9 18 32
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuce stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

Zadatak 4.15 Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog 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 ulaz.txt

Datoteka ulaz.txt ne postoji

Interakcija sa programom:
  Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

Test 4

```
Poziv: ./a.out

Izlaz:
  Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

Zadatak 4.16 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. `Pera Peric 064/123-4567`. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči `KRAJ`, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Primer 1

```
IMENIK.TXT
 Pera Peric 011/3240-987
 Marko Maric 064/1234-987
 Mirko Maric 011/589-333
 Sanja Savkovic 063/321-098
 Zika Zikic 021/759-858

INTERAKCIJA SA PROGRAMOM:
 Unesite ime datoteke: imenik.txt
 Unesite ime i prezime: Pera Peric
 Broj je: 011/3240-987
 Unesite ime i prezime: Marko Markovic
 Broj nije u imeniku!
 Unesite ime i prezime: KRAJ
```

Primer 2

```
DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
 Unesite ime datoteke: imenik1.txt
 Greska: Neuspesno otvaranje datoteke
 imenik1.txt.
```

[Rešenje 4.16]

Zadatak 4.17 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niska od najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

Test 1

```
PRIJEMNI.TXT
 Marko Markovic 45.4 12.3 11
 Milan Jevremovic 35.2 1.3 9
 Maja Agic 60 19 20
 Nadica Zec 54.2 10 15.8
 Jovana Milic 23.3 2 5.6

IZLAZ:
 1. Maja Agic 60.0 19.0 20.0 99.0
 2. Nadica Zec 54.2 10.0 15.8 80.0
 3. Marko Markovic 45.4 12.3 11.0 68.7
 4. Milan Jevremovic 35.2 1.3 9.0 45.5
 ------------------------------------
 5. Jovana Milic 23.3 2.0 5.6 30.9
```

Test 2

```
DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ:
 Greska: Neuspesno otvaranje datoteke
 prijemni.txt.
```

[Rešenje 4.17]

*** Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata `Ime Prezime DD.MM.` i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.`.Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

Primer 1

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

rodjendani.txt
 Marko Markovic 12.12.
 Milan Jevremovic 04.06.
 Maja Agic 23.04.
 Nadica Zec 01.01.
 Jovana Milic 05.05.

Interakcija sa programom:
 Unesite datum: 23.04.
 Slavljenik: Maja Agic
 Unesite datum: 01.01.
 Slavljenik: Nadica Zec
 Unesite datum: 01.05.
 Slavljenik: Jovana Milic 05.05.
 Unesite datum: 20.12.
 Slavljenik: Nadica Zec 01.01.
 Unesite datum:
```

Primer 2

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

Datoteka rodjendani.txt ne postoji

Interakcija sa programom:
 Greska: Neuspesno otvaranje datoteke
 rodjendani.txt.
```

[Rešenje 4.18]

Zadatak 4.19 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor * koren1, Cvor * koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. Napomena: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Primer 1

```
Interakcija sa programom:
 Prvo stablo:
 10 5 15 3 2 4 30 12 14 13 0
 Drugo stablo:
 10 15 5 3 4 2 12 14 13 30 0
 Stabla jesu identicna.
```

Primer 2

```
Interakcija sa programom:
 Prvo stablo:
 10 5 15 4 3 2 30 12 14 13 0
 Drugo stablo:
 10 15 5 3 4 2 12 14 13 30 0
 Stabla nisu identicna.
```

[Rešenje 4.19]

*** Zadatak 4.20** Napisati program za rad sa skupovima u kojem se skupovi predstavljaju pomoću binarnih pretraživačkih stabala. Program za dva skupa čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku skupova. Napomena: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Primer 1

```
Interakcija sa programom:
 Prvi skup: 1 7 8 9 2 2
 Drugi skup: 3 9 6 11 1
 Unija: 1 1 2 2 3 6 7 8 9 9 11
 Presek: 1 9
 Razlika: 2 2 7 8
```

Primer 2

```
Interakcija sa programom:
 Prvi skup: 11 2 7 5
 Drugi skup: 4 3 3 7
 Unija: 2 3 3 4 5 7 7 11
 Presek: 7
 Razlika: 2 5 11
```

[Rešenje 4.20]

Zadatak 4.21 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardni izlaz. Napomena: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Primer 1

```
Interakcija sa programom:
 n: 7
 a: 1 11 8 6 37 25 30
 1 6 8 11 25 30 37
```

Primer 2

```
Interakcija sa programom:
 n: 55
 Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

Zadatak 4.22 Dato je binarno pretraživačko stablo celih brojeva.

(a) Napisati funkciju koja izračunava broj čvorova stabla.

(b) Napisati funkciju koja izračunava broj listova stabla.

(c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.

(d) Napisati funkciju koja izračunava zbir čvorova stabla.

(e) Napisati funkciju koja izračunava najveći element stabla.

(f) Napisati funkciju koja izračunava dubinu stabla.

(g) Napisati funkciju koja izračunava broj čvorova na i-tom nivou stabla.

(h) Napisati funkciju koja ispisuje sve elemente na i-tom nivou stabla.

(i) Napisati funkciju koja izračunava maksimalnu vrednost na i-tom nivou stabla.

(j) Napisati funkciju koja izračunava zbir čvorova na i-tom nivou stabla.

(k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x.

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

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 element na 2. nivou: 30
 Zbir elemenata 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 element na 3. nivou: 55
 Zbir elemenata na 3. nivou: 71
 Zbir elemenata manjih ili jednakih od 31:
 76
```

[Rešenje 4.22]

Zadatak 4.23 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1

```
ULAZ:
 10 5 15 3 2 4 30 12 14 13

IZLAZ:
 0.nivo: 10
 1.nivo: 5 15
 2.nivo: 3 12 30
 3.nivo: 2 4 14
 4.nivo: 13
```

Test 2

```
ULAZ:
 6 11 8 3 -2

IZLAZ:
 0.nivo: 6
 1.nivo: 3 11
 2.nivo: -2 8
```

Test 3

```
ULAZ:
 24 53 61 9 7 55 20 16

IZLAZ:
 0.nivo: 24
 1.nivo: 9 53
 2.nivo: 7 20 61
 3.nivo: 16 55
```

[Rešenje 4.23]

*** Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
 Prvo stablo:  11 20 5 3 0
 Drugo stablo:  8 14 30 1 0
 Stabla su slicna kao u ogledalu.
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
 Prvo stablo:  11 20 5 3 0
 Drugo stablo:  8 20 15 0
 Stabla nisu slicna kao u ogledalu.
```

Zadatak 4.25 AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor * koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1

```
ULAZ:
 10 5 15 2 11 16 1 13

IZLAZ:
 7
```

Test 2

```
ULAZ:
 16 30 40 24 10 18 45 22

IZLAZ:
 6
```

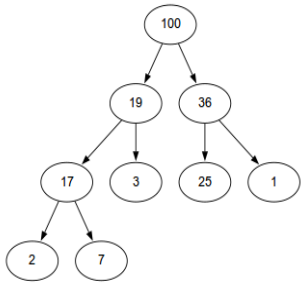
[Rešenje 4.25]

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 zadato slikom 4.1, poziva funkciju `heap` i ispisuje rezultat na standardni izlaz. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

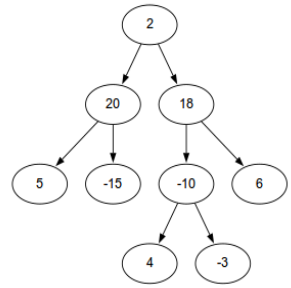
Test 1

```
IZLAZ:
 Zadato stablo je hip!
```

[Rešenje 4.26]



Slika 4.1: *Zadatak 4.26*



Slika 4.2: *Zadatak 4.27*

Zadatak 4.27 Dato je binarno stablo celih brojeva.

(a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

(b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.

(c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.

(d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

Test 1

```
Izlaz:
 Vrednost u cvoru sa maksimalnim desnim zbirom: 18
 Vrednost u cvoru sa minimalnim levim zbirom: 18
 2 18 -10 4
 2 20 -15
```

4.3 Rešenja

Rešenje 4.1

Datoteka 4.1: *lista.h*

```
#ifndef _LISTA_H_
#define _LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
```

```
   greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   pri alokaciji memorije, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

/* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

/* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
int dodaj_iza(Cvor * tekuci, int broj);

/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, 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
```

Datoteka 4.2: *lista.c*

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

Cvor *napravi_cvor(int broj)
{
  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  /* Ako lista nije prazna, onda treba osloboditi memoriju */
  while (*adresa_glave != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor koji predstavlja glavu liste */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);

    /* Sledeci cvor je nova glava liste */
    *adresa_glave = pomocni;
  }
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
```

```
}

Cvor *pronadji_poslednji(Cvor * glava)
{
  /* U praznoj listi nema cvorova pa se vraca NULL */
  if (glava == NULL)
    return NULL;

  /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
     ce pokazivati na cvor liste koji nema sledbenika, tj. na
     poslednji cvor liste pa se vraca vrednost pokazivaca glava.

     Pokazivac glava je argument funkcije i njegove promene nece se
     odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
  while (glava->sledeci != NULL)
    glava = glava->sledeci;

  return glava;
}

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {
    /* Glava nove liste je upravo novi cvor */
    *adresa_glave = novi;
  } else {
    /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
       se dodaje na kraj liste kao sledbenik poslednjeg */
    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
  }

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
  /* U praznoj listi nema takvog mesta i vraca se NULL */
  if (glava == NULL)
    return NULL;

  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
     pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
     vecu ili jednaku vrednosti novog cvora.
```

```
103       Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
          provera da li se doslo do poslednjeg cvora liste pre nego sto se
105       proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
          sledeci ne postoji, pa ni njegova vrednost. */
107   while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
        glava = glava->sledeci;
109
    /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
111       poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
          vrednost vecu od broj. */
113   return glava;
  }
115
  int dodaj_iza(Cvor * tekuci, int broj)
117 {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
119   Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
121   return 1;

123   /* Novi cvor se dodaje iza tekuceg cvora. */
    novi->sledeci = tekuci->sledeci;
125   tekuci->sledeci = novi;

127   /* Vraca se indikator uspesnog dodavanja */
    return 0;
129 }

131 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
  {
133   /* Ako je lista prazna */
    if (*adresa_glave == NULL) {
135   /* Glava nove liste je novi cvor */

137   /* Kreira se novi cvor i proverava se uspesnost kreiranja */
      Cvor *novi = napravi_cvor(broj);
139   if (novi == NULL)
        return 1;
141
      *adresa_glave = novi;
143
      /* Vraca se indikator uspesnog dodavanja */
145   return 0;
    }
147
    /* Inace... */
149
    /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
151       treba dodati na pocetak liste */
    if ((*adresa_glave)->vrednost >= broj) {
153   return dodaj_na_pocetak_liste(adresa_glave, broj);
```

```
  }

  /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  return dodaj_iza(pomocni, broj);
}

Cvor *pretrazi_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  for (; glava != NULL; glava = glava->sledeci)
    /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
       se obustavlja */
    if (glava->vrednost == broj)
      return glava;

  /* Nema trazenog broja u listi i vraca se NULL */
  return NULL;
}

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  /* U uslovu ostanka u petlji, bitan je redosled provera u
     konjunkciji */
  while (glava != NULL && glava->vrednost < broj)
    glava = glava->sledeci;

  /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
     glava->vrednost veca od trazenog broja i tada treba vratiti
     NULL, jer trazen broj nije nadjen medju manjim brojevima pri
     pocetku sortirane liste, pa se moze zakljuciti da ga nema u
     listi. Drugi nacini, tako sto se doslo do kraja liste i glava je
     NULL ili tako sto je glava->vrednost == broj. U oba poslednja
     nacina treba vratiti pokazivac glava bilo da je NULL ili
     pokazivac na konkretan cvor. */
  if (glava->vrednost > broj)
    return NULL;
  else
    return glava;
}

void obrisi_cvor(Cvor ** adresa_glave, int broj)
{
  Cvor *tekuci = NULL;
  Cvor *pomocni = NULL;

  /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
     broju i azurira se pokazivac na glavu liste */
  while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
    {
```

```
205     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
           adresi adresa_glave */
207     pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
209     *adresa_glave = pomocni;
    }
211
    /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
213     funkcije */
    if (*adresa_glave == NULL)
215     return;

217 /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
       pokazuje na cvor cija je vrednost razlicita od trazenog broja.
219    Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
       sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
221    ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
       Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
      */
223 tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL)
225   if (tekuci->sledeci->vrednost == broj) {
        /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
227          prvo cuva u promenljivoj pomocni. */
        pomocni = tekuci->sledeci;
229     /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
           njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
231        sledeci od cvora koji se brise. */
        tekuci->sledeci = pomocni->sledeci;
233     /* Sada treba osloboditi cvor sa vrednoscu broj. */
        free(pomocni);
235   } else {
        /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
237          pomera na sledeci. */
        tekuci = tekuci->sledeci;
239   }
  return;
241 }

243 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
245   Cvor *tekuci = NULL;
    Cvor *pomocni = NULL;
247
    /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
249     broju i azurira se pokazivac na glavu liste. */
    while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
      {
251   /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
         adresi adresa_glave. */
253   pomocni = (*adresa_glave)->sledeci;
      free(*adresa_glave);
```

```
255     *adresa_glave = pomocni;
    }
257
    /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
259      prekida. Isto se radi i ukoliko glava liste sadrzi vrednost koja
         je veca od broja, jer kako je lista sortirana rastuce nema
261      potrebe broj traziti u repu liste. */
    if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
263     return;

265   /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
         na cvor cija vrednost je manja od trazenog broja, kao i svim
267      cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
         je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
269      ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
         cija vrednost je veca od trazenog broja. */
271   tekuci = *adresa_glave;
    while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
       )
273     if (tekuci->sledeci->vrednost == broj) {
        pomocni = tekuci->sledeci;
275     tekuci->sledeci = tekuci->sledeci->sledeci;
        free(pomocni);
277     } else {
        /* Ne treba brisati sledeceg od tekuceg jer je manji od
279        trazenog i tekuci se pomera na sledeci cvor. */
        tekuci = tekuci->sledeci;
281     }
    return;
283 }

285 void ispisi_listu(Cvor * glava)
{
287   /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
         jer se lista nece menjati */
289   putchar('[');
    /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
291      pocetka prema kraju liste. */
    for (; glava != NULL; glava = glava->sledeci) {
293     printf("%d", glava->vrednost);
      if (glava->sledeci != NULL)
295       printf(", ");
    }
297   printf("]\n");
}
```

Datoteka 4.3: *main_a.c*

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

```
4
/* 1) Glavni program */
6 int main()
{
8   /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
10  Cvor *trazeni = NULL;
  int broj;
12
  /* Testiranje dodavanja novog broja na pocetak liste */
14  printf("Unesite brojeve: (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
16    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
18       treba osloboditi pre napustanja programa. */
    if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
20      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
22      exit(EXIT_FAILURE);
    }
24    printf("\tLista: ");
    ispisi_listu(glava);
26  }

28  /* Testiranje funkcije za pretragu liste */
  printf("\nUnesite broj koji se trazi: ");
30  scanf("%d", &broj);

32  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
34    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
36    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

38  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);
40
  exit(EXIT_SUCCESS);
42 }
```

Datoteka 4.4: *main_b.c*

```
#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"
4
/* 2) Glavni program */
6 int main()
{
8   /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
```

```
10    int broj;

12    /* Testira se funkcija za dodavanja novog broja na kraj liste */
      printf("Unesite brojeve: (za kraj CTRL+D)\n");
14    while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16         memorije za nov cvor. Memoriju alociranu za cvorove liste
           treba osloboditi pre napustanja programa. */
18      if (dodaj_na_kraj_liste(&glava, broj) == 1) {
          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
20        oslobodi_listu(&glava);
          exit(EXIT_FAILURE);
22      }
        printf("\tLista: ");
24      ispisi_listu(glava);
      }
26
      /* Testira se funkcije kojom se brise cvor liste */
28    printf("\nUnesite broj koji se brise: ");
      scanf("%d", &broj);
30
      /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
32       procitanom sa ulaza */
      obrisi_cvor(&glava, broj);
34
      printf("Lista nakon brisanja:  ");
36    ispisi_listu(glava);

38    /* Oslobadja se memorija zauzeta listom */
      oslobodi_listu(&glava);
40
      exit(EXIT_SUCCESS);
42  }
```

Datoteka 4.5: *main_c.c*

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include "lista.h"
4
  /* 3) Glavni program */
6 int main()
  {
8   /* Lista je prazna na pocetku */
    Cvor *glava = NULL;
10  Cvor *trazeni = NULL;
    int broj;
12
    /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
14     uredjena neopadajuce */
    printf("Unosite brojeve (za kraj CTRL+D)\n");
```

```
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_sortirano(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcija za pretragu liste */
  printf("\nUnesite broj koji se trazi: ");
  scanf("%d", &broj);

  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  /* Testira se funkcija kojom se brise cvor liste */
  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.2

Datoteka 4.6: *lista.h*

```
#ifndef _LISTA_H_
#define _LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
typedef struct cvor {
```

```
  int vrednost;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
   dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

/* 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);

/* Funkcija dodaje broj u rastuce sortiranu listu tako da nova 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 liste. */
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 ispisuje samo vrednosti cvorova liste razdvojene
   zapetama. */
void ispisi_vrednosti(Cvor * glava);
```

```
58
/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
60    liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);
62
#endif
```

Datoteka 4.7: *lista.c*

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "lista.h"

5  Cvor *napravi_cvor(int broj)
   {
7    /* Alocira se memorija za novi cvor liste i proverava se uspesnost
        alokacije */
9    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
     if (novi == NULL)
11     return NULL;

13   /* Inicijalizacija polja strukture */
     novi->vrednost = broj;
15   novi->sledeci = NULL;

17   /* Vraca se adresa novog cvora */
     return novi;
19 }

21 void oslobodi_listu(Cvor ** adresa_glave)
   {
23   /* Ako je lista vec prazna */
     if (*adresa_glave == NULL)
25     return;

27   /* Ako lista nije prazna, treba osloboditi memoriju. Pre
        oslobadjanja memorije za glavu liste, treba osloboditi rep liste
29    */
     oslobodi_listu(&(*adresa_glave)->sledeci);
31   /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
        glava u pozivajucoj funkciji tako da odgovara praznoj listi */
33   free(*adresa_glave);
     *adresa_glave = NULL;
35 }

37 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
   {
39   /* Kreira se novi cvor i proverava se uspesnost kreiranja */
     Cvor *novi = napravi_cvor(broj);
41   if (novi == NULL)
       return 1;
```

```
43
   /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
45 novi->sledeci = *adresa_glave;
   *adresa_glave = novi;
47
   /* Vraca se indikator uspesnog dodavanja */
49 return 0;
}
51
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
53 {
   /* Ako je lista prazna */
55 if (*adresa_glave == NULL) {

57   /* Novi cvor postaje glava liste */
     Cvor *novi = napravi_cvor(broj);
59   /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
     */
     if (novi == NULL)
61     return 1;

63   /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
        azurira i pokazivacka promenljiva u pozivajucoj funkciji */
65   *adresa_glave = novi;

67   /* Vraca se indikator uspesnog dodavanja */
     return 0;
69 }

71 /* Ako lista nije prazna, broj se dodaje u rep liste. */
   /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
73    novi broj se dodaje u rep liste u rekurzivnom pozivu.
      Informaciju o uspesnosti alokacije u rekurzivnom pozivu funkcija
75    prosledjuje visem rekurzivnom pozivu koji tu informaciju vraca u
      rekurzivni poziv iznad, sve dok se ne vrati u main. Tek je iz
77    main funkcije moguce pristupiti pravom pocetku liste i
      osloboditi je celu, ako ima potrebe. Ako je funkcija vratila 0,
79    onda nije bilo greske. */
   return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
81 }

83 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
85 /* Ako je lista prazna */
   if (*adresa_glave == NULL) {
87
   /* Novi cvor postaje glava liste */
89 Cvor *novi = napravi_cvor(broj);

91 /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1
   */
   if (novi == NULL)
```

```
93        return 1;

95      /* Azurira se glava liste */
        *adresa_glave = novi;
97
        /* Vraca se indikator uspesnog dodavanja */
99      return 0;
    }
101
    /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
103     glavi liste, onda se dodaje na pocetak liste */
    if ((*adresa_glave)->vrednost >= broj)
105   return dodaj_na_pocetak_liste(adresa_glave, broj);

107 /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
        bude sortirana lista. */
109 return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
}
111
Cvor *pretrazi_listu(Cvor * glava, int broj)
113 {
    /* U praznoj listi nema vrednosti */
115 if (glava == NULL)
      return NULL;
117
    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
119 if (glava->vrednost == broj)
      return glava;
121
    /* Inace, pretraga se nastavlja u repu liste */
123 return pretrazi_listu(glava->sledeci, broj);
}
125
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
127 {
    /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
129     vrednosti u glavi liste, jer je lista neopadajuce sortirana */
    if (glava == NULL || glava->vrednost > broj)
131   return NULL;

133 /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
    if (glava->vrednost == broj)
135   return glava;

137 /* Inace, pretraga se nastavlja u repu liste */
    return pretrazi_listu(glava->sledeci, broj);
139 }

141 void obrisi_cvor(Cvor ** adresa_glave, int broj)
    {
143 /* U praznoj listi nema cvorova za brisanje. */
    if (*adresa_glave == NULL)
```

```
145     return;

147   /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
      obrisi_cvor(&(*adresa_glave)->sledeci, broj);
149
      /* Preostaje provera da li glavu liste treba obrisati */
151   if ((*adresa_glave)->vrednost == broj) {
        /* Pomocni pokazuje na cvor koji treba da se obrise */
153     Cvor *pomocni = *adresa_glave;
        /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
155        brise se cvor koji je bio glava liste. */
        *adresa_glave = (*adresa_glave)->sledeci;
157     free(pomocni);
      }
159
}
161
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
163 {
      /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
165        od broja, kako je lista sortirana rastuce nema potrebe broj
          traziti u repu liste i zato se funkcija prekida */
167   if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
        return;
169
      /* Brisu se cvorovi iz repa koji imaju vrednost broj */
171   obrisi_cvor(&(*adresa_glave)->sledeci, broj);

173   /* Preostaje provera da li glavu liste treba obrisati */
      if ((*adresa_glave)->vrednost == broj) {
175     /* Pomocni pokazuje na cvor koji treba da se obrise */
        Cvor *pomocni = *adresa_glave;
177     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
           brise se cvor koji je bio glava liste */
179     *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
181   }
}
183
void ispisi_vrednosti(Cvor * glava)
185 {
      /* Prazna lista */
187   if (glava == NULL)
        return;
189
      /* Ispisuje se vrednost u glavi liste */
191   printf("%d", glava->vrednost);

193   /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
        se poziva ista funkcija za ispis ostalih. */
195   if (glava->sledeci != NULL) {
        printf(", ");
```

```
197     ispisi_vrednosti(glava->sledeci);
  }
199 }

201 void ispisi_listu(Cvor * glava)
{
203   /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
205     na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
        zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
207     rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
        elemente razdvojene zapetom i razmakom. */
209   putchar('[');
  ispisi_vrednosti(glava);
211   printf("]\n");
}
```

Rešenje 4.3

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

5 #define MAX_DUZINA 20

7 /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
     broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
9 typedef struct _Cvor {
    char etiketa[20];
11   unsigned broj_pojavljivanja;
    struct _Cvor *sledeci;
13 } Cvor;

15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
     ili NULL ako alokacija nije uspesno izvrsena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
{
19   /* Alocira se memorija za novi cvor liste i proverava se uspesnost
        alokacije */
21   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
23     return NULL;

25   /* Inicijalizacija polja strukture */
    novi->broj_pojavljivanja = br;
27   strcpy(novi->etiketa, etiketa);
    novi->sledeci = NULL;
29
   /* Vraca se adresa novog cvora */
31   return novi;
}
```

```
33
/* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
35 void oslobodi_listu(Cvor ** adresa_glave)
{
37   Cvor *pomocni = NULL;

39   /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
        azurira se vrednost glave liste */
41   while (*adresa_glave != NULL) {
       pomocni = (*adresa_glave)->sledeci;
43     free(*adresa_glave);
       *adresa_glave = pomocni;
45   }
     /* Nakon izlaska iz petlje pokazivac glava u main funkciji koji se
47      nalazi na adresi adresa_glave bice postavljen na NULL vrednost.
        */
}
49

/* Funkcija dodaje novi cvor na pocetak liste. Povratna vrednost je 1
51    ako je doslo do greske pri alokaciji memorije za nov cvor, odnosno
      0 */
53 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
                             char *etiketa)
55 {
     /* Kreira se novi cvor i proverava se uspesnost alokacije */
57   Cvor *novi = napravi_cvor(br, etiketa);
     if (novi == NULL)
59     return 1;

61   /* Dodaje se novi cvor na pocetak liste */
     novi->sledeci = *adresa_glave;
63   *adresa_glave = novi;

65   /* Vraca se indikator uspesnog dodavanja */
     return 0;
67 }

69 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
      NULL ako takav cvor ne postoji u listi */
71 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
{
73   Cvor *tekuci;

75   /* Obilazi se cvor po cvor liste */
     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
77     /* Ako tekuci cvor sadrzi trazenu etiketu, vracamo njegovu
          vrednost */
79     if (strcmp(tekuci->etiketa, etiketa) == 0)
         return tekuci;
81
     /* Cvor nije pronadjen */
83   return NULL;
```

```
}

/* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Cvor * glava)
{
  /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
     broj njihovog pojavljivanja u HTML datoteci. */
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
}

/* Glavni program */
int main(int argc, char **argv)
{
  /* Provera se da li je program pozvan sa ispravnim brojem
     argumenata. */
  if (argc != 2) {
    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out datoteka.html!\n")
    ;
    exit(EXIT_FAILURE);
  }

  /* Priprema 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);
  }

  char c;
  int i = 0;
  char procitana[MAX_DUZINA];
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;

  /* Cita se karakter po karakter datoteke sve dok se ne procita cela
     datoteka */
  while ((c = fgetc(in)) != EOF) {

    /* Proverava se da li se pocinje sa citanjem nove etikete */
    if (c == '<') {
      /* Proverava se da li se cita zatvarajuca etiketa */
      if ((c = fgetc(in)) == '/') {
        i = 0;
        while ((c = fgetc(in)) != '>')
          procitana[i++] = c;
      }
      /* Cita se otvarajuca etiketa */
      else {
```

```
135         i = 0;
            procitana[i++] = c;
137         while ((c = fgetc(in)) != ' ' && c != '>')
              procitana[i++] = c;
139       }
          procitana[i] = '\0';
141
          /* Trazi se procitana etiketa medju postojecim cvorovima liste.
143          Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
        sa
               brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
145            etikete */
          trazeni = pretrazi_listu(glava, procitana);
147       if (trazeni == NULL) {
            if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
149           fprintf(stderr, "Neuspela alokacija za nov cvor\n");
              oslobodi_listu(&glava);
151           exit(EXIT_FAILURE);
            }
153       } else
            trazeni->broj_pojavljivanja++;
155     }
    }
157
    /* Zatvaranje datoteke */
159 fclose(in);

161 /* Ispisuje se sadrzaj cvorova liste */
    ispisi_listu(glava);
163
    /* Oslobadja se memorija zauzeta listom */
165 oslobodi_listu(&glava);

167 exit(EXIT_SUCCESS);
}
```

Rešenje 4.4

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21

8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
     studentu */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
12  char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
```

```
14    struct _Cvor *sledeci;
} Cvor;

/* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
18     novi cvor ili NULL ukoliko je doslo do greske */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
    /* Alocira se memorija za novi cvor liste i proverava se uspesnost
22       alokacije */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24  if (novi == NULL)
      return NULL;
26
    /* Inicijalizacija polja strukture */
28  strcpy(novi->broj_indeksa, broj_indeksa);
    strcpy(novi->ime, ime);
30  strcpy(novi->prezime, prezime);
    novi->sledeci = NULL;
32
    /* Vraca se adresa novog cvora */
34  return novi;
}
36
/* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void oslobodi_listu(Cvor ** adresa_glave)
{
40  /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
    if (*adresa_glave == NULL)
42    return;

44  /* Rekurzivnim pozivom se oslobadja rep liste */
    oslobodi_listu(&(*adresa_glave)->sledeci);
46
    /* Potom se oslobadja i glava liste */
48  free(*adresa_glave);

50  /* Proglasava se lista praznom */
    *adresa_glave = NULL;
52 }

54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
     do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
                           char *ime, char *prezime)
58 {
    /* Kreira se novi cvor i proverava se uspesnost alokacije */
60  Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
    if (novi == NULL)
62    return 1;

64  /* Dodaje se novi cvor na pocetak liste */
    novi->sledeci = *adresa_glave;
```

```
66    *adresa_glave = novi;

68    /* Vraca se indikator uspesnog dodavanja */
      return 0;
70  }

72  /* Funkcija ispisuje sadrzaj cvorova liste. */
    void ispisi_listu(Cvor * glava)
74  {
      /* Pocevsi od glave liste */
76    for (; glava != NULL; glava = glava->sledeci)
        printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
78             glava->prezime);
    }
80
    /* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj indeksa.
82     U suprotnom funkcija vraca NULL */
    Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
84  {
      /* Ako je lista prazna, ne postoji trazeni cvor */
86    if (glava == NULL)
        return NULL;
88
      /* Poredi se trazeni broj indeksa sa brojem indeksa u glavi liste
90      */
      if (!strcmp(glava->broj_indeksa, broj_indeksa))
        return glava;
92
      /* Ukoliko u glavi liste nije trazeni indeks, pretraga se nastavlja
94       u repu liste */
      return pretrazi_listu(glava->sledeci, broj_indeksa);
96  }

98  /* Glavni program */
    int main(int argc, char **argv)
100 {
      /* Argumenti komandne linije su neophodni jer se iz komandne linije
102      dobija ime datoteke sa informacijama o studentima */
      if (argc != 2) {
104     fprintf(stderr,
                "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
106     exit(EXIT_FAILURE);
      }
108
      /* Priprema datoteke za citanje */
110   FILE *in = NULL;
      in = fopen(argv[1], "r");
112   if (in == NULL) {
        fprintf(stderr,
114             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
        exit(EXIT_FAILURE);
116   }
```

```
118   /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
          listu */
120   char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
      char broj_indeksa[MAX_INDEKS];
122   Cvor *glava = NULL;
      Cvor *trazeni = NULL;
124
      /* Ucitavanje vrednosti u listu */
126   while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
        if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
128       fprintf(stderr, "Neuspela alokacija za nov cvor\n");
          oslobodi_listu(&glava);
130       exit(EXIT_FAILURE);
        }
132
      /* Datoteka vise nije potrebna i zatvara se. */
134   fclose(in);

136   /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
      while (scanf("%s", broj_indeksa) != EOF) {
138     trazeni = pretrazi_listu(glava, broj_indeksa);
        if (trazeni == NULL)
140       printf("ne\n");
        else
142       printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
      }
144
      /* Oslobadja se memorija zauzeta za cvorove liste. */
146   oslobodi_listu(&glava);

148   exit(EXIT_SUCCESS);
    }
```

Rešenje 4.6

```
1   #include <stdio.h>
    #include <stdlib.h>
3   #include "lista.h"

5   /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
        na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7       pokazivaca postojecih cvorova listi */
    Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9   {
      /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
11    Cvor *lista1 = *adresa_glave_1;
      Cvor *lista2 = *adresa_glave_2;
13
      /* Pokazivac na pocetni cvor rezultujuce liste */
15    Cvor *rezultujuca = NULL;
```

```
  Cvor *tekuci = NULL;

  /* Ako su obe liste prazne i rezultat je prazna lista */
  if (lista1 == NULL && lista2 == NULL)
    return NULL;

  /* Ako je prva lista prazna, rezultat je druga lista */
  if (lista1 == NULL)
    return lista2;

  /* Ako je druga lista prazna, rezultat je prva lista */
  if (lista2 == NULL)
    return lista1;

  /* Odredjuje se prvi cvor rezultujuce liste - to je ili prvi cvor
     liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
     koji sadrzi manju vrednost */
  if (lista1->vrednost < lista2->vrednost) {
    rezultujuca = lista1;
    lista1 = lista1->sledeci;
  } else {
    rezultujuca = lista2;
    lista2 = lista2->sledeci;
  }
  /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste, ne
     sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
     koji sadrzi adresu trenutnog cvora rezultujuce liste */
  tekuci = rezultujuca;

  /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi cvor
     tako da bude uredjena neopadajuce. Pokazivac na listu iz koje se
     uzima cvor se azurira tako da pokazuje na sledeci cvor */
  while (lista1 != NULL && lista2 != NULL) {
    if (lista1->vrednost < lista2->vrednost) {
      tekuci->sledeci = lista1;
      lista1 = lista1->sledeci;
    } else {
      tekuci->sledeci = lista2;
      lista2 = lista2->sledeci;
    }
    tekuci = tekuci->sledeci;
  }

  /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
     rezultujucu listu treba nadovezati ostatak druge liste */
  if (lista1 == NULL)
    tekuci->sledeci = lista2;
  else
    /* U suprotnom treba nadovezati ostatak prve liste */
    tekuci->sledeci = lista1;

  /* Preko adresa glava polaznih listi vrednosti pokazivaca u
```

```
     pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
     prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
     se moze doci prateci pokazivace iz glave rezultujuce liste, tako
     da stare pokazivace treba postaviti na NULL. */
  *adresa_glave_1 = NULL;
  *adresa_glave_2 = NULL;

  return rezultujuca;
}

int main(int argc, char **argv)
{
  /* Argumenti komandne linije su neophodni */
  if (argc != 3) {
    fprintf(stderr,
            "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
    exit(EXIT_FAILURE);
  }

  /* Otvaramo datoteke u kojima se nalaze elementi listi */
  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);
  }

  /* Liste su na pocetku prazne */
  int broj;
  Cvor *lista1 = NULL;
  Cvor *lista2 = 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);

  /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
  fclose(in1);
  fclose(in2);

```

```
  /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
     objedinjavanjem listi */
  Cvor *rezultat = objedini(&lista1, &lista2);

  /* Ispis rezultujuce liste. */
  ispisi_listu(rezultat);

  /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.
     Njenim oslobadjanjem bice oslobodjena sva zauzeta memorija. */
  oslobodi_listu(&rezultat);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.8

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

/* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
   predstavlja zagradu koja se koristi i pokazivac na sledeci cvor
   liste */
typedef struct cvor {
  char zagrada;
  struct cvor *sledeci;
} Cvor;

/* Funkcija koja oslobadja memoriju zauzetu stekom */
void oslobodi_stek(Cvor ** stek)
{
  Cvor *tekuci;
  Cvor *pomocni;

  /* Oslobadja se cvor po cvor steka */
  tekuci = *stek;
  while (tekuci != NULL) {
    pomocni = tekuci->sledeci;
    free(tekuci);
    tekuci = pomocni;
  }

  /* Stek se proglasava praznim */
  *stek = NULL;
}

/* Glavni program */
int main()
{
  /* Stek je na pocetku prazan */
  Cvor *stek = NULL;
  FILE *ulaz = NULL;
```

```
  char c;
37  Cvor *pomocni = NULL;

39  /* Otvaranje datotoke za citanje izraza */
    ulaz = fopen("izraz.txt", "r");
41  if (ulaz == NULL) {
      fprintf(stderr,
43            "Greska prilikom otvaranja datoteke izraz.txt!\n");
      exit(EXIT_FAILURE);
45  }

47  /* Cita se karakter po karakter iz datoteke dok se ne dodje do
       kraja */
49  while ((c = fgetc(ulaz)) != EOF) {
      /* Ako je ucitana otvorena zagrada, stavlja se na stek */
51    if (c == '(' || c == '{' || c == '[') {
        /* Alocira se memorija za novi cvor liste i proverava se
53         uspesnost alokacije */
        pomocni = (Cvor *) malloc(sizeof(Cvor));
55      if (pomocni == NULL) {
          fprintf(stderr, "Greska prilikom alokacije memorije!\n");
57        /* Oslobadja se memorija zauzeta stekom */
          oslobodi_stek(&stek);
59        /* I prekida se sa izvrsavanjem programa */
          exit(EXIT_FAILURE);
61      }

63      /* Inicijalizacija polja strukture */
        pomocni->zagrada = c;
65
        /* Promena vrha steka */
67      pomocni->sledeci = stek;
        stek = pomocni;
69    }
      /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
71       prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
         otvorena zagrada */
73    else {
        if (c == ')' || c == '}' || c == ']') {
75        if (stek != NULL && ((stek->zagrada == '(' && c == ')')
                               || (stek->zagrada == '{' && c == '}')
77                             || (stek->zagrada == '[' && c == ']')))
    {
          /* Sa vrha steka se uklanja otvorena zagrada */
79        pomocni = stek->sledeci;
          free(stek);
81        stek = pomocni;
        } else {
83        /* Inace, zakljucujemo da zagrade u izrazu nisu ispravno
             uparene */
85        break;
        }
```

```
87        }
      }
89    }

91    /* Procitana je cela datoteka i treba je zatvoriti */
      fclose(ulaz);
93
      /* Ako je stek prazan i procitana je cela datoteka, zagrade su
95       ispravno uparene */
      if (stek == NULL && c == EOF)
97      printf("Zagrade su ispravno uparene.\n");
      else {
99      /* U suprotnom se zakljucuje da zagrade nisu ispravno uparene */
        printf("Zagrade nisu ispravno uparene.\n");
101     /* Oslobadja se memorija koja je ostala zauzeta stekom */
        oslobodi_stek(&stek);
103   }

105   exit(EXIT_SUCCESS);
}
```

Rešenje 4.9

Datoteka 4.8: *stek.h*

```
#ifndef _STEK_H_
2 #define _STEK_H_

4 #include <stdio.h>
  #include <stdlib.h>
6 #include <string.h>
  #include <ctype.h>
8
  #define MAX 100
10
  #define OTVORENA 1
12 #define ZATVORENA 2

14 #define VAN_ETIKETE 0
  #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2

18 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
      pokazivac na sledeci cvor */
20 typedef struct cvor {
     char etiketa[MAX];
22   struct cvor *sledeci;
   } Cvor;
24
   /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
```

```
   adresu ili NULL ako alokacija nije bila uspesna */
Cvor *napravi_cvor(char *etiketa);

/* Funkcija oslobadja memoriju zauzetu stekom */
void oslobodi_stek(Cvor ** adresa_vrha);

/* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske pri
   alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);

/* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
   pokazivac razlicit od NULL, tada u niz karaktera na koji on
   pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
   suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
   samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
   suprotnom */
int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);

/* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
   steka. Ukoliko je stek prazan, vraca NULL */
char *vrh_steka(Cvor * vrh);

/* Funkcija prikazuje stek od vrha prema dnu */
void prikazi_stek(Cvor * vrh);

/* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
   etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
   Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
   procita etiketa. Vraca OTVORENA, ako je procitana otvorena
   etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
int uzmi_etiketu(FILE * f, char *etiketa);

#endif
```

Datoteka 4.9: *stek.c*

```
#include "stek.h"

Cvor *napravi_cvor(char *etiketa)
{
  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja u novom cvoru */
  if (strlen(etiketa) >= MAX) {
    fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
    etiketa[MAX - 1] = '\0';
  }
```

```
16    strcpy(novi->etiketa, etiketa);
      novi->sledeci = NULL;
18
      /* Vraca se adresa novog cvora */
20    return novi;
  }
22
  void oslobodi_stek(Cvor ** adresa_vrha)
24{
    Cvor *pomocni;
26
    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
28  while (*adresa_vrha != NULL) {
      /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
30       osloboditi cvor koji predstavlja vrh steka */
      pomocni = *adresa_vrha;
32    /* Sledeci cvor je novi vrh steka */
      *adresa_vrha = (*adresa_vrha)->sledeci;
34    free(pomocni);
    }
36
    /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
38     adresa_vrha ce pokazivati na NULL. */
  }
40
  int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
42{
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
44  Cvor *novi = napravi_cvor(etiketa);
    if (novi == NULL)
46    return 1;

48  /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
    novi->sledeci = *adresa_vrha;
50  *adresa_vrha = novi;
    return 0;
52}

54int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
{
56  Cvor *pomocni;

58  /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
       vraca se 0 */
60  if (*adresa_vrha == NULL)
      return 0;
62
    /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
64     adresu kopira etiketa sa vrha steka */
    if (etiketa != NULL)
66    strcpy(etiketa, (*adresa_vrha)->etiketa);
```

```
68    /* Element sa vrha steka se uklanja */
      pomocni = *adresa_vrha;
70    *adresa_vrha = (*adresa_vrha)->sledeci;
      free(pomocni);
72
      /* Vraca se indikator uspesno izvrsene radnje */
74    return 1;
}
76
char *vrh_steka(Cvor * vrh)
78{
      /* Prazan stek nema cvor koji je vrh i vraca se NULL */
80    if (vrh == NULL)
        return NULL;
82
      /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
84       koji je na vrhu steka. */
      return vrh->etiketa;
86}

88void prikazi_stek(Cvor * vrh)
{
90    /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
      for (; vrh != NULL; vrh = vrh->sledeci)
92      printf("<%s>\n", vrh->etiketa);
}
94
int uzmi_etiketu(FILE * f, char *etiketa)
96{
      int c;
98    int i = 0;
      /* Stanje predstavlja informaciju dokle se stalo sa citanjem
100      etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
         nije zapoceto citanje. */
102   /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
         OTVORENA ili ZATVORENA. */
104   int stanje = VAN_ETIKETE;
      int tip;
106
      /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
108      to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
         samo malim slovima. */
110   while ((c = fgetc(f)) != EOF) {
        switch (stanje) {
112     case VAN_ETIKETE:
          if (c == '<')
114         stanje = PROCITANO_MANJE;
          break;
116     case PROCITANO_MANJE:
          if (c == '/') {
118         /* Cita se zatvorena etiketa */
            tip = ZATVORENA;
```

```
120       } else {
            if (isalpha(c)) {
122           /* Cita se otvorena etiketa */
              tip = OTVORENA;
124           etiketa[i++] = tolower(c);
            }
126       }
          /* Od sada se cita etiketa i zato se menja stanje */
128       stanje = U_ETIKETI;
          break;
130     case U_ETIKETI:
          if (isalpha(c) && i < MAX - 1) {
132         /* Ako je procitani karakter slovo i nije prekoracena
               dozvoljena duzina etikete, procitani karakter se smanjuje
134            i smesta u etiketu */
            etiketa[i++] = tolower(c);
136       } else {
            /* Inace, staje se sa citanjem etikete. Korektno se zavrsava
138            niska koja sadrzi procitanu etiketu i vraca se njen tip */
            etiketa[i] = '\0';
140         return tip;
          }
142       break;
        }
144   }
    /* Doslo se do kraja datoteke pre nego sto je procitana naredna
146      etiketa i vraca se EOF. */
  return EOF;
148 }
```

Datoteka 4.10: *main.c*

```
  #include "stek.h"
2
  /* Glavni program */
4 int main(int argc, char **argv)
  {
6   /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
       nije procitana. */
8   Cvor *vrh = NULL;
    char etiketa[MAX];
10  int tip;
    int uparene = 1;
12  FILE *f = NULL;

14  /* Ime datoteke se preuzima iz komandne linije. */
    if (argc < 2) {
16    fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
      exit(EXIT_FAILURE);
18  }
```

```
20  /* Datoteka se otvara za citanje */
    if ((f = fopen(argv[1], "r")) == NULL) {
22    fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
              argv[1]);
24    exit(EXIT_FAILURE);
    }
26
    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
28  while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
      /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
30       etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
         potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
32       koje nemaju sadrzaj (npr link). Zbog jednostavnosti
         pretpostavlja se da njih nema u HTML dokumentu. */
34    if (tip == OTVORENA) {
        if (strcmp(etiketa, "br") != 0
36          && strcmp(etiketa, "hr") != 0
            && strcmp(etiketa, "meta") != 0)
38        if (potisni_na_stek(&vrh, etiketa) == 1) {
            fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40          oslobodi_stek(&vrh);
            exit(EXIT_FAILURE);
42        }
      }
44    /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
         u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
46       uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
         je taj uslov ispunjen, skida se sa steka, jer je upravo
48       zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
         nisu pravilno uparene. */
50    else if (tip == ZATVORENA) {
        if (vrh_steka(vrh) != NULL
52          && strcmp(vrh_steka(vrh), etiketa) == 0)
          skini_sa_steka(&vrh, NULL);
54      else {
          printf("Etikete nisu pravilno uparene\n");
56        printf("(nadjena je etiketa </%s>", etiketa);
          if (vrh_steka(vrh) != NULL)
58          printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh))
    ;
          else
60          printf(" koja nije otvorena)\n");
          uparene = 0;
62        break;
        }
64    }
    }
66  /* Zavrseno je citanje i datoteka se zatvara */
    fclose(f);
68
    /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
70     da bude prazan. Ukoliko nije, tada postoje etikete koje su
```

```
      ostale otvorene */
  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);
    }
  }

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.10

Datoteka 4.11: *red.h*

```
#ifndef _RED_H_
#define _RED_H_

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

#define MAX 1000
#define JMBG_DUZINA 14

/* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
   opis njegovog zahteva. */
typedef struct {
  char jmbg[JMBG_DUZINA];
  char opis[MAX];
} Zahtev;

/* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
   korisnika i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  Zahtev nalog;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
   poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
   greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
   treba smestiti u novi cvor zbog smestanja manjeg podatka na
   sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
   bajtovima(B) u odnosu na strukturu Zahtev. */
Cvor *napravi_cvor(Zahtev * zahtev);

```

```
/* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

/* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
   greske pri alokaciji memorije za novi cvor, inace vraca 0. */
int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                Zahtev * zahtev);

/* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
   pokazivac razlicit od NULL, tada se u strukturu na koju on
   pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
   suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
   suprotnom. */
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

/* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
   na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
Zahtev *pocetak_reda(Cvor * pocetak);

/* Funkcija prikazuje sadrzaj reda. */
void prikazi_red(Cvor * pocetak);

#endif
```

Datoteka 4.12: *red.c*

```
#include "red.h"

Cvor *napravi_cvor(Zahtev * zahtev)
{
  /* Alocira se memorija za novi cvor liste i proverava uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->nalog = *zahtev;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
{
  Cvor *pomocni = NULL;

  /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
  while (*pocetak != NULL) {
```

```
25      /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
           osloboditi cvor sa pocetka reda */
27      pomocni = *pocetak;
        *pocetak = (*pocetak)->sledeci;
29      free(pomocni);
    }
31  /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
       treba postaviti na NULL. */
33  *kraj = NULL;
}
35
int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
37                  Zahtev * zahtev)
{
39  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(zahtev);
41  if (novi == NULL)
      return 1;
43
    /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
45     kraj, to je podjednako efikasno kao dodavanje na pocetak liste
       */
    if (*adresa_kraja != NULL) {
47    (*adresa_kraja)->sledeci = novi;
      *adresa_kraja = novi;
49  } else {
      /* Ako je red bio ranije prazan */
51    *adresa_pocetka = novi;
      *adresa_kraja = novi;
53  }

55  /* Vraca se indikator uspesnog dodavanja */
    return 0;
57  }

59  int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                      Zahtev * zahtev)
61  {
    Cvor *pomocni = NULL;
63
    /* Ako je red prazan */
65  if (*adresa_pocetka == NULL)
      return 0;
67
    /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
69     zahtev koji je na pocetku reda.  */
    if (zahtev != NULL)
71    *zahtev = (*adresa_pocetka)->nalog;

73  /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
       na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
75     u redu. */
```

```
  pomocni = *adresa_pocetka;
  *adresa_pocetka = (*adresa_pocetka)->sledeci;
  free(pomocni);

  /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
     potrebno je azurirati i vrednost pokazivaca na adresi
     adresa_kraja na NULL */
  if (*adresa_pocetka == NULL)
    *adresa_kraja = NULL;

  return 1;
}

Zahtev *pocetak_reda(Cvor * pocetak)
{
  /* U praznom redu nema zahteva */
  if (pocetak == NULL)
    return NULL;

  /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
  return &(pocetak->nalog);
}

void prikazi_red(Cvor * pocetak)
{
  /* Prikazuje se sadrzaj reda od pocetka prema kraju */
  for (; pocetak != NULL; pocetak = pocetak->sledeci)
    printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

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

Datoteka 4.13: *main.c*

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

#define VREME_ZA_PAUZU 5

/* Glavni program */
int main(int argc, char **argv)
{
  /* Red je prazan. */
  Cvor *pocetak = NULL, *kraj = NULL;
  Zahtev nov_zahtev;
  Zahtev *sledeci = NULL;
  char odgovor[3];
  int broj_usluzenih = 0;
```

```
18  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
       broja i opisa potrebne usluge. */
20  printf("Sluzbenik evidentira korisnicke zahteve:\n");
    while (1) {
22
      /* Ucitava se JMBG */
24    printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
      if (scanf("%s", nov_zahtev.jmbg) == EOF)
26      break;

28    /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
         JMBG broja procitao i da bi fgets nakon toga procitala
30       ispravan red sa opisom zahteva */
      getchar();
32
      /* Ucitava se opis problema */
34    printf("\tOpis problema: ");
      fgets(nov_zahtev.opis, MAX - 1, stdin);
36    /* Ako je poslednji karakter nov red, eliminise se */
      if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
38      nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

40    /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
      if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
42      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
        oslobodi_red(&pocetak, &kraj);
44      exit(EXIT_FAILURE);
      }
46  }

48  /* Otvaranje datoteke za dopisivanje izvestaja */
    FILE *izlaz = fopen("izvestaj.txt", "a");
50  if (izlaz == NULL) {
      fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
52    exit(EXIT_FAILURE);
    }
54
    /* Dokle god ima korisnika u redu, treba ih usluziti */
56  while (1) {
      sledeci = pocetak_reda(pocetak);
58    /* Ako nema nikog vise u redu, prekida se petlja */
      if (sledeci == NULL)
60      break;

62    printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
      printf("i zahtevom: %s\n", sledeci->opis);
64
      skini_sa_reda(&pocetak, &kraj, &nov_zahtev);
66
      broj_usluzenih++;
68
      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
```

```
70      scanf("%s", odgovor);

72      if (strcmp(odgovor, "Da") == 0)
          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
74      else
          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
76                nov_zahtev.opis);

78      if (broj_usluzenih == VREME_ZA_PAUZU) {
          printf("\nDa li je kraj smene? [Da/Ne] ");
80        scanf("%s", odgovor);

82        if (strcmp(odgovor, "Da") == 0)
            break;
84        else
            broj_usluzenih = 0;
86      }
    }
88
    /*****************************************************************
90    Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:

92    while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
        printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
94            nov_zahtev.jmbg);
        printf("sa zahtevom: %s\n", nov_zahtev.opis);
96      broj_usluzenih++;

98      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
        scanf("%s", odgovor);
100     if (strcmp(odgovor, "Da") == 0)
          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
102     else
          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
104               nov_zahtev.jmbg, nov_zahtev.opis);

106     if (broj_usluzenih == VREME_ZA_PAUZU) {
          printf("\nDa li je kraj smene? [Da/Ne] ");
108       scanf("%s", odgovor);
          if (strcmp(odgovor, "Da") == 0)
110         break;
          else
112         broj_usluzenih = 0;
        }
114   }
    *****************************************************************/
116 /* Datoteka vise nije potrebna i treba je zatvoriti. */
    fclose(izlaz);
118
    /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
120    neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
       koju zauzima red sa neobradjenim zahtevima korisnika. */
```

```
122   oslobodi_red(&pocetak, &kraj);

124   exit(EXIT_SUCCESS);
}
```

Rešenje 4.11

Datoteka 4.14: *dvostruko_povezana_lista.h*

```
1  #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
   #define _DVOSTRUKO_POVEZANA_LISTA_H_
3
   /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
5     vrednost i pokazivace na sledeci i prethodni cvor liste. */
   typedef struct cvor {
7    int vrednost;
     struct cvor *sledeci;
9    struct cvor *prethodni;
   } Cvor;
11
   /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
13    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
      na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
15 Cvor *napravi_cvor(int broj);

17 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
      ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
19    adresi adresa_kraja. */
   void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);
21
   /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
23    bilo greske pri alokaciji memorije, inace vraca 0. */
   int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
25                             adresa_kraja, int broj);

27 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
      pri alokaciji memorije, inace vraca 0. */
29 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                          int broj);
31
   /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
33    novi cvor sa vrednoscu broj. */
   Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);
35
   /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
37    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
   int dodaj_iza(Cvor * tekuci, int broj);
39
   /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
41    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
```

```
   inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                    broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
   NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
   neopadajuce sortirana. Vraca pokazivac na cvor liste 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 ** adresa_kraja, Cvor *
                   tekuci);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                 broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
   sortirana. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                 adresa_kraja, int broj);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

/* Funkcija prikazuje cvorove liste pocevsi od kraja ka glavi liste.
    */
void ispisi_listu_unazad(Cvor * kraj);

#endif
```

Datoteka 4.15: *dvostruko_povezana_lista.c*

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

Cvor *napravi_cvor(int broj)
```

```
{
  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
{
  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;
  }
  /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
     treba postaviti na NULL */
  *adresa_kraja = NULL;
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
                           adresa_kraja, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Sledbenik novog cvora je glava stare liste */
  novi->sledeci = *adresa_glave;

  /* Ako stara lista nije bila prazna, onda prethodni cvor glave
     treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
     krajnji */
  if (*adresa_glave != NULL)
    (*adresa_glave)->prethodni = novi;
  else
    *adresa_kraja = novi;
```

```
58    /* Novi cvor je nova glava liste */
      *adresa_glave = novi;
60
      /* Vraca se indikator uspesnog dodavanja */
62    return 0;
}
64
int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
66                          int broj)
{
68    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
      Cvor *novi = napravi_cvor(broj);
70    if (novi == NULL)
        return 1;
72
      /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
74       ujedno i cela lista. Azurira se vrednost na koju pokazuju
         adresa_glave i adresa_kraja */
76    if (*adresa_glave == NULL) {
        *adresa_glave = novi;
78      *adresa_kraja = novi;
      } else {
80      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
           sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
82         liste */
        (*adresa_kraja)->sledeci = novi;
84      novi->prethodni = (*adresa_kraja);
        *adresa_kraja = novi;
86    }

88    /* Vraca se indikator uspesnog dodavanja */
      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 sledeci cvor ili ne postoji ili ima
100      vrednost vecu ili jednaku od vrednosti novog cvora.

102      Zbog izracunavanja izraza u C-u prvi deo konjunkcije 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, nailaskom na cvor ciji sledeci ima
          vrednost vecu od broj */
112   return glava;
}
114
int dodaj_iza(Cvor * tekuci, int broj)
116 {
  /* Kreira se novi cvor i provera se uspesnost kreiranja */
118   Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
120     return 1;

122   novi->sledeci = tekuci->sledeci;
  novi->prethodni = tekuci;
124
  /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
126      a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
         na ispravne adrese */
128   if (tekuci->sledeci != NULL)
    tekuci->sledeci->prethodni = novi;
130   tekuci->sledeci = novi;

132   /* Vraca se indikator uspesnog dodavanja */
  return 0;
134 }

136 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                      broj)
138 {
  /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste
     */
140   if (*adresa_glave == NULL) {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
142     Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
144       return 1;

146     /* Azuriraju se vrednosti pocetka i kraja liste */
    *adresa_glave = novi;
148     *adresa_kraja = novi;

150     /* Vraca se indikator uspesnog dodavanja */
    return 0;
152   }

154   /* Ukoliko je vrednost glave liste veca ili jednaka od nove
         vrednosti onda novi cvor treba staviti na pocetak liste */
156   if ((*adresa_glave)->vrednost >= broj) {
    return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
158   }

160   /* Pronazi se cvor iza koga treba uvezati novi cvor */
```

```
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
  if (dodaj_iza(pomocni, broj) == 1)
    return 1;
  /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
     novi cvor poslednji u listi. */
  if (pomocni == *adresa_kraja)
    *adresa_kraja = novi;

  return 0;
}

Cvor *pretrazi_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  for (; glava != NULL; glava = glava->sledeci)
    /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
       se obustavlja */
    if (glava->vrednost == broj)
      return glava;

  /* Nema trazenog broja u listi i vraca se NULL */
  return NULL;
}

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
  for (; glava != NULL && glava->vrednost <= broj;
       glava = glava->sledeci)
    /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
       se obustavlja */
    if (glava->vrednost == broj)
      return glava;

  /* Nema trazenog broja u listi i bice vraceno NULL */
  return NULL;
}

/* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
   lako realizovati jer on sadrzi pokazivace na svog sledbenika i
   prethodnika u listi. U funkciji se bise cvor zadat argumentom
   tekuci */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
                   tekuci)
{
  /* Ako je tekuci NULL pokazivac, nema sta da se brise */
  if (tekuci == NULL)
    return;

  /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
```

```
     njegov sledbenik bude sledbenik tekuceg cvora */
214  if (tekuci->prethodni != NULL)
       tekuci->prethodni->sledeci = tekuci->sledeci;
216
     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
218     treba da bude prethodnik tekuceg cvora */
     if (tekuci->sledeci != NULL)
220    tekuci->sledeci->prethodni = tekuci->prethodni;

222  /* Ako je glava cvor koji se brise, nova glava liste ce biti
        sledbenik stare glave */
224  if (tekuci == *adresa_glave)
       *adresa_glave = tekuci->sledeci;
226
     /* Ako je cvor koji se brise poslednji u listi, azurira se i
228     pokazivac na kraj liste */
     if (tekuci == *adresa_kraja)
230    *adresa_kraja = tekuci->prethodni;

232  /* Oslobadja se dinamicki alociran prostor za cvor tekuci */
     free(tekuci);
234 }

236 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
        )
    {
238   Cvor *tekuci = *adresa_glave;

240   /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
        cvorovi se brisu iz liste. */
242   while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
244 }

246 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                     adresa_kraja, int broj)
248 {
      Cvor *tekuci = *adresa_glave;
250
      /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
252        takvi cvorovi se brisu iz liste. */
      while ((tekuci =
254             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
        obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
256 }

258 void ispisi_listu(Cvor * glava)
    {
260   putchar('[');
      /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
262        pocetka prema kraju liste */
      for (; glava != NULL; glava = glava->sledeci) {
```

```
264     printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
266       printf(", ");
    }
268
    printf("]\n");
270 }

272 void ispisi_listu_unazad(Cvor * kraj)
{
274   putchar('[');
    /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
276      prema pocetku liste */
    for (; kraj != NULL; kraj = kraj->prethodni) {
278     printf("%d", kraj->vrednost);
        if (kraj->prethodni != NULL)
280       printf(", ");
    }
282 printf("]\n");
}
```

Datoteka 4.16: *main_a.c*

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include "dvostruko_povezana_lista.h"

5  /* 1) Glavni program */
   int main()
7  {
     /* Lista je prazna na pocetku */
9    /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
        da bi operacije poput dodavanja na kraj liste i ispisivanja
11      liste unazad bile efikasne poput dodavanja na pocetak liste i
        ispisivanja liste od pocetnog do poslednjeg cvora. */
13   Cvor *glava = NULL;
     Cvor *kraj = NULL;
15   Cvor *trazeni = NULL;
     int broj;
17
     /* Testira se funkcija za dodavanja novog broja na pocetak liste */
19   printf("Unesite brojeve: (za kraj CTRL+D)\n");
     while (scanf("%d", &broj) > 0) {
21     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
          memorije za novi cvor. Memoriju alociranu za cvorove liste
23        treba osloboditi pre napustanja programa */
       if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
25       fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
         oslobodi_listu(&glava, &kraj);
27       exit(EXIT_FAILURE);
       }
```

```
29      printf("\tLista: ");
        ispisi_listu(glava);
31    }

33    /* Testira se funkcija za pretragu liste */
      printf("\nUnesite broj koji se trazi u listi: ");
35    scanf("%d", &broj);

37    /* Pokazivac trazeni dobija vrednost rezultata pretrage */
      trazeni = pretrazi_listu(glava, broj);
39    if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
41    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
43
      /* Ispisuje se lista unazad */
45    printf("\nLista ispisana u nazad: ");
      ispisi_listu_unazad(kraj);
47
      /* Oslobadja se memorija zauzeta za cvorove liste */
49    oslobodi_listu(&glava, &kraj);

51    exit(EXIT_SUCCESS);
  }
```

Datoteka 4.17: *main_b.c*

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include "dvostruko_povezana_lista.h"
4
  /* 2) Glavni program */
6 int main()
  {
8   /* Lista je prazna na pocetku. */
    Cvor *glava = NULL;
10  Cvor *kraj = NULL;
    int broj;
12
    /* Testira se funkcija za dodavanja novog broja na kraj liste */
14  printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
16    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
         memorije za novi cvor. Memoriju alociranu za cvorove liste
18       treba osloboditi pre napustanja programa */
      if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
        oslobodi_listu(&glava, &kraj);
22      exit(EXIT_FAILURE);
      }
24    printf("\tLista: ");
```

```
    ispisi_listu(glava);
  }

  /* Testira se funkcija za brisanje elemenata iz liste */
  printf("\nUnesite broj koji se brise iz liste: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor(&glava, &kraj, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Ispisuje se lista unazad */
  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(kraj);

  /* Oslobadja se memorija zauzeta za cvorove liste */
  oslobodi_listu(&glava, &kraj);

  exit(EXIT_SUCCESS);
}
```

Datoteka 4.18: *main_c.c*

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

/* 3) Glavni program */
int main()
{
  /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
  Cvor *kraj = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
     bude uredjena neopadajuce */
  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 novi cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa */
    if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava, &kraj);
      exit(EXIT_FAILURE);
    }
```

```
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcija za pretragu liste */
  printf("\nUnesite broj koji se trazi u listi: ");
  scanf("%d", &broj);

  /* Pokazivac trazeni dobija vrednost rezultata pretrage */
  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  /* Testira se funkcija za brisanje elemenata iz liste */
  printf("\nUnesite broj koji se brise iz liste: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
  obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Ispisuje se lista unazad */
  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(kraj);

  /* Oslobadja se memorija zauzeta za cvorove liste */
  oslobodi_listu(&glava, &kraj);

  exit(EXIT_SUCCESS);
}
```

Rešenje 4.14

Datoteka 4.19: *stabla.h*

```
#ifndef _STABLA_H_
#define _STABLA_H_ 1

/* 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);

/* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
   funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
   do greske */
int dodaj_u_stablo(Cvor ** adresa_korena, int broj);

/* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
Cvor *pretrazi_stablo(Cvor * koren, int broj);

/* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
Cvor *pronadji_najmanji(Cvor * koren);

/* f) Funkcija koja pronalazi cvor koji sadrzi najvecu vrednost u
   stablu */
Cvor *pronadji_najveci(Cvor * koren);

/* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
void obrisi_element(Cvor ** adresa_korena, int broj);

/* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   postablo - Koren - Desno podstablo ) */
void ispisi_stablo_infiksno(Cvor * koren);

/* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren);

/* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren) */
void ispisi_stablo_postfiksno(Cvor * koren);

/* k) Funkcija koja oslobadja memoriju zauzetu stablom.  */
void oslobodi_stablo(Cvor ** adresa_korena);

#endif
```

Datoteka 4.20: *stabla.c*

```
#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;
}

int dodaj_u_stablo(Cvor ** adresa_korena, int broj)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {

    /* Kreira se novi cvor */
    Cvor *novi_cvor = napravi_cvor(broj);

    /* Proverava se uspesnost kreiranja */
    if (novi_cvor == NULL) {

      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
       */
      return 1;
    }
    /* Inace ... */

    /* Novi cvor se proglasava korenom stabla */
    *adresa_korena = novi_cvor;

    /* I vraca se indikator uspesnosti kreiranja */
    return 0;
  }

  /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati
     broj */

  /* Ako je zadata vrednost manja od vrednosti korena */
  if (broj < (*adresa_korena)->broj)

    /* Broj se dodaje u levo podstablo */
    return dodaj_u_stablo(&(*adresa_korena)->levo, broj);

  else
    /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
       dodaje u desno podstablo */
    return dodaj_u_stablo(&(*adresa_korena)->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) {

    /* Prekidamo pretragu */
    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 */
```

```
114   /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
         najvecu vrednost */
116   if (koren->desno == NULL)
        return koren;
118
      /* Inace, pretragu treba nastaviti u desnom podstablu */
120   return pronadji_najveci(koren->desno);
    }
122
    void obrisi_element(Cvor ** adresa_korena, int broj)
124 {
      Cvor *pomocni_cvor = NULL;
126
      /* Ako je stablo prazno, brisanje nije primenljivo */
128   if (*adresa_korena == NULL)
        return;
130
      /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132       stabla, ona se eventualno nalazi u levom podstablu, pa treba
          rekurzivno primeniti postupak na levo podstablo. Koren ovako
134       modifikovanog stabla je nepromenjen. */
      if (broj < (*adresa_korena)->broj) {
136     obrisi_element(&(*adresa_korena)->levo, broj);
        return;
138   }

140   /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
         stabla, ona se eventualno nalazi u desnom podstablu pa treba
142      rekurzivno primeniti postupak na desno podstablo. Koren ovako
         modifikovanog stabla je nepromenjen. */
144   if ((*adresa_korena)->broj < broj) {
        obrisi_element(&(*adresa_korena)->desno, broj);
146     return;
      }
148
      /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150      jednaka broju koji se brise (tj. slucaj kada treba obrisati
         koren) */
152
      /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154      prazno stablo (vraca se NULL) */
      if ((*adresa_korena)->levo == NULL
156       && (*adresa_korena)->desno == NULL) {
        free(*adresa_korena);
158     *adresa_korena = NULL;
        return;
160   }

162   /* Ako koren ima samo levog sina, tada se brisanje vrsi tako sto se
         brise koren, a novi koren postaje levi sin */
164   if ((*adresa_korena)->levo != NULL
```

```
      && (*adresa_korena)->desno == NULL) {
    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 ispisi_stablo_infiksno(Cvor * koren)
{
  /* Ako stablo nije prazno */
  if (koren != NULL) {

    /* Prvo se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_infiksno(koren->levo);

    /* Zatim se ispisuje vrednost u korenu */
    printf("%d ", koren->broj);

    /* Na kraju se ispisuju cvorovi desno od korena */
    ispisi_stablo_infiksno(koren->desno);
  }
}

void ispisi_stablo_prefiksno(Cvor * koren)
{
  /* Ako stablo nije prazno */
```

```
  if (koren != NULL) {

    /* Prvo se ispisuje vrednost u korenu */
    printf("%d ", koren->broj);

    /* Zatim se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_prefiksno(koren->levo);

    /* Na kraju se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_prefiksno(koren->desno);
  }
}

void ispisi_stablo_postfiksno(Cvor * koren)
{

  /* Ako stablo nije prazno */
  if (koren != NULL) {

    /* Prvo se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_postfiksno(koren->levo);

    /* Zatim se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_postfiksno(koren->desno);

    /* Na kraju se ispisuje vrednost u korenu */
    printf("%d ", koren->broj);
  }
}

void oslobodi_stablo(Cvor ** adresa_korena)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
  if (*adresa_korena == NULL)
    return;

  /* Inace ... */
  /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);

  /* Oslobadja se memorija zauzetu desnim podstablom */
  oslobodi_stablo(&(*adresa_korena)->desno);

  /* Oslobadja se memorija zauzeta korenom */
  free(*adresa_korena);

  /* Proglasava se stablo praznim */
  *adresa_korena = NULL;
}
```

Datoteka 4.21: *main.c*

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

int main()
{
  Cvor *koren;
  int n;
  Cvor *trazeni_cvor;

  /* Proglasava se stablo praznim */
  koren = NULL;

  /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
     dodavanja */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
  while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&koren, n) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
      oslobodi_stablo(&koren);
      return 0;
    }
  }

  /* Generisu se trazeni ispisi: */
  printf("\nInfiksni ispis: ");
  ispisi_stablo_infiksno(koren);
  printf("\nPrefiksni ispis: ");
  ispisi_stablo_prefiksno(koren);
  printf("\nPostfiksni ispis: ");
  ispisi_stablo_postfiksno(koren);

  /* Demonstrira se rad funkcije za pretragu */
  printf("\nTrazi se broj: ");
  scanf("%d", &n);
  trazeni_cvor = pretrazi_stablo(koren, n);
  if (trazeni_cvor == NULL)
    printf("Broj se ne nalazi u stablu!\n");

  else
    printf("Broj se nalazi u stablu!\n");

  /* Demonstrira se rad funkcije za brisanje */
  printf("Brise se broj: ");
  scanf("%d", &n);
  obrisi_element(&koren, n);
  printf("Rezultujuce stablo: ");
  ispisi_stablo_infiksno(koren);
  printf("\n");

  /* Oslobadja se memorija zauzeta stablom */
```

```
  oslobodi_stablo(&koren);

  return 0;
}
```

Rešenje 4.15

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

#define MAX 50

/* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
  char *rec;
  int brojac;
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *rec)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije. */
  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 dodaje novu rec u stablo - ukoliko je dodavanje
   uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
 */
int 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_cvor = napravi_cvor(rec);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
       */
      return 1;
    }
    /* Inace... */
    /* Novi cvor se proglasava korenom stabla */
    *adresa_korena = novi_cvor;

    /* I vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Ako stablo nije prazno, 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)
    return dodaj_u_stablo(&(*adresa_korena)->levo, rec);

  else
    /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
       desno podstablo */
  if (strcmp(rec, (*adresa_korena)->rec) > 0)
    return dodaj_u_stablo(&(*adresa_korena)->desno, rec);

  else
    /* Ako je rec jednaka reci u korenu, uvecava se njen broj
       pojavljivanja */
    (*adresa_korena)->brojac++;
}

/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
  if (*adresa_korena == NULL)
    return;

  /* Inace ... */
  /* Oslobadja se memorija zauzeta levim podstablom */
```

```
96    oslobodi_stablo(&(*adresa_korena)->levo);

98    /* Oslobadja se memorija zauzeta desnim podstablom */
      oslobodi_stablo(&(*adresa_korena)->desno);
100
      /* Oslobadja se memorija zauzeta korenom */
102   free((*adresa_korena)->rec);
      free(*adresa_korena);
104
      /* Stablo se proglasava praznim */
106   *adresa_korena = NULL;
}
108
/* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec (rec
110     sa najvecim brojem pojavljivanja) */
Cvor *nadji_najfrekventniju_rec(Cvor * koren)
112 {
    Cvor *max, *max_levo, *max_desno;
114
    /* Ako je stablo prazno, prekida se sa pretragom */
116 if (koren == NULL)
      return NULL;
118
    /* Pronalazi se najfrekventnija rec u levom podstablu */
120 max_levo = nadji_najfrekventniju_rec(koren->levo);

122 /* Pronalazi se najfrekventnija rec u desnom podstablu */
    max_desno = nadji_najfrekventniju_rec(koren->desno);
124
    /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
126      podstabla, korena i desnog podstabla */
    max = koren;
128 if (max_levo != NULL && max_levo->brojac > max->brojac)
      max = max_levo;
130 if (max_desno != NULL && max_desno->brojac > max->brojac)
      max = max_desno;
132
    /* Vraca se adresa cvora sa najvecim brojem pojavljivanja */
134 return max;
}
136
/* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
138    pracene brojem pojavljivanja */
void prikazi_stablo(Cvor * koren)
140 {
    /* Ako je stablo prazno, zavrsava se sa ispisom */
142 if (koren == NULL)
      return;
144
    /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
146      podstabla */
    prikazi_stablo(koren->levo);
```

```
148
    /* Zatim rec iz korena */
150   printf("%s: %d\n", koren->rec, koren->brojac);

152   /* I nastavlja se sa ispisom reci iz desnog podstabla */
    prikazi_stablo(koren->desno);
154 }

156 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
     niz rec. Maksimalna duzina reci je odredjena argumentom max.
158    Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
     suprotnom. Rec je niz malih ili velikih slova. */
160 int procitaj_rec(FILE * f, char rec[], int max)
{
162   /* Karakter koji se cita */
    int c;
164
    /* Indeks pozicije na koju se smesta procitani karakter */
166   int i = 0;

168   /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
       nije stiglo do kraja datoteke... */
170   while (i < max - 1 && (c = fgetc(f)) != EOF) {
      /* Proverava se da li je procitani karakter slovo */
172     if (isalpha(c))
        /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
174         mala slova jer program treba da bude neosetljiv na razliku
            izmedju malih i velikih slova */
176       rec[i++] = tolower(c);

178     else
        /* Ako nije, proverava se da li je procitano barem jedno slovo
180         nove reci */
        /* Ako jeste, prekida se sa citanjem */
182     if (i > 0)
        break;
184
      /* U suprotnom se ide na sledecu iteraciju */
186   }

188   /* Dodaje se na rec terminirajuca nula */
    rec[i] = '\0';
190
    /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
192   return i > 0 ? 0 : EOF;
}
194
int main(int argc, char **argv)
196 {
    Cvor *koren = NULL, *max;
198   FILE *f;
    char rec[MAX];
```

```
200
    /* Provera da li je navedeno ime datoteke prilikom pokretanja
202     programa */
    if (argc < 2) {
204   fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
      exit(EXIT_FAILURE);
206 }

208 /* Priprema datoteke za citanje */
    if ((f = fopen(argv[1], "r")) == NULL) {
210   fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
              argv[1]);
212   exit(EXIT_FAILURE);
    }
214
    /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
216     pretrage uz proveru uspesnosti dodavanja */
    while (procitaj_rec(f, rec, MAX) != EOF) {
218   if (dodaj_u_stablo(&koren, rec) == 1) {
        fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
220     oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
222   }
    }
224
    /* Posto je citanjem reci zavrseno, zatvara se datoteka */
226 fclose(f);

228 /* Prikazuju se sve reci iz teksta i brojevi njihovih
      pojavljivanja. */
230 prikazi_stablo(koren);

232 /* Pronalazi se najfrekventnija rec */
    max = nadji_najfrekventniju_rec(koren);
234
    /* Ako takve reci nema... */
236 if (max == NULL)

238   /* Ispisuje se odgovarajuce obavestenje */
      printf("U tekstu nema reci!\n");
240
    else
242   /* Inace, ispisuje se broj pojavljivanja reci */
      printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
244           max->rec, max->brojac);

246 /* Oslobadja se dinamicki alociran prostor za stablo */
    oslobodi_stablo(&koren);
248
    exit(EXIT_SUCCESS);
250 }
```

Rešenje 4.16

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

#define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
#define MAX_IME_I_PREZIME 100

/* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime, broj
   telefona i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
  char ime_i_prezime[MAX_IME_I_PREZIME];
  char telefon[MAX_CIFARA];
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije. */
  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
  if (novi_cvor == NULL)
    return NULL;

  /* Inicijalizuju se polja novog cvora */
  strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
  strcpy(novi_cvor->telefon, telefon);
  novi_cvor->levo = NULL;
  novi_cvor->desno = NULL;

  /* Vraca se adresa novog cvora */
  return novi_cvor;
}

/* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
   ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
   povratna vrednost je 1 */
int
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
               char *telefon)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
    /* Kreira se novi cvor */
    Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
```

```
      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
52     */
      return 1;
54   }
   /* Inace... */
56   /* Novi cvor se proglasava korenom stabla */
   *adresa_korena = novi_cvor;
58
   /* I vraca se indikator uspesnog dodavanja */
60   return 0;
 }
62
 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
64    unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
      treba da bude pretrazivacko po ovom polju */
66
 /* Ako je zadato ime i prezime leksikografski manje od imena i
68    prezimena sadrzanog u korenu, podaci se dodaju u levo podstablo
    */
 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
70     < 0)
   return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
72                         telefon);

74 else
   /* Ako je zadato ime i prezime leksikografski vece od imena i
76      prezimena sadrzanog u korenu, podaci se dodaju u desno
      podstablo */
78 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
   return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
80                         telefon);
}
82
/* Funkcija koja oslobadja memoriju zauzetu stablom */
84 void oslobodi_stablo(Cvor ** adresa_korena)
{
86 /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
 if (*adresa_korena == NULL)
88   return;

90 /* Inace ... */
 /* Oslobadja se memorija zauzeta levim podstablom */
92 oslobodi_stablo(&(*adresa_korena)->levo);

94 /* Oslobadja se memorija zauzeta desnim podstablom */
 oslobodi_stablo(&(*adresa_korena)->desno);
96
 /* Oslobadja se memorija zauzeta korenom */
98 free(*adresa_korena);

100 /* Stablo se proglasava praznim */
 *adresa_korena = NULL;
```

```
}

/* Funkcija koja ispisuje imenik u leksikografskom poretku */
/* Napomena: ova funkcija nije trazena u zadatku ali se moze
   koristiti za proveru da li je stablo lepo kreirano ili ne */
void prikazi_stablo(Cvor * koren)
{
  /* Ako je stablo prazno, zavrsava se sa ispisom */
  if (koren == NULL)
    return;

  /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
     podstabla */
  prikazi_stablo(koren->levo);

  /* Zatim se ispisuju podaci iz korena */
  printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

  /* I nastavlja se sa ispisom podataka iz desnog podstabla */
  prikazi_stablo(koren->desno);
}

/* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje ime
   i prezime i broj telefona u odgovarajuce nizove. Maksimalna duzina
   imena i prezimena odredjena je konstantom MAX_IME_PREZIME, a
   maksimalna duzina broja telefona konstantom MAX_CIFARA. Funkcija
   vraca EOF ako nema vise kontakata ili 0 u suprotnom. */
int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
{
  /* Karakter koji se cita */
  int c;

  /* Indeks pozicije na koju se smesta procitani karakter */
  int i = 0;

  /* Linije datoteke koje se obradjuju su formata Ime Prezime
     BrojTelefona */

  /* Preskacu se eventualne praznine sa pocetka linije datoteke */
  while ((c = fgetc(f)) != EOF && isspace(c));

  /* Prvo procitano slovo upisuje se u ime i prezime */
  if (!feof(f))
    ime_i_prezime[i++] = c;

  /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
     cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
     Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
     procitanog karaktera i da se slucajno ne dodje do kraja datoteke
   */
  while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
    if (!isdigit(c))
```

```
154       ime_i_prezime[i++] = c;

156     else if (i > 0)
          break;
158   }

160   /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
         blanko karaktera */
162   ime_i_prezime[--i] = '\0';

164   /* I pocinje se sa citanjem broja telefona */
      i = 0;
166
      /* Upisuje se cifra koja je vec procitana */
168   telefon[i++] = c;

170   /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
         karaktera cije prisustvo nije dozvoljeno u broju telefona */
172   while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
        if (c == '/' || c == '-' || isdigit(c))
174       telefon[i++] = c;
        else
176       break;

178   /* Upisuje se terminirajuca nula */
      telefon[i] = '\0';
180
      /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
182   return !feof(f) ? 0 : EOF;
}
184
/* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
186  */
Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188 {
      /* Ako je imenik prazan, zavrsava se sa pretragom */
190   if (koren == NULL)
        return NULL;
192
      /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
194       zavrsava sa pretragom */
      if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
196     return koren;

198   /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
         korenu pretraga se nastavlja levo */
200   if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
        return pretrazi_imenik(koren->levo, ime_i_prezime);
202
      else
204     /* u suprotnom, pretraga se nastavlja desno */
        return pretrazi_imenik(koren->desno, ime_i_prezime);
```

```
206 }

208 int main(int argc, char **argv)
{
210   char ime_datoteke[MAX_IME_DATOTEKE];
  Cvor *koren = NULL;
212   Cvor *trazeni;
  FILE *f;
214   char ime_i_prezime[MAX_IME_I_PREZIME];
  char telefon[MAX_CIFARA];
216   char c;
  int i;
218
  /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
220   printf("Unesite ime datoteke: ");
  scanf("%s", ime_datoteke);
222   getchar();
  if ((f = fopen(ime_datoteke, "r")) == NULL) {
224     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
            ime_datoteke);
226     exit(EXIT_FAILURE);
  }
228
  /* Citaju se podaci iz datoteke i smestanju u binarno stablo
230     pretrage uz proveru uspesnosti dodavanja */
  while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
232     if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
        fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
234               ime_i_prezime);
        oslobodi_stablo(&koren);
236       exit(EXIT_FAILURE);
      }
238
  /* Zatvara se datoteka */
240   fclose(f);

242   /* Omogucava se pretraga imenika */
  while (1) {
244     /* Ucitavaja se ime i prezime */
    printf("Unesite ime i prezime: ");
246     i = 0;
    while ((c = getchar()) != '\n')
248       ime_i_prezime[i++] = c;
    ime_i_prezime[i] = '\0';
250
    /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
252        funkcionalnost */
    if (strcmp(ime_i_prezime, "KRAJ") == 0)
254       break;

256     /* Inace se ispisuje rezultat pretrage */
    trazeni = pretrazi_imenik(koren, ime_i_prezime);
```

```
258     if (trazeni == NULL)
          printf("Broj nije u imeniku!\n");
260     else
          printf("Broj je: %s \n", trazeni->telefon);
262   }

264   /* Oslobadja se memorija zauzeta imenikom */
      oslobodi_stablo(&koren);
266
      exit(EXIT_SUCCESS);
268 }
```

Rešenje 4.17

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

5  #define MAX 51

7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
      studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9     jezika i redom pokazivace na levo i desno podstablo */
   typedef struct cvor_stabla {
11   char ime[MAX];
     char prezime[MAX];
13   double uspeh;
     double matematika;
15   double jezik;
     struct cvor_stabla *levo;
17   struct cvor_stabla *desno;
   } Cvor;
19
   /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                      double matematika, double jezik)
23 {
     /* Alocira se memorija za novi cvor i proverava se uspesnost
25      alokacije. */
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27   if (novi == NULL)
       return NULL;
29
     /* Inicijalizuju se polja strukture */
31   strcpy(novi->ime, ime);
     strcpy(novi->prezime, prezime);
33   novi->uspeh = uspeh;
     novi->matematika = matematika;
35   novi->jezik = jezik;
     novi->levo = NULL;
37   novi->desno = NULL;
```

```
39    /* Vraca se adresa kreiranog cvora */
      return novi;
41  }

43  /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
       ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
45     povratna vrednost je 1 */
    int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47                     double uspeh, double matematika, double jezik)
    {
49    /* Ako je stablo prazno */
      if (*koren == NULL) {
51      /* Kreira se novi cvor */
        Cvor *novi_cvor =
53          napravi_cvor(ime, prezime, uspeh, matematika, jezik);
        /* Proverava se uspesnost kreiranja novog cvora */
55      if (novi_cvor == NULL) {
          /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57         */
          return 1;
59      }
        /* Inace... */
61      /* Novi cvor se proglasava korenom stabla */
        *koren = novi_cvor;
63
        /* I vraca se indikator uspesnog dodavanja */
65      return 0;
      }
67
      /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69       sortirano po ukupnom broju poena */
      if (uspeh + matematika + jezik >
71        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
        return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
73                              matematika, jezik);
      else
75      return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
                              matematika, jezik);
77  }


79
    /* Funkcija kojom se oslobadja memorija zauzeta stablom */
81  void oslobodi_stablo(Cvor ** koren)
    {
83    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
      if (*koren == NULL)
85      return;

87    /* Inace ... */
      /* Oslobadja se memorija zauzeta levim podstablom */
89    oslobodi_stablo(&(*koren)->levo);
```

```
91    /* Oslobadja se memorija zauzeta desnim podstablom */
      oslobodi_stablo(&(*koren)->desno);
93
      /* Oslobadja se memorija zauzeta korenom */
95    free(*koren);

97    /* Stablo se proglasava praznim */
      *koren = NULL;
99  }


101
    /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
103    polozili jednaka 0 ispisuju se informacije o ucenicima koji nisu
       polozili prijemni, a ako je vrednost argumenta razlicita od nule,
105    ispisuju se informacije o ucenicima koji su polozili prijemni */
    void stampaj(Cvor * koren, int polozili)
107 {
      /* Stablo je prazno - prekida se sa ispisom */
109   if (koren == NULL)
        return;
111
      /* Stampaju se informacije iz levog podstabla */
113   stampaj(koren->levo, polozili);

115   /* Stampaju se informacije iz korenog cvora */
      if (polozili && koren->matematika + koren->jezik >= 10)
117     printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
              koren->prezime, koren->uspeh, koren->matematika,
119           koren->jezik,
              koren->uspeh + koren->matematika + koren->jezik);
121   else if (!polozili && koren->matematika + koren->jezik < 10)
        printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
123           koren->prezime, koren->uspeh, koren->matematika,
              koren->jezik,
125           koren->uspeh + koren->matematika + koren->jezik);

127   /* Stampaju se informacije iz desnog podstabla */
      stampaj(koren->desno, polozili);
129 }

131 /* Funkcija koja odredjuje koliko studenata nije polozilo prijemni
       ispit */
133 int nisu_polozili(Cvor * koren)
    {
135   /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
      if (koren == NULL)
137     return 0;

139   /* Pretraga se vrsi i u levom i u desnom podstablu - ako uslov za
         polaganje nije ispunjen za koreni cvor, broj studenata se
141      uvecava za 1 */
```

```
  if (koren->matematika + koren->jezik < 10)
143   return 1 + nisu_polozili(koren->levo) +
          nisu_polozili(koren->desno);
145
  return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
147 }

149 int main(int argc, char **argv)
{
151   FILE *in;
  Cvor *koren;
153   char ime[MAX], prezime[MAX];
  double uspeh, matematika, jezik;
155
  /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
157   in = fopen("prijemni.txt", "r");
  if (in == NULL) {
159     fprintf(stderr,
              "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
161     exit(EXIT_FAILURE);
  }
163
  /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
165      dodavanja */
  koren = NULL;
167   while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
                  &matematika, &jezik) != EOF) {
169     if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik
    )
          == 1) {
171       fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
                prezime);
173       oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
175     }
  }
177
  /* Zatvaranje datoteke */
179   fclose(in);

181   /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
  stampaj(koren, 1);
183
  /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
185      prijemni */
  if (nisu_polozili(koren) != 0)
187     printf("---------------------------------------\n");

189   /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
  stampaj(koren, 0);
191
  /* Oslobadja se memorija zauzeta stablom */
```

```
193   oslobodi_stablo(&koren);

195   exit(EXIT_SUCCESS);
}
```

Rešenje 4.18

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

#define MAX_NISKA 51

/* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
   osobe, dan i mesec rodjenja i redom pokazivace na levo i desno
   podstablo */
typedef struct cvor_stabla {
  char ime[MAX_NISKA];
  char prezime[MAX_NISKA];
  int dan;
  int mesec;
  struct cvor_stabla *levo;
  struct cvor_stabla *desno;
} Cvor;

/* Funkcija koja kreira novi cvor */
Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
{
  /* Alocira se memorija */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizuju se polja strukture */
  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
  novi->dan = dan;
  novi->mesec = mesec;
  novi->levo = NULL;
  novi->desno = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

/* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
   uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
   dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
   vrednost je 1 */
int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                   int dan, int mesec)
```

```
45 {
     /* Ako je stablo prazno */
47   if (*koren == NULL) {

49     /* Kreira se novi cvor */
       Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51     /* Proverava se uspesnost kreiranja novog cvora */
       if (novi_cvor == NULL) {
53       /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
          */
55       return 1;
       }
57     /* Inace... */
       /* Novi cvor se proglasava korenom stabla */
59     *koren = novi_cvor;

61     /* I vraca se indikator uspesnog dodavanja */
       return 0;
63   }

65   /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
        meseca */
67   if (mesec < (*koren)->mesec)
       return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
69   else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
       return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
71   else
       return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec)
         ;
73 }

75 /* Funkcija vrsi pretragu stabla i vraca cvor sa trazenim datumom */
   Cvor *pretrazi(Cvor * koren, int dan, int mesec)
77 {
     /* Stablo je prazno, obustavlja se pretraga */
79   if (koren == NULL)
       return NULL;
81
     /* Ako je trazeni datum u korenu */
83   if (koren->dan == dan && koren->mesec == mesec)
       return koren;
85
     /* Ako je mesec trazenog datuma manji od meseca sadrzanog u korenu
87        ili ako su meseci isti ali je dan trazenog datuma manji od
          aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
89        svakako proverava da li leva grana postoji - ako ne postoji
          treba vratiti prvi sledeci, a to je bas vrednost uocenog korena
        */
91   if (mesec < koren->mesec
         || (mesec == koren->mesec && dan < koren->dan)) {
93     if (koren->levo == NULL)
         return koren;
```

```
95      else
96        return pretrazi(koren->levo, dan, mesec);
97    }
98
99    /* Inace se nastavlja pretraga u desnom delu */
100   return pretrazi(koren->desno, dan, mesec);
101 }
102
103 /* Funkcija koja pronalazi najmanji datum u stablu */
104 Cvor *pronadji_najmanji_datum(Cvor * koren)
105 {
106   /* Stablo je prazno, obustavlja se pretraga */
107   if (koren == NULL)
108     return NULL;
109
110   /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
111      sadrzi najmanji datum */
112   if (koren->levo == NULL)
113     return koren;
114   else
115     /* Inace, trazimo manji datum u levom podstablu */
116     return pronadji_najmanji_datum(koren->levo);
117 }
118
119 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
120    */
121 void datum_u_nisku(int dan, int mesec, char datum[])
122 {
123   if (dan < 10) {
124     datum[0] = '0';
125     datum[1] = dan + '0';
126   } else {
127     datum[0] = dan / 10 + '0';
128     datum[1] = dan % 10 + '0';
129   }
130   datum[2] = '.';
131
132   if (mesec < 10) {
133     datum[3] = '0';
134     datum[4] = mesec + '0';
135   } else {
136     datum[3] = mesec / 10 + '0';
137     datum[4] = mesec % 10 + '0';
138   }
139   datum[5] = '.';
140   datum[6] = '\0';
141 }
142
143 /* Funkcija koja oslobadja memoriju zauzetu stablom */
144 void oslobodi_stablo(Cvor ** adresa_korena)
145 {
146   /* Stablo je prazno */
```

```
  if (*adresa_korena == NULL)
147   return;

149 /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
  if ((*adresa_korena)->levo)
151   oslobodi_stablo(&(*adresa_korena)->levo);

153 /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
  if ((*adresa_korena)->desno)
155   oslobodi_stablo(&(*adresa_korena)->desno);

157 /* Oslobadja se memorija zauzeta korenom */
  free(*adresa_korena);
159
  /* Proglasava se stablo praznim */
161 *adresa_korena = NULL;
}
163
int main(int argc, char **argv)
165 {
  FILE *in;
167 Cvor *koren;
  Cvor *slavljenik;
169 char ime[MAX_NISKA], prezime[MAX_NISKA];
  int dan, mesec;
171 char datum[7];

173 /* Provera da li je zadato ime ulazne datoteke */
  if (argc < 2) {
175   /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
        programa */
177   fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
    exit(EXIT_FAILURE);
179 }

181 /* Inace, priprema se datoteka za citanje */
  in = fopen(argv[1], "r");
183 if (in == NULL) {
    fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
185             argv[1]);
    exit(EXIT_FAILURE);
187 }

189 /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
      */
  koren = NULL;
191 while (fscanf
         (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
193   if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
      fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
195             prezime);
      oslobodi_stablo(&koren);
```

```
197       exit(EXIT_FAILURE);
    }
199
  /* Datoteka se zatvara */
201 fclose(in);

203 /* Omogucuje se pretraga podataka */
  while (1) {
205
    /* Ucitava se novi datum */
207   printf("Unesite datum: ");
    if (scanf("%d.%d.", &dan, &mesec) == EOF)
209     break;

211   /* Pretrazuje se stablo */
    slavljenik = pretrazi(koren, dan, mesec);
213
    /* Ispisuju se pronadjeni podaci */
215
    /* Ako slavljenik nije pronadjen, to moze znaci da: */
217   /* 1. drvo je prazno */
    if (slavljenik == NULL && koren == NULL) {
219     printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
      continue;
221   }
    /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223       ovom slucaju se pretraga vrsi pocevsi od naredne godine i
       ispisuje se najmanji datum */
225   if (slavljenik == NULL) {
      slavljenik = pronadji_najmanji_datum(koren);
227     datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229             slavljenik->prezime, datum);
      continue;
231   }

233   /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
    /* 1. Pronadjeni su tacni podaci */
235   if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
      printf("Slavljenik: %s %s\n", slavljenik->ime,
237             slavljenik->prezime);
      continue;
239   }

241   /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
    datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
243   printf("Slavljenik: %s %s %s\n", slavljenik->ime,
          slavljenik->prezime, datum);
245 }

247 /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);
```

```
249
  exit(EXIT_SUCCESS);
251 }
```

Rešenje 4.19

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja proverava da li su dva stabla koja sadrze cele
   brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
int identitet(Cvor * koren1, Cvor * koren2)
{
  /* Ako su oba stabla prazna, jednaka su */
  if (koren1 == NULL && koren2 == NULL)
    return 1;

  /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
  if (koren1 == NULL || koren2 == NULL)
    return 0;

  /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
     vrednosti, moze se zakljuciti da se razlikuju */
  if (koren1->broj != koren2->broj)
    return 0;

  /* Inace, proverava se da li vazi i jednakost levih i desnih
     podstabala */
  return (identitet(koren1->levo, koren2->levo)
          && identitet(koren1->desno, koren2->desno));
}

int main()
{
  int broj;
  Cvor *koren1, *koren2;

  /* Ucitavaju se elementi prvog stabla */
  koren1 = NULL;
  printf("Prvo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    if (dodaj_u_stablo(&koren1, broj) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
      oslobodi_stablo(&koren1);
      return 0;
    }
```

```
    scanf("%d", &broj);
  }

  /* Ucitavaju se elementi drugog stabla */
  koren2 = NULL;
  printf("Drugo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    if (dodaj_u_stablo(&koren2, broj) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
      oslobodi_stablo(&koren2);
      return 0;
    }
    scanf("%d", &broj);
  }

  /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
     njen rezultat. */
  if (identitet(koren1, koren2))
    printf("Stabla jesu identicna.\n");
  else
    printf("Stabla nisu identicna.\n");

  /* Oslobadja se memorija zauzeta stablima */
  oslobodi_stablo(&koren1);
  oslobodi_stablo(&koren2);

  return 0;
}
```

Rešenje 4.20

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
   Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
   odnosno 1 ukoliko je doslo do greske */
int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
{
  /* Izlaz iz rekurzije */
  if (koren == NULL) {
    *duplikat = NULL;
    return 0;
  }

  /* Duplira se koren stabla i postavlja da bude koren novog stabla
    */
```

```
  *duplikat = napravi_cvor(koren->broj);
  if (*duplikat == NULL) {
    return 1;
  }

  /* Rekurzivno se duplirju levo i desno podstablo i njihove adrese
     se cuvaju redom u pokazivacima na levo i desno podstablo korena
     duplikata */
  int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
  int kopija_desno =
      kopiraj_stablo(koren->desno, &(*duplikat)->desno);

  /* Ako je uspesno duplirano i levo i desno podstablo */
  if (kopija_levo == 0 && kopija_desno == 0)
    /* Uspesno je duplirano i celo stablo */
    return 0;
  /* Inace, prijavljuje se da je doslo do greske */
  return 1;

}

/* Funkcija izracunava uniju dva skupa predstavljena stablima -
   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
   Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
   uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
{
  /* Ako drugo stablo nije prazno */
  if (koren2 != NULL) {
    /* 1. Dodaje se njegov koren u prvo stablo */
    if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
      return 1;
    }

    /* 2. Rekurzivno se racuna unija levog i desnog podstabla drugog
       stabla sa prvim stablom */
    int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
    int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);

    /* Ako je unija podstabala uspesno kreirana */
    if (unija_levo == 0 && unija_desno == 0)
      /* Uspesno je kreirana i unija stabala */
      return 0;

    /* U suprotnom se prijavljuje da je doslo do greske */
    return 1;
  }

  /* Ako je drugo stablo prazno, nista se ne preduzima */
  return 0;
}

```

```
/* Funkcija izracunava presek dva skupa predstavljana stablima -
   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
   Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka
   uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
{
  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
  if (*adresa_korena1 == NULL)
    return 0;

  /* 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 */
  int presek_levo = kreiraj_presek(&(*adresa_korena1)->levo, koren2);
  int presek_desno =
      kreiraj_presek(&(*adresa_korena1)->desno, koren2);
  if (presek_levo == 0 && presek_desno == 0) {
    /* 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);

    /* Presek stabala je uspesno kreiran */
    return 0;
  }
  /* Inece, prijavljuje se da je doslo do greske */
  return 1;
}

/* Funkcija izracunava razliku dva skupa predstavljana stablima -
   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
   Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike
   uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
{
  /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
  if (*adresa_korena1 == NULL)
    return 0;

  /* 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 */
  int razlika_levo =
      kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
  int razlika_desno =
      kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
  if (razlika_levo == 0 && razlika_desno == 0) {
    /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on
       uklanja se iz prvog stabla */
    if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
```

```
      obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

    /* Razlika stabala je uspesno kreirana */
    return 0;
  }

  /* Inece, prijavljuje se da je doslo do greske */
  return 1;
}

int main()
{
  Cvor *skup1;
  Cvor *skup2;
  Cvor *pomocni_skup = NULL;
  int n;

  /* Ucitavaju se elementi prvog skupa */
  skup1 = NULL;
  printf("Prvi skup: ");
  while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&skup1, n) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
      oslobodi_stablo(&skup1);
      return 0;
    }
  }

  /* Ucitavaju se elementi drugog skupa */
  skup2 = NULL;
  printf("Drugi skup: ");
  while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&skup2, n) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
      oslobodi_stablo(&skup2);
      return 0;
    }
  }

  /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
     bi se isti mogao iskoristiti i za preostale operacije */
  if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    return 0;
  }
  if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
    return 0;
  }
  printf("Unija: ");
```

```
    ispisi_stablo_infiksno(pomocni_skup);
176 putchar('\n');

178 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
       operacije */
180 oslobodi_stablo(&pomocni_skup);

182 /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
       kako bi se isti mogao iskoristiti i za preostale operacije */
184 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
      oslobodi_stablo(&skup1);
186   oslobodi_stablo(&pomocni_skup);
      return 0;
188 }
    if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190   oslobodi_stablo(&pomocni_skup);
      oslobodi_stablo(&skup2);
192   return 0;
    }
194 printf("Presek: ");
    ispisi_stablo_infiksno(pomocni_skup);
196 putchar('\n');

198 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
       operacije */
200 oslobodi_stablo(&pomocni_skup);

202 /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
       kako bi se isti mogao iskoristiti i za preostale operacije */
204 if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
      oslobodi_stablo(&skup1);
206   oslobodi_stablo(&pomocni_skup);
      return 0;
208 }
    if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
210   oslobodi_stablo(&pomocni_skup);
      oslobodi_stablo(&skup2);
212   return 0;
    }
214 printf("Razlika: ");
    ispisi_stablo_infiksno(pomocni_skup);
216 putchar('\n');

218 /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
       operacije */
220 oslobodi_stablo(&pomocni_skup);

222 /* Oslobadja se memorija zauzeta polaznim skupovima */
    oslobodi_stablo(&skup1);
224 oslobodi_stablo(&skup2);

226 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 na
   desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
   kreiran i sortiran, a 1 ukoliko je doslo do greske.

   Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
   sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
   sortiranje vrsi u pomocnoj dinamickoj strukturi, a ne razmenom
   elemenata niza. */
int sortiraj(int a[], int n)
{
  int i;
```

```
  Cvor *koren;

  /* Kreira se stablo smestanjem elemenata iz niza u stablo */
  koren = NULL;
  for (i = 0; i < n; i++) {
    if (dodaj_u_stablo(&koren, a[i]) == 1) {
      oslobodi_stablo(&koren);
      return 1;
    }
  }
  /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
     a */
  kreiraj_niz(koren, a);

  /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
    */
  oslobodi_stablo(&koren);

  /* Vraca se indikator uspesnog sortiranja */
  return 0;
}

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

  /* Ucitavaju se dimenzija i elementi niza */
  printf("n: ");
  scanf("%d", &n);
  if (n < 0 || n > MAX) {
    printf("Greska: pogresna dimenzija niza!\n");
    return 0;
  }

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

  /* Poziva se funkcija za sortiranje */
  if (sortiraj(a, n) == 0) {
    /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
    for (i = 0; i < n; i++)
      printf("%d ", a[i]);
    printf("\n");
  } else {
    /* Inace, obavestava se korisnik da je doslo do greske */
    printf("Greska: problem prilikom sortiranja niza!\n");
  }

  return 0;
}
```

Rešenje 4.22

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* a) Funkcija koja izracunava broj cvorova stabla */
int broj_cvorova(Cvor * koren)
{
  /* Ako je stablo prazno, broj cvorova je nula */
  if (koren == NULL)
    return 0;

  /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
     levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
     zato sto treba racunati i koren */
  return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
}

/* b) Funkcija koja izracunava broj listova stabla */
int broj_listova(Cvor * koren)
{
  /* Ako je stablo prazno, broj listova je nula */
  if (koren == NULL)
    return 0;

  /* Proverava se da li je tekuci cvor list */
  if (koren->levo == NULL && koren->desno == NULL)
    /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
       uvecati broj listova za 1 */
    return 1;

  /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
   */
  return broj_listova(koren->levo) + broj_listova(koren->desno);
}

/* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
void pozitivni_listovi(Cvor * koren)
{
  /* Slucaj kada je stablo prazno */
  if (koren == NULL)
    return;

  /* Ako je cvor list i sadrzi pozitivnu vrednost */
  if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
    /* Stampa se */
    printf("%d ", koren->broj);

  /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
```

```
  pozitivni_listovi(koren->levo);
  pozitivni_listovi(koren->desno);
}

/* d) Funkcija koja izracunava zbir cvorova stabla */
int zbir_svih_cvorova(Cvor * koren)
{
  /* Ako je stablo prazno, zbir cvorova je 0 */
  if (koren == NULL)
    return 0;

  /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
     elemenata u podstablima */
  return koren->broj + zbir_svih_cvorova(koren->levo) +
      zbir_svih_cvorova(koren->desno);
}

/* e) Funkcija koja izracunava najveci element stabla */
Cvor *najveci_element(Cvor * koren)
{
  /* Ako je stablo prazno, obustavlja se pretraga */
  if (koren == NULL)
    return NULL;

  /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena se
     nalaze u desnom podstablu */

  /* Ako desnog podstabla nema */
  if (koren->desno == NULL)
    /* Najveca vrednost je koren */
    return koren;

  /* Inace, najveca vrednost se trazi desno */
  return najveci_element(koren->desno);
}

/* f) 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;
}

/* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
{
  /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
     nivoa */

  /* Ako nema vise cvorova, nema spustanja niz stablo */
  if (koren == NULL)
    return 0;

  /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
     rekurzivnih poziva uvecati broj cvorova za 1 */
  if (i == 0)
    return 1;

  /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
    */
  return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
      + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
}

/* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispis_nivo(Cvor * koren, int i)
{
  /* Ideja je slicna ideji iz prethodne funkcije */

  /* Nema vise cvorova, nema spustanja kroz stablo */
  if (koren == NULL)
    return;

  /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
  if (i == 0) {
    printf("%d ", koren->broj);
    return;
  }
  /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
     desnom podstablu */
  ispis_nivo(koren->levo, i - 1);
  ispis_nivo(koren->desno, i - 1);
}

/* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
   stabla */
Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
{
  /* Ako je stablo prazno, obustavlja se pretraga */
  if (koren == NULL)
    return NULL;
```

```
154   /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
      if (i == 0)
156     return koren;

158   /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
      Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
160
      /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
162   Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);

164   /* Trazi se i vraca maksimum izracunatih vrednosti */
      if (a == NULL && b == NULL)
166     return NULL;
      if (a == NULL)
168     return b;
      if (b == NULL)
170     return a;
      return a->broj > b->broj ? a : b;
172 }

174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
    int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
      /* Ako je stablo prazno, zbir je nula */
178   if (koren == NULL)
        return 0;
180
      /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
182   if (i == 0)
        return koren->broj;
184
      /* Inace, spustanje se nastavlja za jedan nivo nize i traze se sume
186       iz levog i desnog podstabla */
      return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188       + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
    }
190

192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
       manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
    {
196   /* Ako je stablo prazno, zbir je nula */
      if (koren == NULL)
198     return 0;

200   /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
        prirode pretrazivackog stabla treba obici i levo i desno
202     podstablo */
      if (koren->broj <= x)
204     return koren->broj + zbir_manjih_od_x(koren->levo, x) +
          zbir_manjih_od_x(koren->desno, x);
```

```
206
  /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
208     medju njima jedino moze biti onih koje zadovoljavaju uslov */
  return zbir_manjih_od_x(koren->levo, x);
210 }

212 int main(int argc, char **argv)
{
214   /* Analiza argumenata komandne linije */
  if (argc != 3) {
216     fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out nivo
    broj_za_pretragu\n");
218     return 1;
  }
220   int i = atoi(argv[1]);
  int x = atoi(argv[2]);
222
  /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
224    */
  Cvor *koren = NULL;
226   int broj;
  while (scanf("%d", &broj) != EOF) {
228     if (dodaj_u_stablo(&koren, broj) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
230       oslobodi_stablo(&koren);
      return 0;
232     }
  }
234
  /* ispisuju se rezultati rada funkcija */
236   printf("Broj cvorova: %d\n", broj_cvorova(koren));
  printf("Broj listova: %d\n", broj_listova(koren));
238   printf("Pozitivni listovi: ");
  pozitivni_listovi(koren);
240   printf("\n");
  printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
242   if (najveci_element(koren) == NULL)
    printf("Najveci element: ne postoji\n");
244   else
    printf("Najveci element: %d\n", najveci_element(koren)->broj);
246
  printf("Dubina stabla: %d\n", dubina_stabla(koren));
248
  printf("Broj cvorova na %d. nivou: %d\n", i,
250         broj_cvorova_na_itom_nivou(koren, i));
  printf("Elementi na %d. nivou: ", i);
252   ispis_nivo(koren, i);
  printf("\n");
254   if (najveci_element_na_itom_nivou(koren, i) == NULL)
    printf("Nema elemenata na %d. nivou!\n", i);
256   else
```

```
    printf("Maksimalni element na %d. nivou: %d\n", i,
258           najveci_element_na_itom_nivou(koren, i)->broj);

260  printf("Zbir elemenata na %d. nivou: %d\n", i,
         zbir_cvorova_na_itom_nivou(koren, i));
262  printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
         zbir_manjih_od_x(koren, x));
264
  /* Oslobadja se memorija zauzeta stablom */
266  oslobodi_stablo(&koren);

268  return 0;
}
```

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)
28 {
  /* Ideja je slicna ideji iz prethodne funkcije */
30   /* Nema vise cvorova, nema spustanja niz stablo */
  if (koren == NULL)
32     return;

34   /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
  if (i == 0) {
```

```
36      printf("%d ", koren->broj);
        return;
38    }
    /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u desnom
40       podstablu */
    ispisi_nivo(koren->levo, i - 1);
42    ispisi_nivo(koren->desno, i - 1);
}
44
/* Funkcija koja ispisuje stablo po nivoima */
46 void ispisi_stablo_po_nivoima(Cvor * koren)
{
48    int i;

50    /* Prvo se izracunava dubina stabla */
    int dubina;
52    dubina = dubina_stabla(koren);

54    /* Ispisuje se nivo po nivo stabla */
    for (i = 0; i < dubina; i++) {
56      printf("%d. nivo: ", i);
      ispisi_nivo(koren, i);
58      printf("\n");
    }
60 }

62 int main(int argc, char **argv)
{
64    Cvor *koren;
    int broj;
66
    /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
68       uspesnosti dodavanja */
    koren = NULL;
70    while (scanf("%d", &broj) != EOF) {
      if (dodaj_u_stablo(&koren, broj) == 1) {
72        fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
        oslobodi_stablo(&koren);
74        return 0;
      }
76    }

78    /* Ispisuje se stablo po nivoima */
    ispisi_stablo_po_nivoima(koren);
80
    /* Oslobadja se memorija zauzeta stablom */
82    oslobodi_stablo(&koren);

84    return 0;
}
```

Rešenje 4.25

```c
#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 uz proveru
     uspesnosti dodavanja */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    if (dodaj_u_stablo(&koren, broj) == 1) {
      fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
      oslobodi_stablo(&koren);
      return 0;
    }
  }

  /* 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 hip. Ideja koja ce biti implementirana u osnovi ima
   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
   desnog podstabla - ako je vrednost u korenu veca od izracunatih
   vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
   ce funkcija vracati maksimalne vrednosti iz uocenog podstabala ili
   vrednost -1 ukoliko se zakljuci da stablo nije hip. */
int heap(Cvor * koren)
{
  int max_levo, max_desno;

  /* Prazno sablo je hip - kao rezultat se vraca 0 kao najmanji
     pozitivan broj */
  if (koren == NULL) {
    return 0;
```

```
  }
  /* Ukoliko je stablo list... */
  if (koren->levo == NULL && koren->desno == NULL) {
    /* Vraca se njegova vrednost */
    return koren->broj;
  }
  /* Inace... */

  /* Proverava se svojstvo za levo podstablo */
  max_levo = heap(koren->levo);

  /* Proverava se svojstvo za desno podstablo */
  max_desno = heap(koren->desno);

  /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
     ni celo stablo */
  if (max_levo == -1 || max_desno == -1) {
    return -1;
  }

  /* U suprotonom proverava se da li svojstvo vazi za uoceni cvor */
  if (koren->broj > max_levo && koren->broj > max_desno) {
    /* Ako vazi, vraca se vrednost korena */
    return koren->broj;
  }

  /* U suprotnom zakljucuje se da stablo nije hip */
  return -1;
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int hip_indikator;

  /* Kreira se stablo prema zadatoj slici */
  koren = NULL;
  koren = napravi_cvor(100);
  koren->levo = napravi_cvor(19);
  koren->levo->levo = napravi_cvor(17);
  koren->levo->levo->levo = napravi_cvor(2);
  koren->levo->levo->desno = napravi_cvor(7);
  koren->levo->desno = napravi_cvor(3);
  koren->desno = napravi_cvor(36);
  koren->desno->levo = napravi_cvor(25);
  koren->desno->desno = napravi_cvor(1);

  /* Poziva se funkcija kojom se proverava da li je stablo hip */
  hip_indikator = heap(koren);

  /* Ispisuje se rezultat */
  if (hip_indikator == -1) {
```

```
74      printf("Zadato stablo nije hip!\n");
    } else {
76      printf("Zadato stablo je hip!\n");
    }
78
    /* Oslobadja se memorija zauzeta stablom */
80  oslobodi_stablo(&koren);

82  return 0;
}
```

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 za grešku ispisati vrednost `-1` i prekinuti izvršavanje programa.

Test 1

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

ulaz.txt
  5
  Programiranje
  Matematika
  12345
  dInAmiCnArEc
  Ispit

Izlaz:
  Ispit
  Matematika
  Programiranje
```

Test 2

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

ulaz.txt
  2
  maksimalano
  poena

Izlaz:
```

Test 3

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

Datoteka ulaz.txt ne postoji

Izlaz:
  -1
```

Test 4

```
Poziv: ./a.out

Izlaz za grešku:
 -1
```

[Rešenje 5.1]

Zadatak 5.2 Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na n-tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj n, a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `sumirajN` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test 1

```
Ulaz:
 2 8 10 3 6 14 13 7 4 0
Izlaz:
  20
```

Test 2

```
Ulaz:
 0 50 14 5 2 4 56 8 52 7 1 0
Izlaz:
  50
```

[Rešenje 5.2]

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A, a zatim i elementi matrice A. Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od `50`. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

Test 1

```
ULAZ:
 4 5
 1 2 3 4 5
  -1 2 -3 4 -5
  -5 -4 -3 -2 1
 -1 0 0 0 0
IZLAZ:
 0
```

Test 2

```
ULAZ:
 2 3
 0 0 -5
 1 2 -4
IZLAZ:
 2
```

Test 3

```
ULAZ:
 -2
IZLAZ ZA GREŠKU:
 -1
```

[Rešenje 5.3]

5.2 Programiranje 2, praktični deo ispita, jul 2015.

Zadatak 5.4 Napisati program koji kao prvi arugment komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.
Potpis funkcije `strstr`:
`char *strstr(const char *haystack, const char *needle);`
Funkcija traži prvo pojavljivanje podniske needle u nisci haystack, i vraća pokazivač na početak podniske, ili NULL ako podniska nije pronađena.

Test 1

```
POZIV: ./a.out ulaz.txt test

ULAZ.TXT
   Ovo je test primer.
   U njemu se rec test javlja
  vise puta. testtesttest

IZLAZ:
 5
```

Test 2

```
POZIV: ./a.out

IZLAZ ZA GREŠKU:
 -1
```

Test 3

```
POZIV: ./a.out ulaz.txt foo

DATOTEKA ULAZ.TXT NE POSTOJI

IZLAZ ZA GREŠKU:
 -1
```

Test 4

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

DATOTEKA ULAZ.TXT JE PRAZNA

IZLAZ:
 0
```

[Rešenje 5.4]

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.

Test 1

```
TROUGLOVI.TXT
 4
  0 0 0 1.2 1 0
  0.3 0.3 0.5 0.5 0.9 1
 -2 0 0 0 0 1
 -2 0 0 0 0 1

IZLAZ:
 2 0 2 2 -1 -1
 -2 0 0 0 0 1
 0 0 0 1.2 1 0
 0.3 0.3 0.5 0.5 0.9 1
```

Test 2

```
TROUGLOVI.TXT
 3
  1.2 3.2 1.1 4.3

IZLAZ ZA GREŠKU:
 -1
```

Test 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI

IZLAZ ZA GREŠKU:
 -1
```

Test 4

```
TROUGLOVI.TXT
 0

IZLAZ:
```

[Rešenje 5.5]

Zadatak 5.6 Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Napisati funkciju
`int prebrojN(Cvor *koren, int n)`
koja u datom stablu prebrojava čvorove na n-tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou `0`. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

Test 1

```
ULAZ:
  1 5 3 6 1 4 7 9
IZLAZ:
 1
```

Test 2

```
ULAZ:
 2 5 3 6 1 0 4 7 9
IZLAZ:
 2
```

Test 3

```
ULAZ:
 0 4 2 5
IZLAZ:
 0
```

Test 4

```
ULAZ:
  3
IZLAZ:
 0
```

Test 5

```
ULAZ:
 -1 4 5 1 7
IZLAZ:
 0
```

[Rešenje 5.6]

5.3 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
```

[Rešenje 5.7]

Zadatak 5.8 Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

Test 1

```
ULAZ:
 1 2 3 4 5
IZLAZ:
 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00
```

Test 2

```
ULAZ:
 12
IZLAZ:
 12.00
```

Test 3

```
ULAZ:
 prazna lista
IZLAZ:
```

Test 4

```
ULAZ:
 13.3 15.8
IZLAZ:
 13.30 14.55
```

[Rešenje 5.8]

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 su sume brojeva u svim redovima i kolonama međusobno jednake). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati -". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati `-1` na standardni izlaz za grešku. NAPOMENA: *Rešenje koristi biblioteku za rad sa matricama iz zadatka 2.19.*

Test 1

```
ULAZ:
 4
 1 2 3 4
 2 1 4 3
 3 4 2 1
 4 3 1 2
IZLAZ:
 10
```

Test 2

```
ULAZ:
 3
 1 1 1
 1 1 1
 1 1 1
IZLAZ:
 3
```

Test 3

```
ULAZ:
 2
 1 1
 2 2
IZLAZ:
 -
```

Test 4

```
ULAZ:
 2
 1 2
 1 2
IZLAZ:
 -
```

Test 5

```
ULAZ:
 1
 5
IZLAZ:
 5
```

Test 6

```
ULAZ:
 0
IZLAZ ZA GREŠKU:
 -1
```

[Rešenje 5.9]

5.4 Rešenja

Rešenje 5.1

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

/* Funkcija vrsi dinamicku alokaciju memorije potrebne n linija tj.
   n niski od kojih nijedna nije duza od MAX karaktera. */
char **alociranje_memorije(int n)
{
  char **linije = NULL;
  int i, j;
  /* Alocira se prostor za niz vrsti matrice */
  linije = (char **) malloc(n * sizeof(char *));
  /* U slucaju neuspesnog otvaranja ispisuje se -1 na stderr i
     program zavrsava. */
  if (linije == NULL)
    return NULL;
  /* Alocira se prostor za svaku vrstu matrice. Niska nije duza od
     MAX karaktera, a 1 se dodaje zbog terminirajuce nule. */
  for (i = 0; i < n; i++) {
    linije[i] = malloc((MAX + 1) * sizeof(char));
    /* Ako alokacija nije prosla uspesno, oslobadjaju se svi
       prethodno alocirani resursi, i povratna vrednost je NULL */
    if (linije[i] == NULL) {
      for (j = 0; j < i; j++) {
        free(linije[j]);
      }
      free(linije);
      return NULL;
    }
  }
  return linije;
}

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

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

```
}

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

  /* Proverava argumenata komandne linije. */
  if (argc != 2) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  /* Otvaranje datoteke cije ime je navedeno kao argument komandne
     linije neposredno nakon imena programa koji se poziva. U slucaju
     neuspesnog otvaranja ispisuje se -1 na stderr i program zavrsava
    . */
  ulaz = fopen(argv[1], "r");
  if (ulaz == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }
  /* Ucitavanje broja linija. */
  fscanf(ulaz, "%d", &n);

  /* Alociranje memorije na osnovu ucitanog broja linija. */
  linije = alociranje_memorije(n);

  /* U slucaju neuspesne alokacije ispisuje se -1 na stderr i program
     zavrsava. */
  if (linije == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavanje svih n linija iz datoteke. */
  for (i = 0; i < n; i++) {
    fscanf(ulaz, "%s", linije[i]);
  }

  /* Ispisivanje u odgovarajucem poretku ucitane linije koje
     zadovoljavaju kriterijum. */
  for (i = n - 1; i >= 0; i--) {
    if (isupper(linije[i][0])) {
      printf("%s\n", linije[i]);
    }
  }
  /* Oslobadjanje memorije koja je dinamicki alocirana. */
  linije = oslobadjanje_memorije(linije, n);

  /* Zatvaranje datoteku. */
```

```
98    fclose(ulaz);
      exit(EXIT_SUCCESS);
100 }
```

Rešenje 5.2

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Datoteka 5.1: *main.c*

```
#include <stdio.h>
2 #include <stdlib.h>
  #include "stabla.h"
4
  int sumirajN(Cvor * koren, int n)
6 {
    /* Ako je stablo prazno, suma je nula */
8   if (koren == NULL)
      return 0;
10  /* Inace ... */
    /* Ako je n jednako nula, vraca se broj iz korena */
12  if (n == 0)
      return koren->broj;
14  /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
       kao i suma na (n-1)-om nivou u desnom podstablu i vraca se zbir
16     te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
       n-tom nivou u pocetnom stablu */
18  return sumirajN(koren->levo, n - 1) + sumirajN(koren->desno, n - 1)
      ;
  }
20
  int main()
22 {
    Cvor *koren = NULL;
24  int n;
    int nivo;
26
    /* Ucitava se vrednost nivoa */
28  scanf("%d", &nivo);
    while (1) {
30    scanf("%d", &n);
      /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
32    if (n == 0)
        break;
34    /* Ako nije, dodaje se procitani broj u stablo. */
      if (dodaj_u_stablo(&koren, n) == 1) {
36      fprintf(stderr, "-1\n", n);
        oslobodi_stablo(&koren);
38      exit(EXIT_FAILURE);
```

```
    }
  }

  /* Ispisuje se rezultat rada trazene funkcije */
  printf("%d\n", sumirajN(koren, nivo));

  /* Oslobadja se memorija */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

Rešenje 5.3

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50

/* Funkcija ucitava elemenate matrice sa standardnog ulaza */
void ucitaj_matricu(int m[][MAX], int v, int k)
{
  int i, j;
  for (i = 0; i < v; i++) {
    for (j = 0; j < k; j++) {
      scanf("%d", &m[i][j]);
    }
  }
}

/* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
   koja ima v vrsta */
int broj_negativnih_u_koloni(int m[][MAX], int v, int k)
{
  int broj_negativnih = 0;
  int i;
  for (i = 0; i < v; i++) {
    if (m[i][k] < 0)
      broj_negativnih++;
  }
  return broj_negativnih;
}

int max_indeks(int m[][MAX], int v, int k)
{
  int i, j;
  int broj_negativnih;
  /* Inicijalizujemo na nulu indeks kolone sa maksimalnim brojem
     negativnih (max_indeks_kolone), kao i maksimalni broj negativnih
     elemenata u nekoj koloni (max_broj_negativnih) */
  int max_indeks_kolone = 0;
  int max_broj_negativnih = 0;
```

```
38
   for (j = 0; j < k; j++) {
40   /* Racunamo broj negativnih u j-oj koloni */
     broj_negativnih = broj_negativnih_u_koloni(m, v, j);
42   /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
        maksimalnog, azuriramo trenutni maksimalni broj negativnih
44      kao i indeks kolone sa maksimalnim brojem negativnih */
     if (max_broj_negativnih < broj_negativnih) {
46     max_indeks_kolone = j;
       max_broj_negativnih = broj_negativnih;
48   }
   }
50 return max_indeks_kolone;
}
52
int main()
54 {
   int m[MAX][MAX];
56 int v, k;

58 /* Ucitavanje dimenzije matrice */
   scanf("%d", &v);
60 if (v < 0 || v > MAX) {
     fprintf(stderr, "-1\n");
62   exit(EXIT_FAILURE);
   }
64 scanf("%d", &k);
   if (k < 0 || k > MAX) {
66   fprintf(stderr, "-1\n");
     exit(EXIT_FAILURE);
68 }
   /* Ucitavanje elemenata matrice */
70 ucitaj_matricu(m, v, k);

72 /* Pronalazenje kolone koja sadrzi najveci broj negativnih
      elemenata i ispisivanje trazenog rezultata */
74 printf("%d\n", max_indeks(m, v, k));
   exit(EXIT_SUCCESS);
76 }
```

Rešenje 5.4

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #define MAX 128

6 int main(int argc, char **argv)
{
8   FILE *f;
    int brojac = 0;
```

```
  char linija[MAX], *p;

  /* Provera da li je broj argumenata komandne linije 3 */
  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);
  }
  /* Ucitavanje iz otvorene datoteke - liniju po liniju */
  while (fgets(linija, MAX, f) != NULL) {
    p = linija;
    while (1) {
      p = strstr(p, argv[2]);

      /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
      if (p == NULL)
        break;
      /* Inace se uvecava brojac */
      brojac++;
      /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
         linije nakon uocene podniske */
      p = p + strlen(argv[2]);
    }
  }
  /* Zatvaranje datoteke */
  fclose(f);
  /* Ispisivanje vrednosti brojaca */
  printf("%d\n", brojac);

  exit(EXIT_SUCCESS);
}
```

Rešenje 5.5

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

/* Struktura trougao */
typedef struct _trougao {
  double xa, ya, xb, yb, xc, yc;
} trougao;

/* Funkcija racuna duzinu duzi */
double duzina(double x1, double y1, double x2, double y2)
{
```

```
13    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
15
/* Funkcija racuna povrsinu trougla koristeci Heronov obrazac */
17 double povrsina(trougao t)
{
19    /* Racunanje duzina stranica trougla */
    double a = duzina(t.xb, t.yb, t.xc, t.yc);
21    double b = duzina(t.xa, t.ya, t.xc, t.yc);
    double c = duzina(t.xa, t.ya, t.xb, t.yb);
23    /* Poluobim */
    double s = (a + b + c) / 2;
25    /* Primena Heronovog obrasca */
    return sqrt(s * (s - a) * (s - b) * (s - c));
27 }

29 /* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
     prvi argument funkcije manja od povrsine trougla koji je drugi
31     element funkcije funcija vraca 1, ukoliko je veca -1, a ukoliko
     su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
33     napisana tako da se moze proslediti funkciji qsort da se niz
     trouglova sortira po povrsini opadajuce. */
35 int poredi(const void *a, const void *b)
{
37    trougao x = *(trougao *) a;
    trougao y = *(trougao *) b;
39    double xp = povrsina(x);
    double yp = povrsina(y);
41    if (xp < yp)
      return 1;
43    if (xp > yp)
      return -1;
45    return 0;
}
47
int main()
49 {
    FILE *f;
51    int n, i;
    trougao *niz;
53
    /* Otvaranje datoteke ciji je naziv trouglovi.txt */
55    if ((f = fopen("trouglovi.txt", "r")) == NULL) {
      fprintf(stderr, "-1\n");
57      exit(EXIT_FAILURE);
    }
59
    /* Ucitavanje podtaka o broju trouglova iz datoteke */
61    if (fscanf(f, "%d", &n) != 1) {
      fprintf(stderr, "-1\n");
63      exit(EXIT_FAILURE);
    }
```

```
65
  /* Dinamicka alokacija memotije za niz trouglova duzine n */
67  if ((niz = malloc(n * sizeof(trougao))) == NULL) {
      fprintf(stderr, "-1\n");
69    exit(EXIT_FAILURE);
    }
71
  /* Ucitavanje podataka u niz iz otvorene datoteke */
73  for (i = 0; i < n; i++) {
      if (fscanf(f, "%lf%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
75                &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
      {
        fprintf(stderr, "-1\n");
77      exit(EXIT_FAILURE);
      }
79  }

81  /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
       poredi */
83  qsort(niz, n, sizeof(trougao), &poredi);

85  /* Ispisivanje sortiranog niza na standardni izlaz */
    for (i = 0; i < n; i++)
87    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);
89
  /* Oslobadjanje dinamicki alocirane memorije */
91  free(niz);

93  /* Zatvranje datoteke */
    fclose(f);
95  exit(EXIT_SUCCESS);
}
```

Rešenje 5.6

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Datoteka 5.2: *main.c*

```
#include <stdio.h>
2 #include <stdlib.h>
  #include "stabla.h"
4
  /* Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
6    stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa u
     stablo, a inace 0. */
8 int ucitaj_stablo(Cvor ** koren)
  {
```

```
10    *koren = NULL;
      int x;
12    /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
         dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
14       je i povratna vrednost iz funkcije ucitaj_stablo 1. */
      while (scanf("%d", &x) == 1)
16      if (dodaj_u_stablo(koren, x) == 1)
          return 1;
18    return 0;
}
20
/* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
22 int prebrojN(Cvor * koren, int n)
{
24    /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
         negativan broj, na tom nivou nema cvorova (rezultat je nula). */
26    if (koren == NULL || n < 0)
        return 0;
28    /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima jednog
         potomka funkcija vraca 1, inace 0 */
30    if (n == 0) {
        if (koren->levo == NULL && koren->desno != NULL)
32        return 1;
        if (koren->levo != NULL && koren->desno == NULL)
34        return 1;
        return 0;
36    }
      /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
38       (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
         levog podstabla */
40    return prebrojN(koren->levo, n - 1) + prebrojN(koren->desno, n - 1)
        ;
}
42
int main()
44 {
      Cvor *koren;
46    int n;
      scanf("%d", &n);
48
      /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
50       oslobodja se alocirana memorija i izlazi se iz programa. */
      if (ucitaj_stablo(&koren) == 1) {
52      fprintf(stderr, "-1\n");
        oslobodi_stablo(&koren);
54      exit(EXIT_FAILURE);
      }
56
      /* Ispisivanje rezultata */
58    printf("%d\n", prebrojN(koren, n));

60    /* Oslobadjanje dinamicki alociranog stabla */
```

```
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

Rešenje 5.7

```
#include <stdio.h>

/* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
   zapis argumenta x rotira za n mesta udesno */
unsigned int Rotiraj(unsigned int x, unsigned int n)
{
  int i;
  unsigned int maska = 1;
  /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
     izgledala: 000...00001111
   Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
   nastavku je drugi nacin. */
  for (i = 1; i < n; i++)
    maska = (maska << 1) | 1;
  /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n bitova
     binarne reprezentacije broja x ce "ispasti". Za rotaciju je
     potrebno da se tih n bitova postavi na pocetak broja. Kreirana
     maska omogucava da se tih n bitova izdvoji sa (maska & x), a
     zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta ulevo
    tih
     n bitova postavlja na pocetak. Primenom logicke disjunkcije
     dobija se rotirani broj. */
  return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
}

int main()
{
  unsigned int x, n;

  /* Ucitavanje brojeva sa standardnog ulaza */
  scanf("%u%u", &x, &n);

  /* Ispisivanje rezultata */
  printf("%u\n", Rotiraj(x, n));
  return 0;
}
```

Rešenje 5.8

Datoteka 5.3: *liste.h*

```
#ifndef _LISTE_H_
#define _LISTE_H_ 1

/* Struktura koja predstavlja cvor liste */
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 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. Vraca 1 ukoliko je bilo
                                    greske pri alokaciji memorije,
                                    inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
void ispisi_listu(Cvor * glava);

#endif
```

Datoteka 5.4: *liste.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)
```

```
10      return NULL;
    /* inicijalizacija polja u novom cvoru */
12    novi->vrednost = broj;
    novi->sledeci = NULL;
14    return novi;

16  }

18  /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
       ciji se pocetni cvor nalazi na adresi adresa_glave. */
20  void oslobodi_listu(Cvor ** adresa_glave)
    {
22    Cvor *pomocni = NULL;
    while (*adresa_glave != NULL) {
24      pomocni = (*adresa_glave)->sledeci;
      free(*adresa_glave);
26      *adresa_glave = pomocni;

28  }

30  }

32  /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
       ili NULL kao je lista prazna */
34  Cvor *pronadji_poslednji(Cvor * glava)
    {
36    /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
         funkcija vraca NULL. */
38    if (glava == NULL)
      return NULL;
40    while (glava->sledeci != NULL)
      glava = glava->sledeci;
42    return glava;

44  }

46  /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
       greske pri alokaciji memorije, inace vraca 0. */
48  int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
    {
50    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
52      return 1;
    if (*adresa_glave == NULL) {
54      *adresa_glave = novi;
      return 0;
56
  }
58    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
60
  }
```

```
62
/* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
64 void ispisi_listu(Cvor * glava)
{
66   for (; glava != NULL; glava = glava->sledeci)
       printf("%.2lf ", glava->vrednost);
68   putchar('\n');

70 }
```

Datoteka 5.5: *main.c*

```
#include <stdio.h>
2 #include <stdlib.h>
#include "liste.h"
4
/* Funkcija umece novi cvor iza tekuceg u listi */
6 void dodaj_iza(Cvor * tekuci, Cvor * novi)
{
8   /* Novi cvor se dodaje iza tekuceg cvora. */
   novi->sledeci = tekuci->sledeci;
10  tekuci->sledeci = novi;
}
12


14 /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
    Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
16  0. */
int dopuni_listu(Cvor ** adresa_glave)
18 {
   Cvor *tekuci;
20  Cvor *novi;
   double aritmeticka_sredina;
22  /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
   if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
24   return 1;
   /* Promenljiva tekuci se inicijalizacuje da pokazuje na pocetni
26     cvor */
   tekuci = *adresa_glave;
28  /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
      vrednosti u susednim cvorovims i kreira cvor sa tom vrednoscu. U
30    slucaju neupele alokacije novog cvora, funkcija vraca 1. Inace,
      novi cvor se umece izmedju dva cvora za koje racunata
32    aritmeticka sredina */
   while (tekuci->sledeci != NULL) {
34   aritmeticka_sredina =
        ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
36   novi = napravi_cvor(aritmeticka_sredina);
     if (novi == NULL)
38     return 1;
```

```
    /* Poziva se funkcija koja umece novi cvor iza tekuceg cvora */
    dodaj_iza(tekuci, novi);
    /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
       cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz
       polazne liste */
    tekuci = tekuci->sledeci;
    tekuci = tekuci->sledeci;
  }
  return 0;
}

int main()
{
  Cvor *glava = NULL;
  double broj;

  /* Ucitavanje se vrsi do kraja ulaza. Elementi se dodaju na kraj
     liste! */
  while (scanf("%lf", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
      fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
  }

  /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
     onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
     alociranu za cvorove liste treba osloboditi. */
  if (dopuni_listu(&glava) == 1) {
    fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
    oslobodi_listu(&glava);
    exit(EXIT_FAILURE);
  }

  /* Ispisivanje liste */
  ispisi_listu(glava);

  /* Oslobadjanje liste */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

Rešenje 5.9

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

/* Funkcija racuna zbir elemenata v-te vrste */
int zbir_vrste(int **M, int n, int v)
{
  int j, zbir = 0;
  for (j = 0; j < n; j++)
    zbir += M[v][j];
  return zbir;
}

/* Funkcija racuna zbir elemenata k-te kolone */
int zbir_kolone(int **M, int n, int k)
{
  int i, zbir = 0;
  for (i = 0; i < n; i++)
    zbir += M[i][k];
  return zbir;
}

/* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
   argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
   Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
   kolone ukoliko je kvadrat magican. */
int magicni_kvadrat(int **M, int n, int *zbirmagicnog)
{
  int i, j;
  int zbir = 0, zbir_pom;
  /* Promenljivu zbir inicijalizujemo na zbir 0-te vreste */
  zbir = zbir_vrste(M, n, 0);

  /* Racunaju se zbirovi u ostalim vrstama i ako neki razlikuje od
     vrednosti promeljive zbir funkcija vraca 1 */
  for (i = 1; i < n; i++) {
    zbir_pom = zbir_vrste(M, n, i);
    if (zbir_pom != zbir)
      return 0;
  }
  /* Racunaju se zbirovi u svim kolonama i ako neki razlikuje od
     vrednosti promeljive zbir funkcija vraca 1 */
  for (j = 0; j < n; j++) {
    zbir_pom = zbir_kolone(M, n, j);
    if (zbir_pom != zbir)
      return 0;
  }
  /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
     vresnost u zbirmagicnog i funkcija vraca 1 */
  *zbirmagicnog = zbir;
  return 1;
}
```

```
54 int main()
{
56   int n, i, j;
   int **matrica = NULL;
58   int zbir = 0, zbirmagicnog = 0;
   scanf("%d", &n);
60
   /* Provera da li je n strogo pozitivan */
62   if (n <= 0) {
     printf("-1\n");
64     exit(EXIT_FAILURE);
   }
66
   /* Dinamicka alokacija matrice dimenzije nxn */
68   matrica = alociraj_matricu(n);
   if (matrica == NULL) {
70     printf("-1\n");
     exit(EXIT_FAILURE);
72   }

74   /* Ucitavanje elemenata matrice sa standardnog ulaza */
   ucitaj_matricu(matrica, n);
76
   /* Ispisivanje rezultata na osnovu fukcije magicni_kvadrat */
78   if (magicni_kvadrat(matrica, n, &zbirmagicnog)) {
     printf("%d\n", zbirmagicnog);
80   } else
     printf("-\n");
82
   /* Oslobadjanje dinamicki alocirane memorije */
84   matrica = dealociraj_matricu(matrica, n);

86   exit(EXIT_SUCCESS);
}
```