

Univerzitet u Beogradu
Matematički fakultet

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

Programiranje 2 Zbirka zadataka sa rešenjima

Beograd, 2015.

Predgovor

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

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

...

Autori

Sadržaj

1	Uvodni zadaci	3
1.1	Podela koda po datotekama	3
1.2	Algoritmi za rad sa bitovima	6
1.3	Rekurzija	11
1.4	Rešenja	18
2	Pokazivači	61
2.1	Pokazivačka aritmetika	61
2.2	Višedimenzioni nizovi	65
2.3	Dinamička alokacija memorije	70
2.4	Pokazivači na funkcije	75
2.5	Rešenja	76
3	Algoritmi pretrage i sortiranja	113
3.1	Pretraživanje	113
3.2	Sortiranje	117
3.3	Bibliotečke funkcije pretrage i sortiranja	126
3.4	Rešenja	131
4	Dinamičke strukture podataka	205
4.1	Liste	205
4.2	Stabla	215
4.3	Rešenja	224
5	Ispitni rokovi	317
5.1	Programiranje 2, praktični deo ispita, jun 2015.	317
5.2	Programiranje 2, praktični deo ispita, jul 2015.	319
5.3	Programiranje 2, praktični deo ispita, septembar 2015.	321
5.4	Rešenja	323

Glava 1

Uvodni zadaci

1.1 Podela koda po datotekama

Zadatak 1.1 Napisati program za rad sa kompleksnim brojevima.

- (a) Definirati strukturu `KompleksanBroj` koja opisuje kompleksan broj njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `ucitaj_kompleksan_broj` koja učitava kompleksan broj sa standardnog ulaza.
- (c) Napisati funkciju `ispisi_kompleksan_broj` koja ispisuje kompleksan broj na standardni izlaz u odgovarajućem formatu (npr. broj čiji je realan deo 2, a imaginarni -3 ispisati kao $(2 - 3i)$ na standardni izlaz).
- (d) Napisati funkciju `realan_deo` koja vraća vrednost realnog dela broja.
- (e) Napisati funkciju `imaginaran_deo` koja vraća vrednost imaginarnog dela broja.
- (f) Napisati funkciju `moduo` koja računa moduo kompleksnog broja.
- (g) Napisati funkciju `konjugovan` koja računa konjugovano-kompleksni broj svog argumenta.
- (h) Napisati funkciju `saberi` koja sabira dva kompleksna broja.
- (i) Napisati funkciju `oduzmi` koja oduzima dva kompleksna broja.
- (j) Napisati funkciju `mnozi` koja množi dva kompleksna broja.

1 Uvodni zadaci

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

Napisati program koji testira prethodno napisane funkcije. Program najpre za kompleksan broj z_1 koji se unosi sa standardnog ulaza ispisuje njegov realni deo, imaginarni deo i moduo. Zatim za naredni kompleksan broj z_2 koji se unosi sa standardnog ulaza ispisuje njegov konjugovano-kompleksan broj i argument. Na kraju program ispisuje zbir, razliku i proizvod brojeva z_1 i z_2 .

Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite realan i imaginaran deo kompleksnog broja: 1 -3
(1.00 - 3.00 i)
realan_deo: 1
imaginaran_deo: -3.000000
moduo 3.162278
Unesite realan i imaginaran deo kompleksnog broja: -1 4
(-1.00 + 4.00 i)
Njegov konjugovano kompleksan broj: (-1.00 - 4.00 i)
Argument kompleksnog broja: 1.815775
(1.00 - 3.00 i) + (-1.00 + 4.00 i) = (1.00 i)
(1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)
(1.00 - 3.00 i) * (-1.00 + 4.00 i) = (11.00 + 7.00 i)
```

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

Primer 1

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

Zadatak 1.3 Napisati biblioteku za rad sa polinomima.

- (a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20. UPUTSTVO: *Struktura sadrži stepen i niz koeficijenata. Redosled navođenja koeficijenata u nizu treba da bude takav da na nultoj poziciji u nizu bude koeficijent uz slobodan član, na prvoj koeficijent uz prvi stepen, itd.*
- (b) Napisati funkciju koja ispisuje polinom na standardni izlaz.
- (c) Napisati funkciju koja učitava polinom sa standardnog ulaza. Za polinom se najpre unosi stepen pa njegovi koeficijenti.
- (d) Napisati funkciju za izračunavanje vrednosti polinoma u datoj tački koristeći Hornerov algoritam.

(e) Napisati funkciju koja sabira dva polinoma.

(f) Napisati funkciju koja množi dva polinoma.

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

Primer 1

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

Zadatak 1.4 Napisati biblioteku za rad sa razlomcima.

- (a) Definisati strukturu za reprezentovanje razlomaka.
- (b) Napisati funkcije za učitavanje i ispis razlomaka.
- (c) Napisati funkcije koje vraćaju brojilac i imenilac razlomka.
- (d) Napisati funkciju koja vraća odgovarajuću realnu vrednost razlomka (tipa `double`).
- (e) Napisati funkciju koja izračunava recipročnu vrednost razlomka.
- (f) Napisati funkciju koja skraćuje dati razlomak.
- (g) Napisati funkcije koje sabiraju, oduzimaju, množe i dele dva razlomka.

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

INTERAKCIJA

ULAZ:

ULAZ:

ULAZ:

ULAZ:

Test 1

```

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

```

Test 2

```

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

```

Test 3

```

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

```

Test 4

```

ULAZ:
  0xABCD123
IZLAZ:
  Broj jedinica u zapisu je
  funkcija count_bits1: 17
  funkcija count_bits2: 17

```

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

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

Test 1

```

ULAZ:
  0x7F
IZLAZ:
  Najveci:
  11111110000000000000000000000000
  Najmanji:
  000000000000000000000000111111

```

Test 2

```

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

```

Test 3

```

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

```

Test 4

```

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

```

Zadatak 1.8 Napisati funkcije za rad sa bitovima.

- (a) Napisati funkciju koja određuje broj koji se dobija kada se n bitova datog

1 Uvodni zadaci

broja, počevši od pozicije p , postave na 0.

- (b) Napisati funkciju koja određuje broj koji se dobija kada se n bitova datog broja, počevši od pozicije p , postave na 1.
- (c) Napisati funkciju koja određuje broj koji se dobija od n bitova datog broja, počevši od pozicije p , i vraća ih kao bitove najmanje težine rezultata.
- (d) Napisati funkciju koja vraća broj koji se dobija upisivanjem poslednjih n bitova broja y u broj x , počevši od pozicije p .
- (e) Napisati funkciju koja vraća broj koji se dobija invertovanjem n bitova broja x počevši od pozicije p .

Napisati program koji testira prethodno napisane funkcije nad neoznačenim celim brojem koji se unosi sa standardnog ulaza. NAPOMENA: *Pozicije se broje počev od pozicije bita najmanje težine, pri čemu je bit najmanje težine na poziciji nula.*

Test 1

```

ULAZ:
    235 5 10 127

IZLAZ:
    Broj 235 = 00000000000000000000000011101011
    reset(235, 5, 10) = 0000000000000000000000000000101011
    set(235, 5, 10) = 0000000000000000000000001111101011
    get_bits(235, 5, 10) = 000000000000000000000000000000011
    y = 127 = 00000000000000000000000000001111111
    set_n_bits(235, 5, 10, 127) = 0000000000000000000000001111101011
    invert(235, 5, 10) = 0000000000000000000000000011100101011

```

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

- Napisati funkciju `rotate_left` koja određuje broj koji se dobija rotiranjem `k` puta ulevo datog celog broja `x`.
- Napisati funkciju `rotate_right` koja određuje broj koji se dobija rotiranjem `k` puta udesno datog celog neoznačenog broja `x`.
- 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 heksadekaskom formatu sa standardnog ulaza.

Test 1

```

|| ULAZ:
|| B10011A7 5
|| IZLAZ:
|| x = 10110001000000000001000110100111
|| rotate_left(2969571751, 5) = 001000000000000100011010011110110
|| rotate_right(2969571751, 5) = 0011110110001000000000010001101
|| rotate_right_signed(2969571751, 5) = 0011110110001000000000010001101

```

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 heksadekaskom 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:
|| 000000000000000000000001001010101
|| 10101010010000000000000000000000

```

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

Test 1

```

|| ULAZ:
|| 10
|| IZLAZ:
|| 0

```

Test 2

```

|| ULAZ:
|| 2147377146
|| IZLAZ:
|| 1

```

Test 3

```

|| ULAZ:
|| 1111111115
|| IZLAZ:
|| 0

```

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 11 IZLAZ: 1 </pre>	<pre> ULAZ: 1024 IZLAZ: 0 </pre>	<pre> ULAZ: 2147377146 IZLAZ: 22 </pre>

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 $+$, $-$, $/$, $*$, $\%$.

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
<pre> Poziv: ./a.out 1 2 INTERAKCIJA PROGRAMA: 11 13 </pre>	<pre> Poziv: ./a.out 1 2 INTERAKCIJA PROGRAMA: 1024 1024 </pre>	<pre> Poziv: ./a.out 12 12 INTERAKCIJA PROGRAMA: 12345 12345 </pre>

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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 11 IZLAZ: 0000000B </pre>	<pre> ULAZ: 1024 IZLAZ: 00000400 </pre>	<pre> ULAZ: 12345 IZLAZ: 00003039 </pre>

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	Test 2	Test 3
ULAZ: 123 10 IZLAZ: 4294967285	ULAZ: 3251 0 IZLAZ: 4294967295	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	Test 2	Test 3
ULAZ: 123 IZLAZ: 0	ULAZ: 3245 IZLAZ: 2	ULAZ: 100328 IZLAZ: 1

1.3 Rekurzija

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

Test 1	Test 2	Test 3
ULAZ: 2 10 IZLAZ: 1024	ULAZ: 5 3 IZLAZ: 125	ULAZ: 9 4 IZLAZ: 6561

Zadatak 1.18 Koristeći uzajamnu (posrednu) rekurziju napisati:

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

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

Test 1

```
|| ULAZ:
|| 11
|| IZLAZ:
|| Uneti broj ima paran broj cifara
```

Test 2

```
|| ULAZ:
|| 123
|| IZLAZ:
|| Uneti broj ima neparan broj cifara
```

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

Primer 1

```
|| INTERAKCIJA PROGRAMA:
|| Unesite n (<= 12): 5
|| 5! = 120
```

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

$$F(0) = 0$$

$$F(1) = 1$$

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

Napisati rekurzivnu funkciju koja računa n -ti element u nizu F , ali tako da se problemi manje dimenzije rešavaju samo jedan put. Napisati program koji testira napisanu funkciju za vrednosti koeficijenata i prirodan broj n koji se unose sa standardnog ulaza.

Primer 1

```
|| INTERAKCIJA PROGRAMA:
|| Unesite koeficijente 2 3
|| Unesite koji clan niza se racuna 5
|| F(5) = 61
```

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

Test 1

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

Test 2

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

Test 3

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


Test 4

```

ULAZ:
  1
IZLAZ:
  1

```

Test 5

```

ULAZ:
  0
IZLAZ:
  0

```

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

Test 1

```

ULAZ:
  5 1 2 3 4 5
IZLAZ:
  Suma elemenata je 15

```

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

Test 1

```

ULAZ:
  3 2 1 4 21
IZLAZ:
  21

```

Test 2

```

ULAZ:
  2 -1 0 -5 -10
IZLAZ:
  2

```

Test 3

```

ULAZ:
  1 11 3 5 8 1
IZLAZ:
  11

```

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

Test 1

```

ULAZ:
  3 1 2 3 1 2 3
IZLAZ:
  14

```

Test 2

```

ULAZ:
  2 3 5 2 6
IZLAZ:
  36

```

Test 3

```

ULAZ:
  0
IZLAZ:
  0

```

Zadatak 1.25 Napisati rekurzivnu funkciju **br_pojave** koja računa broj

1 Uvodni zadaci

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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 4 1 2 3 4 IZLAZ: 1</pre>	<pre> ULAZ: 11 3 2 11 14 11 43 1 IZLAZ: 2</pre>	<pre> ULAZ: 1 3 21 5 6 IZLAZ: 0</pre>

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

<i>Test 1</i>	<i>Test 2</i>
<pre> ULAZ: 1 2 3 4 1 2 3 4 5 IZLAZ: da</pre>	<pre> ULAZ: 1 2 3 11 1 2 4 3 6 IZLAZ: ne</pre>

Zadatak 1.27 Napisati rekursivnu 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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 0x7F IZLAZ: 7</pre>	<pre> ULAZ: 0x00FF00FF IZLAZ: 16</pre>	<pre> ULAZ: 0xFFFFFFFF IZLAZ: 32</pre>

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

Test 1

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

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

Test 1

ULAZ:
5
IZLAZ:
5

Test 2

ULAZ:	125
IZLAZ:	7

Test 3

ULAZ:	8
IZLAZ:	1

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

Test 1

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

Test 2

ULAZ:
16
IZLAZ:
1

Test 3

ULAZ:	18
IZLAZ:	2

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

Test 1

ULAZ:	a
IZLAZ:	da

Test 2

ULAZ:	<i>aa</i>
IZLAZ:	<i>da</i>

Test 3

ULAZ:	aba
IZLAZ:	da

Test 4

```

ULAZ:
programiranje
IZLAZ:
ne

```

Test 5

```

ULAZ:
anavolimitovana
IZLAZ:
da

```

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

Primer 1

```

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

```

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

```

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

```

- Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta $\binom{n}{k}$ koristeći osobine Paskalovog trougla.
- Napisati rekurzivnu funkciju koja izračunava d_n kao sumu elemenata n -te hipotenuze Paskalovog trougla.

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

Test 1

```

ULAZ:
5 3
IZLAZ:

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

8

```

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

Test 1

```

ULAZ:
3
IZLAZ:
a a a
a a b
a b a
a b b
b a a
b a b
b b a
b b b

```

Zadatak 1.35 *Hanojske kule*: Data su tri vertikalna štapa, na jednom se nalazi n diskova poluprečnika $1, 2, 3, \dots$ 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 $1, 2, 3, \dots$ 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

```
1 #include <stdio.h>
2 #include <math.h>
3
4 /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
5    imaginaran deo kompleksnog broja */
6 typedef struct {
7     float real;
8     float imag;
9 } KompleksanBroj;
10
11 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
12    kompleksnog broja i smesta ih u strukturu cija adresa je argument
13    funkcije */
14 void ucitaj_kompleksan_broj(KompleksanBroj * z)
15 {
16     printf("Unesite realan i imaginaran deo kompleksnog broja: ");
17     scanf("%f", &z->real);
18     scanf("%f", &z->imag);
19 }
20
21 /* Funkcija ispisuje na standardan izlaz kompleksan broj z koji joj
22    se salje kao argument u obliku (x + i y) Ovoj funkciji se
23    kompleksan broj prenosi po vrednosti (za ispis nije neophodna
24    adresa) */
25 void ispsi_kompleksan_broj(KompleksanBroj z)
26 {
27     printf("(");
28     if (z.real != 0) {
29         printf("%.2f", z.real);
30         if (z.imag > 0)
31             printf(" +");
32     }
33     if (z.imag != 0)
34         printf(" %.2f i ", z.imag);
35
36     if (z.imag == 0 && z.real == 0)
37         printf("0 ");
38
39     printf(")");
40 }
41 }
```

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

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

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

61
63 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
    odgovara kompleksnom broju poslatom kao argument */
KompleksanBroj konjugovan(KompleksanBroj z)
65 {
    KompleksanBroj z1 = z;
67
    z1.imag *= -1;
69
    return z1;
71 }

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

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

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

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

```
95 }

97 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
   argumenata funkcije */
99 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
100 {
101     KompleksanBroj z;

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

105     return z;
106 }

109 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
   kao argument */
111 float argument(KompleksanBroj z)
112 {
113     return atan2(z.imag, z.real);
114 }

115 int main()
116 {
117     /* Deklaracija 2 promenljive tipa KompleksanBroj */
118     KompleksanBroj z1, z2;

121     /* Ucitavanje prvog kompleksnog broja u promenljivu z1, a potom
       njegovo ispisivanje na standardni izlaz */
123     ucitaj_kompleksan_broj(&z1);
124     ispisi_kompleksan_broj(z1);

125     /* Ispisuje se na standardni izlaz realan, imaginaran deo i moduo
       kompleksnog broja z1 */
127     printf("\nrealan_deo: %.f\nimaginaran_deo: %.f\nmoduo %.f\n",
128           realan_deo(z1), imaginaran_deo(z1), moduo(z1));
129     printf("\n");

131     /* Ucitavanje drugog kompleksnog broja u promenljivu z2, a potom
       njegovo ispisivanje na standardni izlaz */
133     ucitaj_kompleksan_broj(&z2);
134     ispisi_kompleksan_broj(z2);

137     /* Racunanje i ispisivanje konjugovano kompleksan broj od z2 */
138     printf("\nNjegov konjugovano kompleksan broj: ");
139     ispisi_kompleksan_broj(konjugovan(z2));
140     printf("\n");

141     /* Sabiranje kompleksnih brojeva */
143     printf("\n");
144     ispisi_kompleksan_broj(z1);
145     printf(" + ");
146     ispisi_kompleksan_broj(z2);
```



```

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

    /* Oduzimanje kompleksnih brojeva */
    printf("\n");
153 ispisi_kompleksan_broj(z1);
    printf(" - ");
155 ispisi_kompleksan_broj(z2);
    printf(" = ");
157 ispisi_kompleksan_broj(oduzmi(z1, z2));
    printf("\n");

159     /* Mnozenje kompleksnih brojeva */
    printf("\n");
161 ispisi_kompleksan_broj(z1);
    printf(" * ");
163 ispisi_kompleksan_broj(z2);
    printf(" = ");
165 ispisi_kompleksan_broj(mnozi(z1, z2));

    /* Testiranje funkcije koja racuna argument kompleksnih brojeva */
169 printf("\n");
    ispisi_kompleksan_broj(z2);
171 printf("\nArgument kompleksnog broja %f\n", argument(z2));

173 return 0;
}

```

Rešenje 1.2

```

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

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

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

```

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

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

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

35  printf(")");
    }

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

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

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

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

65  /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
    argumenata funkcije */
67  KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
69  {
    KompleksanBroj z = z1;

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

```

73     z.imag += z2.imag;

75     return z;
76 }

77 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
78    argumenata funkcije */
79 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
80 {
81     KompleksanBroj z = z1;
82
83     z.real -= z2.real;
84     z.imag -= z2.imag;
85
86     return z;
87 }

88 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
89    argumenata funkcije */
90 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
91 {
92     KompleksanBroj z;
93
94     z.real = z1.real * z2.real - z1.imag * z2.imag;
95     z.imag = z1.real * z2.imag + z1.imag * z2.real;
96
97     return z;
98 }

99 /* Funkcija vraca argument kompleksnog broja koji je funkciji poslat
100    kao argument */
101 float argument(KompleksanBroj z)
102 {
103     return atan2(z.imag, z.real);
104 }

105
106
107
108
109
110
111 /*
112    Zaglavlje complex.h sadrzi definiciju tipa KompleksanBroj i
113    deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje
114    nikada ne treba da sadrzi definicije funkcija. Da bi neki program
115    mogao da koristi ove brojeve i funkcije iz ove biblioteke,
116    neophodno je da ukljuci ovo zaglavlje. */

117
118 /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i time
119    onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci. Niska
120    posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
121    narednoj pretprocesorskoj define direktivi. */
122 #ifndef _COMPLEX_H
123 #define _COMPLEX_H

124
125 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
126    koje se koriste u definicijama funkcija navedenim u complex.c */

```

1 Uvodni zadaci

```
17 #include <stdio.h>
   #include <math.h>
19
   /* Struktura KompleksanBroj */
21 typedef struct {
   float real;
23 float imag;
   } KompleksanBroj;
25
   /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
27   definisane u complex.c */
   void ucitaj_kompleksan_broj(KompleksanBroj * z);
29
   void ispisi_kompleksan_broj(KompleksanBroj z);
31
   float realan_deo(KompleksanBroj z);
33
   float imaginaran_deo(KompleksanBroj z);
35
   float moduo(KompleksanBroj z);
37
   KompleksanBroj konjugovan(KompleksanBroj z);
39
   KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
41
   KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
43
   KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
45
   float argument(KompleksanBroj z);
47
   /* Kraj zakljucanog dela */
49 #endif
```

```
1  /*****
   Ovaj program koristi korektno definisanu biblioteku kompleksnih
3  brojeva. U zaglavlju complex.h nalazi se definicija kompleksnog
   broja i popis deklaracija podrzanih funkcija, a u complex.c se
5  nalaze njihove definicije.

7  Kompilacija programa se najjednostavnije postize naredbom
   gcc -Wall -lm -o izvrsni complex.c main.c
9
   Kompilacija se moze uraditi i na sledeci nacin:
11 gcc -Wall -c -o complex.o complex.c
   gcc -Wall -c -o main.o main.c
13 gcc -lm -o complex complex.o main.o
   *****/
15

17 #include <stdio.h>
```

```

/* Uključuje aw zaglavlje neophodno za rad sa kompleksnim brojevima
*/
19 #include "complex.h"

21 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
   polarni oblik */
23 int main()
24 {
25     KompleksanBroj z;

27     /* Učitavamo kompleksan broj */
    ucitaj_kompleksan_broj(&z);

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

33     return 0;
}

```

Rešenje 1.3

```

1  #include <stdio.h>
   #include <stdlib.h>
3  #include "polinom.h"

5
6  /* Funkcija koja ispisuje polinom na standardan izlaz u citljivom
7     obliku. Kako bi uštedeli kopiranje cele strukture, polinom
8     prenosimo po adresi */
9  void ispisi(const Polinom * p)
10 {
11     int i;
12     for (i = p->stepen; i >= 0; i--) {
13         if (p->koef[i]) {
14             if (p->koef[i] >= 0 && i != p->stepen)
15                 putchar('+');
16             if (i > 1)
17                 printf("%.2fx~d", p->koef[i], i);
18             else if (i == 1)
19                 printf("%.2fx", p->koef[i]);
20             else
21                 printf("%.2f", p->koef[i]);
22         }
23     }
24     putchar('\n');
25 }

27 /* Funkcija koja učitava polinom sa tastature */
Polinom ucitaj()
29 {
    int i;

```

1 Uvodni zadaci

```
31 Polinom p;

33 /* Ucitavamo stepen polinoma */
scanf("%d", &p.stepen);

35
37 /* Ponavljamo učitavanje stepena sve dok ne unesemo stepen iz
   dozwoljenog opsega */
while (p.stepen > MAX_STEPEN || p.stepen < 0) {
39     printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
    scanf("%d", &p.stepen);
41 }

43 /* Unosimo koeficijente polinoma */
for (i = p.stepen; i >= 0; i--)
45     scanf("%lf", &p.koef[i]);
return p;
47 }

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

61 /* Funkcija koja sabira dva polinoma */
Polinom saberi(const Polinom * p, const Polinom * q)
63 {
    Polinom rez;
65     int i;

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

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

    return rez;
75 }

77
79 /* Funkcija mnozi dva polinoma p i q */
Polinom pomnozi(const Polinom * p, const Polinom * q)
{
81     int i, j;
    Polinom r;
```

```
83     r.stepen = p->stepen + q->stepen;
85     if (r.stepen > MAX_STEPEN) {
86         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
87         exit(EXIT_FAILURE);
88     }
89
90     for (i = 0; i <= r.stepen; i++)
91         r.koef[i] = 0;
92
93     for (i = 0; i <= p->stepen; i++)
94         for (j = 0; j <= q->stepen; j++)
95             r.koef[i + j] += p->koef[i] * q->koef[j];
96
97     return r;
98 }
99
100 /* Funkcija racuna izvod polinoma p */
101 Polinom izvod(const Polinom * p)
102 {
103     int i;
104     Polinom r;
105
106     if (p->stepen > 0) {
107         r.stepen = p->stepen - 1;
108
109         for (i = 0; i <= r.stepen; i++)
110             r.koef[i] = (i + 1) * p->koef[i + 1];
111     } else
112         r.koef[0] = r.stepen = 0;
113
114     return r;
115 }
116
117 /* Funkcija racuna n-ti izvod polinoma p */
118 Polinom nIzvod(const Polinom * p, int n)
119 {
120     int i;
121     Polinom r;
122
123     if (n < 0) {
124         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
125         exit(EXIT_FAILURE);
126     }
127
128     if (n == 0)
129         return *p;
130
131     r = izvod(p);
132     for (i = 1; i < n; i++)
133         r = izvod(&r);
```

1 Uvodni zadaci

```
135     return r;
    }
```

```
2  /* Ovim pretrocesorskim direktivama zakljucavamo zaglavlje i time
   onemogucujemo da se sadrzaj zaglavlja vise puta ukljuci */
4  #ifndef _POLINOM_H
   #define _POLINOM_H
6
   #include <stdio.h>
8  #include <stdlib.h>

10 /* Maksimalni stepen polinoma */
   #define MAX_STEPEN 20
12

14 /* Polinome predstavljamo strukturom koja cuva koeficijente (koef[i]
   je koeficijent uz clan x^i) i stepen polinoma */
16 typedef struct {
   double koef[MAX_STEPEN + 1];
18   int stepen;
   } Polinom;
20

22 /* Funkcija koja ispisuje polinom na stdout u citljivom obliku
   Polinom prenosimo po adresi, da bi ustedeli kopiranje cele
   strukture, vec samo prenosimo adresu na kojoj se nalazi polinom
24   kog ispisujemo */
   void ispisi(const Polinom * p);
26

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

30 /* Funkcija racuna vrednost polinoma p u tacki x Hornerovim
   algoritmom */
32 /*  $x^4 + 2x^3 + 3x^2 + 2x + 1 = ((x+2)x + 3)x + 2)x + 1$  */
   double izracunaj(const Polinom * p, double x);
34

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

38 /* Funkcija mnozi dva polinoma p i q */
   Polinom pomnozi(const Polinom * p, const Polinom * q);
40

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

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

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



```
2 #include "polinom.h"
4 /*
6     Prevodjenje: gcc -o test-polinom polinom.c main.c
8     ili: gcc -c polinom.c gcc -c main.c gcc -o test-polinom polinom.o
9     main.o */
10 int main(int argc, char **argv)
11 {
12     Polinom p, q, r;
13     double x;
14     int n;
16     /* Unos polinoma */
17     printf
18     ("Unesite polinom (prvo stepen, pa zatim koeficijente od
19      najveceg stepena do nultog):\n");
20     p = ucitaj();
22     /* Ispis polinoma */
23     ispisi(&p);
25     printf("Unesite tacku u kojoj racunate vrednost polinoma\n");
26     scanf("%lf", &x);
28     /* Ispisujemo vrednost polinoma u toj tacki */
29     printf("Vrednost polinoma u tacki je %.2f\n", izracunaj(&p, x));
31     /* Unesimo drugi polinom */
32     printf
33     ("Unesite drugi polinom (prvo stepen, pa zatim koeficijente od
34      najveceg stepena do nultog):\n");
35     q = ucitaj();
37     /* Sabiramo polinome i ispisujemo zbir ta dva polinoma */
38     r = saberi(&p, &q);
39     printf("Zbir polinoma je: ");
40     ispisi(&r);
42     /* Mnozimo polinome i ispisujemo proizvod ta dva polinoma */
43     r = pomnozi(&p, &q);
44     printf("Prozvod polinoma je: ");
45     ispisi(&r);
47     /* Izvod polinoma */
48     printf("Unesite izvod polinoma koji zelite:\n");
49     scanf("%d", &n);
50     r = nIzvod(&p, n);
51     printf("%d. izvod prvog polinoma je: ", n);
52     ispisi(&r);
```

1 Uvodni zadaci

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

Rešenje 1.5

```
#include <stdio.h>

2
/* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
4  celog broja u memoriji. Bitove koji predstavljaju binarnu
   reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
6  najveće težine ka bitu najmanje težine. */
void print_bits(unsigned x)
8 {
    /* Broj bitova celog broja */
    unsigned velicina = sizeof(unsigned) * 8;
    /* Maska koja se koristi za "ocitavanje" bitova */
    unsigned maska;

    /* Početna vrednost maske se postavlja na broj čiji binarni zapis
16   na mestu bita najveće težine sadrži jedinicu, a na svim ostalim
   mestima sadrži nulu. U svakoj iteraciji maska se menja tako što
18   se jedini bit jedinica pomera udesno, kako bi se ocitao naredni
   bit broja x koji je argument funkcije. Odgovarajući karakter,
20   ('0' ili '1'), ispisuje se na standardnom izlazu. Neophodno je
   da promenljiva maska bude deklarirana kao neoznačen ceo broj
22   kako bi se siftovanjem u desno vrsilo logičko siftovanje
   (popunjavanje nulama) a ne aritmetičko siftovanje (popunjavanje
24   znakom broja). */
    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
26         putchar(x & maska ? '1' : '0');

    putchar('\n');
28 }

30

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

    return 0;
38 }
}
```

Rešenje 1.6

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

```
2
3
4 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
5    celog broja u memoriji */
6 void print_bits(int x)
7 {
8     unsigned velicina = sizeof(int) * 8;
9     unsigned maska;
10
11     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
12         putchar(x & maska ? '1' : '0');
13
14     putchar('\n');
15 }
16
17 /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja x
18    kreiranjem odgovarajuće maske i njenim pomeranjem */
19 int count_bits1(int x)
20 {
21     int br = 0;
22     unsigned wl = sizeof(unsigned) * 8 - 1;
23
24     /* Formiranje se maska čija binarna reprezentacija izgleda
25        100000...0000000, koja služi za očitavanje bita najveće težine.
26        U svakoj iteraciji maska se pomera u desno za 1 mesto, i
27        očitavamo sledeći bit. Petlja se završava kada više nema
28        jedinica tj. kada maska postane nula. */
29     unsigned maska = 1 << wl;
30     for (; maska != 0; maska >>= 1)
31         x & maska ? br++ : 1;
32
33     return br;
34 }
35
36 /* Funkcija vraća broj jedinica u binarnoj reprezentaciji broja x
37    formiranjem odgovarajuće maske i pomeranjem promenljive x */
38 int count_bits2(int x)
39 {
40     int br = 0;
41     unsigned wl = sizeof(int) * 8 - 1;
42
43     /* Kako je argument funkcije označen ceo broj x naredba x>>=1
44        vrsila bi aritmetičko pomeranje u desno, tj. popunjavanje bita
45        najveće težine bitom znaka. U tom slučaju nikad ne bi bio
46        ispunjen uslov x!=0 i program bi bio zarobljen u beskončnoj
47        petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
48        koja očitava bit najveće težine. */
49
50     unsigned maska = 1 << wl;
51     for (; x != 0; x <= 1)
52         x & maska ? br++ : 1;
53
54     return br;
55 }
```

```
54 }  
  
56  
57  
58 int main()  
59 {  
60     int x;  
61     scanf("%x", &x);  
62     printf("Broj jedinica u zapisu je\n");  
63     printf("funkcija count_bits1: %d\n", count_bits1(x));  
64     printf("funkcija count_bits2: %d\n", count_bits2(x));  
65     return 0;  
66 }
```

Rešenje 1.7

```
1  #include <stdio.h>  
  
3  /* Funkcija vraća najveći neoznačeni broj sastavljen od istih bitova  
   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */  
5  unsigned najveći(unsigned x)  
6  {  
7      unsigned velicina = sizeof(unsigned) * 8;  
  
9      /* Formira se maska 100000...0000000 */  
10     unsigned maska = 1 << (velicina - 1);  
  
11     /* Rezultat se inicijalizuje vrednoscu 0 */  
12     unsigned rezultat = 0;  
  
13     /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj  
       binarnoj reprezentaciji (tj. sve dok je promenljiva x različita  
       od nule). */  
14     for (; x != 0; x <<= 1) {  
15         /* Za svaku jedinicu koja se korišćenjem maske detektuje na  
           poziciji najveće težine u binarnoj reprezentaciji promenjive  
           x, potiskuje se jedna nova jedinicu sa leva u rezultat */  
16         if (x & maska) {  
17             rezultat >>= 1;  
18             rezultat |= maska;  
19         }  
20     }  
21     return rezultat;  
22 }  
  
23  
24  
25  
26  
27  
28  
29  
30  
31 /* Funkcija vraća najmanji neoznačeni broj sastavljen od istih bitova  
   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */  
32 unsigned najmanji(unsigned x)  
33 {  
34     /* Rezultat se inicijalizuje vrednoscu 0 */  
35     unsigned rezultat = 0;
```

```

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

51  return rezultat;
53 }

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

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

65     putchar('\n');
}

67 int main()
69 {
    int broj;
71     scanf("%x", &broj);

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

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

79     return 0;
}

```

Rešenje 1.8

```

#include <stdio.h>

2

4  /*****

```

1 Uvodni zadaci

```

6      Funkcija postavlja na nulu n bitova pocev od pozicije p.
      Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
      se broji od nule . Npr, za n=5, p=10 1010 1011 1100 1101 1110
8      1010 1110 0111 1010 1011 1100 1101 1110 1000 0010 0111 */
      unsigned reset(unsigned x, unsigned n, unsigned p)
10     {
      /******
12      Cilj je anulirati samo zeljene bitove, a da ostali
      ostanu nepromenjeni. Maska koja ce se koristiti je ona cija
14      binarna reprezentacija ima n bitova
      postavljenih na 0 pocev od pozicije p, dok su svi ostali
16      postavljeni na 1.

      Na primer, za n=5 i p=10 cilj je maska oblika
18      1111 1111 1111 1111 1111 1000 0011 1111
      To se postize na sledeci nacin:
20      ~0              1111 1111 1111 1111 1111 1111 1111 1111
22      (~0 << n)       1111 1111 1111 1111 1111 1111 1110 0000
      ~(~0 << n)        0000 0000 0000 0000 0000 0000 0001 1111
24      ~(~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
26      *****/
      unsigned maska = ~(~0 << n) << (p - n + 1));
28
      return x & maska;
30     }

32
      /******
34      Funkcija postavlja na 1 n bitova pocev od pozicije p.
      Pozicije se broje pocev od pozicije najnizeg bita, pri cemu
36      se broji od nule . Npr, za n=5, p=10
      1010 1011 1100 1101 1110 1010 1110 0111
38      1010 1011 1100 1101 1110 1111 1110 0111
      *****/
40     unsigned set(unsigned x, unsigned n, unsigned p)
      {
42
      /******
44      Cilj je samo odredjenih n bitova postaviti na 1, dok
      ostali treba da ostanu netaknuti. Na primer, za n=5 i p=10
46      formira se maska oblika
      0000 0000 0000 0000 0000 0111 1100 0000
48      prateci vrlo slican postupak kao za prethodnu funkciju
      *****/
50     unsigned maska = ~(~0 << n) << (p - n + 1);

52     return x | maska;
      }

54
56     /******
```

```

58   Funkcija vraca celobrojno polje bitova, desno poravnato, koje
predstavlja n bitova pocev od pozicije p u binarnoj
60   reprezentaciji broja x, pri cemu se pozicija broji sa desna
ulevo, gde je pocetna pozicija 0. Na primer za n = 5 i p = 10
62   i broj cija je binarna reprezentacija:
1010 1011 1100 1101 1110 1010 1110 0111
trazi se
64   0000 0000 0000 0000 0000 0000 0000 1011
*****/
66   unsigned get_bits(unsigned x, unsigned n, unsigned p)
{
68   /*
*****/
70   Kreira se maska kod koje su poslednjih n bitova 1, a
ostali su 0. Na primer za n=5
72   0000 0000 0000 0000 0000 0000 0001 1111
*****/
74   unsigned maska = ~(~0 << n);

76   /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
78   zeljenih n i funkcija vraca tako dobijenu vrednost */
return maska & (x >> (p - n + 1));
80 }

82 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
84   postavljeni na vrednosti n bitova najnize tezine binarne
reprezentacije broja y */
86   unsigned set_n_bits(unsigned x, unsigned n, unsigned p, unsigned y)
{
88   /*
*****/
90   Kreira se maska kod kod koje su poslednjih n bitova 1, a
ostali su 0. Na primer za n=5
92   0000 0000 0000 0000 0000 0000 0001 1111
*****/
94   unsigned last_n_1 = ~(~0 << n);
*****/
96   Kao sto je i u funkciji reset, i ovde se kreira masku koja ima n
bitova postavljenih na 0 pocevsi od pozicije p, dok su
ostali bitovi 1. Na primer za n=5 i p =10
98   1111 1111 1111 1111 1111 1000 0011 1111
*****/
100   unsigned middle_n_0 = ~(~0 << n) << (p - n + 1);

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

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

```

```

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

114     return x_reset ^ y_shift_middle;
}

116

118 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
        njih n */
120 unsigned invert(unsigned x, unsigned n, unsigned p)
{
122     /******
        Formira se maska sa n jedinica pocev od pozicije p.
        Na primer za n=5 i p=10
        0000 0000 0000 0000 0000 0111 1100 0000
        *****/
124     unsigned maska = ~(~0 << n) << (p - n + 1);

126     /* Operator ekskluzivno ili invertuje sve bitove gde je
        odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
128     return maska ^ x;
130 }

132

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

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

144     putchar('\n');
146 }

148

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

156

158     printf("reset(%u,%u,%u)%s = ", broj, n, p, "");
        print_bits(reset(broj, n, p));
160

```



```

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

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

166
168 printf("y = %u = ", y);
    print_bits(y);
    printf("set_n_bits(%u,%u,%u,%u) = ", broj, n, p, y);
170 print_bits(set_n_bits(broj, n, p, y));

172 printf("invert(%u,%u,%5u)%s = ", broj, n, p, "");
    print_bits(invert(broj, n, p));
174
176 return 0;
}

```

Rešenje 1.9

```

1  #include <stdio.h>

3  /*****
   Funkcija binarnu reprezentaciju svog argumenta x rotira u
5  levo za n mesta i vraća odgovarajući neoznačen ceo broj čija
   je binarna reprezentacija dobijena nakon rotacije.
7  Na primer za n =5 i x čija je interna reprezentacija
   1010 1011 1100 1101 1110 0001 0010 0011
9  funkcija vraća neoznačen ceo broj čija je binarna
   reprezentacija:
11  0111 1001 1011 1100 0010 0100 0111 0101
   *****/
13 unsigned rotate_left(int x, unsigned n)
{
15     unsigned first_bit;
    /* Maska koja ima samo najvisi bit postavljen na 1 neophodna da bi
17     pre siftovanja u levo za 1 najvisi bit bio sačuvan. */
    unsigned first_bit_mask = 1 << (sizeof(unsigned) * 8 - 1);
19     int i;

21     /* n puta se vrši rotaciju za jedan bit u levo. U svakoj iteraciji
       se odredi prvi bit, a potom se pomera binarna reprezentacija
23     trenutne vrednosti promenljive x u levo za 1. Nakon toga, potom
       najnizi bit se postavlja na vrednost koju je imao prvi bit koji
25     je istisnut siftovanjem */
    for (i = 0; i < n; i++) {
27         first_bit = x & first_bit_mask;
        x = x << 1 | first_bit >> (sizeof(unsigned) * 8 - 1);
29     }
    return x;
31 }

```

1 Uvodni zadaci

```
33  /*****
    Funkcija neoznaceni broj x rotira u desno za n.
    Na primer za n=5 i x cija je binarna reprezentacija
    1010 1011 1100 1101 1110 0001 0010 0011
    funkcija vraca neoznaceni ceo broj cija je binarna
    reprezentacija:
    0001 1101 0101 1110 0110 1111 0000 1001
    *****/
41  unsigned rotate_right(unsigned x, unsigned n)
    {
43      unsigned last_bit;
45      int i;

47      /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
        iteraciji se odredjuje bit najmanje tezine broja x, zatm tako
        odredjeni bit se siftuje u levo tako da najnizi bit dode do
        pozicije najviseg bita. Zatim, nakon siftovanja binarne
        reprezentacije trenutne vrednosti promenljive x za 1 u desno,
        najvisi bit se postavlja na vrednost vec zapamcenog bita koji je
        bio na poziciji najmanje tezine. */
53      for (i = 0; i < n; i++) {
        last_bit = x & 1;
55        x = x >> 1 | last_bit << (sizeof(unsigned) * 8 - 1);
      }

57      return x;
59  }

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

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

77        if (x < 0)
79          /*****
            Siftovanjem u desno broja koji je negativan dobija se 1
            kao bit na najvisoj poziciji. Na primer ako je x
            1010 1011 1100 1101 1110 0001 0010 001b
            (sa b je oznaceni ili 1 ili 0 na najnižoj poziciji)
            *****/

```

```

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

109    return x;
111}

113 /* Funkcija prikazuje na standardni ekran binarnu reprezentaciju
      celog broja u memoriji */
115 void print_bits(int x)
116 {
117     unsigned velicina = sizeof(int) * 8;
118     unsigned maska;
119     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
120         putchar(x & maska ? '1' : '0');
121
122     putchar('\n');
123 }

125 int main()
126 {
127     unsigned x, k;
128     scanf("%x%x", &x, &k);
129     printf("x %s = ", "");
130     print_bits(x);
131     printf("rotate_left(%u,%u)%s = ", x, k, "");
132     print_bits(rotate_left(x, k));
133
134     printf("rotate_right(%u,%u)%s = ", x, k, "");
135     print_bits(rotate_right(x, k));

```

1 Uvodni zadaci

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

Rešenje 1.10

```
1  #include <stdio.h>

3  /*****
   Funkcija vraća vrednost cija je binarna reprezentacija slika
   u ogledalu binarne reprezentacije broja x koji se prosledjuje
   kao argument funkcije. Na primer za x
   cija binarna reprezentacija izgleda ovako
   101010111100110111100100100100011
   funkcija treba da vrati broj cija binarna reprezentacija
   izgleda:
   11000100100001111011001111010101
   *****/
13 unsigned mirror(unsigned x)
{
15     unsigned najnizi_bit;
    unsigned rezultat = 0;

17     int i;
19     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
       vrednosti broja x se određuje i pamti u promenljivoj
       najnizi_bit, nakon čega se na promenljivu x primeni siftovanje u
       desno. */
23     for (i = 0; i < sizeof(x) * 8; i++) {
        najnizi_bit = x & 1;
25         x >>= 1;
        /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
        postavljeni bitovi dobijaju veću poziciju. Novi bit se
        postavlja na najnižu poziciju */
27         rezultat <<= 1;
        rezultat |= najnizi_bit;
31     }
    return rezultat;
33 }

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

```
45     putchar('\n');
46 }
47
48 int main()
49 {
50     int broj;
51     scanf("%x", &broj);
52
53     /* Ispisuje se binarna reprezentaciju unetog broja */
54     print_bits(broj);
55
56     /* Ispisuje se binarna reprezentaciju broja dobijenog pozivom
57        funkcije mirror */
58     print_bits(mirror(broj));
59
60     return 0;
61 }
```

Rešenje 1.11

```
1  #include <stdio.h>
2
3  /* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
4     jedinica veci od broja nula. U suprotnom funkcija vraca 0 */
5  int Broj01(unsigned int n)
6  {
7
8     int broj_nula, broj_jedinica;
9     unsigned int maska;
10
11     broj_nula = 0;
12     broj_jedinica = 0;
13
14     /* Maska je inicijalizovana tako da moze da analizira bit najvece
15        tezine */
16     maska = 1 << (sizeof(unsigned int) * 4 - 1);
17
18     /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
19        iteraciji pomera u desno pa ce jedini bit koji je postavljen na
20        1 biti na svim pozicijama u binarnoj reprezentaciji maske */
21     while (maska != 0) {
22
23         /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
24            1 i uveca se odgovarajuci brojac */
25         if (n & maska) {
26             broj_jedinica++;
27         } else {
28             broj_nula++;
29         }
30     }
```

1 Uvodni zadaci

```
32     /* Pomera se maska u desnu stranu */
33     maska = maska >> 1;
34 }
35
36 /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
37    suprotnom vraca 0 */
38 return (broj_jedinica > broj_nula) ? 1 : 0;
39 }
40
41 int main()
42 {
43     unsigned int n;
44
45     scanf("%u", &n);
46
47     printf("%d\n", Broj01(n));
48
49     return 0;
50 }
```

Rešenje 1.12

```
#include <stdio.h>
2
3 int broj_parova(unsigned int x)
4 {
5
6     int broj_parova;
7     unsigned int maska;
8
9     /* Vrednost promenljive koja predstavlja broj parova se
10        inicijalizuje na 0 */
11     broj_parova = 0;
12
13     /* Postavlja se maska tako da moze da procitamo da li su dva
14        najmanja bita u zapisu broja x 11 */
15     /* Binarna reprezentacija broja 3 je 000...00011 */
16     maska = 3;
17
18     while (x != 0) {
19
20         /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
21            */
22         if ((x & maska) == maska) {
23             broj_parova++;
24         }
25
26         /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
27            proveravao sledeci par bitova. Pomeranjem u desno bit najvece
28            tezine se popunjava nulom jer je x neoznaceni broj. */
29     }
```

```

28     x = x >> 1;
    }
30     return broj_parova;
32 }
34 int main()
35 {
36     unsigned int x;
38     scanf("%u", &x);
40     printf("%d\n", broj_parova(x));
42     return 0;
44 }

```

Rešenje 1.14

```

#include <stdio.h>
2
/*
4     Niska koja se formiramo je duzine (sizeof(unsigned int)*8)/4 +1
    jer su za svaku heksadekadnu cifru potrebne 4 binarne cifre i
6     jedna dodatna pozicija za terminirajucu nulu.
8     Prethodni izraz je identican sa sizeof(unsigned int)*2+1. */
10 #define MAX_DUZINA sizeof(unsigned int)*2 +1
12
13 void prevod(unsigned int x, char s[])
14 {
15     int i;
16     unsigned int maska;
17     int vrednost;
18
19     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
20     maska za citanje 4 uzastopne cifre */
21     maska = 15;
22
23     /******
24     Broj se posmatra od pozicije najmanje tezine ka poziciji
25     najvece tezine. Na primer za broj
26     00000000001101000100001111010101
27     u prvom koraku se citaju bitovi izdvojeni sa <...>:
28     0000000000110100010000111101<0101>
29     u drugom koraku:
30     000000000011010001000011<1101>0101

```

1 Uvodni zadaci

```
32     u trecem koraku:
33     00000000001101000100<0011>11010101 i tako redom
34
35     Indeks i oznacava poziciju na koju se smesta vrednost.
36
37     */
38     for (i = MAX_DUZINA - 2; i >= 0; i--) {
39         /* Vrednost izdvojene cifre */
40         vrednost = x & maska;
41
42         /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
43            dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
44            10 do 15 odgovarajuci karakter se dobija tako sto se prvo
45            oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
46            dobijenu vrednost doda ASCII kod 'A' (time se dobija
47            odgovarajuće slovo 'A', 'B', ... 'F') */
48         if (vrednost < 10) {
49             s[i] = vrednost + '0';
50         } else {
51             s[i] = vrednost - 10 + 'A';
52         }
53
54         /* Primenljiva x se pomera za 4 bita u desnu stranu i time ce u
55            narednoj iteraciji biti posmatrane sledeće 4 cifre */
56         x = x >> 4;
57     }
58
59     s[MAX_DUZINA - 1] = '\\0';
60 }
61
62 int main()
63 {
64     unsigned int x;
65     char s[MAX_DUZINA];
66
67     scanf("%u", &x);
68
69     prevod(x, s);
70
71     printf("%s\\n", s);
72
73     return 0;
74 }
```

Rešenje 1.17

```
1 #include <stdio.h>
2
3
4 /******
```



```

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

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

16      /*
18         Celo telo funkcije se moze ovako kratko zapisati
19         return k == 0 ? 1 : x * stepen(x,k-1);
20
21         Druga verzija prethodne funkcije. Obratiti paznju na
22         efikasnost u odnosu na prvu verziju!
23         Logaritamsko resenje je zasnovano na cinjenicama:
24         x^0 =1;
25         x^k = x * (x^2 )^(k/2) , za neparno k
26         x^k = (x^2)^(k/2) , za parno k
27         Ovom resenju ce biti potrebno manje rekursivnih poziva da bi
28         doslo do rezultata, i stoga je efikasnije.
29         *****/
30      int stepen2(int x, int k)
        {
32         // printf("Racunam stepen2 (%d, %d)\n",x,k);
34         if (k == 0)
            return 1;

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

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

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

52         printf("%d\n", stepen(x, k));
53         // printf("\n-----\n");
54         // printf("%d\n",stepen2(2,10));
56         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 deklarirana i u telu funkcije
12 paran. */

14 unsigned neparan(unsigned n);

16 /* Funkcija vraca 1 ako broj n ima paran broj cifara inace vraca 0.
   */
   unsigned paran(unsigned n)
18 {
20     if (n <= 9)
22         return 0;
   else
24         return neparan(n / 10);
26 }

28 /* Funkcija vraca 1 ako broj n ima neparan broj cifara inace vraca
   0. */
   unsigned neparan(unsigned n)
30 {
32     if (n <= 9)
34         return 1;
   else
36         return paran(n / 10);
38 }

40 /* Glavna funkcija za testiranje */
   int main()
42 {
44     int n;
   scanf("%d", &n);
   printf("Uneti broj ima %sparan broj cifara\n",
46         (paran(n) == 1 ? "" : "ne"));
   return 0;
}
```

Rešenje 1.19

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

   return faktoriijelRepna(n - 1, n * result);
11 }

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

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

23 int faktoriijelRepna_v1(int n, int result)
   {
25   pocetak:
       if (n == 0)
27         return result;

       result = n * result;
       n = n - 1;
       goto pocetak;
31   }

33 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
35 praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
   iterativno resenje bez bezuslovnih skokova: */
37 int faktoriijelRepna_v2(int n, int result)
   {
39   while (n != 0) {
       result = n * result;
41     n = n - 1;
   }

43   return result;
45 }

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

1 Uvodni zadaci

```
51      zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
53      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
55      poziv faktorijelRepna sa pozivom faktorijelRepna_v1, a zatim sa
      pozivom funkcije faktorijelRepna_v2. */
57  int faktorijel(int n)
  {
59      return faktorijelRepna(n, 1);
  }
61
  /* Test program */
63  int main()
  {
65      int n;

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

71      return 0;
  }
```

Rešenje 1.21

```
#include <stdio.h>

2
int zbir_cifara(unsigned int x)
4  {
    /* Izlazak iz rekurzije: ako je broj jednocifren */
6    if (x < 10)
        return x;
8
    /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
10     poslednje cifre + poslednja cifra tog broja */
    return zbir_cifara(x / 10) + x % 10;
12 }

14 int main()
  {
16     unsigned int x;

18     /* Ucitava se ceo broj sa ulaza */
    scanf("%u", &x);
20
    /* Ispisuje se zbir cifara ucitanog broja */
22    printf("%d\n", zbir_cifara(x));

24    return 0;
  }
```

Rešenje 1.22

```

#include <stdio.h>
#define MAX_DIM 1000

/*
Ako je n==0, onda je suma(a,0) = 0 Ako je n>0, onda je suma(a,n) =
a[n-1] + suma(a,n-1) Suma celog niza je jednaka sumi prvih n-1
elementa uvećenoj za poslednji element celog niza. */
int sumaNiza(int *a, int n)
{
/* Nije postavljena stroga jednakost n==0, za slucaj da korisnik
prilikom prvog poziva, posalje negativan broj za velicinu niza.
*/
if (n <= 0)
return 0;

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

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

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

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

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

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

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

return 0;
}

```

Rešenje 1.23

```
1  #include <stdio.h>
2  #define MAX_DIM 256
3
4  /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
   * dimenzije n */
5
6  int maksimum_niza(int niz[], int n)
7  {
8      /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
   * ujedno i jedini element niza */
9
10     if (n == 1)
11         return niz[0];
12
13     /* Resavanje problema manje dimenzije */
14     int max = maksimum_niza(niz, n - 1);
15
16     /* Na osnovu poznatog resenja problema dimenzije n-1, resava se
   * problem dimenzije n */
17     return niz[n - 1] > max ? niz[n - 1] : max;
18 }
19
20 int main()
21 {
22     int brojevi[MAX_DIM];
23     int n;
24
25     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
   * Promenljiva i predstavlja indeks tekućeg broja. */
26     int i = 0;
27     while (scanf("%d", &brojevi[i]) != EOF) {
28         i++;
29     }
30     n = i;
31
32     /* Stampa se maksimum unetog niza brojeva */
33     printf("%d\n", maksimum_niza(brojevi, n));
34     return 0;
35 }
```

Rešenje 1.24

```
1  #include <stdio.h>
2  #define MAX_DIM 256
3
4  int skalarno(int a[], int b[], int n)
5  {
```

```

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

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

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

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

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

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

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

32   return 0;
34 }

```

Rešenje 1.25

```

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

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

10     int bp = br_pojave(x, a, n - 1);
     return a[n - 1] == x ? 1 + bp : bp;
12 }

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

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

```

1 Uvodni zadaci

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

Rešenje 1.26

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

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

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

16 int main()
{
18     int x, y, z, a[MAX_DIM];
    int n;

20
    /* Učitavaju se tri cela broja za koje se ispituje da li su
22     uzastopni članovi niza */
    scanf("%d%d%d", &x, &y, &z);

24
    int i = 0;
    while (scanf("%d", &a[i]) != EOF) {
26        i++;
    }
    n = i;
28

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

36    return 0;
```



```
}

```

Rešenje 1.27

```

1  #include <stdio.h>
2
3  /*****
4      Funkcija koja broji bitove svog argumenta
5
6      ako je x ==0, onda je count(x) = 0
7      inace count(x) = najvisi_bit +count(x<<1)
8
9      Za svaki naredni rekurzivan poziv prosledjuje se x<<1. Kako se
10     siftovanjem sa desne strane uvek dopisuju 0, argument x ce u
11     nekom rekurzivnom pozivu biti bas 0 i izacicemo iz rekurzije.
12 *****/
13 int count(int x)
14 {
15     /* Izlaz iz rekurzije */
16     if (x == 0)
17         return 0;
18
19     /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
20     postavljen na 1. Koriscenjem odgovarajuce maske proverava se
21     vrednost najviseg bita. Rezultat koliko ima jedinica u ostatku
22     binarnog zapisa broja x se uvecava za 1. Najvisi bit je 0. Stoga
23     je broj jedinica u zapisu x isti kao broj jedinica u zapisu
24     broja x<<1, jer se siftovanjem u levo sa desne strane dopisuju 0.
25     Za rekurzivni poziv se salje vrednost koja se dobija kada se x
26     siftuje u levo. Napomena: argument funkcije x je oznacen ceo
27     broj, usled cega se ne koristi siftovanje udesno, jer funkciji
28     moze biti prosleden i negativan broj. Iz tog razloga, odlucujemo
29     se da proveramo najvisi, umesto najnizeg bita */
30     if (x & (1 << (sizeof(x) * 8 - 1)))
31         return 1 + count(x << 1);
32     else
33         return count(x << 1);
34 }
35
36 /*****
37     Telo prethodne funkcije je moglo biti zapisano i krace:
38     jednolinijska return naredba sa proverom i rekurzivnim pozivom
39     return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + count(x<<1);
40 *****/
41
42
43
44 int main()
45 {
46     int x;
47     scanf("%x", &x);

```

1 Uvodni zadaci

```
48     printf("%d\n", count(x));
50     return 0;
}
```

Rešenje 1.29

```
#include<stdio.h>

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

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

Rešenje 1.30

```
#include<stdio.h>

2
/* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
4   */
int max_heksadekadna_cifra(unsigned x)
6 {
    /* Izlazak iz rekurzije */
8     if (x == 0)
        return 0;
10    /* Odredjivanje poslednje heksadekadne cifre u broju */
    int poslednja_cifra = x & 15;
12    /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
```

```

14     njega izbrise poslednja heksadekadna cifra */
15     int max_bez_poslednje_cifre = max_heksadekadna_cifra(x >> 4);
16     return poslednja_cifra >
17         max_bez_poslednje_cifre ? poslednja_cifra :
18         max_bez_poslednje_cifre;
19 }
20
21 int main()
22 {
23     unsigned x;
24     scanf("%u", &x);
25     printf("%d\n", max_heksadekadna_cifra(x));
26     return 0;
27 }

```

Rešenje 1.31

```

1  #include<stdio.h>
2  #include<string.h>
3  /* Niska moze imati najvise 32 karaktera + 1 za terminalnu nulu */
4  #define MAX_DIM 33
5
6  int palindrom(char s[], int n)
7  {
8      if ((n == 1) || (n == 0))
9          return 1;
10     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
11 }
12
13 int main()
14 {
15     char s[MAX_DIM];
16     int n;
17
18     scanf("%s", s);
19
20     /* Odredjuje se duzina niske */
21     n = strlen(s);
22
23     /* Ispisuje se poruka da li je niska palindrom ili nije */
24     if (palindrom(s, n))
25         printf("da\n");
26     else
27         printf("ne\n");
28
29     return 0;
30 }

```

Rešenje 1.32

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

4 void ispisiNiz(int a[], int n)
6 {
    int i;

8     for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
    printf("\n");
12 }

14 /* Funkcija proverava da li se x vec nalazi u permutaciji na
    prethodnih 1...n mesta */
16 int koriscen(int a[], int n, int x)
18 {
    int i;
    for (i = 1; i <= n; i++)
20         if (a[i] == x)
                return 1;

22     return 0;
24 }

26 /* F-ja koja ispisuje sve permutacije od skupa {1,2,...,n} a[] je niz
    u koji smesta permutacije m - oznacava da se na m-tu poziciju u
28 permutaciji smesta jedan od preostalih celih brojeva n- je
    velicina skupa koji se permutuje Funkciju se poziva sa argumentom
30 m=1 jer formiranje permutacije pocinje od 1. pozicije. Stoga, nece
    se koristiti a[0]. */
32 void permutacija(int a[], int m, int n)
34 {
    int i;

36     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
        premasila velicinu skupa, onda se svi brojevi vec nalaze u
38 permutaciji i ispisuje se permutacija. */
    if (m > n) {
40         ispisiNiz(a, n);
        return;
42     }

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

```

52     ali sada sa drugacije postavljenim prefiksom. */
54
55     for (i = 1; i <= n; i++) {
56         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
57            pozicije, onda se on postavlja na poziciju m i poziva se
58            funkcija da napravi permutaciju za jedan vece duzine, tj. m+1.
59            Inace, nastavlja se dalje, trazeci broj koji se nije pojavio
60            do sada u permutaciji. */
61         if (!koriscen(a, m - 1, i)) {
62             a[m] = i;
63             /* Poziva se ponovo funkcija da dopuni ostatak permutacije
64                posle upisivanja i na poziciju m. */
65             permutacija(a, m + 1, n);
66         }
67     }
68 }
69
70 int main(void)
71 {
72     int n;
73     int a[MAX_DUZINA_NIZA];
74
75     printf("Unesite duzinu permutacije: ");
76     scanf("%d", &n);
77     if (n < 0 || n >= MAX_DUZINA_NIZA) {
78         fprintf(stderr,
79             "Duzina permutacije mora biti broj veci od 0 i manji od %
80             d!\n",
81             MAX_DUZINA_NIZA);
82         exit(EXIT_FAILURE);
83     }
84
85     permutacija(a, 1, n);
86
87     exit(EXIT_SUCCESS);
88 }

```

Rešenje 1.33

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Rekurzivna funkcija za racunanje binomnog koeficijenta. */
5  /* ako je k=0 ili k=n, onda je binomni koeficijent 0 ako je k izmedju
6     0 i n, onda je bk(n,k) = bk(n-1,k-1) + bk(n-1,k) */
7  int binomniKoeficijent(int n, int k)
8  {
9      return (0 < k
10             && k < n) ? binomniKoeficijent(n - 1,
11             k - 1) +

```

```
        binomniKoeficijent(n - 1, k) : 1;
13 }

15 /*****
    Iterativno izracunavanje datog binomnog koeficijenta.
17
    int binomniKoeficijent (int n, int k) {
19         int i, j, b;
        for (b=i=1, j=n; i<=k; b =b * j-- / i++)
21             ;
        return b;
23     }
    *****/

25 /* Prostim opaZanjem se uocava da se svaki element n-te hipotenuze
27     (osim ivicnih 1) dobija kao zbir 2 elementa iz n-1 hipotenuze. Uz
    pomenute dve nove ivicne jedinice lako se zakljucuje da ce suma
29     elementa n-te hipotenuze biti tacno 2 puta veća. */
    int sumaElemenataHipotenuze(int n)
31 {
        return n > 0 ? 2 * sumaElemenataHipotenuze(n - 1) : 1;
33 }

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

39
        scanf("%d %d", &d, &r);

41
        /* Ispisivanje Paskalovog trougla */
43         putchar('\n');
        for (n = 0; n <= d; n++) {
45             for (i = 0; i < d - n; i++)
                printf(" ");
47             for (k = 0; k <= n; k++)
                printf("%4d", binomniKoeficijent(n, k));
49             putchar('\n');
        }

51
        if (r < 0) {
53             fprintf(stderr,
                "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
            );
            exit(EXIT_FAILURE);
55         }
        printf("%d\n", sumaElemenataHipotenuze(r));
57
        exit(EXIT_SUCCESS);
59     }
```

Glava 2

Pokazivači

2.1 Pokazivačka aritmetika

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

- (a) korišćenjem indeksne sintakse,
- (b) korišćenjem pokazivačke sintakse.

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

Primer 1

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

Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

Zadatak 2.2 Dat je niz realnih brojeva dimenzije n .

- (a) Napisati funkciju **zbir** koja izračunava zbir elemenata niza.

- (b) Napisati funkciju `proizvod` koja izračunava proizvod elemenata niza.
- (c) Napisati funkciju `min_element` koja izračunava najmanji element niza.
- (d) Napisati funkciju `max_element` koja izračunava najveći element 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 učitano niza.

Primer 1

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

Primer 2

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

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

Primer 1

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

Primer 2

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

Primer 3

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 0
Greska: neodgovarajuca dimenzija niza.
```

Primer 4

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 101
Greska: neodgovarajuca dimenzija niza.
```

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

- (a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

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

Primer 1

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

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

Primer 2

```
POZIV: ./a.out

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

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

Primer 1

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

INTERAKCIJA PROGRAMA:
  Broj argumenata komandne linije
  koji su palindromi je 4.
```

Primer 2

```
POZIV: ./a.out

INTERAKCIJA PROGRAMA:
  Broj argumenata komandne linije koji
  koji su palindromi je 0.
```

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

Primer 1

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

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

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

Primer 2

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

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

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

Primer 3

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

DATOTEKA ULAZ.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

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

Primer 1

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

ULAZ.TXT
Ovo je sadržaj datoteke i u njoj ima reci
koje se završavaju na ke

INTERAKCIJA PROGRAMA:
Broj reci koje se završavaju na ke je 2.
```

Primer 2

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

ULAZ.TXT
Ovo je sadržaj datoteke i u njoj ima reci
koje počinju sa sa

INTERAKCIJA PROGRAMA:
Broj reci koje počinju na sa je 3.
```

Primer 3

```

Poziv: ./a.out ulaz.txt sa -p
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA PROGRAMA:
  Greska: Neuspesno otvaranje
  datoteke ulaz.txt.

```

Primer 4

```

Poziv: ./a.out ulaz.txt
ULAZ.TXT
  Ovo je sadrzaj ulaza.
INTERAKCIJA PROGRAMA:
  Greska: Nedovoljan broj argumenata
  komandne linije.
  Program se poziva sa
  ./a.out ime_dat suf/pref -s/-p.

```

2.2 Višedimenzioni nizovi

Zadatak 2.8 Data je kvadratna matrica dimenzije n .

- Napisati funkciju koja izračunava trag matrice (sumu elemenata na glavnoj dijagonali).
- Napisati funkciju koja izračunava euklidsku normu matrice (koren sume kvadrata svih elemenata).
- Napisati funkciju koja izračunava gornju vandijagonalnu normu matrice (sumu apsolutnih vrednosti elemenata iznad glavne dijagonale).

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

Primer 1

```

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

```

Primer 2

```

INTERAKCIJA PROGRAMA:
  Unesite dimenziju matrice: 0
  Greska: neodgovarajuca dimenzija matrice.

```

Zadatak 2.9 Date su dve kvadratne matrice istih dimenzija n .

- Napisati funkciju koja proverava da li su matrice jednake.
- Napisati funkciju koja izračunava zbir matrica.

- (c) Napisati funkciju koja izračunava proizvod matrica.

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

Primer 1

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

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

- (a) Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- (b) Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- (c) Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.
- (d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).
- (e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).

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

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

Primer 1

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

ULAZ.TXT
4
1 0 0 0
0 1 1 0
0 0 1 0
0 0 0 0

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

Zadatak 2.11 Data je kvadratna matrica dimenzije n .

- Napisati funkciju koja određuje najveći element matrice na sporednoj dijagonali.
- Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.
- Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.
- Napisati funkciju koja određuje broj negativnih elemenata matrice.

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

Primer 1

```
Poziv: ./a.out 3

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

Primer 2

```
Poziv: ./a.out 4

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

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

Primer 1

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

Primer 2

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

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

- (a) Napsiati funkciju koja učitava elemente matrice sa standardnog ulaza
- (b) Napsiati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice.

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

Primer 1

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

Primer 2

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

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

Primer 1

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

2.3 Dinamička alokacija memorije

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

Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite dimenziju niza: 3
Unesite elemente niza:
1 -2 3
Niz u obrnutom poretku je: 3 -2 1
```

Primer 2

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

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

- (a) realokaciju memorije niza vršiti korišćenjem `malloc()` funkcije,
- (b) realokaciju memorije niza vršiti korišćenjem `realloc()` funkcije.

Primer 1

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

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

Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite dve niske karaktera:
Jedan Dva
Nadovezane niske: JedanDva
```

Zadatak 2.18 Napisati program koji sa standardnog ulaza učitava matricu realnih brojeva. Prvo se učitavaju dimenzije matrice n i m (ne praviti nikakve

pretpostavke o njihovoj veličini), a zatim i elementi matrice. Na standardni izlaz ispisati trag matrice.

Primer 1

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

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

- (a) Napisati funkciju koja vrši učitavanje matrice sa standardnog ulaza.
- (b) Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu).

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

Primer 1

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

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

Primer 1

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

Primer 2

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

Zadatak 2.21 Data je realna kvadratna matrica dimenzije n .

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

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

Primer 1

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

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

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

Zadatak 2.22 Napisati program koji na osnovu dve realne matrice dimenzija $m \times n$ formira matricu dimenzije $2 \cdot m \times n$ tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci „matrice.txt“. U prvom redu se nalaze dimenzije matrica m i n , u narednih m redova se nalaze vrste prve matrice, a u narednih m redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz.

Primer 1

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

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

```
INTERAKCIJA PROGRAMA:
```

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

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

Primer 1

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

Zadatak 2.24 Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci „slicice.txt“ se nalaze informacije o sličicama koje mu nedostaju u formatu:

`redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`

Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronađe ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. NAPOMENA: Za realokaciju memorije koristiti *realloc()* funkciju.

Primer 1

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

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

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

Primer 1

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

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

2.4 Pokazivači na funkcije

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

Primer 1

```
Poziv: ./a.out sin

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

Primer 2

```
Poziv: ./a.out cos

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

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

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

Primer 1

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

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

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

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

Primer 1

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

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

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

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

Primer 1

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

2.5 Rešenja

Rešenje 2.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija obrće elemente niza koriscenjem indekse sintakse */
7 void obrni_niz_v1(int a[], int n)
8 {
9     int i, j;
10
11     for (i = 0, j = n - 1; i < j; i++, j--) {
12         int t = a[i];
13         a[i] = a[j];
14         a[j] = t;
15     }
16 }
17
18 /* Funkcija obrće elemente niza koriscenjem pokazivacke sintakse */
19 void obrni_niz_v2(int *a, int n)
20 {
21     /* Pokazivaci na elemente niza */
22     int *prvi, *poslednji;
23
24     /* Vrsi se obrtanje niza */
```

```

25  for (prvi = a, poslednji = a + n - 1; prvi < poslednji;) {
26      int t = *prvi;
27
28      /* Na adresu na koju pokazuje pokazivac "prvi" postavlja se
29       vrednost koja se nalazi na adresi na koju pokazuje pokazivac
30       "poslednji". Nakon toga se pokazivac "prvi" uvecava za jedan
31       sto za posledicu ima da "prvi" pokazuje na sledeci element u
32       nizu */
33      *prvi++ = *poslednji;
34
35      /* Vrednost promenljive "t" se postavlja na adresu na koju
36       pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
37       umanjuje za jedan, sto za posledicu ima da pokazivac
38       "poslednji" sada pokazuje na element koji mu prethodi u nizu
39       */
40      *poslednji-- = t;
41  }
42
43  /******
44   Drugi nacin za obrtanje niza
45   *****/
46
47  for (prvi = a, poslednji = a + n - 1; prvi < poslednji;
48       prvi++, poslednji--) {
49      int t = *prvi;
50      *prvi = *poslednji;
51      *poslednji = t;
52  }
53  /******
54
55  int main()
56  {
57      /* Deklarise se niz od najvise MAX elemenata */
58      int a[MAX];
59
60      /* Broj elemenata niza a */
61      int n;
62
63      /* Pokazivac na elemente niza */
64      int *p;
65
66      printf("Unesite dimenziju niza: ");
67      scanf("%d", &n);
68
69      /* Proverava se da li je doslo do prekoracenja ogranicenja
70       dimenzije */
71      if (n <= 0 || n > MAX) {
72          fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
73          exit(EXIT_FAILURE);
74      }

```

```
75 printf("Unesite elemente niza:\n");
76 for (p = a; p - a < n; p++)
77     scanf("%d", p);
78
79 obrni_niz_v1(a, n);
80
81 printf("Nakon obrtanja elemenata, niz je:\n");
82
83 for (p = a; p - a < n; p++)
84     printf("%d ", *p);
85 printf("\n");
86
87 obrni_niz_v2(a, n);
88
89 printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
90
91 for (p = a; p - a < n; p++)
92     printf("%d ", *p);
93 printf("\n");
94
95 return 0;
}
```

Rešenje 2.2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija izracunava zbir elemenata niza */
7 double zbir(double *a, int n)
8 {
9     double s = 0;
10    int i;
11
12    for (i = 0; i < n; i++) s += *(a + i);
13
14    return s;
15 }
16
17 /* Funkcija izracunava proizvod elemenata niza */
18 double proizvod(double *a, int n)
19 {
20    double p = 1;
21
22    for (; n; n--)
23        p *= *(a + n - 1);
24
25    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
       minimalni */
36     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--)
54         if (*a > max)
            max = *a;

56     return max;
58 }

60 int main()
61 {
62     double a[MAX];
63     int n, i;
64
65     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) {
71         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72         exit(EXIT_FAILURE);
73     }

74     printf("Unesite elemente niza:\n");
75     for (i = 0; i < n; i++)
76         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
return 0;
86 }
```

Rešenje 2.3

```
1 #include <stdio.h>
#include <stdlib.h>
3 #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
7 ima neparan broj elemenata, srednji element ostaje nepromenjen */
void povecaj_smanji(int *a, int n)
9 {
    int *prvi = a;
11    int *poslednji = a + n - 1;

13    while (prvi < poslednji) {

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

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

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

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

23    }

25    /* Drugi nacin */
    while (prvi < poslednji) {
        (*prvi++)++;
        (*poslednji--)--;
31    }
33 }

35 int main()
37 {
    int a[MAX];
39    int n;
```

```

41     int *p;

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

44     /* Proverava se da li je doslo do prekoračenja ograničenja
45        dimenzije */
46     if (n <= 0 || n > MAX) {
47         fprintf(stderr, "Greska: neodgovarajuća dimenzija niza.\n");
48         exit(EXIT_FAILURE);
49     }

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

53     povecaj_smanji(a, n);

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

58     return 0;
59 }

```

Rešenje 2.4

```

#include <stdio.h>

2 int main(int argc, char *argv[])
3 {
4     int i;
5     char tip_ispisa;

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

8     printf("Kako zelite da ispisete argumente, ");
9     printf("koriscenjem indeksne ili pokazivacke sintakse (I ili P)? ");
10    ;
11    scanf("%c", &tip_ispisa);

12    printf("Argumenti komandne linije su:\n");
13    if (tip_ispisa == 'I') {
14        /* Ispisuju se argumenti komandne linije koriscenjem indeksne
15           sintakse */
16        for (i = 0; i < argc; i++)
17            printf("%d %s\n", i, argv[i]);
18    } else if (tip_ispisa == 'P') {
19        /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
20           sintakse */
21        for (i = 0; i < argc; i++)
22            printf("%s\n", argv[i]);
23    }
24 }

```

```

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

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

36     printf("Pocetna slova argumenata komandne linije su:\n");
38     if (tip_ispisa == 'I') {
        /* koristeci indeksnu sintaksu */
40         for (i = 0; i < argc; i++)
            printf("%c ", argv[i][0]);
42         printf("\n");
    } else if (tip_ispisa == 'P') {
44         /* koristeci pokazivacku sintaksu */
        for (i = 0; i < argc; i++)
46             printf("%c ", **argv++);
        printf("\n");
48    }

50    return 0;
}
```

Rešenje 2.5

```

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

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

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

```

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

23
    printf
25      ("Broj argumenata komandne linije koji su palindromi je %d.\n",
        n);
27  return 0;
}

```

Rešenje 2.6

```

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

4  #define MAX_KARAKTERA 100

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

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

20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
        */
    if (argc < 3) {
22         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
24         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
            argv[0]);
        exit(EXIT_FAILURE);
26     }

28
    /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
        komandne linije. */
30     in = fopen(*(argv + 1), "r");
    if (in == NULL) {
32         fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
34         exit(EXIT_FAILURE);
    }
36 }

```

2 Pokazivači

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

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

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

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

Rešenje 2.7

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

4 #define MAX_KARAKTERA 100

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

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

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

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

```

{
36     if (duzina_niske(sufiks) <= duzina_niske(niska) &&
        poredjenje_niski(niska + duzina_niske(niska) -
38                     duzina_niske(sufiks), sufiks) == 0)
        return 1;
40     return 0;
}

42 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
43     prefiks niske zadate prvi argumentom funkcije */
44 int prefiks_niske(char *niska, char *prefiks)
45 {
    int i;
48     if (duzina_niske(prefiks) <= duzina_niske(niska)) {
        for (i = 0; i < duzina_niske(prefiks); i++)
50             if (*(prefiks + i) != *(niska + i))
                    return 0;
52         return 1;
    } else
54         return 0;
}

56 int main(int argc, char **argv)
57 {
    /* Ukoliko korisnik nije uneo trazene argumente, prijavljuje se
60     greska */
    if (argc < 4) {
62         printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
64         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
            argv[0]);
66         exit(EXIT_FAILURE);
    }

68     FILE *in;
    int br = 0;
    char rec[MAX_KARAKTERA];

72     in = fopen(*(argv + 1), "r");
    if (in == NULL) {
74         fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
76         exit(EXIT_FAILURE);
    }

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

```

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

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

Rešenje 2.8

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 #define MAX 100
6
7 /* Deklaracija funkcija koje ce kasnije biti definisane */
8 double euklidska_norma(int M[][MAX], int n);
9 int trag(int M[][MAX], int n);
10 int gornja_vandijagonalna_norma(int M[][MAX], int n);
11
12 int main()
13 {
14     int A[MAX][MAX];
15     int i, j, n;
16
17     printf("Unesite dimenziju matrice: ");
18     scanf("%d", &n);
19
20     /* Provera prekoracenja dimenzije matrice */
21     if (n > MAX || n <= 0) {
22         fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
23         exit(EXIT_FAILURE);
24     }
25
26     printf("Unesite elemente matrice, vrstu po vrstu:\n ");
27     for (i = 0; i < n; i++)
28         for (j = 0; j < n; j++)
29             scanf("%d", &A[i][j]);
30
31     /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
32     for (i = 0; i < n; i++) {
33         /* Ispis elemenata i-te vrste */
34         for (j = 0; j < n; j++)
35             printf("%d ", A[i][j]);
36         printf("\n");
37     }
```



```

39  /*****
41  Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
43  Kod ovako definisane matrice, elementi su uzastopno smesteni u
45  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.

47  for( i = 0; i < n ; i++) {
    for ( j=0 ; j<n ; j++)
49      printf("%d ", *((A+i)+j));
    printf("\n");
51  }
    *****/

53

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

57

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

61

    return 0;
63 }

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

67 /* Funkcija izracunava trag matrice */
    int trag(int M[][MAX], int n)
69 {
    int trag = 0, i;
71    for (i = 0; i < n; i++)
        trag += M[i][i];
73    return trag;
    }

75

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

81

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

85

    return sqrt(norma);
87 }

```

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

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

    return norma;
99 }
101 }
```

Rešenje 2.9

```
1 #include <stdio.h>
#include <stdlib.h>
3
#define MAX 100
5
/* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
7 standardnog ulaza */
void ucitaj_matricu(int m[][MAX], int n)
9 {
    int i, j;

    for (i = 0; i < n; i++)
11         for (j = 0; j < n; j++)
13             scanf("%d", &m[i][j]);
15 }

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

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
25             printf("%d ", m[i][j]);
        printf("\n");
27     }
}

29
/* Funkcija proverava da li su zadate kvadratne matrice a i b
31 dimenzije n jednake */
int jednake_matrice(int a[][MAX], int b[][MAX], int n)
33 {
    int i, j;
35 }
```

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

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

45 /* Funkcija izracunava zbir dve kvadratne matice */
47 void saberi(int a[][MAX], int b[][MAX], int c[][MAX], int n)
{
49     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 dimenziju matrica:\n");
    scanf("%d", &n);
83

    /* Proverava se da li je doslo do prekoračenja 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

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

99

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

105

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

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

111

113    return 0;
}
```

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

16

18    return 1;
}

20

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

```
24     dopunjene jedinicama na glavnoj dijagonali */
25 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
26 {
27     int i, j;
28
29     /* Prepisuju se vrednosti elemenata pocetne matrice */
30     for (i = 0; i < n; i++)
31         for (j = 0; j < n; j++)
32             zatvorenje[i][j] = m[i][j];
33
34     /* Na glavnoj dijagonali se postavljaju jedinice */
35     for (i = 0; i < n; i++)
36         zatvorenje[i][i] = 1;
37 }
38
39 /* Funkcija proverava da li je relacija simetricna. Relacija je
40 simetricna ako za svaki par elemenata vazi: ako je element "i" u
41 relaciji sa elementom "j", onda je i element "j" u relaciji sa
42 elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
43 dijagonalu */
44 int simetricnost(int m[][MAX], int n)
45 {
46     int i, j;
47
48     /* Obilaze se elementi ispod glavne dijagonale matrice i uporeduju
49 se sa njima simetricnim elementima */
50     for (i = 0; i < n; i++)
51         for (j = 0; j < i; j++)
52             if (m[i][j] != m[j][i])
53                 return 0;
54
55     return 1;
56 }
57
58 /* Funkcija odredjuje simetricno zatvorenje zadate relacije. Ono je
59 odredjeno matricom koja sadrzi sve elemente polazne matrice
60 dopunjene tako da matrica postane simetricna u odnosu na glavnu
61 dijagonalu */
62 void sim_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
63 {
64     int i, j;
65
66     for (i = 0; i < n; i++)
67         for (j = 0; j < n; j++)
68             zatvorenje[i][j] = m[i][j];
69
70     for (i = 0; i < n; i++)
71         for (j = 0; j < n; j++)
72             if (zatvorenje[i][j] == 1)
73                 zatvorenje[j][i] = 1;
74 }
```

```
76  /* Funkcija proverava da li je relacija tranzitivna. Relacija je
77     tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
78     relaciji sa elementom "j" i element "j" u relaciji sa elementom
79     "k", onda je i element "i" u relaciji sa elementom "k" */
80  int tranzitivnost(int m[][MAX], int n)
81  {
82      int i, j, k;
83
84      for (i = 0; i < n; i++)
85          for (j = 0; j < n; j++)
86              /* Ispituje se da li postoji element koji narusava *
87                 tranzitivnost */
88                  for (k = 0; k < n; k++)
89                      if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
90                          return 0;
91
92      return 1;
93  }
94
95  /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje zadate
96     relacije koriscenjem Varsalovog algoritma */
97  void ref_tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
98  {
99      int i, j, k;
100
101      /* Prepisuju se vrednosti elemenata pocetne matrice */
102      for (i = 0; i < n; i++)
103          for (j = 0; j < n; j++)
104              zatvorenje[i][j] = m[i][j];
105
106      /* Odredjuje se reflektivno zatvorenje matrice */
107      for (i = 0; i < n; i++)
108          zatvorenje[i][i] = 1;
109
110      /* Primenom Varsalovog algoritma odredjuje se tranzitivno
111         zatvorenje matrice */
112      for (k = 0; k < n; k++)
113          for (i = 0; i < n; i++)
114              for (j = 0; j < n; j++)
115                  if ((zatvorenje[i][k] == 1) && (zatvorenje[k][j] == 1)
116                      && (zatvorenje[i][j] == 0))
117                      zatvorenje[i][j] = 1;
118  }
119
120  /* Funkcija ispisuje elemente matrice */
121  void pisi_matricu(int m[][MAX], int n)
122  {
123      int i, j;
124
125      for (i = 0; i < n; i++) {
```

```
128     for (j = 0; j < n; j++)
        printf("%d ", m[i][j]);
        printf("\n");
130     }
    }
132
133 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

160     /* Proverava se da li je doslo do prekoracenja dimenzije */
    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

172     /* Ispisuje se rezultat */
    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 ? "jest" : "nije");
180
181     printf("Refleksivno zatvorenje relacije:\n");
182     ref_zatvorenje(m, n, pomocna);
183     pisi_matricu(pomocna, n);
184
185     printf("Simetricno zatvorenje relacije:\n");
186     sim_zatvorenje(m, n, pomocna);
187     pisi_matricu(pomocna, n);
188
189     printf("Refleksivno-tranzitivno zatvorenje relacije:\n");
190     ref_tran_zatvorenje(m, n, pomocna);
191     pisi_matricu(pomocna, n);
192
193     /* Zatvara se datoteka */
194     fclose(ulaz);
195
196     return 0;
197 }
```

Rešenje 2.11

```
#include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
5
6 /* Funkcija izracunava najveći element na sporednoj dijagonali. Za
   elemente sporedne dijagonale vazi da je zbir indeksa vrste i
   indeksa kolone jednak n-1 */
8 int max_sporedna_dijagonala(int m[][MAX], int n)
10 {
11     int i;
12     int max_na_sporednoj_dijagonali = m[0][n - 1];
13
14     for (i = 1; i < n; i++)
15         if (m[i][n - 1 - i] > max_na_sporednoj_dijagonali)
16             max_na_sporednoj_dijagonali = m[i][n - 1 - i];
17
18     return max_na_sporednoj_dijagonali;
19 }
20
21 /* Funkcija izracunava indeks kolone najmanjeg elementa */
22 int indeks_min(int m[][MAX], int n)
23 {
24     int i, j;
25     int min = m[0][0], indeks_kolone = 0;
26
27     for (i = 0; i < n; i++)
28         for (j = 0; j < n; j++)
```



```

    if (m[i][j] < min) {
30         min = m[i][j];
        indeks_kolone = j;
32     }

34     return indeks_kolone;
}

36
/* Funkcija izracunava indeks vrste najveceg elementa */
38 int indeks_max(int m[][MAX], int n)
{
40     int i, j;
    int max = m[0][0], indeks_vrste = 0;

42     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
44             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;
    int broj_negativnih = 0;

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

60     return broj_negativnih;
62 }

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 dim_matrice.\n", argv[0]);
        exit(EXIT_FAILURE);
78     }

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

```

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

94
    printf("Najveci element matrice na sporednoj dijagonali je %d.\n",
96        max_sporedna_dijagonala(m, n));

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

100
    printf("Indeks vrste koja sadrzi najveći element matrice %d.\n",
102        indeks_max(m, n));

104
    printf("Broj negativnih elemenata matrice je %d.\n",
        broj_negativnih(m, n));

106
    return 0;
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++) {
22         for (j = 0; j < n; j++)
23             printf("%d ", m[i][j]);
24         printf("\n");
25     }
26 }

27 /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
28    da li je normirana i ortogonalna. Matrica je normirana ako je
29    proizvod svake vrste matrice sa samom sobom jednak jedinici.
30    Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
31    vrste matrice jednak nuli */
32 int ortonormirana(int m[][MAX], int n)
33 {
34     int i, j, k;
35     int proizvod;

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

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

42         if (proizvod != 1)
43             return 0;
44     }

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

49             proizvod = 0;

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

53             if (proizvod != 0)
54                 return 0;
55         }
56     }

57     /* Ako su oba uslova ispunjena, matrica je ortonormirana */
58     return 1;
59 }

60 int main()
61 {
62     int A[MAX][MAX];
63     int n;
```

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

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

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

83
84 printf("Matrica %s ortonormirana.\n",
85        ortonormirana(A, n) ? "je" : "nije");
86 return 0;
87 }
```

Rešenje 2.13

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_V 10
5  #define MAX_K 10
6
7  /* Funkcija proverava da li su ispisani svi elementi iz matrice,
8     odnosno da li se narušio prirodan poredak medju granicama */
9  int krajIspisa(int top, int bottom, int left, int right)
10 {
11     return !(top <= bottom && left <= right);
12 }
13
14 /* Funkcija spiralno ispisuje elemente matrice */
15 void ispisi_matricu_spiralno(int a[][MAX_K], int n, int m)
16 {
17     int i, j, top, bottom, left, right;
18
19     top = left = 0;
20     bottom = n - 1;
21     right = m - 1;
22
23     while (!krajIspisa(top, bottom, left, right)) {
24
25         for (j = left; j <= right; j++)
26             printf("%d ", a[top][j]);
27
28         /* Spusta se prvi red */
29         top++;
30
31         if (krajIspisa(top, bottom, left, right))
```

```
        break;
33
    for (i = top; i <= bottom; i++)
35        printf("%d ", a[i][right]);

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

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

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

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

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

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

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

63 void ucitaj_matricu(int a[][MAX_K], int n, int m)
65 {
    int i, j;

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

71 }

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

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

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

```
        exit(EXIT_FAILURE);
85    }

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

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

93    return 0;
}
```

Rešenje 2.15

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

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

11
       /* Alocira se prostor za n celih brojeva */
13     if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
           fprintf(stderr, "malloc(): ");
15         fprintf(stderr, "greska pri alokaciji memorije.\n");
           exit(EXIT_FAILURE);
17     }

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

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

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

31     return 0;
}
```

Rešenje 2.16

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define KORAK 10
4
5  int main(void)
6  {
7      /* Adresa prvog alociranog bajta */
8      int *a = NULL;
9
10     /* Velicina alocirane memorije */
11     int alocirano;
12
13     /* Broj elemenata niza */
14     int n;
15
16     /* Broj koji se ucitava sa ulaza */
17     int x;
18     int i;
19     int *b = NULL;
20
21     /* Inicijalizacija */
22     alocirano = n = 0;
23
24     printf("Unesite brojeve, nulu za kraj:\n");
25     scanf("%d", &x);
26
27     while (x != 0) {
28         if (n == alocirano) {
29             alocirano = alocirano + KORAK;
30
31             /* Vrsi se realokacija memorije sa novom velicinom */
32             /* Resenje sa funkcijom malloc() */
33             b = (int *) malloc(alocirano * sizeof(int));
34
35             if (b == NULL) {
36                 fprintf(stderr, "malloc(): ");
37                 fprintf(stderr, "greska pri alokaciji memorije.\n");
38                 free(a);
39                 exit(EXIT_FAILURE);
40             }
41
42             /* Svih n elemenata koji pocinju na adresi a prepisujemo na
43              novu adresu b */
44             for (i = 0; i < n; i++)
45                 b[i] = a[i];
46
47             free(a);
48
49             /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
50              bloka koji je prilikom alokacije zapamcen u promenljivoj b
51              */
52         }
53     }
```

```
52     a = b;

54     /******
    Resenje sa funkcijom realloc()

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

58     a = (int*) realloc(a, alocirano * sizeof(int));
60     if(a == NULL) {
        fprintf(stderr, "realloc(): ");
62         fprintf(stderr, "greska pri alokaciji memorije.\n");
        exit(EXIT_FAILURE);
64     }
    /******/
66 }

68     a[n++] = x;

70     scanf("%d", &x);
72 }

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

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

80     exit(EXIT_SUCCESS);
82 }
```

Rešenje 2.17

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX 1000
6
7  /* Funkcija dinamicki kreira niz karaktera u koji smesta rezultat
8   nadovezivanja niski. Adresa niza se vraca kao povratna vrednost.
9   */
10 char *nadovezi(char *s, char *t)
11 {
12     char *p = (char *) malloc((strlen(s) + strlen(t) + 1)
13                               * sizeof(char));
14
15     /* Proverava se da li je memorija uspesno alocirana */
16     if (p == NULL) {
```



```

16     fprintf(stderr, "malloc(): ");
17     fprintf(stderr, "greska pri alokaciji memorije.\n");
18     exit(EXIT_FAILURE);
19 }
20
21 /* Kopiraju se i nadovezuju niske karaktera */
22 strcpy(p, s);
23 strcat(p, t);
24
25 return p;
26 }
27
28 int main()
29 {
30     char *s = NULL;
31     char s1[MAX], s2[MAX];
32
33     printf("Unesite dve niske karaktera:\n");
34     scanf("%s", s1);
35     scanf("%s", s2);
36
37     /* Poziva se funkcija koja nadovezuje niske */
38     s = nadovezi(s1, s2);
39
40     /* Prikazuje se rezultat */
41     printf("Nadovezane niske: %s\n", s);
42
43     /* Oslobadja se memorija alocirana u funkciji nadovezi() */
44     free(s);
45
46     return 0;
47 }

```

Rešenje 2.18

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  int main()
6  {
7      int i, j;
8
9      /* Pokazivac na dinamički alociran niz pokazivaca na vrste matrice
10       */
11     double **A = NULL;
12
13     /* Broj vrsta i broj kolona */
14     int n = 0, m = 0;
15
16     /* Trag matice */

```

```
double trag = 0;

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

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

23

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

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

35     /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
    potrebno je osloboditi svih i-1 prethodno alociranih vrsta, i
    alocirani niz pokazivaca */
37     if (A[i] == NULL) {
39         for (j = 0; j < i; j++)
41             free(A[j]);
        free(A);
        exit(EXIT_FAILURE);
43     }
}

45

47 printf("Unesite elemente matrice, vrstu po vrstu:\n");
for (i = 0; i < n; i++)
49     for (j = 0; j < m; j++)
        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

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

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

63

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

67 return 0;
```

```
}

```

Rešenje 2.19

```

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

4
5  /* Funkcija ucitava matricu sa ulaza */
6  void ucitaj_matricu(int **M, int n, int m)
7  {
8      int i, j;
9
10     for (i = 0; i < n; i++)
11         for (j = 0; j < m; j++)
12             scanf("%d", &M[i][j]);
13 }

14
15 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
16 {
17     int i, j;
18
19     for (i = 0; i < n; i++) {
20         for (j = 0; j <= i; j++)
21             printf("%d ", M[i][j]);
22         printf("\n");
23     }
24 }

25
26 int main()
27 {
28     int m, n, i, j;
29     int **matrica = NULL;

30
31     printf("Unesite broj vrsta i broj kolona matrice: ");
32     scanf("%d %d", &n, &m);
33
34     /* Alocira se prostor za niz pokazivaca na vrste matrice */
35     matrica = (int **) malloc(n * sizeof(int *));
36     if (matrica == NULL) {
37         fprintf(stderr, "malloc(): Neuspela alokacija\n");
38         exit(EXIT_FAILURE);
39     }

40
41     /* Alocira se prostor za svaku vrstu matrice */
42     for (i = 0; i < n; i++) {
43         matrica[i] = (int *) malloc(m * sizeof(int));
44
45         if (matrica[i] == NULL) {
46             fprintf(stderr, "malloc(): Neuspela alokacija\n");
47             for (j = 0; j < i; j++)

```

```
        free(matrica[j]);
49     free(matrica);
        exit(EXIT_FAILURE);
51     }
    }

53
    printf("Unesite elemente matrice, vrstu po vrstu:\n");
55     učitaj_matricu(matrica, n, m);

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

59
    /* Oslobadja se dinamički alocirana memorija za matricu. Prvo se
61     oslobadja memorija rezervisana za svaku vrstu */
    for (j = 0; j < n; j++)
63         free(matrica[j]);

65     /* Zatim se oslobadja memorija za niz pokazivaca na vrste matrice
        */
    free(matrica);

67     return 0;
69 }
```

Rešenje 2.21

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

4
/* Funkcija izvršava tražene transformacije nad matricom */
6 void izmeni(float **a, int n)
{
8     int i, j;

10     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
12         if (i < j)
            a[i][j] /= 2;
14         else if (i > j)
            a[i][j] *= 2;
16 }

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

```
26     for (i = 0; i < n; i++)
27         for (j = 0; j < n; j++)
28             if (i + j > n - 1)
29                 zbir += fabs(m[i][j]);
30
31     return zbir;
32 }
33
34 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
35    datoteke */
36 void ucitaj_matricu(FILE * ulaz, float **m, int n)
37 {
38     int i, j;
39
40     for (i = 0; i < n; i++)
41         for (j = 0; j < n; j++)
42             fscanf(ulaz, "%f", &m[i][j]);
43 }
44
45 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
46    standardni izlaz */
47 void ispisi_matricu(float **m, int n)
48 {
49     int i, j;
50
51     for (i = 0; i < n; i++) {
52         for (j = 0; j < n; j++)
53             printf("%.2f ", m[i][j]);
54         printf("\n");
55     }
56 }
57
58 /* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
59 float **alociraj_memoriju(int n)
60 {
61     int i, j;
62     float **m;
63
64     m = (float **) malloc(n * sizeof(float *));
65     if (m == NULL) {
66         fprintf(stderr, "malloc(): Neuspela alokacija\n");
67         exit(EXIT_FAILURE);
68     }
69
70     for (i = 0; i < n; i++) {
71         m[i] = (float *) malloc(n * sizeof(float));
72
73         if (m[i] == NULL) {
74             printf("malloc(): neuspela alokacija memorije!\n");
75             for (j = 0; j < i; j++)
76                 free(m[j]);
```

```
78     free(m);
79     exit(EXIT_FAILURE);
80 }
81 }
82 return m;
83 }
84
85 /* Funkcija oslobadja memoriju zauzetu kvadratnom matricom dimenzije
86    n */
87 void oslobodi_memoriju(float **m, int n)
88 {
89     int i;
90
91     for (i = 0; i < n; i++)
92         free(m[i]);
93     free(m);
94 }
95
96 int main(int argc, char *argv[])
97 {
98     FILE *ulaz;
99     float **a;
100     int n;
101
102     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
103        */
104     if (argc < 2) {
105         printf("Greska: ");
106         printf("Nedovoljan broj argumenata komandne linije.\n");
107         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
108         exit(EXIT_FAILURE);
109     }
110
111     /* Otvara se datoteka za citanje */
112     ulaz = fopen(argv[1], "r");
113     if (ulaz == NULL) {
114         fprintf(stderr, "Greska: ");
115         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
116         exit(EXIT_FAILURE);
117     }
118
119     /* Cita se dimenzija matrice */
120     fscanf(ulaz, "%d", &n);
121
122     /* Alocira se memorija */
123     a = alociraj_memoriju(n);
124
125     /* Ucitavaju se elementi matrice */
126     ucitaj_matricu(ulaz, a, n);
127
128     float zbir = zbir_ispod_sporodne_dijagonale(a, n);
```

```

130  /* Poziva se funkcija za transformaciju matrice */
    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

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

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

144

    return 0;
146 }

```

Rešenje 2.26

```

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 {
32     double a, b;
33     int n;
34
35     char ime_fje[6];
36
37     /* Pokazivac na funkciju koja ima jedan argument tipa double i
38        povratnu vrednost istog tipa */
39     double (*fp) (double);
40
41     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
42        */
43     if (argc < 2) {
44         printf("Greska: ");
45         printf("Nedovoljan broj argumenata komandne linije.\n");
46         printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
47             argv[0]);
48         exit(EXIT_FAILURE);
49     }
50
51     /* Niska ime_fje sadrzi ime trazene funkcije koja je navedena u
52        komandnoj liniji */
53     strcpy(ime_fje, argv[1]);
54
55     /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
56        */
57     if (strcmp(ime_fje, "sin") == 0)
58         fp = &sin;
59     else if (strcmp(ime_fje, "cos") == 0)
60         fp = &cos;
61     else if (strcmp(ime_fje, "tan") == 0)
62         fp = &tan;
63     else if (strcmp(ime_fje, "atan") == 0)
64         fp = &atan;
65     else if (strcmp(ime_fje, "acos") == 0)
66         fp = &acos;
67     else if (strcmp(ime_fje, "asin") == 0)
68         fp = &asin;
69     else if (strcmp(ime_fje, "exp") == 0)
70         fp = &exp;
71     else if (strcmp(ime_fje, "log") == 0)
72         fp = &log;
73     else if (strcmp(ime_fje, "log10") == 0)
74         fp = &log10;
75     else if (strcmp(ime_fje, "sqrt") == 0)
76         fp = &sqrt;
77     else if (strcmp(ime_fje, "floor") == 0)
78         fp = &floor;
79     else if (strcmp(ime_fje, "ceil") == 0)
80         fp = &ceil;
81     else if (strcmp(ime_fje, "sqr") == 0)
82         fp = &sqr;
```



```
83     else {
84         printf("Program jos uvek ne podrzava trazenu funkciju!\n");
85         exit(EXIT_SUCCESS);
86     }
87
88     printf("Unesite krajeve intervala:\n");
89     scanf("%lf %lf", &a, &b);
90
91     printf("Koliko tacaka ima na ekvidistantnoj mrezi ");
92     printf("(ukljucujuci krajeve intervala)?\n");
93     scanf("%d", &n);
94
95     /* Mreza mora da ukljucuje bar krajeve intervala, tako da se mora
96        uneti broj veci od 2 */
97     if (n < 2) {
98         fprintf(stderr, "Broj tacaka mreze mora biti bar 2!\n");
99         exit(EXIT_FAILURE);
100     }
101
102     /* Ispisuje se ime funkcije */
103     printf("      x %10s(x)\n", ime_fje);
104
105     /* Prosledjuje se funkciji tabela() funkcija zadata kao argument
106        komandne linije */
107     tabela(a, b, n, fp);
108
109     exit(EXIT_SUCCESS);
110 }
```


Glava 3

Algoritmi pretrage i sortiranja

3.1 Pretraživanje

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

- (a) Napisati funkciju `linarna_pretraga` koja vrši linearnu pretragu niza celih brojeva `a`, dužine `n`, tražeći u njemu broj `x`.
- (b) Napisati funkciju `binarna_pretraga` koja vrši binarnu pretragu sortiranog niza `a`, dužine `n`, tražeći u njemu broj `x`.
- (c) Napisati funkciju `interpolaciona_pretragakoja` vrši interpolacionu pretragu sortiranog niza `a`, dužine `n`, tražeći u njemu broj `x`.

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

Test 1

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

```
Izlaz:
```

```
Linearna pretraga  
Element nije u nizu
```

```
-----  
Binarna pretraga  
Element nije u nizu
```

```
-----  
Interpolaciona pretraga  
Element nije u nizu
```

Test 2

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

```
Izlaz:
```

```
Linearna pretraga  
Element nije u nizu
```

```
-----  
Binarna pretraga  
Element nije u nizu
```

```
-----  
Interpolaciona pretraga  
Element nije u nizu
```

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

Primer 1

```
INTERAKCIJA PROGRAMA:
```

```
Unesite traženi broj: 11  
Unesite sortirani niz elemenata:  
2 5 6 8 10 11 23  
Linearna pretraga  
Pozicija elementa je 5.  
Binarna pretraga  
Pozicija elementa je 5.  
Interpolaciona pretraga  
Pozicija elementa je 5.
```

Primer 2

```
INTERAKCIJA PROGRAMA:
```

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

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

Primer 1

```

POZIV: ./a.out datoteka.txt

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

INTERAKCIJA PROGRAMA:
Unesite indeks studenta cije informacije želite: 20140076
Indeks: 20140076, Ime i prezime: Sonja Stevanovic
Unesite prezime studenta cije informacije želite: Popovic
Indeks: 20140032, Ime i prezime: Dejan Popovic

```

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

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

Test 1

```

POZIV: ./a.out dat.txt -x

DAT.TXT
12 53
2.342 34.1
-0.3 23
-1 23.1
123.5 756.12

IZLAZ:
-0.3 23

```

Test 2

```

POZIV: ./a.out dat.txt

DAT.TXT
12 53
2.342 34.1
-0.3 23
-1 2.1
123.5 756.12

IZLAZ:
-1 2.1

```

Test 3

```

POZIV: ./a.out dat.txt -y

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

IZLAZ:
-0.3 0.23

```

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

Test 1

```
|| IZLAZ:  
|| 1.57031
```

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

Test 1

```
|| ULAZ:  
|| -151 -44 5 12 13 15  
||  
|| IZLAZ:  
|| 2
```

Test 2

```
|| ULAZ:  
|| -100 -15 -11 -8 -7 -5  
||  
|| IZLAZ:  
|| -1
```

Test 3

```
|| ULAZ:  
|| -100 -15 0 13 55 124  
|| 258 315 516 7000  
||  
|| IZLAZ:  
|| 3
```

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

Test 1

```
|| ULAZ:  
|| 151 44 5 -12 -13 -15  
||  
|| IZLAZ:  
|| 3
```

Test 2

```
|| ULAZ:  
|| 100 55 15 0 -15 -124  
|| -155 -258 -315 -516  
||  
|| IZLAZ:  
|| 4
```

Test 3

```
|| ULAZ:  
|| 100 15 11 8 7 5 4 3 2  
||  
|| IZLAZ:  
|| -1
```

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

- Napisati funkciju linearne složenosti koja određuje logaritam pomeranjem broja udesno.
- 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	Test 2	Test 3
<pre> ULAZ: 4 IZLAZ: 2 2 </pre>	<pre> ULAZ: 17 IZLAZ: 4 4 </pre>	<pre> ULAZ: 1031 IZLAZ: 10 10 </pre>

**** Zadatak 3.10** U prvom kvadrantu dato je $1 \leq N \leq 10000$ duži svojim koordinatama (duži mogu da se seku, preklapaju, itd.). Napisati program koji pronalazi najmanji ugao $0 \leq \alpha \leq 90^\circ$, 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 $[0, 90^\circ]$.*

Primer 1	Primer 2	Primer 3
<pre> INTERAKCIJA PROGRAMA: Unesi broj tacaka: 2 Unesi koordinate tacaka: 2 0 2 1 1 2 2 2 26.57 </pre>	<pre> INTERAKCIJA PROGRAMA: Unesi broj tacaka: 2 Unesi koordinate tacaka: 1 0 1 1 0 1 1 1 45 </pre>	<pre> INTERAKCIJA PROGRAMA: Unesi broj tacaka: 3 Unesi koordinate tacaka: 1 0 1 1 2 0 2 1 1 2 2 2 26.57 </pre>

3.2 Sortiranje

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

Test 1	Test 2	Test 3
<pre> ULAZ: 23 64 123 76 22 7 IZLAZ: 1 </pre>	<pre> ULAZ: 21 654 65 123 65 12 61 IZLAZ: 0 </pre>	<pre> ULAZ: 34 30 IZLAZ: 4 </pre>

3 Algoritmi pretrage i sortiranja

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

Primer 1

```
|| INTERAKCIJA PROGRAMA:
|| Unesite prvu nisku anagram
|| Unesite drugu nisku rangana
|| jesu
```

Primer 2

```
|| INTERAKCIJA PROGRAMA:
|| Unesite prvu nisku anagram
|| Unesite drugu nisku anagrmm
|| nisu
```

Primer 3

```
|| INTERAKCIJA PROGRAMA:
|| Unesite prvu nisku test
|| Unesite drugu nisku tset
|| jesu
```

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

Test 1

```
|| ULAZ:
|| 4 23 5 2 4 6 7 34 6 4 5
||
|| IZLAZ:
|| 4
```

Test 2

```
|| ULAZ:
|| 2 4 6 2 6 7 99 1
||
|| IZLAZ:
|| 2
```

Test 3

```
|| ULAZ:
|| 123
||
|| IZLAZ:
|| 123
```

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

Primer 1

```
|| INTERAKCIJA PROGRAMA:
|| Unesite trazeni zbir: 34
|| Unesite elemente niza:
|| 134 4 1 6 30 23
|| da
```

Primer 2

```
|| INTERAKCIJA PROGRAMA:
|| Unesite trazeni zbir: 12
|| Unesite elemente niza:
|| 53 1 43 3 56 13
|| ne
```

Primer 3

```
|| INTERAKCIJA PROGRAMA:
|| Unesite trazeni zbir: 52
|| Unesite elemente niza:
|| 52
|| ne
```

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

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

Test 1

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

Test 2

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

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

Primer 1

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

Primer 2

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

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

Test 1

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

PRVI-DEO.TXT                                CEO-TOK.TXT
Andrija Petrovic                             Aleksandra Cvetic
Anja Ilic                                    Andrija Petrovic
Ivana Markovic                               Anja Ilic
Lazar Micic                                 Bojan Golubovic
Nenad Brankovic                             Dragan Markovic
Sofija Filipovic                            Filip Dukic
Vladimir Savic                             Ivana Stankovic
Uros Milic                                  Ivana Markovic
                                              Lazar Micic
DRUGI-DEO.TXT                               Marija Stankovic
Aleksandra Cvetic                           Nenad Brankovic
Bojan Golubovic                             Ognjen Peric
Dragan Markovic                             Sofija Filipovic
Filip Dukic                                 Uros Milic
Ivana Stankovic                             Vladimir Savic
Marija Stankovic
Ognjen Peric
```

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

Test 1

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

IN.TXT
3 4
11 6
7 3
2 82
-1 6

OUT.TXT
-1 6
2 82
3 4
7 3
11 6
```

Test 2

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

IN.TXT
3 4
11 6
7 3
2 82
-1 6

OUT.TXT
3 4
-1 6
7 3
11 6
2 82
```

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

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

Test 1

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

IZLAZ:
3
```

Test 2

```
BIRACKI-SPISAK.TXT
Milan Milicevic

IZLAZ:
1
```

Test 3

```
DATOTEKA BIRACKI-SPISAK.TXT
NE POSTOJI

IZLAZ:
Problem pri otvaranju
datoteke.
```

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

Test 1

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

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

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

Test 2

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

IN.OUT
Milijana Maric 2009
```

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

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

3 Algoritmi pretrage i sortiranja

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

Test 1

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

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

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

Primer 1

```

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

INTERAKCIJA PROGRAMA:
Asortiman:
KOD Naziv artikla Ime proizvođača 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 traženim kodom!
Unesite kod artikla [ili 0 za prekid]: 2530
Trazili ste: Napolitanke Bambi 230.00
Unesite kod artikla [ili 0 za prekid]: 0

UKUPNO: 230.00 dinara.

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

Kraj rada kase!

```

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

3 Algoritmi pretrage i sortiranja

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

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

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

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

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

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<code>Poziv: ./a.out</code>	<code>Poziv: ./a.out -i</code>	<code>Poziv: ./a.out -n</code>
<code>PESME.TXT</code>	<code>PESME.TXT</code>	<code>PESME.TXT</code>
<code>5</code>	<code>5</code>	<code>5</code>
<code>Ana - Nebo, 2342</code>	<code>Ana - Nebo, 2342</code>	<code>Ana - Nebo, 2342</code>
<code>Laza - Oblaci, 29</code>	<code>Laza - Oblaci, 29</code>	<code>Laza - Oblaci, 29</code>
<code>Pera - Ptice, 327</code>	<code>Pera - Ptice, 327</code>	<code>Pera - Ptice, 327</code>
<code>Jelena - Sunce, 92321</code>	<code>Jelena - Sunce, 92321</code>	<code>Jelena - Sunce, 92321</code>
<code>Mika - Kisa, 5341</code>	<code>Mika - Kisa, 5341</code>	<code>Mika - Kisa, 5341</code>
<code>IZLAZ:</code>	<code>IZLAZ:</code>	<code>IZLAZ:</code>
<code>Jelena - Sunce, 92321</code>	<code>Ana - Nebo, 2342</code>	<code>Mika - Kisa, 5341</code>
<code>Mika - Kisa, 5341</code>	<code>Jelena - Sunce, 92321</code>	<code>Ana - Nebo, 2342</code>
<code>Ana - Nebo, 2342</code>	<code>Laza - Oblaci, 29</code>	<code>Laza - Oblaci, 29</code>
<code>Pera - Ptice, 327</code>	<code>Mika - Kisa, 5341</code>	<code>Pera - Ptice, 327</code>
<code>Laza - Oblaci, 29</code>	<code>Pera - Ptice, 327</code>	<code>Jelena - Sunce, 92321</code>

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

Primer 1

```
INTERAKCIJA PROGRAMA:
Unesi niz operacija: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
0 2 8 2 6
```

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

3 5 2 1

3 Algoritmi pretrage i sortiranja

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

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

Test 1

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

3.3 Bibliotečke funkcije pretrage i sortiranja

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

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

Test 1

```
|| ULAZ:
|| R
||
|| IZLAZ:
|| Dusko Radovic
```

Test 2

```
|| ULAZ:
|| E
||
|| IZLAZ:
|| Dobrica Eric
```

Test 3

```
|| ULAZ:
|| A
||
|| IZLAZ:
|| Mika Antic
```

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

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

Primer 1

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

Primer 2

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

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

Primer 1

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

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

- (a) leksikografski,
- (b) po dužini.

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

Primer 1

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

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

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

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

Primer 1

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

```
ULAZNA DATOTEKA (KOLOKVIJUM.TXT):
```

```
Aleksandra Cvetic 15
Bojan Golubovic 30
Dragan Markovic 25
Filip Dukic 20
Ivana Stankovic 25
Marija Stankovic 15
Ognjen Peric 20
Uros Milic 10
Andrija Petrovic 0
Anja Ilic 5
Ivana Markovic 5
Lazar Micic 20
Nenad Brankovic 15
```

```
INTERAKCIJA PROGRAMA:
```

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

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

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

Primer 1

```
INTERAKCIJA PROGRAMA:
```

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

Primer 2

```
INTERAKCIJA PROGRAMA:
```

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

Primer 3

```
INTERAKCIJA PROGRAMA:
```

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

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

3 Algoritmi pretrage i sortiranja

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

Test 1

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

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

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

IZLAZ:
mm13321 Marija Radic 12
```

Test 2

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

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

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

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

Primer 1

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

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

```
INTERAKCIJA PROGRAMA:
Unesi sledeci datum: 13.12.2016.
postoji
Unesi sledeci datum: 10.5.2015.
ne postoji
Unesi sledeci datum: 5.2.2009.
postoji
```

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

Test 1

```

INTERAKCIJA PROGRAMA:
Unesite dimenzije matrice: 3 2
Unesite elemente matrice po vrstama:
6 -5
-4 3
2 1
Sortirana matrica je:
-4 3
6 -5
2 1

```

Test 2

```

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

```

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

Primer 1

```

INTERAKCIJA PROGRAMA:
Unesite dimenziju matrice: 2
Unesite elemente matrice po vrstama:
6 -5
-4 3
Sortirana matrica je:
-5 6
3 -4

```

Primer 2

```

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

```

3.4 Rešenja

Rešenje 3.1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define MAX 1000000
5
6 /* Pri prevodjenju program linkovati sa bibliotekom librt opcijom
   -lrt zbog funkcije clock_gettime() */
7
8 /* Funkcija pretrazuje niz a[] celih brojeva duzine n, trazeci u

```

3 Algoritmi pretrage i sortiranja

```
10      njemu element x. Pretraga se vrši prostom iteracijom kroz niz. Ako
12      se element pronadje funkcija vraća indeks pozicije na kojoj je
      pronadjen. Ovaj indeks je uvek nenegativan. Ako element nije
      pronadjen u nizu, funkcija vraća -1, kao indikator neuspesne
14      pretrage. */
      int linearna_pretraga(int a[], int n, int x)
16  {
      int i;
18      for (i = 0; i < n; i++)
          if (a[i] == x)
20              return i;
      return -1;
22  }

24  /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
      pozicije nadjenog elementa ili -1, ako element nije pronadjen. */
26  int binarna_pretraga(int a[], int n, int x)
  {
28      int levi = 0;
      int desni = n - 1;
30      int srednji;
      /* Dokle god je indeks levi levo od indeksa desni */
32      while (levi <= desni) {
          /* Srednji indeks je njihova aritmeticka sredina */
34          srednji = (levi + desni) / 2;
          /* Ako je element sa sredisnjim indeksom veci od x, tada se x
36             mora nalaziti u levoj polovini niza */
          if (x < a[srednji])
38              desni = srednji - 1;
          /* Ako je element sa sredisnjim indeksom manji od x, tada se x
40             mora nalaziti u desnoj polovini niza */
          else if (x > a[srednji])
42              levi = srednji + 1;
          else
44              /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
                  x pronadjen na poziciji srednji */
46              return srednji;
      }
48      /* Ako element x nije pronadjen, vraća se -1 */
      return -1;
50  }

52  /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca indeks
      pozicije nadjenog elementa ili -1, ako element nije pronadjen */
54  int interpolaciona_pretraga(int a[], int n, int x)
  {
56      int levi = 0;
      int desni = n - 1;
58      int srednji;
      /* Dokle god je indeks levi levo od indeksa desni... */
60      while (levi <= desni) {
          /* Ako je trazeni element manji od pocetnog ili veci od
```

```

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

104 int main(int argc, char **argv)
{
106     int a[MAX];
    int n, i, x;
108     struct timespec time1, time2, time3, time4, time5, time6;
    FILE *f;

110     /* Provera argumenata komandne linije */
112     if (argc != 3) {

```

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

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

124

126 /* Broj koji se trazi */
x = atoi(argv[2]);

128 /* Elementi niza se generisu slucajno, tako da je svaki sledeci
    veci od prethodnog. srandom() funkcija obezbedjuje novi seed za
130 pozivanje random() funkcije. Kako generisani niz ne bi uvek isto
    izgledao, seed se postavlja na tekuce vreme u sekundama od Nove
132 godine 1970. random()%100 daje brojeve izmedju 0 i 99 */
srandom(time(NULL));
134 for (i = 0; i < n; i++)
    a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
136

138 /* Linearna pretraga */
printf("Linearna pretraga\n");
/* Vreme proteklo od Nove godine 1970 */
140 clock_gettime(CLOCK_REALTIME, &time1);
i = linearna_pretraga(a, n, x);
142 /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
    linearnu pretragu */
144 clock_gettime(CLOCK_REALTIME, &time2);
if (i == -1)
146     printf("Element nije u nizu\n");
else
148     printf("Element je u nizu na poziciji %d\n", i);
printf("-----\n");
150

152 /* Binarna pretraga */
printf("Binarna pretraga\n");
clock_gettime(CLOCK_REALTIME, &time3);
154 i = binarna_pretraga(a, n, x);
clock_gettime(CLOCK_REALTIME, &time4);
156 if (i == -1)
    printf("Element nije u nizu\n");
158 else
    printf("Element je u nizu na poziciji %d\n", i);
160 printf("-----\n");

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



```

164 clock_gettime(CLOCK_REALTIME, &time5);
    i = interpolaciona_pretraga(a, n, x);
166 clock_gettime(CLOCK_REALTIME, &time6);
    if (i == -1)
168     printf("Element nije u nizu\n");
    else
170     printf("Element je u nizu na poziciji %d\n", i);
    printf("-----\n");
172
    /* Podaci o izvršavanju programa bivaju upisani u log fajl */
174 if ((f = fopen("vremena.txt", "a")) == NULL) {
    fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
176     exit(EXIT_FAILURE);
    }
178
    fprintf(f, "Dimenzija niza: %d\n", n);
180 fprintf(f, "\t\tLinearna pretraga:%10ld ns\n",
        (time2.tv_sec - time1.tv_sec) * 1000000000 +
182         time2.tv_nsec - time1.tv_nsec);
    fprintf(f, "\t\tBinarna: %19ld ns\n",
184         (time4.tv_sec - time3.tv_sec) * 1000000000 +
        time4.tv_nsec - time3.tv_nsec);
186 fprintf(f, "\t\tInterpolaciona: %12ld ns\n\n",
        (time6.tv_sec - time5.tv_sec) * 1000000000 +
188         time6.tv_nsec - time5.tv_nsec);
190
    /* Zatvaranje datoteke */
    fclose(f);
192
    return 0;
194 }

```

Rešenje 3.2

```

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

```

```
18  int lin_pretraga_rek_prefiks(int a[], int n, int x)
19  {
20      /* Izlaz iz rekurzije */
21      if (n <= 0)
22          return -1;
23      /* Ako je poslednji element trazeni */
24      if (a[n - 1] == x)
25          return n - 1;
26      /* Pretraga ostatka niza */
27      return lin_pretraga_rek_prefiks(a, n - 1, x);
28  }
29
30  int bin_pretraga_rek(int a[], int l, int d, int x)
31  {
32      int srednji;
33      if (l > d)
34          return -1;
35      /* Sredisnja pozicija na kojoj se trazi vrednost x */
36      srednji = (l + d) / 2;
37      /* Ako je element na sredisnjoj poziciji trazeni */
38      if (a[srednji] == x)
39          return srednji;
40      /* Ako je trazeni broj veci od broja na sredisnjoj poziciji,
41         pretrazuje se desna polovina niza */
42      if (a[srednji] < x)
43          return bin_pretraga_rek(a, srednji + 1, d, x);
44      /* Ako je trazeni broj manji od broja na sredisnjoj poziciji,
45         pretrazuje se leva polovina niza */
46      else
47          return bin_pretraga_rek(a, l, srednji - 1, x);
48  }
49
50
51  int interp_pretraga_rek(int a[], int l, int d, int x)
52  {
53      int p;
54      if (x < a[l] || x > a[d])
55          return -1;
56      if (a[d] == a[l])
57          return l;
58      /* Pozicija na kojoj se trazi vrednost x */
59      p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
60      if (a[p] == x)
61          return p;
62      if (a[p] < x)
63          return interp_pretraga_rek(a, p + 1, d, x);
64      else
65          return interp_pretraga_rek(a, l, p - 1, x);
66  }
67
68  int main()
```

```

70 {
    int a[MAX];
72     int x;
    int i, indeks;

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

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

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

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

102    /* Interpolaciona pretraga */
104    printf("Interpolaciona pretraga\n");
    indeks = interp_pretraga_rek(a, 0, i - 1, x);
106    if (indeks == -1)
        printf("Element se ne nalazi u nizu.\n");
108    else
        printf("Pozicija elementa je %d.\n", indeks);
110    return 0;
112 }

```

Rešenje 3.3

```

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

5 #define MAX_STUDENATA 128

```

3 Algoritmi pretrage i sortiranja

```
#define MAX_DUZINA 16

7
/* 0 svakom studentu postoje 3 informacije i one su objedinjene u
9  strukturi kojom se predstavlja svaki student. */
typedef struct {
11  /* Indeks mora biti tipa long jer su podaci u datoteci preveliki za
    int, npr. 20140123 */
13  long indeks;
    char ime[MAX_DUZINA];
15  char prezime[MAX_DUZINA];
} Student;

17
/* Učitani niz studenata će biti sortirani rastuće prema indeksu, jer
19  su studenti u datoteci već sortirani. Iz tog razloga pretraga po
    indeksu se vrši binarno, dok pretraga po prezimenu mora linearno,
21  jer nema garancije da postoji uredjenje po prezimenu. */

23  /* Funkcija traži u sortiranom nizu studenata a[] dužine n studenta
    sa indeksom x i vraća indeks pozicije nadjenog člana niza ili -1,
25  ako element nije pronađen. */
int binarna_pretraga(Student a[], int n, long x)
27 {
    int levi = 0;
29  int desni = n - 1;
    int srednji;
31  /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
33      /* Računa se srednja pozicija */
        srednji = (levi + desni) / 2;
35      /* Ako je indeks studenta na toj poziciji veći od traženog, tada
        se traženi indeks mora nalaziti u levoj polovini niza */
37      if (x < a[srednji].indeks)
        desni = srednji - 1;
39      /* Ako je pak manji od traženog, tada se on mora nalaziti u
        desnoj polovini niza */
41      else if (x > a[srednji].indeks)
        levi = srednji + 1;
43      else
        /* Ako je jednak traženom indeksu x, tada je pronađen student
45      sa traženom indeksom na poziciji srednji */
        return srednji;
47  }
    /* Ako nije pronađen, vraća se -1 */
49  return -1;
}

51
/* Linearnom pretragom niza studenata traži se prezime x */
53 int linearna_pretraga(Student a[], int n, char x[])
{
55  int i;
    for (i = 0; i < n; i++)
57      /* Poređenje prezimena i-tog studenta i poslatog x */
```

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

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

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

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

107    /* Unos indeksa koji se binarno traži u nizu */
    printf("Unesite indeks studenta čije informacije želite: ");
```

3 Algoritmi pretrage i sortiranja

```
109 scanf("%ld", &trazen_indeks);
    i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
111 /* Rezultat binarne pretrage */
    if (i == -1)
113         printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
115         printf("Indeks: %ld, Ime i prezime: %s %s\n",
                dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
117
    /* Unos prezimena koje se linearno trazi u nizu */
119 printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
121 i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    /* Rezultat linearne pretrage */
123 if (i == -1)
        printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
125 else
        printf("Indeks: %ld, Ime i prezime: %s %s\n",
127                dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
129 return 0;
}
```

Rešenje 3.4

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX_STUDENATA 128
6 #define MAX_DUZINA 16

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

14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
                                long x)
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 trazen */
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);
}

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 trazanim 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 }

int linearna_pretraga_rekurzivna(Student a[], int n, char x[])
52 {
54     /* Ako je niz prazan, vraca se -1 */
    if (n == 0)
56         return -1;
    /* Ako se trazi poslednji student sa trazanim 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 imate argumente jer se ime datoteke prosledjuje
kao argument 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 trazeni_indeks = 0;
    char trazeno_prezime[MAX_DUZINA];
74
    /* Provera da li je korisnik prilikom poziva prosledio ime datoteke
76 sa informacijama o studentima */
    if (argc != 2) {
78         fprintf(stderr,

```

3 Algoritmi pretrage i sortiranja

```

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

84 /* Otvaranje datoteke */
fin = fopen(argv[1], "r");
86 if (fin == NULL) {
    fprintf(stderr,
88         "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
    exit(EXIT_FAILURE);
90 }

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

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

108

110 /* Unos indeksa koji se binarno traži u nizu */
printf("Unesite indeks studenta čije informacije želite: ");
scanf("%ld", &trazen_indeks);
112 i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata - 1,
                                trazen_indeks);

114 if (i == -1)
    printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
116 else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
118         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

120 /* Unos prezimena koje se linearno traži u nizu */
printf("Unesite prezime studenta čije informacije želite: ");
122 scanf("%s", trazeno_prezime);
i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
                                    trazeno_prezime);
124

126 if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n", trazeno_prezime);
else
128     printf
        ("Prvi takav student:\nIndeks: %ld, Ime i prezime: %s %s\n",
130         dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
```



```
132     return 0;
    }
```

Rešenje 3.5

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  /* Struktura koja opisuje tacku u ravni */
7  typedef struct Tacka {
8      float x;
9      float y;
10 } Tacka;
11
12 /* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
13    pocetka (0,0) */
14 float rastojanje(Tacka A)
15 {
16     return sqrt(A.x * A.x + A.y * A.y);
17 }
18
19 /* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u nizu
20    zadatih tacaka t dimenzije n */
21 Tacka najbliza_koordinatnom(Tacka t[], int n)
22 {
23     Tacka najbliza;
24     int i;
25     najbliza = t[0];
26     for (i = 1; i < n; i++) {
27         if (rastojanje(t[i]) < rastojanje(najbliza)) {
28             najbliza = t[i];
29         }
30     }
31     return najbliza;
32 }
33
34 /* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih tacaka
35    t dimenzije n */
36 Tacka najbliza_x_osi(Tacka t[], int n)
37 {
38     Tacka najbliza;
39     int i;
40     najbliza = t[0];
41     for (i = 1; i < n; i++) {
42         if (fabs(t[i].x) < fabs(najbliza.x)) {
43             najbliza = t[i];
44         }
45     }
```

```

    }
47     return najbliza;
    }
49
    /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih tacaka
    t dimenzije n */
51     Tacka najbliza_y_osi(Tacka t[], int n)
52     {
53         Tacka najbliza;
54         int i;
55         najbliza = t[0];
56         for (i = 1; i < n; i++) {
57             if (fabs(t[i].y) < fabs(najbliza.y)) {
58                 najbliza = t[i];
59             }
60         }
61         return najbliza;
62     }
63 }
64
65 #define MAX 1024
66
67 int main(int argc, char *argv[])
68 {
69     FILE *ulaz;
70     Tacka tacke[MAX];
71     Tacka najbliza;
72     int i, n;
73
74     /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
75     ime datoteke sa tackama */
76     if (argc < 2) {
77         fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
79         return EXIT_FAILURE;
80     }
81
82     /* Otvaranje datoteke za citanje */
83     ulaz = fopen(argv[1], "r");
84     if (ulaz == NULL) {
85         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
87         return EXIT_FAILURE;
88     }
89
90     /* Sve dok ima tacaka u datoteci, one bivaju smestane u niz sa
91     tackama; i predstavlja indeks tekuce tacke */
92     i = 0;
93     while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
94         i++;
95     }
96     n = i;
97 }
```

```

99  /* Proverava se koji su dodatni argumenti komandne linije. Ako nema
    dodatnih argumenata */
101 if (argc == 2)
    /* Trazi se najbliza tacka u odnosu na koordinatni pocetak */
    najbliza = najbliza_koordinatnom(tacke, n);
103 /* Inace proverava se koji je dodatni argument prosledjen. Ako je u
    pitanju opcija -x */
105 else if (strcmp(argv[2], "-x") == 0)
    /* Racuna se rastojanje u odnosu na x osu */
107     najbliza = najbliza_x_osi(tacke, n);
    /* Ako je u pitanju opcija -y */
109 else if (strcmp(argv[2], "-y") == 0)
    /* Racuna se rastojanje u odnosu na y osu */
111     najbliza = najbliza_y_osi(tacke, n);
    else {
113     /* Ako nije zadata opcija -x ili -y, ispisuje se obavestenje za
        korisnika i prekida se izvršavanje programa */
115     fprintf(stderr, "Pogresna opcija\n");
        return EXIT_FAILURE;
117     }

119 /* Stampanje koordinata trazene tacke */
    printf("%g %g\n", najbliza.x, najbliza.y);
121
    /* Zatvaranje datoteke */
123     fclose(ulaz);

125     return 0;
}

```

Rešenje 3.6

```

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

4  /* Tacnost */
#define EPS 0.001

6
int main()
8  {
    double l, d, s;

10
    /* Kako je u pitanju interval [0, 2] leva granica je 0, a desna 2
       */
12     l = 0;
        d = 2;

14
    /* Sve dok se ne pronadje trazena vrednost argumenta */
16     while (1) {
        /* Polovi se interval */
18         s = (l + d) / 2;
    }

```

3 Algoritmi pretrage i sortiranja

```
20      /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
21         tacnosti, prekida se pretraga */
22      if (fabs(cos(s)) < EPS) {
23          break;
24      }
25      /* Ako je nula u levom delu intervala, nastavlja se pretraga na
26         [l, s] */
27      if (cos(l) * cos(s) < 0)
28          d = s;
29      else
30          /* Inace, na intervalu [s, d] */
31          l = s;
32      }
33
34      /* Stapanje vrednost trazene tacke */
35      printf("%g\n", s);
36
37      return 0;
38  }
```

Rešenje 3.7

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 256
5
6  int prvi_veci_od_nule(int niz[], int n)
7  {
8      /* Granice pretrage */
9      int l = 0, d = n - 1;
10     int s;
11     /* Sve dok je leva manja od desne granice */
12     while (l <= d) {
13         /* Racuna se sredisnja pozicija */
14         s = (l + d) / 2;
15         /* Ako je broj na toj poziciji veci od nule, a eventualni njegov
16            prethodnik manji ili jednak nuli, pretraga je završena */
17         if (niz[s] > 0 && ((s > 0 && niz[s - 1] <= 0) || s == 0))
18             return s;
19         /* U slucaju broja manjeg ili jednakog nuli, pretrazuje se desna
20            polovina niza */
21         if (niz[s] <= 0)
22             l = s + 1;
23         /* A inace, leva polovina */
24         else
25             d = s - 1;
26     }
27     return -1;
28 }
```

```

30 int main()
31 {
32     int niz[MAX];
33     int n = 0;
34
35     /* Unos niza */
36     while (scanf("%d", &niz[n]) == 1)
37         n++;
38
39     /* Stampanje rezultata */
40     printf("%d\n", prvi_veci_od_nule(niz, n));
41
42     return 0;
43 }

```

Rešenje 3.8

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 256
5
6  int prvi_manji_od_nule(int niz[], int n)
7  {
8      /* Granice pretrage */
9      int l = 0, d = n - 1;
10     int s;
11     /* Sve dok je leva manja od desne granice */
12     while (l <= d) {
13         /* Racuna se sredisnja pozicija */
14         s = (l + d) / 2;
15         /* Ako je broj na toj poziciji manji od nule, a eventualni njegov
16            prethodnik veci ili jednak nuli, pretraga se zavrшава */
17         if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
18             return s;
19         /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
20            niza */
21         if (niz[s] >= 0)
22             l = s + 1;
23         /* A inace leva */
24         else
25             d = s - 1;
26     }
27     return -1;
28 }
29
30 int main()
31 {
32     int niz[MAX];
33     int n = 0;

```

3 Algoritmi pretrage i sortiranja

```
35  /* Unos niza */
    while (scanf("%d", &niz[n]) == 1)
37      n++;

39  /* Stapanje rezultata */
    printf("%d\n", prvi_manji_od_nule(niz, n));
41
    return 0;
43 }
```

Rešenje 3.9

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

3
   unsigned int logaritam_a(unsigned int x)
5  {
   /* Izlaz iz rekurzije */
7   if (x == 1)
       return 0;
9   /* Rekurzivni korak */
       return 1 + logaritam_a(x >> 1);
11 }

13 unsigned int logaritam_b(unsigned int x)
   {
15   /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
       najvece vaznosti, tj. najlevlja. Pretragu se vrsi od pozicije 0
17   do 31 */
       int d = 0, l = sizeof(unsigned int) * 8 - 1;
       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;
       /* Pretraga desne polovine binarnog zapisa */
       if ((1 << s) > x)
27           l = s - 1;
       /* Pretraga leve polovine binarnog zapisa */
       else
31           d = s + 1;
       }
33   }
       return s;
35 }

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

```

41  /* Unos podatka */
    scanf("%u", &x);

43

45  /* Provera da li je uneti broj pozitivan */
    if (x == 0) {
        fprintf(stderr, "Logaritam od nule nije definisan\n");
47        exit(EXIT_FAILURE);
    }

49

51  /* Ispis povratnih vrednosti funkcija */
    printf("%u %u\n", logaritam_a(x), logaritam_b(x));

53  return 0;
}

```

Rešenje 3.11

```

1  #include<stdio.h>
   #define MAX 256

3

4  /* Iterativna verzija funkcije koja sortira niz celih brojeva,
   primenom algoritma Selection Sort */
5  void selectionSort(int a[], int n)
6  {
7      int i, j;
8      int min;
9      int pom;

11

13     /* U svakoj iteraciji ove petlje se pronalazi najmanji element
       medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
       poziciju i, dok se element na poziciji i premesta na poziciju
15     min, na kojoj se nalazio najmanji od gore navedenih elemenata.
       */
17     for (i = 0; i < n - 1; i++) {
        /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
           najmanji od elemenata a[i],...,a[n-1]. */
19         min = i;
21         for (j = i + 1; j < n; j++)
             if (a[j] < a[min])
                 min = j;

23         /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
           su (i) i min razliciti, inace je nepotrebno. */
25         if (min != i) {
             pom = a[i];
             a[i] = a[min];
             a[min] = pom;
27         }
29     }
31 }

```

3 Algoritmi pretrage i sortiranja

```
33 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
    sortiranom nizu celih brojeva */
35 int najmanje_rastojanje(int a[], int n)
{
37     int i, min;
    min = a[1] - a[0];
39     for (i = 2; i < n; i++)
        if (a[i] - a[i - 1] < min)
41         min = a[i] - a[i - 1];
    return min;
43 }

45
47 int main()
{
49     int i, a[MAX];

    /* Ucitavaju se elementi niza sve do kraja ulaza */
51     i = 0;
    while (scanf("%d", &a[i]) != EOF)
53         i++;

55     /* Sortiranje */
    selectionSort(a, i);

57     /* Ispis rezultata */
59     printf("%d\n", najmanje_rastojanje(a, i));

61     return 0;
}
```

Rešenje 3.12

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

4 #define MAX_DIM 128

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



```

    s[min] = pom;
20     }
    }
22 }

24 /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
26 {
    int i;
28
    /* Ako dve niske imaju razlicit broj karaktera onda one nisu
30     anagrami */
    if (strlen(s) != strlen(t))
32         return 0;

    /* Sortiramo niske */
    selectionSort(s);
36     selectionSort(t);

    /* Dve sortirane niske su anagrami ako i samo ako su jednake */
38     for (i = 0; s[i] != '\0'; i++)
        if (s[i] != t[i])
40             return 0;
    return 1;
42 }

44
int main()
46 {
    char s[MAX_DIM], t[MAX_DIM];
48
    /* Ucitavanje niski sa ulaza */
    printf("Unesite prvu nisku: ");
    scanf("%s", s);
52     printf("Unesite drugu nisku: ");
    scanf("%s", t);
54
    /* Poziv funkcije */
    if (anagrami(s, t))
56         printf("jesu\n");
    else
58         printf("nisu\n");

    return 0;
60 }
62

```

Rešenje 3.13

```

1  #include<stdio.h>
   #define MAX_DIM 256
3
   /* Funkcija za sortiranje niza */

```

3 Algoritmi pretrage i sortiranja

```
5 void selectionSort(int s[], int n)
{
7     int i, j, min;
    char pom;
9     for (i = 0; i < n; i++) {
        min = i;
11        for (j = i + 1; j < n; j++)
            if (s[j] < s[min])
13                min = j;
        if (min != i) {
15            pom = s[i];
            s[i] = s[min];
17            s[min] = pom;
        }
19    }
}

21 /* Funkcija za odredjivanje onog elementa sortiranog niza koji se
23    najvise puta pojavio u tom nizu */
int najvise_puta(int a[], int n)
25 {
    int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
27    /* Za i-ti element izracunava se koliko puta se pojavio u nizu */
    for (i = 0; i < n; i = j) {
29        br_pojava = 1;
        for (j = i + 1; j < n && a[i] == a[j]; j++)
31            br_pojava++;
        /* Ispitivanje da li se do tog trenutka i-ti element pojavio
33            najvise puta u nizu */
        if (br_pojava > max_br_pojava) {
35            max_br_pojava = br_pojava;
            i_max_pojava = i;
37        }
    }
39    /* Vraca se element koji se najvise puta pojavio u nizu */
    return a[i_max_pojava];
41 }

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

47     /* Ucitavanje elemenata niza sve do kraja ulaza */
    i = 0;
49     while (scanf("%d", &a[i]) != EOF)
        i++;
51
    /* Niz se sortira */
53     selectionSort(a, i);

55     /* Odredjuje se broj koji se najvise puta pojavio u nizu */
    printf("%d\n", najvise_puta(a, i));
```

```
57     return 0;
59 }
```

Rešenje 3.14

```
1  #include<stdio.h>
2  #define MAX_DIM 256
3
4  /* Funkcija za sortiranje niza */
5  void selectionSort(int a[], int n)
6  {
7      int i, j, min, pom;
8      for (i = 0; i < n - 1; i++) {
9          min = i;
10         for (j = i + 1; j < n; j++)
11             if (a[j] < a[min])
12                 min = j;
13         if (min != i) {
14             pom = a[i];
15             a[i] = a[min];
16             a[min] = pom;
17         }
18     }
19 }
20
21 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
22    u nizu, a 0 inace. Pretpostavlja se da je niz sortiran u rastucem
23    poretku */
24 int binarna_pretraga(int a[], int n, int x)
25 {
26     int levi = 0, desni = n - 1, srednji;
27
28     while (levi <= desni) {
29         srednji = (levi + desni) / 2;
30         if (a[srednji] == x)
31             return 1;
32         else if (a[srednji] > x)
33             desni = srednji - 1;
34         else if (a[srednji] < x)
35             levi = srednji + 1;
36     }
37     return 0;
38 }
39
40 int main()
41 {
42     int a[MAX_DIM], n = 0, zbir, i;
43
44     /* Ucitava se trazeni zbir */
45     printf("Unesite trazeni zbir: ");
```

3 Algoritmi pretrage i sortiranja

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

47
/* Ucitavaju se elementi niza sve do kraja ulaza */
49 i = 0;
printf("Unesite elemente niza: ");
51 while (scanf("%d", &a[i]) != EOF)
    i++;
53 n = i;

/* Sortira se niz */
selectionSort(a, n);

57
for (i = 0; i < n; i++)
59     /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
        niza nalazi element koji sabran sa njim ima ucitanu vrednost
        zbira */
61     if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
63         printf("da\n");
        return 0;
65     }
printf("ne\n");
67
return 0;
69 }
```

Rešenje 3.15

```
/* Datoteka sort.h */
2 #ifndef __SORT_H__
#define __SORT_H__ 1
4
/* Selection sort */
6 void selectionsort(int a[], int n);
/* Insertion sort */
8 void insertionsort(int a[], int n);
/* Bubble sort */
10 void bubblesort(int a[], int n);
/* Shell sort */
12 void shellsort(int a[], int n);
/* Merge sort */
14 void mergesort(int a[], int l, int r);
/* Quick sort */
16 void quicksort(int a[], int l, int r);
18 #endif
```

```
/* Datoteka sort.c */
2
#include "sort.h"
4
```

```

6  #define MAX 1000000
8  /* 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
10 iteraciji, pronalazi se najmanji element, i dovodi na nulto mesto
   u nizu. U i-toj iteraciji najmanjih i elemenata su vec na svojim
12 pozicijama, pa se od i+1 do n-1 elementa trazi najmanji, koji se
   dovodi na i+1 poziciju. */
14 void selectionsort(int a[], int n)
   {
16     int i, j;
17     int min;
18     int pom;

20     /* U svakoj iteraciji ove petlje pronalazi se najmanji element
       medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
22 poziciju i, dok se element na poziciji i premesta na poziciju
       min, na kojoj se nalazio najmanji od gore navedenih elemenata.
       */
24     for (i = 0; i < n - 1; i++) {
       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
26        najmanji od elemenata a[i],...,a[n-1]. */
       min = i;
28       for (j = i + 1; j < n; j++)
         if (a[j] < a[min])
30           min = j;

32       /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
         su (i) i min razliciti, inace je nepotrebno. */
34       if (min != i) {
         pom = a[i];
36         a[i] = a[min];
         a[min] = pom;
38       }
40     }

42     /* Funkcija sortira niz celih brojeva metodom sortiranja umetanjem.
       Ideja algoritma je sledeca: neka je na pocetku i-te iteracije niz
44 prvih i elemenata (a[0],a[1],...,a[i-1]) sortirano. U i-toj
       iteraciji treba element a[i] umetnuti na pravu poziciju medju
46 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
48 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
50 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
52 dalje potiskivanje elementa u levo, poredeci ga sa njegovim novim
       levim susedom. Ovim uzastopnim premanjanjem se a[i] umece na pravo
54 mesto u nizu. */
   void insertion sort(int a[], int n)

```

3 Algoritmi pretrage i sortiranja

```
56 {
57     int i, j;
58
59     /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
60        sortiran */
61     for (i = 1; i < n; i++) {
62
63         /* U ovoj petlji se redom potiskuje element a[i] ulevo koliko je
64            potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...,a[i]
65            bude sortiran. Indeks j je trenutna pozicija na kojoj se
66            element koji se umeće nalazi. Petlja se završava ili kada
67            element dodje do levog kraja (j==0) ili kada se naidje na
68            element a[j-1] koji je manji od a[j]. */
69         for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
70             int temp = a[j];
71             a[j] = a[j - 1];
72             a[j - 1] = temp;
73         }
74     }
75 }
76
77 /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
78    algoritma je sledeca: prolazi se kroz niz redom poredeci susedne
79    elemente, i pri tom ih zamenjujuci ako su u pogresnom poretku.
80    Ovim se najveći element poput mehurica istiskuje na "povrsinu",
81    tj. na krajnju desnu poziciju. Nakon toga je potrebno ovaj
82    postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih n-1
83    elemenata niza bez poslednjeg koji je postavljen na pravu
84    poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
85    kracim prefiksima niza, cime se jedan po jedan istiskuju
86    elementi na svoje prave pozicije. */
87 void bubblesort(int a[], int n)
88 {
89     int i, j;
90     int ind;
91
92     for (i = n, ind = 1; i > 1 && ind; i--)
93     {
94         /* Poput "mehurica" potiskuje se najveći element medju elementima
95            od a[0] do a[i-1] na poziciju i-1 upoređujući susedne
96            elemente niza i potiskujući veci u desno */
97         for (j = 0, ind = 0; j < i - 1; j++)
98             if (a[j] > a[j + 1]) {
99                 int temp = a[j];
100                 a[j] = a[j + 1];
101                 a[j + 1] = temp;
102                 ind = 1;
103             }
104
105         /* Promenljiva ind registruje da je bilo premestanja. Samo u
106            tom slucaju ima smisla ici na sledecu iteraciju, jer ako
107            nije bilo premestanja, znaci da su svi elementi vec u
108            dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
109            niza. Algoritam moze biti i bez ovoga, sortiranje bi bilo
```

```

108         ispravno, ali manje efikasano, jer bi se cesto nepotrebno
110         vrsila mnoga uporedjivanja, kada je vec jasno da je
            sortiranje zavrшено. */
112     ind = 1;
113 }
114
115 /* Selsort je jednostavno prosirenje sortiranja umetanjem koje
116 dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
117 sastoji u tome da se kroz algoritam umetanja prolazi vise puta; u
118 prvom prolazu, umesto koraka 1 uzima se neki korak h koji je manji
119 od n (sto omogucuje razmenu udaljenih elemenata) i tako se dobija
120 h-sortiran niz, tj. niz u kome su elementi na rastojanju h
121 sortirani, mada susedni elementi to ne moraju biti. U drugom
122 prolazu kroz isti algoritam sprovodi se isti postupak ali za manji
123 korak h. Sa prolazima se nastavlja sve do koraka h = 1, u kome se
124 dobija potpuno sortirani niz. Izbor pocetne vrednosti za h, i
125 nacina njegovog smanjivanja menja u nekim slucajevima brzinu
126 algoritma, ali bilo koja vrednost ce rezultovati ispravnim
127 sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
128 vrednost 1. */
129 void shellsort(int a[], int n)
130 {
131     int h = n / 2, i, j;
132     while (h > 0) {
133         /* Insertion sort sa korakom h */
134         for (i = h; i < n; i++) {
135             int temp = a[i];
136             j = i;
137             while (j >= h && a[j - h] > temp) {
138                 a[j] = a[j - h];
139                 j -= h;
140             }
141             a[j] = temp;
142         }
143         h = h / 2;
144     }
145 }
146
147 /* Funkcija sortira niz celih brojeva a[] ucesljavanjem. Sortiranje
148 se vrsi od elementa na poziciji l do onog na poziciji d. Na
149 pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a d je
150 jednako poslednjem validnom indeksu u nizu. Funkcija niz podeli na
151 dve polovine, levu i desnu, koje zatim rekurzivno sortira. Od ova
152 dva sortirana podniza, sortiran niz se dobija ucesljavanjem, tj.
153 istovremenim prolaskom kroz oba niza i izborom trenutnog manjeg
154 elementa koji se smesta u pomocni niz. Na kraju algoritma,
155 sortirani elementi su u pomocnom nizu, koji se kopira u originalni
156 niz. */
157 void mergesort(int a[], int l, int d)
158 {
159     int s;

```

3 Algoritmi pretrage i sortiranja

```
160 static int b[MAX];          /* Pomocni niz */
161 int i, j, k;
162
163 /* Izlaz iz rekurziije */
164 if (l >= d)
165     return;
166
167 /* Odredjivanje sredisnjeg indeksa */
168 s = (l + d) / 2;
169
170 /* Rekurzivni pozivi */
171 mergesort(a, l, s);
172 mergesort(a, s + 1, d);
173
174 /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
175 niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
176 prolazi kroz pomocni niz b[] */
177 i = l;
178 j = s + 1;
179 k = 0;
180
181 /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
182 while (i <= s && j <= d) {
183     if (a[i] < a[j])
184         b[k++] = a[i++];
185     else
186         b[k++] = a[j++];
187 }
188
189 /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive j
190 iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
191 polovine niza */
192 while (i <= s)
193     b[k++] = a[i++];
194
195 /* U slucaju da se prethodna petlja zavrсила izlaskom promenljive i
196 iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
197 polovine niza */
198 while (j <= d)
199     b[k++] = a[j++];
200
201 /* Prepisuje se "ucesljani" niz u originalni niz */
202 for (k = 0, i = l; i <= d; i++, k++)
203     a[i] = b[k];
204 }
205
206 /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
207 void swap(int a[], int i, int j)
208 {
209     int tmp = a[i];
210     a[i] = a[j];
211     a[j] = tmp;
```



```

212 }
214
216 /* Funkcija sortira deo niza brojeva a izmedju pozicija l i r. Njena
218 ideja sortiranja je izbor jednog elementa niza, koji se naziva
220 pivot, i koji se dovodi na svoje mesto. Posle ovog koraka, svi
222 elementi levo od njega bice manji, a svi desno bice veci od njega.
224 Kako je pivot doveden na svoje mesto, da bi niz bio kompletno
226 sortiran, potrebno je sortirati elemente levo (manje) od njega, i
228 elemente desno (vece). Kako su dimenzije ova dva podniza manje od
230 dimenzije pocetnog niza koji je trebalo sortirati, ovaj deo moze
232 se uraditi rekurzivno. */
234 void quicksort(int a[], int l, int r)
236 {
238     int i, pivot_position;
240
242     /* Izlaz iz rekurzije -- prazan niz */
244     if (l >= r)
246         return;
248
250     /* Particionisanje niza. Svi elementi na pozicijama levo od
252     pivot_position (izuzev same pozicije l) su strogo manji od
254     pivota. Kada se pronadje neki element manji od pivota, uvecava
256     se promenljiva pivot_position i na tu poziciju se premesta
258     nadjeni element. Na kraju ce pivot_position zaista biti pozicija
260     na koju treba smestiti pivot, jer ce svi elementi levo od te
262     pozicije biti manji a desno biti veci ili jednaki od pivota. */
264     pivot_position = l;
266     for (i = l + 1; i <= r; i++)
268         if (a[i] < a[l])
270             swap(a, ++pivot_position, i);
272
274     /* Postavljanje pivota na svoje mesto */
276     swap(a, l, pivot_position);
278
280     /* Rekurzivno sortiranje elementa manjih od pivota */
282     quicksort(a, l, pivot_position - 1);
284     /* Rekurzivno sortiranje elementa vecih od pivota */
286     quicksort(a, pivot_position + 1, r);
288 }

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include <time.h>
4 #include "sort.h"

6 /* Maksimalna duzina niza */
#define MAX 1000000

8
int main(int argc, char *argv[])
10 {

```

3 Algoritmi pretrage i sortiranja

```
12  /******  
13      tip_sortiranja == 0 => selectionsort, (podrazumevano)  
14      tip_sortiranja == 1 => insertionsort, -i opcija komandne linije  
15      tip_sortiranja == 2 => bubblesort,    -b opcija komandne linije  
16      tip_sortiranja == 3 => shellsort,      -s opcija komandne linije  
17      tip_sortiranja == 4 => mergesort,      -m opcija komandne linije  
18      tip_sortiranja == 5 => quicksort,      -q opcija komandne linije  
19  *****/  
20  int tip_sortiranja = 0;  
21  /******  
22      tip_niza == 0 => slucajno generisani nizovi, (podrazumevano)  
23      tip_niza == 1 => rastuce sortirani nizovi,    -r opcija  
24      tip_niza == 2 => opadajuce soritrani nizovi, -o opcija  
25  *****/  
26  int tip_niza = 0;  
27  
28  /* Dimenzija niza koji se sortira */  
29  int dimenzija;  
30  int i;  
31  int niz[MAX];  
32  
33  /* Provera argumenata komandne linije */  
34  if (argc < 2) {  
35      fprintf(stderr,  
36          "Program zahteva bar 2 argumenta komandne linije!\n");  
37      exit(EXIT_FAILURE);  
38  }  
39  
40  /* Ocitavanje opcija i argumenata prilikom poziva programa */  
41  for (i = 1; i < argc; i++) {  
42      /* Ako je u pitanju opcija... */  
43      if (argv[i][0] == '-') {  
44          switch (argv[i][1]) {  
45              case 'i':  
46                  tip_sortiranja = 1;  
47                  break;  
48              case 'b':  
49                  tip_sortiranja = 2;  
50                  break;  
51              case 's':  
52                  tip_sortiranja = 3;  
53                  break;  
54              case 'm':  
55                  tip_sortiranja = 4;  
56                  break;  
57              case 'q':  
58                  tip_sortiranja = 5;  
59                  break;  
60              case 'r':  
61                  tip_niza = 1;  
62                  break;  
63              case 'o':
```

```

        tip_niza = 2;
64      break;
      default:
66        printf("Pogresna opcija -%c\n", argv[i][1]);
        return 1;
68        break;
    }
70 }
/* Ako je u pitanju argument, onda je to duzina niza koji treba
72 da se sortira */
else {
74     dimenzija = atoi(argv[i]);
    if (dimenzija <= 0 || dimenzija > MAX) {
76         fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
        exit(EXIT_FAILURE);
78     }
    }
80 }

82 /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
niza dobijenom iz komandne linije. srandom() funkcija
84 obezbedjuje novi seed za pozivanje random funkcije, i kako
generisani niz ne bi uvek bio isti seed je postavljen na tekuce
86 vreme u sekundama od Nove godine 1970. random()%100 daje brojeve
izmedju 0 i 99 */
88 srandom(time(NULL));
if (tip_niza == 0)
90     for (i = 0; i < dimenzija; i++)
        niz[i] = random();
92 else if (tip_niza == 1)
    for (i = 0; i < dimenzija; i++)
94         niz[i] = i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
else
96     for (i = 0; i < dimenzija; i++)
        niz[i] = i == 0 ? random() % 100 : niz[i - 1] - random() % 100;
98

/* Ispisivanje elemenata niza */
100 /******
Ovaj deo je iskomentaran jer sledeci ispis ne treba da se nadje
102 na standardnom izlazu. Njegova svrha je samo bila provera da li
je niz generisan u skladu sa opcijama komandne linije.

104
printf("Niz koji sortiramo je:\n");
106 for (i = 0; i < dimenzija; i++)
    printf("%d\n", niz[i]);
108 *****/

110

/* Sortiranje niza na odgovarajuci nacin */
112 if (tip_sortiranja == 0)
    selectionsort(niz, dimenzija);
114 else if (tip_sortiranja == 1)

```

3 Algoritmi pretrage i sortiranja

```
        insertionsort(niz, dimenzija);
116 else if (tip_sortiranja == 2)
        bubblesort(niz, dimenzija);
118 else if (tip_sortiranja == 3)
        shellsort(niz, dimenzija);
120 else if (tip_sortiranja == 4)
        mergesort(niz, 0, dimenzija - 1);
122 else
        quicksort(niz, 0, dimenzija - 1);

124
/* Ispis elemenata niza */
126 /*****
    Ovaj deo je iskomentarisan jer vreme potrebno za njegovo
128    izvršavanje ne bi trebalo da bude ukljuceno u vreme izmereno
    programom time. Takodje, kako je svrha ovog programa da prikaze
130    vremena razlicitih algoritama sortiranja, dimenzije nizova ce
    biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
132    od toliko elemenata. Ovaj deo je koriscen u razvoju programa
    zarad testiranja korektnosti.

134    printf("Sortiran niz je:\n");
136    for (i = 0; i < dimenzija; i++)
        printf("%d\n", niz[i]);
138    *****/
140 return 0;
}
```

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
        od njih */
14    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
31 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 argumenata 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 */
fout = fopen("ceo-tok.txt", "w");
34 if (fout == NULL) {
    fprintf(stderr,
36        "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
    exit(EXIT_FAILURE);
38 }

40
/* Citanje narednog studenta iz prve datoteke */
42 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
    kraj1 = 1;

44
/* Citanje narednog studenta iz druge datoteke */
46 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
    kraj2 = 1;

48
/* Sve dok nije dostignut kraj neke datoteke */
50 while (!kraj1 && !kraj2) {
    if (strcmp(ime1, ime2) < 0) {
52        /* Ime i prezime iz prve datoteke je leksikografski ranije, i
           biva upisano u izlaznu datoteku */
54        fprintf(fout, "%s %s\n", ime1, prezime1);
        /* Citanje narednog studenta iz prve datoteke */
56        if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
            kraj1 = 1;
```

```

58     } 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
           druge datoteke, onda ima jos studenata u prvoj datoteci, koje
           treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.
           */
72     while (!kraj1) {
           fprintf(fout, "%s %s\n", ime1, prezime1);
74         if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
           kraj1 = 1;
76     }

78     /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
           datoteke, onda ima jos studenata u drugoj datoteci, koje treba
           prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
80     while (!kraj2) {
           fprintf(fout, "%s %s\n", ime2, prezime2);
82         if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
           kraj2 = 1;
84     }

86     /* Zatvaranje datoteka */
88     fclose(fin1);
           fclose(fin2);
           fclose(fout);
90

92     return 0;
}

```

Rešenje 3.18

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6  #define MAX_BR_TACAKA 128
7
8  /* Struktura koja reprezentuje koordinate tacke */
9  typedef struct Tacka {
10     int x;
11     int y;
12 } Tacka;

```

```
13  /* Funkcija racuna rastojanje zadate tacke od koordinatnog pocetka
15  (0,0) */
16  float rastojanje(Tacka A)
17  {
18      return sqrt(A.x * A.x + A.y * A.y);
19  }

21  /* Funkcija koja sortira niz tacaka po rastojanju od koordinatnog
22  pocetka */
23  void sortiraj_po_rastojanju(Tacka t[], int n)
24  {
25      int min, i, j;
26      Tacka tmp;

27      for (i = 0; i < n - 1; i++) {
28          min = i;
29          for (j = i + 1; j < n; j++) {
30              if (rastojanje(t[j]) < rastojanje(t[min])) {
31                  min = j;
32              }
33          }
34          if (min != i) {
35              tmp = t[i];
36              t[i] = t[min];
37              t[min] = tmp;
38          }
39      }
40  }

41  /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
42  void sortiraj_po_x(Tacka t[], int n)
43  {
44      int min, i, j;
45      Tacka tmp;

46      for (i = 0; i < n - 1; i++) {
47          min = i;
48          for (j = i + 1; j < n; j++) {
49              if (abs(t[j].x) < abs(t[min].x)) {
50                  min = j;
51              }
52          }
53          if (min != i) {
54              tmp = t[i];
55              t[i] = t[min];
56              t[min] = tmp;
57          }
58      }
59  }

60  /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
61  void sortiraj_po_y(Tacka t[], int n)
62  {
63      int min, i, j;
64      Tacka tmp;
```



```
65 void sortiraj_po_y(Tacka t[], int n)
66 {
67     int min, i, j;
68     Tacka tmp;
69
70     for (i = 0; i < n - 1; i++) {
71         min = i;
72         for (j = i + 1; j < n; j++) {
73             if (abs(t[j].y) < abs(t[min].y)) {
74                 min = j;
75             }
76         }
77         if (min != i) {
78             tmp = t[i];
79             t[i] = t[min];
80             t[min] = tmp;
81         }
82     }
83 }
84
85 int main(int argc, char *argv[])
86 {
87     FILE *ulaz;
88     FILE *izlaz;
89     Tacka tacke[MAX_BR_TACAKA];
90     int i, n;
91
92     /* Proveravanje broja argumenata komandne linije: ocekuje se ime
93        izvrsnog programa, opcija, ime ulazne datoteke i ime izlazne
94        datoteke, tj. 4 argumenta */
95     if (argc != 4) {
96         fprintf(stderr,
97             "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
98         return 0;
99     }
100
101     /* Otvaranje datoteke u kojoj su zadate tacke */
102     ulaz = fopen(argv[2], "r");
103     if (ulaz == NULL) {
104         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
105             argv[2]);
106         return 0;
107     }
108
109     /* Otvaranje datoteke u koju treba upisati rezultat */
110     izlaz = fopen(argv[3], "w");
111     if (izlaz == NULL) {
112         fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
113             argv[3]);
114         return 0;
115     }
116 }
```

3 Algoritmi pretrage i sortiranja

```
117  /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
119      koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
      brojacem i. */
      i = 0;
121  while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
      i++;
123  }

125  /* Ukupan broj procitanih tacaka */
      n = i;
127
129  /* 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
131      sortiranja */
      switch (argv[1][1]) {
133      case 'x':
          /* Sortiranje po vrednosti x koordinate */
135          sortiraj_po_x(tacke, n);
          break;
137      case 'y':
          /* Sortiranje po vrednosti y koordinate */
139          sortiraj_po_y(tacke, n);
          break;
141      case 'o':
          /* Sortiranje po udaljenosti od koordinatnog pocetka */
143          sortiraj_po_rastojanju(tacke, n);
          break;
145      }

147  /* Upisivanje dobijenog niza u izlaznu datoteku */
      for (i = 0; i < n; i++) {
149          fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
      }

151
153  /* Zatvaranje otvorenih datoteka */
      fclose(ulaz);
      fclose(izlaz);
155
157  return 0;
}
```

Rešenje 3.19

```
#include <stdio.h>
2 #include <string.h>
#include <stdlib.h>
4
#define MAX 1000
6 #define MAX_DUZINA 16
```

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

14 /* Funkcija sortira niz gradjana rastuce po imenima */
void sort_ime(Gradjanin a[], int n)
16 {
    int i, j;
18     int min;
    Gradjanin pom;

20     for (i = 0; i < n - 1; i++) {
22         /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
            najmanji od elemenata a[i].ime,...,a[n-1].ime. */
24         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 */
63 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
64 {
65     int i;
66     for (i = 0; i < n; i++)
67         if (strcmp(a[i].ime, x->ime) == 0
68             && strcmp(a[i].prezime, x->prezime) == 0)
69             return i;
70     return -1;
71 }

72

74 int main()
75 {
76     Gradjanin spisak1[MAX], spisak2[MAX];
77     int isti_rbr = 0;
78     int i, n;
79     FILE *fp = NULL;

80     /* Otvaranje datoteke */
81     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
82         fprintf(stderr,
83             "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
84         exit(EXIT_FAILURE);
85     }

86     /* Citanje sadrzaja */
87     for (i = 0;
88         fscanf(fp, "%s %s", spisak1[i].ime,
89             spisak1[i].prezime) != EOF; i++)
90         spisak2[i] = spisak1[i];
91     n = i;

92     /* Zatvaranje datoteke */
93     fclose(fp);

94     sort_ime(spisak1, n);

95     /******
96     Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
97     sortiranih nizova. Koriscen je samo u fazi testiranja programa.
98     *****/

99     printf("Biracki spisak [uredjen prema imenima]:\n");
100     for(i=0; i<n; i++)
101         printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
102     /******

103     sort_prezime(spisak2, n);

104     /******
```

```

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 */
    printf("%d\n", isti_rbr);
134
136    exit(EXIT_SUCCESS);
}

```

Rešenje 3.21

```

1  #include <stdio.h>
      #include <string.h>
3  #include <ctype.h>

5  #define MAX_BR_RECII 128
      #define MAX_DUZINA_RECII 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_REC], int n)
33 {
    int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
35    duzina_j, duzina_min;
    char tmp[MAX_DUZINA_REC];
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]);

                if (duzina_j < duzina_min)
51                    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,
```

```

75     a drugi maksimalnu duzinu pojedinačne reci */
char reci[MAX_BR_RECII][MAX_DUZINA_RECII];

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 može pročitati sledeća rec */
while (fscanf(ulaz, "%s", reci[i]) != EOF) {
87     /* Proverava se da li učitani maksimalan broj reci, i ako jeste,
        prekida se učitavanje */
89     if (i == MAX_BR_RECII)
        break;
91     /* Priprema broja za narednu iteraciju */
    i++;
93 }

95 /* n je dužina niza reci i predstavlja poslednju vrednost
    korišćenog broja */
97 n = i;
/* Poziv funkcije za sortiranje reci */
99 sortiraj_reci(reci, n);

101 /* Ispis sortiranih niza reci */
for (i = 0; i < n; i++) {
103     printf("%s ", reci[i]);
}
105 printf("\n");

107 /* Zatvaranje datoteke */
fclose(ulaz);

109 return 0;
111 }

```

Rešenje 3.22

```

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

4

#define MAX_ARTIKALA 100000

6

/* Struktura koja predstavlja jedan artikal */
8 typedef struct art {
    long kod;
10     char naziv[20];

```

3 Algoritmi pretrage i sortiranja

```
char proizvodjac[20];
12 float cena;
} Artikal;

14
/* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
16 trazenim bar kodom */
int binarna_pretraga(Artikal a[], int n, long x)
18 {
    int levi = 0;
    20 int desni = n - 1;

    22 /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {
        24 /* Racuna se sredisnji indeks */
        int srednji = (levi + desni) / 2;
        26 /* 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 */
    return -1;
    40 }

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 }
```



```

64 int main()
65 {
66     Artikl asortiman[MAX_ARTIKALA];
67     long kod;
68     int i, n;
69     float racun;
70
71     FILE *fp = NULL;
72
73     /* Otvaranje datoteke */
74     if ((fp = fopen("artikli.txt", "r")) == NULL) {
75         fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
76         exit(EXIT_FAILURE);
77     }
78
79     /* Ucitavanje artikala */
80     i = 0;
81     while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
82         asortiman[i].naziv, asortiman[i].proizvodjac,
83         &asortiman[i].cena) == 4)
84         i++;
85
86     /* Zatvaranje datoteke */
87     fclose(fp);
88
89     n = i;
90
91     /* Sortira se celokupan asortiman prodavnice prema kodovima jer ce
92     pri kucanju racuna prodavac unositi kod artikla. Prilikom
93     kucanja svakog racuna pretrazuje se asortiman, da bi se utvrdila
94     cena artikla. Kucanje racuna obuhvata vise pretraga asortimana i
95     cilj je da ta operacija bude sto efikasnija. Zato se koristi
96     algoritam binarne pretrage prilikom pretrazivanja po kodu
97     artikla. Iz tog razloga, potrebno je da asortiman bude sortiran
98     po kodovima i to ce biti uradjeno primenom selection sort
99     algoritma. Sortiranje se vrši samo jednom na pocetku, ali se
100     zato posle artikli mogu brzo pretrazivati prilikom kucanja
101     proizvoljno puno racuna. Vreme koje se utrosi na sortiranje na
102     pocetku izvorsavanja programa, kasnije se isplati jer se za
103     brojna trazjenja artikla umesto linearne moze koristiti
104     efikasnija binarna pretraga. */
105     selection_sort(asortiman, n);
106
107     /* Ispis stanja u prodavnici */
108     printf
109     ( "Asortiman:\nKOD          Naziv artikla      Ime
110     proizvodjaca      Cena\n");
111     for (i = 0; i < n; i++)
112         printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
113             asortiman[i].naziv, asortiman[i].proizvodjac,
114             asortiman[i].cena);

```

3 Algoritmi pretrage i sortiranja

```
114 kod = 0;
115 while (1) {
116     printf("-----\n");
117     printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
118     printf("- Za nov racun unesite kod artikla!\n\n");
119     /* Unos bar koda provog artikla sledeceg kupca */
120     if (scanf("%ld", &kod) == EOF)
121         break;
122     /* Trenutni racun novog kupca */
123     racun = 0;
124     /* Za sve artikle trenutnog kupca */
125     while (1) {
126         /* Vrsi se njihov pronalazak u nizu */
127         if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
128             printf("\tGRESKA: Ne postoji proizvod sa trazanim kodom!\n");
129         } else {
130             printf("\tTrazili ste:\t%s %s %12.2f\n",
131                 asortiman[i].naziv, asortiman[i].proizvodjac,
132                 asortiman[i].cena);
133             /* I dodavanje na ukupan racun */
134             racun += asortiman[i].cena;
135         }
136         /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
137             nema vise artikla */
138         printf("Unesite kod artikla [ili 0 za prekid]: \t");
139         scanf("%ld", &kod);
140         if (kod == 0)
141             break;
142     }
143     /* Stampanje ukupnog racuna trenutnog kupca */
144     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
145 }
146 printf("Kraj rada kase!\n");
147
148 exit(EXIT_SUCCESS);
149
150 }
```

Rešenje 3.23

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 500
6
7 /* Struktura sa svim informacijama o pojedinacnom studentu */
8 typedef struct {
9     char ime[20];
10    char prezime[25];
11    int prisustvo;
```

```
13     int zadaci;
14 } Student;

15 /* Funkcija za sortiranje niza struktura po prezimenu leksikografski
16    rastuce */
17 void sort_ime_leksikografski(Student niz[], int n)
18 {
19     int i, j;
20     int min;
21     Student pom;

22     for (i = 0; i < n - 1; i++) {
23         min = i;
24         for (j = i + 1; j < n; j++)
25             if (strcmp(niz[j].ime, niz[min].ime) < 0)
26                 min = j;

27         if (min != i) {
28             pom = niz[min];
29             niz[min] = niz[i];
30             niz[i] = pom;
31         }
32     }
33 }

34 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
35    zadataka opadajuće, a ukoliko neki studenti imaju isti broj
36    uradjenih zadataka sortiraju se po dužini imena rastuce. */
37 void sort_zadatke_pa_imena(Student niz[], int n)
38 {
39     int i, j;
40     int max;
41     Student pom;

42     for (i = 0; i < n - 1; i++) {
43         max = i;
44         for (j = i + 1; j < n; j++)
45             if (niz[j].zadaci > niz[max].zadaci)
46                 max = j;
47             else if (niz[j].zadaci == niz[max].zadaci
48                     && strlen(niz[j].ime) < strlen(niz[max].ime))
49                 max = j;

50         if (max != i) {
51             pom = niz[max];
52             niz[max] = niz[i];
53             niz[i] = pom;
54         }
55     }
56 }

57 /* Funkcija za sortiranje niza struktura po broju casova na kojima
58    su bili opadajuće. Ukoliko neki studenti imaju isti broj casova,
59    sortiraju se opadajuće 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_prezimana(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;
            else if (niz[j].prisustvo == niz[max].prisustvo
79                     && 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
97     FILE *fp = NULL;

    /* Otvaranje datoteke za citanje */
99     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke aktivnost.txt.\n");
101         exit(EXIT_FAILURE);
    }

103
    /* Ucitavanje sadrzaja */
105     for (i = 0;
        fscanf(fp, "%s%s%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_lexikografski(praktikum, br_studenata);
115     /* Otvaranje datoteke za pisanje */
```

```
117     if ((fp = fopen("dat1.txt", "w")) == NULL) {
118         fprintf(stderr, "Neuspesno otvaranje datoteke dat1.txt.\n");
119         exit(EXIT_FAILURE);
120     }
121     /* Upis niza u datoteku */
122     fprintf
123     (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
124     for (i = 0; i < br_studenata; i++)
125         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
126                 praktikum[i].prezime, praktikum[i].prisustvo,
127                 praktikum[i].zadaci);
128     /* Zatvaranje datoteke */
129     fclose(fp);
130
131     /* Kreiranje drugog spiska studenata po drugom kriterijumu */
132     sort_zadatke_pa_imena(praktikum, br_studenata);
133     /* Otvaranje datoteke za pisanje */
134     if ((fp = fopen("dat2.txt", "w")) == NULL) {
135         fprintf(stderr, "Neuspesno otvaranje datoteke dat2.txt.\n");
136         exit(EXIT_FAILURE);
137     }
138     /* Upis niza u datoteku */
139     fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
140     fprintf(fp, "pa po duzini imena rastuce:\n");
141     for (i = 0; i < br_studenata; i++)
142         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
143                 praktikum[i].prezime, praktikum[i].prisustvo,
144                 praktikum[i].zadaci);
145     /* Zatvaranje datoteke */
146     fclose(fp);
147
148     /* Kreiranje treceg spiska studenata po trecem kriterijumu */
149     sort_prisustvo_pa_zadatke_pa_prezimenama(praktikum, br_studenata);
150     /* Otvaranje datoteke za pisanje */
151     if ((fp = fopen("dat3.txt", "w")) == NULL) {
152         fprintf(stderr, "Neuspesno otvaranje datoteke dat3.txt.\n");
153         exit(EXIT_FAILURE);
154     }
155     /* Upis niza u datoteku */
156     fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
157     fprintf(fp, "pa po broju zadataka,\n");
158     fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
159     for (i = 0; i < br_studenata; i++)
160         fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
161                 praktikum[i].prezime, praktikum[i].prisustvo,
162                 praktikum[i].zadaci);
163     /* Zatvaranje datoteke */
164     fclose(fp);
165
166     return 0;
167 }
```

Rešenje 3.24

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

4
#define KORAK 10

6
/* Struktura koja opisuje jednu pesmu */
8 typedef struct {
    char *izvodjac;
10    char *naslov;
    int broj_gledanja;
12 } Pesma;

14 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
    rad qsort funkcije) */
16 int uporedi_gledanost(const void *pp1, const void *pp2)
{
18     Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

20     return p2->broj_gledanja - p1->broj_gledanja;
22 }

24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
    funkcije) */
26 int uporedi_naslave(const void *pp1, const void *pp2)
{
28     Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

30     return strcmp(p1->naslov, p2->naslov);
32 }

34 /* Funkcija za uporedjivanje pesama po izvodjaku (potrebna za rad
    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
{
38     Pesma *p1 = (Pesma *) pp1;
    Pesma *p2 = (Pesma *) pp2;

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

44 int main(int argc, char *argv[])
{
46     FILE *ulaz;
    Pesma *pesme;                                /* Pokazivac na deo memorije za
                                                    cuvanje pesama */
48
    int alocirano_za_pesme;                       /* Broj mesta alociranih za pesme */
50    int i;                                        /* Redni broj pesme cije se
```

```
52     informacije citaju */
53     int n; /* Ukupan broj pesama */
54     int j, k;
55     char c;
56     int alocirano; /* Broj mesta alociranih za propratne
57                    informacije o pesmama */
58     int broj_gledanja;
59
60     /* Priprema datoteke za citanje */
61     ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
62     if (ulaz == NULL) {
63         printf("Greska pri otvaranju ulazne datoteke!\n");
64         return 0;
65     }
66
67     /* Citanje informacija o pesmama */
68     pesme = NULL;
69     alocirano_za_pesme = 0;
70     i = 0;
71
72     while (1) {
73
74         /* Proverava da li je dostignut kraj datoteke */
75         c = fgetc(ulaz);
76         if (c == EOF) {
77             /* Nema vise sadrzaja za citanje */
78             break;
79         } else {
80             /* Inace, vracamo procitani karakter nazad */
81             ungetc(c, ulaz);
82         }
83
84         /* Provera da li postoji dovoljno memorije za citanje nove pesme
85         */
86         if (alocirano_za_pesme == i) {
87
88             /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
89             novih KORAK mesta */
90             alocirano_za_pesme += KORAK;
91             pesme =
92                 (Pesma *) realloc(pesme,
93                                   alocirano_za_pesme * sizeof(Pesma));
94
95             /* Proverava da li je nova memorija uspesno realocirana */
96             if (pesme == NULL) {
97                 /* Ako nije ispisuje se obavestenje */
98                 printf("Problem sa alokacijom memorije!\n");
99                 /* I oslobadja sva memorija zauzeta do ovog koraka */
100                 for (k = 0; k < i; k++) {
101                     free(pesme[k].izvodjac);
102                     free(pesme[k].naslov);
103                 }
104             }
105         }
106     }
```

3 Algoritmi pretrage i sortiranja

```
102         free(pesme);
103         return 0;
104     }
105 }
106
107 /* Ako jeste, nastavlja se sa citanjem pesama ... */
108 /* Cita se ime izvodjaca */
109 j = 0;                                /* Pozicija na koju treba smestiti
110                                         procitani karakter */
111
112 alocirano = 0;                         /* Broj alociranih mesta */
113 pesme[i].izvodjac = NULL;             /* Memorija za smestanje procitanih
114                                         karaktera */
115
116 /* Sve do prve beline u liniji (beline koja se nalazi nakon imena
117   izvodjaca) citaju se karakteri iz datoteke */
118 while ((c = fgetc(ulaz)) != ' ') {
119     /* Proverav da li postoji dovoljno memorije za smestanje
120       procitanog karaktera */
121     if (j == alocirano) {
122
123         /* Ako ne, ako je potrosena sva alocirana memorija, alocira
124           se novih KORAK mesta */
125         alocirano += KORAK;
126         pesme[i].izvodjac =
127             (char *) realloc(pesme[i].izvodjac,
128                             alocirano * sizeof(char));
129
130         /* Provera da li je nova alokacija uspesna */
131         if (pesme[i].izvodjac == NULL) {
132             /* Ako nije oslobadja se sva memorija zauzeta do ovog
133               koraka */
134             for (k = 0; k < i; k++) {
135                 free(pesme[k].izvodjac);
136                 free(pesme[k].naslov);
137             }
138             free(pesme);
139             /* I prekida sa izvorsavanjem programa */
140             return 0;
141         }
142     }
143     /* Ako postoji dovoljno memorije, smestamo procitani karakter
144     */
145     pesme[i].izvodjac[j] = c;
146     j++;
147     /* I nastavlja se sa citanjem */
148 }
149
150 /* Upis terminirajuće nule na kraj reci */
151 pesme[i].izvodjac[j] = '\0';
152
153 /* Preskace se karakter - */
```



```
154     fgetc(ulaz);
156     /* Preskace se razmak */
156     fgetc(ulaz);
158     /* Cita se naslov pesme */
158     j = 0;                                /* Pozicija na koju treba smestiti
160                                           procitani karakter */
160     alocirano = 0;                        /* Broj alociranih mesta */
162     pesme[i].naslov = NULL;              /* Memoriya za smestanje procitanih
162                                           karaktera */
164
164     /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
166     karakteri iz datoteke */
166     while ((c = fgetc(ulaz)) != ',') {
168         /* Provera da li postoji dovoljno memorije za smestanje
168         procitanog karaktera */
170         if (j == alocirano) {
172             /* Ako ne, ako je potrosena sva alocirana memorija, alocira
172             se novih KORAK mesta */
174             alocirano += KORAK;
174             pesme[i].naslov =
174                 (char *) realloc(pesme[i].naslov,
176                                 alocirano * sizeof(char));
178
178             /* Provera da li je nova alokacija uspesna */
180             if (pesme[i].naslov == NULL) {
182                 /* Ako nije, oslobadja se sva memorija zauzeta do ovog
182                 koraka */
184                 for (k = 0; k < i; k++) {
184                     free(pesme[k].izvodjac);
184                     free(pesme[k].naslov);
186                 }
186                 free(pesme[i].izvodjac);
186                 free(pesme);
188
188                 /* I prekida izvršavanje programa */
190                 return 0;
192             }
192         }
192         /* Ako postoji dovoljno memorije, smesta se procitani karakter
192         */
194         pesme[i].naslov[j] = c;
194         j++;
196         /* I nastavlja dalje sa citanjem */
196     }
198     /* Upisuje se terminirajuca nula na kraj reci */
198     pesme[i].naslov[j] = '\0';
200
200     /* Preskace se razmak */
202     fgetc(ulaz);
```

3 Algoritmi pretrage i sortiranja

```
204      /* Cita se broj gledanja */
      broj_gledanja = 0;

206

      /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
208      datoteke */
      while ((c = fgetc(ulaz)) != '\n') {
210          broj_gledanja = broj_gledanja * 10 + (c - '0');
      }
      pesme[i].broj_gledanja = broj_gledanja;

212

      /* Prelazi se na citanje sledece pesme */
      i++;
216  }

218  /* Informacija o broju procitanih pesama */
      n = i;
220  /* Zatvaranje nepotrebne datoteke */
      fclose(ulaz);

222

      /* Analiza argumenta komandne linije */
224  if (argc == 1) {
      /* Nema dodatnih opcija => sortiranje po broju gledanja */
226      qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
  } else {
228      if (argc == 2 && strcmp(argv[1], "-n") == 0) {
          /* Sortiranje po naslovu */
230          qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
      } else {
232          if (argc == 2 && strcmp(argv[1], "-i") == 0) {
              /* Sortiranje po izvodjaku */
234              qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
          } else {
236              printf("Nedozvoljeni argumenti!\n");
              free(pesme);
238              return 0;
          }
      }
240  }

242

      /* Ispis rezultata */
244  for (i = 0; i < n; i++) {
      printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
246          pesme[i].broj_gledanja);
  }

248

      /* Oslobadjanje memorije */
250  for (i = 0; i < n; i++) {
      free(pesme[i].izvodjac);
252      free(pesme[i].naslov);
  }
254  free(pesme);
```

```

256     return 0;
    }

```

Rešenje 3.28

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <search.h>
5
6  #define MAX 100
7
8  /* Funkcija poredjenja dva cela broja */
9  int compare_int(const void *a, const void *b)
10 {
11     /* Potrebno je konvertovati void pokazivace u int pokazivace koji
12        se zatim dereferenciraju. Vraca se razlika dobijenih int-ova. */
13
14     /* Zbog moguceg prekoracenja opsega celih brojeva, sledece
15        oduzimanje treba izbegavati return *((int *)a) - *((int *)b); */
16
17     int b1 = *((int *) a);
18     int b2 = *((int *) b);
19
20     if (b1 > b2)
21         return 1;
22     else if (b1 < b2)
23         return -1;
24     else
25         return 0;
26 }
27
28 int compare_int_desc(const void *a, const void *b)
29 {
30     /* Za obrnuti poredak treba samo oduzimati a od b */
31     /* return *((int *)b) - *((int *)a); */
32
33     /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
34        funkcija */
35     return -compare_int(a, b);
36 }
37
38 int main()
39 {
40     size_t n;
41     int i, x;
42     int a[MAX], *p = NULL;
43
44     /* Unos dimenzije */
45     printf("Uneti dimenziju niza: ");
46     scanf("%ld", &n);

```

3 Algoritmi pretrage i sortiranja

```
47  if (n > MAX)
    n = MAX;
49
/* Unos elementa niza */
51  printf("Uneti elemente niza:\n");
    for (i = 0; i < n; i++)
53      scanf("%d", &a[i]);
55
/* Sortiranje niza celih brojeva */
    qsort(a, n, sizeof(int), &compare_int);
57
/* Prikaz sortiranog niz */
59  printf("Sortirani niz u rastucem poretku:\n");
    for (i = 0; i < n; i++)
61      printf("%d ", a[i]);
    putchar('\n');
63
/* Pretrazivanje niza */
65  /* Vrednost koja ce biti trazena u nizu */
    printf("Uneti element koji se trazi u nizu: ");
67      scanf("%d", &x);
69
/* Binarna pretraga */
    printf("Binarna pretraga: \n");
71    p = bsearch(&x, a, n, sizeof(int), &compare_int);
    if (p == NULL)
73        printf("Elementa nema u nizu!\n");
    else
75        printf("Element je nadjen na poziciji %ld\n", p - a);
77
/* Linearna pretraga */
    printf("Linearna pretraga (lfind): \n");
79    p = lfind(&x, a, &n, sizeof(int), &compare_int);
    if (p == NULL)
81        printf("Elementa nema u nizu!\n");
    else
83        printf("Element je nadjen na poziciji %ld\n", p - a);
85    return 0;
}
```

Rešenje 3.29

```
1  #include <stdio.h>
   #include <stdlib.h>
3  #include <math.h>
   #include <search.h>
5
   #define MAX 100
7
/* Funkcija racuna broj delilaca broja x */
```

```
9  int no_of_deviders(int x)
10 {
11     int i;
12     int br;
13
14     /* Negativni brojevi imaju isti broj delilaca kao i pozitivni */
15     if (x < 0)
16         x = -x;
17     if (x == 0)
18         return 0;
19     if (x == 1)
20         return 1;
21     /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
22     br = 2;
23     for (i = 2; i < sqrt(x); i++)
24         if (x % i == 0)
25             /* Ako i deli x onda su delioci: i, x/i */
26             br += 2;
27     /* Ako je broj x bas kvadrat, onda se iz petlje izaslo kada je
28        promenljiva i bila bas jednaka korenu od x, i tada broj x ima
29        jos jednog delioca */
30     if (i * i == x)
31         br++;
32
33     return br;
34 }
35
36 /* Funkcija poredjenja dva cela broja po broju delilaca */
37 int compare_no_deviders(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = no_of_deviders(ak);
42     int n_d_b = no_of_deviders(bk);
43
44     if (n_d_a > n_d_b)
45         return 1;
46     else if (n_d_a < n_d_b)
47         return -1;
48     else
49         return 0;
50 }
51
52 int main()
53 {
54     size_t n;
55     int i;
56     int a[MAX];
57
58     /* Unos dimenzije */
59     printf("Uneti dimenziju niza: ");
60     scanf("%ld", &n);
```

3 Algoritmi pretrage i sortiranja

```
61     if (n > MAX)
62         n = MAX;
63
64     /* Unos elementa niza */
65     printf("Uneti elemente niza:\n");
66     for (i = 0; i < n; i++)
67         scanf("%d", &a[i]);
68
69     /* Sortiranje niza celih brojeva prema broju delilaca */
70     qsort(a, n, sizeof(int), &compare_no_deviders);
71
72     /* Prikaz sortiranog niza */
73     printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
74     for (i = 0; i < n; i++)
75         printf("%d ", a[i]);
76     putchar('\n');
77
78     return 0;
79 }
```

Rešenje 3.30

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>
5
6  #define MAX_NISKI 1000
7  #define MAX_DUZINA 30
8
9  /*****
10   Niz nizova karaktera ovog potpisa
11   char niske[3][4];
12   se moze graficki predstaviti ovako:
13   -----
14   | a | b | c | \0 || d | e | \0|   || f | g | h | \0||
15   -----
16   Dakle kao tri reci (abc, de, fgh), nadovezane jedna na drugu. Za
17   svaku je rezervisano po 4 karaktera ukljucujuci \0. Druga rec sa
18   nalazi na adresi koja je za 4 veka od prve reci, a za 4 manja od
19   adrese na kojoj se nalazi treca rec. Adresa i-te reci je niske[i]
20   i ona je tipa char*.
21
22   Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
23   koji trebaju biti uporedjeni, (npr. pri porecenju prve i poslednje
24   reci, pokazivac a ce pokazivati na slovo 'a', a pokazivac b na
25   slovo 'f') treba ih kastovati na char*, i pozvati funkciju strcmp
26   nad njima.
27   *****/
28   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
41 int main()
{
    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 */
    fclose(fp);
    n = i;

61     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
    prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
    niske po duzini */
63     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);

65     printf("Leksikografski sortirane niske:\n");
    for (i = 0; i < n; i++)
67         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

```

3 Algoritmi pretrage i sortiranja

```
83     if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
               p, (p - (char *) niske) / MAX_DUZINA);
85     else
        printf("Niska nije pronadjena u nizu\n");
87
    /* Linearna pretraga */
89     p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
               &poredi_leksikografski);
91
    if (p != NULL)
93        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
               p, (p - (char *) niske) / MAX_DUZINA);
95     else
        printf("Niska nije pronadjena u nizu\n");
97
    /* Sortiranje po duzini */
99     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

101    printf("Niske sortirane po duzini:\n");
    for (i = 0; i < n; i++)
103        printf("%s ", niske[i]);
    printf("\n");
105
    exit(EXIT_SUCCESS);
107 }
```

Rešenje 3.31

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <search.h>
5
6  #define MAX_NISKI 1000
7  #define MAX_DUZINA 30
8
9  /******
10   Niz pokazivaca na karaktere ovog potpisa
11   char *niske[3];
12   posle alokacije u main-u se moze graficki predstaviti ovako:
13   ----
14   | X | -----> | a | b | c | \0|
15   ----
16   | Y | -----> | d | e | \0|
17   ----
18   | Z | -----> | f | g | h | \0|
19   ----
20   Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti
21   njegov element pokazivac koji pokazuje na alocirane karaktere i-te
   reci. Njegov tip je char*.
```



```

23     Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata
25     koji trebaju biti uporedjeni (recimo adresu od X i adresu od Z), i
27     kako su X i Z tipa char*, onda a i b su tipa char**, pa se tako
29     moraju i kastovati. Da bi se leksikografski uporedili elementi X i
31     Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se
32     u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,
33     kastovani na odgovarajuci tip.
34     *****/
35     int poredi_leksikografski(const void *a, const void *b)
36     {
37         return strcmp(*(char **) a, *(char **) b);
38     }
39
40     /* Funkcija slicna prethodnoj, osim sto elemente ne uporedjuje
41     leksikografski, vec po duzini */
42     int poredi_duzine(const void *a, const void *b)
43     {
44         return strlen(*(char **) a) - strlen(*(char **) b);
45     }
46
47     /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
48     element u nizu sa kojim se poredi, pa njega treba kastovati na
49     char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
50     ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
51     main funkciji je to x, koji je tipa char*, tako da pokazivac a
52     ovde samo treba kastovati i ne dereferencirati. */
53     int poredi_leksikografski_b(const void *a, const void *b)
54     {
55         return strcmp((char *) a, *(char **) b);
56     }
57
58     int main()
59     {
60         int i;
61         size_t n;
62         FILE *fp = NULL;
63         char *niske[MAX_NISKI];
64         char **p = NULL;
65         char x[MAX_DUZINA];
66
67         /* Otvaranje datoteke */
68         if ((fp = fopen("niske.txt", "r")) == NULL) {
69             fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
70             exit(EXIT_FAILURE);
71         }
72
73         /* Citanje sadrzaja datoteke */
74         i = 0;
75         while (fscanf(fp, "%s", x) != EOF) {
76             /* Alociranje dovoljne memorije za i-tu nisku */
77             if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {

```

3 Algoritmi pretrage i sortiranja

```
75     fprintf(stderr, "Greska pri alociranju niske\n");
76     exit(EXIT_FAILURE);
77 }
78 /* Kopiranje procitane niske na svoje mesto */
79 strcpy(niske[i], x);
80 i++;
81 }
82
83 /* Zatvaranje datoteke */
84 fclose(fp);
85 n = i;
86
87 /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
88    prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
89    niske po duzini */
90 qsort(niske, n, sizeof(char *), &poredi_leksikografski);
91
92 printf("Leksikografski sortirane niske:\n");
93 for (i = 0; i < n; i++)
94     printf("%s ", niske[i]);
95 printf("\n");
96
97 /* Unos trazene niske */
98 printf("Uneti trazenu nisku: ");
99 scanf("%s", x);
100
101 /* Binarna pretraga */
102 p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografski_b);
103 if (p != NULL)
104     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
105           *p, p - niske);
106 else
107     printf("Niska nije pronadjena u nizu\n");
108
109 /* Linearna pretraga */
110 p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografski_b);
111 if (p != NULL)
112     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
113           *p, p - niske);
114 else
115     printf("Niska nije pronadjena u nizu\n");
116
117 /* Sortiramo po duzini */
118 qsort(niske, n, sizeof(char *), &poredi_duzine);
119
120 printf("Niske sortirane po duzini:\n");
121 for (i = 0; i < n; i++)
122     printf("%s ", niske[i]);
123 printf("\n");
124
125 /* Oslobadjanje zauzete memorije */
126 for (i = 0; i < n; i++)
```

```

127     free(niske[i]);
129     exit(EXIT_SUCCESS);
}

```

Rešenje 3.32

```

#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 poredil(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. */
46 int poredi3(const void *a, const void *b)
{
48     char *prezime = (char *) a;
    Student *s = (Student *) b;
50     return strcmp(prezime, s->prezime);
}

52
int main(int argc, char *argv[])
54 {
    Student kolokvijum[MAX];
56     int i;
    size_t br_studenata = 0;
58     Student *nadjen = NULL;
    FILE *fp = NULL;
60     int bodovi;
    char prezime[21];
62

    /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
64     informacija korisniku kako se program koristi i prekida se
    izvršavanje. */
66     if (argc < 2) {
        fprintf(stderr,
68             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
                argv[0]);
70         exit(EXIT_FAILURE);
    }
72

    /* Otvaranje datoteke */
74     if ((fp = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Neuspješno otvaranje datoteke %s\n", argv[1]);
76         exit(EXIT_FAILURE);
    }
78

    /* Ucitavanje sadržaja */
80     for (i = 0;
        fscanf(fp, "%s%s%d", kolokvijum[i].ime,
82             kolokvijum[i].prezime,
                &kolokvijum[i].bodovi) != EOF; i++);
84

    /* Zatvaranje datoteke */
86     fclose(fp);
    br_studenata = i;
88

    /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
90     studenata sa istim brojem bodova sortiranje vrši po prezimenu */
    qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
92

    printf("Studenti sortirani po broju poena opadajuće, ");
94     printf("pa po prezimenu rastuće:\n");
    for (i = 0; i < br_studenata; i++)
96         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
    return 0;
132 }

```

Rešenje 3.33

```

#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

```

3 Algoritmi pretrage i sortiranja

```
/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
14 int anagrami(char s[], char t[])
{
16     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
    if (strlen(s) != strlen(t))
18         return 0;

20     /* Sortiranje niski */
    qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
22     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);

24     /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
        suprotnom, nisu */
26     return !strcmp(s, t);
}

28
int main()
30 {
    char s[MAX], t[MAX];

32
    /* Unos niski */
34     printf("Unesite prvu nisku: ");
    scanf("%s", s);
36     printf("Unesite drugu nisku: ");
    scanf("%s", t);

38
    /* Ispituje se da li su niske anagrami */
40     if (anagrami(s, t))
        printf("jesu\n");
42     else
        printf("nisu\n");
44
    return 0;
46 }
```

Rešenje 3.34

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

5 #define MAX 10
  #define MAX_DUZINA 32
7
/* Funkcija porenjenja */
9 int uporedi_niske(const void *pa, const void *pb)
{
11     return strcmp((char *) pa, (char *) pb);
}

13
int main()
```

```

15 {
16     int i, n;
17     char S[MAX][MAX_DUZINA];

18     /* Unos broja niski */
19     printf("Unesite broj niski:");
20     scanf("%d", &n);

21     /* Unos niza niski */
22     printf("Unesite niske:\n");
23     for (i = 0; i < n; i++)
24         scanf("%s", S[i]);

25     /* Sortiranje niza niski */
26     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

27     /******
28      Ovaj deo je iskomentaran jer se u zadatku ne trazi ispis
29      sortiranih niski. Koriscen je samo u fazi testiranja programa.
30      *****/

31     printf("Sortirane niske su:\n");
32     for(i = 0; i < n; i++)
33         printf("%s ", S[i]);
34     /******

35     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
36        niza biti jedna do druge */
37     for (i = 0; i < n - 1; i++)
38         if (strcmp(S[i], S[i + 1]) == 0) {
39             printf("ima\n");
40             return 0;
41         }

42     printf("nema\n");
43     return 0;
44 }

```

Rešenje 3.35

```

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

4
5  #define MAX 21

6
7  /* Struktura koja predstavlja jednog studenta */
8  typedef struct student {
9      char nalog[8];
10     char ime[MAX];
11     char prezime[MAX];
12     int poeni;

```

```
13 } Student;

15 /* Funkcija poredi studente prema broju poena, rastuce */
16 int uporedi_poeni(const void *a, const void *b)
17 {
18     Student s = *(Student *) a;
19     Student t = *(Student *) b;
20     return s.poeni - t.poeni;
21 }

23 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i na
    kraju prema indeksu */
24 int uporedi_nalog(const void *a, const void *b)
25 {
26     Student s = *(Student *) a;
27     Student t = *(Student *) b;
28     /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
        broj indeksa */
29     int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
30     int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
31     char smer1 = s.nalog[1];
32     char smer2 = t.nalog[1];
33     int indeks1 =
34         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
35         s.nalog[6] - '0';
36     int indeks2 =
37         (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
38         t.nalog[6] - '0';
39     if (godina1 != godina2)
40         return godina1 - godina2;
41     else if (smer1 != smer2)
42         return smer1 - smer2;
43     else
44         return indeks1 - indeks2;
45 }

46 int uporedi_bsearch(const void *a, const void *b)
47 {
48     /* Nalog studenta koji se trazi */
49     char *nalog = (char *) a;
50     /* Kljuc pretrage */
51     Student s = *(Student *) b;
52
53     int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
54     int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
55     char smer1 = nalog[1];
56     char smer2 = s.nalog[1];
57     int indeks1 =
58         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
59         ;
60     int indeks2 =
61         (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 otvaranja 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

```

3 Algoritmi pretrage i sortiranja

```
115  /* Ako je prisutna opcija -p, vrsi se sortiranje po poenima */
    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    return 0;
}
```

Rešenje 3.37

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

4 /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
    standardnog ulaza */
6 void ucitaj_matricu(int **a, int n, int m)
{
8     printf("Unesite elemente matrice po vrstama:\n");
    int i, j;
```

```
10     for (i = 0; i < n; i++) {
12         for (j = 0; j < m; j++) {
13             scanf("%d", &a[i][j]);
14         }
15     }
16 }

18 /* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
20 int zbir_vrste(int **a, int v, int m)
21 {
22     int i, zbir = 0;

23     for (i = 0; i < m; i++) {
24         zbir += a[v][i];
25     }
26     return zbir;
27 }

30 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
   osnovu zbira koriscenjem selection sort algoritma */
32 void sortiraj_vrste(int **a, int n, int m)
33 {
34     int i, j, min;

35     for (i = 0; i < n - 1; i++) {
36         min = i;
37         for (j = i + 1; j < n; j++) {
38             if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
39                 min = j;
40             }
41         }
42         if (min != i) {
43             int *tmp;
44             tmp = a[i];
45             a[i] = a[min];
46             a[min] = tmp;
47         }
48     }
49 }

50 }

52 /* Funkcija koja ispisuje elemente matrice a dimenzije nxm na
   standardni izlaz */
54 void ispisi_matricu(int **a, int n, int m)
55 {
56     int i, j;

57     for (i = 0; i < n; i++) {
58         for (j = 0; j < m; j++) {
59             printf("%d ", a[i][j]);
60         }
61     }
62 }
```

3 Algoritmi pretrage i sortiranja

```
62     printf("\n");
63 }
64 }
65
66 /* Funkcija koja alokira memoriju za matricu dimenzija nxm */
67 int **alociraj_memoriju(int n, int m)
68 {
69     int i, j;
70     int **a;
71
72     a = (int **) malloc(n * sizeof(int *));
73     if (a == NULL) {
74         fprintf(stderr, "Problem sa alokacijom memorije!\n");
75         exit(EXIT_FAILURE);
76     }
77     /* Za svaku vrstu ponaosob */
78     for (i = 0; i < n; i++) {
79         /* Alocira se memorija */
80         a[i] = (int *) malloc(m * sizeof(int));
81         /* Proverava se da li je doslo do greske prilikom alokacije */
82         if (a[i] == NULL) {
83             /* Ako jeste, ispisuje se poruka */
84             fprintf(stderr, "Problem sa alokacijom memorije!\n");
85             /* I oslobadja memorija zauzeta do ovog koraka */
86             for (j = 0; j < i; j++) {
87                 free(a[j]);
88             }
89             free(a);
90             exit(EXIT_FAILURE);
91         }
92     }
93
94     return a;
95 }
96
97 /* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije nxm
98    */
99 void oslobodi_memoriju(int **a, int n, int m)
100 {
101     int i;
102     for (i = 0; i < n; i++) {
103         free(a[i]);
104     }
105     free(a);
106 }
107
108 int main(int argc, char *argv[])
109 {
110     int **a;
111     int n, m;
112
113     /* Unos dimenzija matrice */
```

```
114     printf("Unesite dimenzije matrice: ");
    scanf("%d %d", &n, &m);

116     /* Alokacija memorije */
    a = alociraj_memoriju(n, m);

118     /* Ucitavanje elementa matrice */
120     ucitaj_matricu(a, n, m);

122     /* Poziv funkcije koja sortira vrste matrice prema zbiru */
    sortiraj_vrste(a, n, m);

124     /* Ispis rezultujuce matrice */
126     printf("Sortirana matrica je:\n");
    ispisi_matricu(a, n, m);

128     /* Oslobadjanje memorije */
130     oslobodi_memoriju(a, n, m);

132     return 0;
}
```


Glava 4

Dinamičke strukture podataka

4.1 Liste

Zadatak 4.1 Napisati biblioteku za rad sa jednostruko povezanom listom čiji čvorovi sadrže cele brojeve.

- (a) Definirati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje mu polja i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor *pronadji_poslednji(Cvor * glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.
- (f) Napisati funkciju `Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

- (g) Napisati funkciju `void dodaj_iza(Cvor * tekuci, Cvor * novi)` koja uvezuje u postojeću listu čvor `novi` iza čvora `tekuci`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradama `[,]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.
- (l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.
- (m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.
- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. NAPOMENA: *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

Primer 1

```

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [3, 2]
14
Lista: [14, 3, 2]
5
Lista: [5, 14, 3, 2]
3
Lista: [3, 5, 14, 3, 2]
17
Lista: [17, 3, 5, 14, 3, 2]

Unosite broj koji se trazi: 5
Trazeni broj 5 je u listi!

```

Primer 2

```

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unosite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

Primer 1

```

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 14, 3]
3
Lista: [2, 3, 14, 3, 3]
17
Lista: [2, 3, 14, 3, 3, 17]
3
Lista: [2, 3, 14, 3, 3, 17, 3]

Unosite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]

```

Primer 2

```

INTERAKCIJA PROGRAMA:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unosite broj koji se brise: 3
Lista nakon brisanja: [23, 14, 35]

```

- (3) U glavnom programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. *NAPOMENA: Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

Primer 1

```
INTERAKCIJA PROGRAMA:
Unesite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se traži: 14
Trazeni broj 14 je u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 5, 14]
```

Primer 2

```
INTERAKCIJA PROGRAMA:
Unesite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [14, 23, 35]

Unesite broj koji se traži: 8
Broj 8 se ne nalazi u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [14, 23, 35]
```

Zadatak 4.2 Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekursivno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1.*

Zadatak 4.3 Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 4.1. Dopuniti biblioteku novim funkcijama.

- (a) Napisati funkciju `void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)` koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave`.
- (b) Napisati funkciju `void ispisi_listu_unazad(Cvor * glava)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. NAPOMENA: *Koristiti iste main programe i test primere iz zadatka 4.1. Ove programe dopuniti pozivom funkcije koja ispisuje listu unazad.*

Zadatak 4.4 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

Test 1

```

IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23

IZLAZ:
Zagrade su ispravno uparene.

```

Test 2

```

IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}

IZLAZ:
Zagrade su ispravno uparene.

```

Test 3

```

IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)

IZLAZ:
Zagrade nisu ispravno uparene.

```

Test 4

```

IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)

IZLAZ:
Zagrade nisu ispravno uparene.

```

Test 5

```

DATOTEKA IZRAZ.TXT JE PRAZNA

IZLAZ:
Zagrade su ispravno uparene.

```

Test 6

```

DATOTEKA IZRAZ.TXT NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja
datoteke izraz.txt!

```

Zadatak 4.5 Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.*

Test 1

```

POZIV: ./a.out datoteka.html

DATOTEKA.HTML
<html>
  <head>
    <title>Primer</title>
  </head>
  <body>
  </body>

IZLAZ:
Etikete nisu pravilno uparene
(etiketa <html> nije zatvorena)

```

Test 2

```

POZIV: ./a.out datoteka.html

DATOTEKA.HTML
<head>
  <title>Primer</title>
</head>
<body>
</body>
</html>

IZLAZ:
Etikete nisu pravilno uparene
(nadjena je etiketa </html> koja nije otvorena)

```

Test 3

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

DATOTEKA.HTML
<html>
<head>
  <title>Primer</title>
</head>
<body>
  <h1>Naslov</h1>
  Danas je lep i suncan dan. <br>
  Sutra ce biti jos lepsi.
  <a link='http://www.math.rs'>Link</a>
</body>
</html>

IZLAZ:
  Etikete su pravilno uparene!
```

Test 4

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

DATOTEKA.HTML
<html>
<head>
  <title>Primer</title>
</head>
<body>
</html>

IZLAZ:
  Etikete nisu pravilno uparene
  (nadjena je etiketa </html>, a poslednja
  otvorena je <body>)
```

Test 5

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

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
  Greska prilikom otvaranja
  datoteke datoteka.html.
```

Test 6

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

DATOTEKA.HTML JE PRAZNA

IZLAZ:
  Etikete su pravilno uparene!
```

Zadatak 4.6 Napisati program kojim se simulira rad jednog šaltera na kojem se prvo kod službenika zakazuju termini, a potom službenik uslužuje korisnike. Službenik evidentira korisničke JMBG brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (EOF). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Postavlja mu se pitanje **Da li korisnika vratate na kraj reda?** i ukoliko on da odgovor **Da**, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije **Da**, tada službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke **izvestaj.txt**. Ova datoteka, za svaki obrađen zahtev, sadrži JMBG i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nezvezano od broja korisnika koji i dalje čekaju u redu. UPUTSTVO: *Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.*

Primer 1

```

INTERAKCIJA PROGRAMA:
Sluzbenik evidentira korisnicke zahteve:
Novi zahtev [CTRL+D za kraj]
JMBG: 1234567890123
Opis problema: Otvaranje racuna

Novi zahtev [CTRL+D za kraj]
JMBG: 2345678901234
Opis problema: Podizanje novca

Novi zahtev [CTRL+D za kraj]
JMBG: 3456789012345
Opis problema: Reklamacija

Novi zahtev [CTRL+D za kraj]
JMBG:

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 2345678901234
i zahtevom: Podizanje novca
Da li ga vracate na kraj reda? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Da

Sledeci je korisnik sa JMBG: 3456789012345
i zahtevom: Reklamacija
Da li ga vracate na kraj reda? [Da/Ne] Ne

Da li je kraj smene? [Da/Ne] Ne

Sledeci je korisnik sa JMBG: 1234567890123
i zahtevom: Otvaranje racuna
Da li ga vracate na kraj reda? [Da/Ne] Ne

IZVESTAJ.TXT
JMBG: 2345678901234 Zahtev: Podizanje novca
JMBG: 3456789012345 Zahtev: Reklamacija
JMBG: 1234567890123 Zahtev: Otvaranje racuna

```

Zadatak 4.7 Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
```

```
{
    unsigned broj_pojavljivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;
```

Test 1

```
POZIV: ./a.out datoteka.html

DATOTEKA.HTML
<html>
<head><title>Primer</title></head>
<body>
  <h1>Naslov</h1>
  Danas je lep i suncan dan. <br>
  A sutra ce biti jos lepsi.
  <a link='http://www.google.com'> Link 1</a>
  <a link='http://www.math.rs'> Link 2</a>
</body>
</html>

IZLAZ:
a - 4
br - 1
h1 - 2
body - 2
title - 2
head - 2
html - 2
```

Test 2

```
POZIV: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja
datoteke datoteka.html.
```

Zadatak 4.8 U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku **da** ili **ne**, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (EOF). Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: *Pretpostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

Primer 1

```

Poziv: ./a.out studenti.txt

STUDENTI.TXT
123/2014 Marko Lukic
3/2014 Ana Sokic
43/2013 Jelena Ilic
41/2009 Marija Zaric
13/2010 Milovan Lazic

INTERAKCIJA PROGRAMA:
3/2014 da: Ana Sokic
235/2008 ne
41/2009 da: Marija Zaric

```

Primer 2

```

Poziv: ./a.out studenti.txt

DATOTEKA STUDENTI.TXT JE PRAZNA

INTERAKCIJA PROGRAMA:
3/2014 ne
235/2008 ne
41/2009 ne

```

Zadatak 4.9 Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. *NAPOMENA: Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.*

Test 1

```

Poziv: ./a.out dat1.txt dat2.txt

DAT1.TXT
2 4 6 10 15

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
[2, 4, 5, 6, 6, 10, 11, 12, 14, 15, 16]

```

Test 2

```

Poziv: ./a.out dat1.txt dat2.txt

DAT1.TXT
2 4 6 10 15

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ:
Greska prilikom otvaranja datoteke
dat2.txt.

```

Test 3

```

Poziv: ./a.out dat1.txt dat2.txt

DATOTEKA DAT1.TXT JE PRAZNA

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
[5, 6, 11, 12, 14, 16]

```

Test 4

```

Poziv: ./a.out dat1.txt

IZLAZ:
Program se poziva sa:
./a.out dat1.txt dat2.txt!

```

Zadatak 4.10 Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od ovih listi formira novu listu L koja sadrži naizmenično raspoređene čvorove listi L1 i L2: prvi čvor iz L1, prvi čvor iz L2, drugi čvor L1, drugi čvor

4 Dinamičke strukture podataka

L2, itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 6 za zadatak 4.9.*

Test 1

```
Poziv: ./a.out dat1.txt dat2.txt

DAT1.TXT
 2 4 6 10 15

DAT2.TXT
 5 6 11 12 14 16

IZLAZ:
 2 5 4 6 6 11 10 12 15 14 16
```

Zadatak 4.11 Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

- (a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.
- (b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

Test 1

```
BROJEVI.TXT
43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
12 15 16
```

Test 2

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAZ:
REZULTAT.TXT
  Greska prilikom otvaranja
  datoteke brojevi.txt.
```

Test 3

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
 Rezultat.txt ce biti prazna.
```

Zadatak 4.12 Grupa od n plesača na kostimima ima brojeve od 1 do n , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, opet u smeru kazaljke na satu. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n , k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz

ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti jednostruko povezanu kružnu listu.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 5 3 IZLAZ: 3 1 5 2 4 </pre>	<pre> ULAZ: 8 4 IZLAZ: 4 8 5 2 1 3 7 6 </pre>	<pre> ULAZ: 3 8 IZLAZ: n mora biti uvek vece od k, a 3 < 8! </pre>

Zadatak 4.13 Grupa od n plesača na kostimima ima brojeve od 1 do n , redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k -ti plesač. Odbrojavanje sada počinje od sledećeg suseda prethodno izbačenog, uz promenu smera. Ukoliko se prilikom prethodnog izbacivanja odbrojavalo u smeru kazaljke na satu sada će se obrojavati u suprotnom smeru, i obrnuto. Izlasci iz kruga se nastavljaju sve dok svi plesači ne budu isključeni. Celi brojevi n , k ($k < n$) se učitavaju sa standardnog ulaza. Napisati program koji će na standardni izlaz ispisati redne brojeve plesača u redosledu napuštanja kruga. UPUTSTVO: *Pri implementaciji koristiti dvostruko povezanu kružnu listu.* NAPOMENA: *Iskoristiti test 3 iz 4.12. zadatka.*

<i>Test 1</i>	<i>Test 2</i>
<pre> ULAZ: 5 3 IZLAZ: 3 5 4 2 1 </pre>	<pre> ULAZ: 8 4 IZLAZ: 4 8 5 7 6 3 2 1 </pre>

4.2 Stabla

Zadatak 4.14 Napisati program za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu `Cvor` kojom se opisuje čvor binarnog pretraživačkog stabla koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.
- Napisati funkciju `Cvor* napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.

- (c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.
- (d) Napisati funkciju `Cvor* pretrazi_stablo(Cvor* koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili NULL ukoliko takav čvor ne postoji.
- (e) Napisati funkciju `Cvor* pronadji_najmanji(Cvor* koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.
- (f) Napisati funkciju `Cvor* pronadji_najveci(Cvor* koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor** koren, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `koren`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor* koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor* koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor* koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void oslobodi_stablo(Cvor** koren)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `koren`.

Korišćenjem prethodnih funkcija, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

Primer 1

```

INTERAKCIJA PROGRAMA:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Traži se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće stablo: 1 2 9 18 32

```

Primer 2

```

INTERAKCIJA PROGRAMA:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Traži se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuće stablo: -3 -2 6 8 13 24

```

Zadatak 4.15 Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku **Nedostaje ime ulazne datoteke!**. Može se pretpostaviti da dužina reči neće biti veća od 50 karaktera.

Test 1

```

POZIV: ./a.out test.txt

TEST.TXT
Sunce utorak raCunar SUNCE programiranje
jabuka PROGramiranje sunCE JABUKa

IZLAZ:
jabuka: 2
programiranje: 2
racunar: 1
sunce: 3
utorak: 1

Najcesca rec: sunce (pojavljuje se 3 puta)

```

Test 2

```

POZIV: ./a.out suma.txt

SUMA.TXT
lipa zova hrast ZOVA breza LIPA

IZLAZ:
breza: 1
hrast: 1
lipa: 2
zova: 2

Najcesca rec: lipa
(pojavljuje se 2 puta)

```

Test 3

```

POZIV: ./a.out

IZLAZ:
Nedostaje ime ulazne datoteke!

```

Zadatak 4.16 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. **Pera Peric** 064/123-4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose

4 Dinamičke strukture podataka

sve do unosa reči KRAJ, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Primer 1

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858

INTERAKCIJA PROGRAMA:
Unesite ime datoteke: imenik.txt
Unesite ime i prezime: Pera Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ
```

Primer 2

```
DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
Unesite ime datoteke: imenik1.txt
Greska prilikom otvaranja datoteke
imenik1.txt!
```

Zadatak 4.17 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niz najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

Test 1

```

PRIJEMNI.TXT
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6

IZLAZ:
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9

```

Test 2

```

PRIJEMNI.TXT
[Ova datoteka ne postoji]

IZLAZ:
Greska prilikom otvaranja datoteke!

```

* **Zadatak 4.18** Napisati program koji implementira podsetnik za roden-dane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argu-ment komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY.`

Primer 1

```

POZIV: ./a.out rodjendani.txt

RODJENDANI.TXT
Marko Markovic 12.12.1990.
Milan Jevremovic 04.06.1989.
Maja Agic 23.04.2000.
Nadica Zec 01.01.1993.
Jovana Milic 05.05.1990.

INTERAKCIJA PROGRAMA:
Unesite datum: 23.04.
Slavljenik: Maja Agic
Unesite datum: 01.01.
Slavljenik: Nadica Zec
Unesite datum: 01.05.
Slavljenik: Jovana Milic 05.05.
Unesite datum:

```

Primer 2

```

POZIV: ./a.out rodjendani1.txt

INTERAKCIJA PROGRAMA:
Greska prilikom otvaranja datoteke!

```

Zadatak 4.19 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla.

4 Dinamičke strukture podataka

Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. NAPOMENA: *Skup funkcija koje smo napisali u prvom zadatku možemo iskoristiti kao malu biblioteku za rad sa binarnim pretraživačkim stablima celih brojeva. Tako će u zadacima koji slede, datoteka `stabla.h` predstavljati popis funkcija biblioteke, a datoteka `stabla.c` njihove implementacije. Programme koji koriste ovu biblioteku treba prevoditi i pokretati u skladu sa smernicama iz poglavlja 1.1.*

Primer 1

```
INTERAKCIJA PROGRAMA:
Prvo stablo:
10 5 15 3 2 4 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla jesu identicna.
```

Primer 2

```
INTERAKCIJA PROGRAMA:
Prvo stablo:
10 5 15 4 3 2 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla nisu identicna.
```

* **Zadatak 4.20** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

Primer 1

```
INTERAKCIJA PROGRAMA:
Prvo stablo: 1 7 8 9 2 2
Drugo stablo: 3 9 6 11 1
Unija: 1 1 2 2 3 6 7 8 9 9 11
Presek: 1 9
Razlika: 2 2 7 8
```

Primer 2

```
INTERAKCIJA PROGRAMA:
Prvo stablo: 11 2 7 5
Drugo stablo: 4 3 3 7
Unija: 2 3 3 4 5 7 7 11
Presek: 7
Razlika: 2 5 11
```

Zadatak 4.21 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

Primer 1

```

|| INTERAKCIJA PROGRAMA:
||   n: 7
||   a: 1 11 8 6 37 25 30
||   1 6 8 11 25 30 37

```

Primer 2

```

|| INTERAKCIJA PROGRAMA:
||   n: 55
||   Greska: pogresna dimenzija niza!

```

Zadatak 4.22 Dato je binarno pretraživačko stablo celih brojeva.

- Napisati funkciju koja izračunava broj čvorova stabla.
- Napisati funkciju koja izračunava broj listova stabla.
- Napisati funkciju koja štampa pozitivne vrednosti listova stabla.
- Napisati funkciju koja izračunava zbir čvorova stabla.
- Napisati funkciju koja izračunava najveći element stabla.
- Napisati funkciju koja izračunava dubinu stabla.
- Napisati funkciju koja izračunava broj čvorova na i -tom nivou stabla.
- Napisati funkciju koja ispisuje sve elemente na i -tom nivou stabla.
- Napisati funkciju koja izračunava maksimalnu vrednost na i -tom nivou stabla.
- Napisati funkciju koja izračunava zbir čvorova na i -tom nivou stabla.
- Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x .

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije.

Test 1

```
Poziv: ./a.out 2 15

ULAZ:
  10 5 15 3 2 4 30 12 14 13

IZLAZ:
  Broj cvorova: 10
  Broj listova: 4
  Pozitivni listovi: 2 4 13 30
  Zbir cvorova: 108
  Najveci element: 30
  Dubina stabla: 5
  Broj cvorova na 2. nivou: 3
  Elementi na 2. nivou: 3 12 30
  Maksimalni na 2. nivou: 30
  Zbir na 2. nivou: 45
  Zbir elemenata manjih ili jednakih od 15: 78
```

Test 2

```
Poziv: ./a.out 3 31

ULAZ:
  24 53 61 9 7 55 20 16

IZLAZ:
  Broj cvorova: 8
  Broj listova: 3
  Pozitivni listovi: 7 16 55
  Zbir cvorova: 245
  Najveci element: 61
  Dubina stabla: 4
  Broj cvorova na 3. nivou: 2
  Elementi na 3. nivou: 16 55
  Maksimalni na 3. nivou: 55
  Zbir na 3. nivou: 71
  Zbir elemenata manjih ili jednakih od 31: 76
```

Zadatak 4.23 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza.

Test 1

```
ULAZ:
  10 5 15 3 2 4 30 12 14 13

IZLAZ:
  0.nivo: 10
  1.nivo: 5 15
  2.nivo: 3 12 30
  3.nivo: 2 4 14
  4.nivo: 13
```

Test 2

```
ULAZ:
  6 11 8 3 -2

IZLAZ:
  0.nivo: 6
  1.nivo: 3 11
  2.nivo: -2 8
```

Test 3

```
ULAZ:
  24 53 61 9 7 55 20 16

IZLAZ:
  0.nivo: 24
  1.nivo: 9 53
  2.nivo: 7 20 61
  3.nivo: 16 55
```

* **Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1

```
INTERAKCIJA PROGRAMA:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 14 30 1 0
  Stabla su slicna kao u ogledalu.
```

Primer 2

```
INTERAKCIJA PROGRAMA:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 20 15 0
  Stabla nisu slicna kao u ogledalu.
```


Zadatak 4.25 AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

Test 1

```
ULAZ:
10 5 15 2 11 16 1 13

IZLAZ:
7
```

Test 2

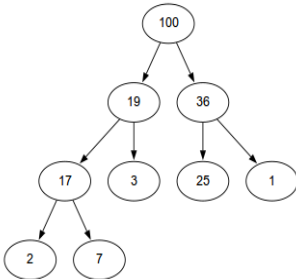
```
ULAZ:
16 30 40 24 10 18 45 22

IZLAZ:
6
```

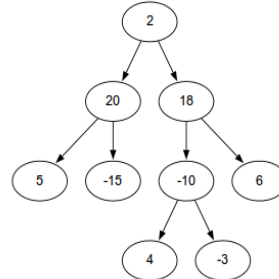
Zadatak 4.26 Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablama. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i glavni program koji kreira stablo kao na slici 4.1, poziva funkciju `heap` i ispisuje rezultat na standardnom izlazu.

Test 1

```
IZLAZ:
Zadato stablo je heap!
```



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

Zadatak 4.27 Dato je binarno stablo celih brojeva.

- (a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardnom izlazu.

Test 1

```
IZLAZ:
Vrednost u cvoru sa maksimalnim desnim zbirom:: 18
Vrednost u cvoru sa minimalnim levim zbirom: 18
2 18 -10 4
2 20 -15
```

4.3 Rešenja

Rešenje 4.1

```
1  #ifndef _LISTA_H
2  #define _LISTA_H
3
4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
5
6  typedef struct cvor {
7      int vrednost;
8      struct cvor *sledeci;
9  } Cvor;
10
11 /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12 dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
   na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
13
14 Cvor *napravi_cvor(int broj);
15
16 /* Funkcija oslobadja dinamicu memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
17
18 void oslobodi_listu(Cvor ** adresa_glave);
19
20 /* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
21
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
```

```

24 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
    NULL ukoliko je lista prazna. */
26 Cvor *pronadji_poslednji(Cvor * glava);

28 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
    pri alokaciji memorije, inace vraca 0. */
30 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
    nov cvor sa vrednoscu broj. */
34 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

36 /* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
void dodaj_iza(Cvor * tekuci, Cvor * novi);

38 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
    inace vraca 0. */
40 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

42 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
    NULL u slučaju da takav cvor ne postoji u listi. */
44 Cvor *pretrazi_listu(Cvor * glava, int broj);

46 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
    U pretrazi oslanja se na činjenicu da je lista koja se pretražuje
    neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
    sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
    */
48 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

50 /* Funkcija briše iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
    obriše stara glava. */
52 void obrisi_cvor(Cvor ** adresa_glave, int broj);

54 /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista sortirana
    neopadajuće. Azurira pokazivac na glavu liste, koji može biti
    promenjen ukoliko se obriše stara glava liste. */
56 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

58 /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
    liste, razdvojene zarezima i uokvirene zagradama. */
60 void ispisi_listu(Cvor * glava);

62 #endif

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"

```

```
4  Cvor *napravi_cvor(int broj)
6  {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    8  if (novi == NULL)
        return NULL;

    10  novi->vrednost = broj;
    12  novi->sledeci = NULL;
    14  return novi;
}

16  void oslobodi_listu(Cvor ** adresa_glave)
{
    18  Cvor *pomocni = NULL;

    20  /* Ako lista nije prazna, onda treba osloboditi memoriju. */
    while (*adresa_glave != NULL) {
        22  /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
           osloboditi cvor koji predstavlja glavu liste */
        24  pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
        26  /* Sledeci cvor je nova glava liste. */
        *adresa_glave = pomocni;
    }
    28 }

30  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
32  {
    34  /* Kreira se nov cvor i proverava se da li je bilo greske pri
       alokaciji. */
    Cvor *novi = napravi_cvor(broj);
    36  if (novi == NULL)
        return 1;

    38  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste. */
    40  novi->sledeci = *adresa_glave;
    *adresa_glave = novi;

    42  return 0;
    44  }

46  Cvor *pronadji_poslednji(Cvor * glava)
{
    48  /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
    if (glava == NULL)
        50  return NULL;

    52  /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
       glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
       54  ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
       cvor liste i vraca se vrednost pokazivaca glava.
```

```
56     Pokazivac glava je argument funkcije i njegove promene neće se
57     odraziti na vrednost pokazivaca glava u pozivajućoj funkciji. */
58     while (glava->sledeci != NULL)
59         glava = glava->sledeci;
60
61     return glava;
62 }
63
64 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
65 {
66     Cvor *novi = napravi_cvor(broj);
67     if (novi == NULL)
68         return 1;
69
70     /* U slučaju prazne liste, glava nove liste je upravo novi cvor i
71     ujedno i cela lista. Azurira se vrednost na koju pokazuje
72     adresa_glave i tako se azurira i pokazivačka promenljiva u
73     pozivajućoj funkciji. */
74     if (*adresa_glave == NULL) {
75         *adresa_glave = novi;
76         return 0;
77     }
78
79     /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
80     se dodaje na kraj liste kao sledbenik poslednjeg. */
81     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
82     poslednji->sledeci = novi;
83
84     return 0;
85 }
86
87 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
88 {
89     /* U praznoj listi nema takvog mesta i vraća se NULL. */
90     if (glava == NULL)
91         return NULL;
92
93     /* Pokazivac glava se pomera na sledeći cvor sve dok ne bude
94     pokazivala na cvor čiji je sledeći ili ne postoji ili ima
95     vrednost veću ili jednaku vrednosti novog cvora.
96
97     Zbog izračunavanja izraza u C-u prvi deo konjukcije mora biti
98     provera da li se doslo do poslednjeg cvora liste pre nego što se
99     proverí vrednost u sledećem cvoru, jer u slučaju poslednjeg,
100    sledeći ne postoji, pa ni njegova vrednost. */
101    while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
102        glava = glava->sledeci;
103
104    /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
105    poslednjeg cvora ili, ranije, na cvoru čiji sledeći ima vrednost
106    veću od broj. */

```

```
108     return glava;
109 }
110
111 void dodaj_iza(Cvor * tekuci, Cvor * novi)
112 {
113     /* Novi cvor se dodaje iza tekuceg cvora. */
114     novi->sledeci = tekuci->sledeci;
115     tekuci->sledeci = novi;
116 }
117
118 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
119 {
120     /* U slucaju prazne liste glava nove liste je novi cvor. Ukoliko je
121        doslo do greske pri alokaciji memorije čvraa se 1. */
122     if (*adresa_glave == NULL) {
123         Cvor *novi = napravi_cvor(broj);
124         if (novi == NULL)
125             return 1;
126         *adresa_glave = novi;
127         return 0;
128     }
129
130     /* Lista nije prazna. */
131     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
132        treba dodati na pocetak liste. */
133     if ((*adresa_glave)->vrednost >= broj) {
134         return dodaj_na_pocetak_liste(adresa_glave, broj);
135     }
136
137     /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
138        tada se pronalazi cvor liste iza koga treba uvezati nov cvor. */
139     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
140     Cvor *novi = napravi_cvor(broj);
141     if (novi == NULL)
142         return 1;
143
144     /* Uvezuje se novi cvor iza pomocnog. */
145     dodaj_iza(pomocni, novi);
146     return 0;
147 }
148
149 Cvor *pretrazi_listu(Cvor * glava, int broj)
150 {
151     for (; glava != NULL; glava = glava->sledeci)
152         if (glava->vrednost == broj)
153             return glava;
154
155     /* Nema traženog broja u listi i vraća se NULL. */
156     return NULL;
157 }
158
159 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
```

```

160 {
161     /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
162     for (; glava != NULL && glava->vrednost <= broj;
163           glava = glava->sledeci)
164         if (glava->vrednost == broj)
165             return glava;
166
167     return NULL;
168 }
169
170 void obrisi_cvor(Cvor ** adresa_glave, int broj)
171 {
172     Cvor *tekuci = NULL;
173     Cvor *pomocni = NULL;
174
175     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
176        broju, i azurira se pokazivac na glavu liste. */
177     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
178     {
179         /* Adresi repa liste treba sacuvati pre oslobadjanja cvora na
180            adresi adresa_glave. */
181         pomocni = (*adresa_glave)->sledeci;
182         free(*adresa_glave);
183         *adresa_glave = pomocni;
184     }
185
186     /* Ako je nakon toga lista ostala prazna, izlazi se iz funkcije. */
187     if (*adresa_glave == NULL)
188         return;
189
190     /* Od ovog trenutka, u svakoj iteraciji petlje tekuci pokazuje na
191        cvor cija vrednost je razlicita od trazeneog broja. Isto vazi i
192        za sve cvorove levo od tekućeg. Poredi se vrednost sledećeg
193        cvora (ako postoji) sa trazanim brojem. Cvor se brise ako je
194        jednak, ili, ako je razlicit, prelazi se na sledeci cvor. Ovaj
195        postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
196     tekuci = *adresa_glave;
197     while (tekuci->sledeci != NULL)
198     {
199         if (tekuci->sledeci->vrednost == broj) {
200             /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
201                prvo cuva u pomocni. */
202             pomocni = tekuci->sledeci;
203             /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
204                njegovog trenutnog sledećeg. Njegov novi sledeci ce biti
205                sledeci od cvora koji se brise. */
206             tekuci->sledeci = pomocni->sledeci;
207             /* Sada treba osloboditi cvor sa vrednoscu broj. */
208             free(pomocni);
209         } else {
210             /* Inace, ne treba brisati sledećeg od tekućeg i pokazivac se
211                pomera na sledeci. */
212             tekuci = tekuci->sledeci;

```

```
    }
212     return;
213 }
214
215 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
216 {
217     Cvor *tekuci = NULL;
218     Cvor *pomocni = NULL;
219
220     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
221        broju i azurira se pokazivac na glavu liste. */
222     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
223     {
224         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
225            adresi adresa_glave. */
226         pomocni = (*adresa_glave)->sledeci;
227         free(*adresa_glave);
228         *adresa_glave = pomocni;
229     }
230
231     /* Ako je nakon toga lista ostala prazna, funkcija se prekida. Isto
232        se radi i ukoliko glava liste sadrzi vrednost koja je veca od
233        broja, jer kako je lista sortirana rastuce nema potrebe broj
234        traziti u repu liste. */
235     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
236         return;
237
238     /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
239        na cvor cija vrednost je manja od trazenog broja, kao i svim
240        cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
241        je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
242        ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
243        cija vrednost je veca od trazenog broja. */
244     tekuci = *adresa_glave;
245     while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj)
246     {
247         if (tekuci->sledeci->vrednost == broj) {
248             pomocni = tekuci->sledeci;
249             tekuci->sledeci = tekuci->sledeci->sledeci;
250             free(pomocni);
251         } else {
252             /* Ne treba brisati sledeceg od tekuceg jer je manji od
253                trazenog i tekuci se pomera na sledeci cvor. */
254             tekuci = tekuci->sledeci;
255         }
256     }
257     return;
258 }
259
260 void ispisi_listu(Cvor * glava)
261 {
262     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
263        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
```



```

    na glavu liste iz pozivajuće funkcije. */
262 putchar('[');
    for (; glava != NULL; glava = glava->sledeci) {
264     printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
266         printf(", ");
    }
268
    printf("]\n");
270 }

```

```

#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
            treba osloboditi pre napustanja programa. */
18        if (dodaj_na_pocetak_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    }

    printf("\nUnesite broj koji se trazi: ");
28    scanf("%d", &broj);
30
    trazeni = pretrazi_listu(glava, broj);
32    if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
34    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
36
    oslobodi_listu(&glava);
38
    return 0;
40 }

```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* 2) Glavni program */
6 int main()
7 {
8     Cvor *glava = NULL;
9     int broj;
10
11     /* Testiranje dodavanja novog broja na kraj liste. */
12     printf("Unesite brojeve: (za kraj CTRL+D)\n");
13     while (scanf("%d", &broj) > 0) {
14         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
15            memorije za nov cvor. Memoriju alociranu za cvorove liste
16            treba osloboditi pre napustanja programa. */
17         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
18             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
19             oslobodi_listu(&glava);
20             exit(EXIT_FAILURE);
21         }
22         printf("\tLista: ");
23         ispisi_listu(glava);
24     }
25
26     printf("\nUnesite broj koji se brise: ");
27     scanf("%d", &broj);
28
29     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
30        procitanom sa ulaza */
31     obrisi_cvor(&glava, broj);
32
33     printf("Lista nakon brisanja: ");
34     ispisi_listu(glava);
35
36     oslobodi_listu(&glava);
37
38     return 0;
39 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* 3) Glavni program */
6 int main()
7 {
8     Cvor *glava = NULL;
9     Cvor *trazeni = NULL;
10     int broj;
11 }
```

```

13  /* Testira se dodavanje u listu tako da ona bude neopadajuće
    uredjena */
printf("Unosite brojeve (za kraj CTRL+D)\n");
15  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. */
17      if (dodaj_sortirano(&glava, broj) == 1) {
          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21          oslobodi_listu(&glava);
          exit(EXIT_FAILURE);
23      }
      printf("\tLista: ");
25      ispisi_listu(glava);
    }

27  printf("\nUnosite broj koji se trazi: ");
29  scanf("%d", &broj);

31  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
33      printf("Broj %d se ne nalazi u listi!\n", broj);
  else
35      printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

37  printf("\nUnosite broj koji se brise: ");
  scanf("%d", &broj);

39

  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
  procitanom sa ulaza */
41  obrisi_cvor_sortirane_liste(&glava, broj);

43

  printf("Lista nakon brisanja: ");
45  ispisi_listu(glava);

47  oslobodi_listu(&glava);

49  return 0;
}

```

Rešenje 4.2

```

1  #ifndef _LISTA_H
2  #define _LISTA_H

4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste. */
6  typedef struct cvor {
    int vrednost;
8    struct cvor *sledeci;
   } Cvor;

```

```
10  /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
12     dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
     na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
14  Cvor *napravi_cvor(int broj);

16  /* Funkcija oslobadja dinamičku memoriju zauzetu za cvorove liste
     ciji se pokazivac glava nalazi na adresi adresa_glave. */
18  void oslobodi_listu(Cvor ** adresa_glave);

20  /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
     bilo greske pri alokaciji memorije, inace vraca 0. */
22  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24  /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
     pri alokaciji memorije, inace vraca 0. */
26  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

28  /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
     ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
     memorije, inace vraca 0. */
30  int dodaj_sortirano(Cvor ** adresa_glave, int broj);
32
34  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
     Vraca pokazivac na cvor liste u kome je sadržan traženi broj ili
     NULL u slučaju da takav cvor ne postoji u listi. */
36  Cvor *pretrazi_listu(Cvor * glava, int broj);

38  /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
     U pretrazi oslanja se na činjenicu da je lista koja se pretražuje
     neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
     sadržan traženi broj ili NULL u slučaju da takav cvor ne postoji.
     */
42  Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

44  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj. Azurira
     pokazivac na glavu liste, koji može biti promenjen u slučaju da se
     obriše stara glava liste. */
46  void obrisi_cvor(Cvor ** adresa_glave, int broj);

48
50  /* Funkcija briše iz liste sve cvorove koji sadrže dati broj,
     oslanjajući se na činjenicu da je prosledjena lista sortirana
     neopadajuće. Azurira pokazivac na glavu liste, koji može biti
     promenjen ukoliko se obriše stara glava liste. */
52  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
54
56  /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
     zapetama. */
     void ispisi_vrednosti(Cvor * glava);
58
60  /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
     liste, razdvojene zapetama i uokvirene zagradama. */
```

```

void ispisi_listu(Cvor * glava);
62 #endif

1 #include <stdio.h>
#include <stdlib.h>
3 #include "lista.h"

5 Cvor *napravi_cvor(int broj)
{
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
9         return NULL;

11     novi->vrednost = broj;
    novi->sledeci = NULL;
13     return novi;
}

15 void oslobodi_listu(Cvor ** adresa_glave)
17 {
    /* Lista je vec prazna */
19     if (*adresa_glave == NULL)
        return;

21     /* Ako lista nije prazna, treba osloboditi memoriju. Pre
23     oslobadjanja memorije za glavu liste, treba osloboditi rep
        liste. */
25     oslobodi_listu(&(*adresa_glave)->sledeci);
    /* Nakon oslobodjenog repa, oslobadja se glava liste, i azurira se
27     glava u pozivajucoj funkciji tako da odgovara praznoj listi */
    free(*adresa_glave);
29     *adresa_glave = NULL;
}

31 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
33 {
    /* Kreira se nov cvor i proverava se da li je bilo greske pri
35     alokaciji */
    Cvor *novi = napravi_cvor(broj);
37     if (novi == NULL)
        return 1;

39     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
41     novi->sledeci = *adresa_glave;
    *adresa_glave = novi;
43     return 0;
}

45 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
47 {
    if (*adresa_glave == NULL) {

```

```

49      /* Glava liste je upravo novi cvor i ujedno i cela lista. */
      Cvor *novi = napravi_cvor(broj);
51      /* Ukoliko je bilo greske pri alokaciji vraća se 1. */
      if (novi == NULL)
53          return 1;

55      /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
        azurira i pokazivačka promenljiva u pozivajućoj funkciji. */
57      *adresa_glave = novi;
      return 0;
59  }

61  /* Ako lista nije prazna, broj se dodaje u rep liste. */
  /* Prilikom dodavanja u listu na kraj u velikoj većini slučajeva
63      nov broj se dodaje u rep liste u rekurzivnom pozivu. Informacija
        o uspesnosti alokacije u rekurzivnom pozivu funkcija prosledjuje
65      visem rekurzivnom pozivu koji tu informaciju vraća u rekurzivni
        poziv iznad, sve dok se ne vrati u main. Tek je iz main funkcije
67      moguće pristupiti pravom početku liste i osloboditi je celu, ako
        ima potrebe. Ako je funkcija vratila 0, onda nije bilo greske.
        */
69      return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
  }

71  int dodaj_sortirano(Cvor ** adresa_glave, int broj)
73  {
75      /* U slučaju prazne liste, glava nove liste je upravo novi cvor. */
      if (*adresa_glave == NULL) {
          Cvor *novi = napravi_cvor(broj);
77          if (novi == NULL)
              return 1;

79          *adresa_glave = novi;
81          return 0;
      }

83      /* Lista nije prazna. Ako je broj manji ili jednak vrednosti u
        glavi liste, onda se dodaje na početak liste i vraća se
85          informacija o uspesnosti alokacije. */
      if ((*adresa_glave)->vrednost >= broj)
87          return dodaj_na_pocetak_liste(adresa_glave, broj);

89      /* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
        bude sortirana lista. */
91      return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
93  }

95  Cvor *pretrazi_listu(Cvor * glava, int broj)
  {
97      /* U praznoj listi ga sigurno nema */
      if (glava == NULL)
99          return NULL;
  
```

```

101  /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
    */
    if (glava->vrednost == broj)
103      return glava;

105  /* Inace, pretraga se nastavlja u repu liste. */
    return pretrazi_listu(glava->sledeci, broj);
107 }

109 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
    {
111     /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
        vrednosti u glavi liste, jer je lista neopadajuće sortirana. */
113     if (glava == NULL || glava->vrednost > broj)
        return NULL;

115     /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava.
        */
117     if (glava->vrednost == broj)
        return glava;

119     /* Inace, pretraga se nastavlja u repu. */
121     return pretrazi_listu(glava->sledeci, broj);
    }

123 void obrisi_cvor(Cvor ** adresa_glave, int broj)
    {
125     /* U praznoj listi, nema cvorova za brisanje. */
127     if (*adresa_glave == NULL)
        return;

129     /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj. */
131     obrisi_cvor(&(*adresa_glave)->sledeci, broj);

133     /* Preostaje provera da li glavu liste treba obrisati. */
135     if ((*adresa_glave)->vrednost == broj) {
        /* pomocni pokazuje na cvor koji treba da se obrise. */
        Cvor *pomocni = *adresa_glave;
137        /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
            brise se cvor koji je bio glava liste. */
139        *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
141    }
    }

143 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
    {
145     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
        od broja, kako je lista sortirana rastuce nema potrebe broj
        traziti u repu liste i zato se funkcija prekida. */
147     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
149

```

```
        return;
151
    /* Brisu se cvorovi iz repa koji imaju vrednost broj. */
153    obrisi_cvor(&(*adresa_glave)->sledeci, broj);

155    /* Preostaje provera da li glavu liste treba obrisati. */
    if ((*adresa_glave)->vrednost == broj) {
157        /* pomocni pokazuje na cvor koji treba da se obrise. */
        Cvor *pomocni = *adresa_glave;
159        /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
            brise se cvor koji je bio glava liste. */
        *adresa_glave = (*adresa_glave)->sledeci;
        free(pomocni);
163    }
165 }

166 void ispisi_vrednosti(Cvor * glava)
167 {
    /* Prazna lista */
169    if (glava == NULL)
        return;

171    /* Ispisuje se vrednost u glavi liste. */
173    printf("%d", glava->vrednost);

175    /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
        se poziva ista funkcija za ispis ostalih. */
177    if (glava->sledeci != NULL) {
        printf(", ");
179        ispisi_vrednosti(glava->sledeci);
    }
181 }

182 void ispisi_listu(Cvor * glava)
183 {
185    /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
        jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
        na glavu liste iz pozivajuće funkcije. Ona ispisuje samo
        zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
        rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
        elemente razdvojene zapetom i razmakom. */
191    putchar('[');
        ispisi_vrednosti(glava);
193    printf("]\n");
195 }
```

Rešenje 4.3

```
#ifndef _LISTA_H
2 #define _LISTA_H
```



```
4  /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   * vrednost i pokazivace na sledeci i prethodni cvor liste. */
6  typedef struct cvor {
   *     int vrednost;
   *     struct cvor *sledeci;
   *     struct cvor *prethodni;
10 } Cvor;

12 /* 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. */
14 Cvor *napravi_cvor(int broj);

16 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   * ciji se pocetni cvor nalazi na adresi adresa_glave. */
18 void oslobodi_listu(Cvor ** adresa_glave);

20 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   * bilo greske pri alokaciji memorije, inace vraca 0. */
22 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

24 /* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste, ili
   * NULL ukoliko je lista prazna. */
26 Cvor *pronadji_poslednji(Cvor * glava);

28 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
   * pri alokaciji memorije, inace vraca 0. */
30 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

32 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   * nov cvor sa vrednoscu broj. */
34 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

36 /* Funkcija uvezuje cvor novi iza postojeceg cvora tekuci. */
38 void dodaj_iza(Cvor * tekuci, Cvor * novi);

40 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   * sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
   * inace vraca 0. */
42 int dodaj_sortirano(Cvor ** adresa_glave, int broj);

44 /* 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. */
46 Cvor *pretrazi_listu(Cvor * glava, int broj);

48 /* 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.
   */
50 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
```

```

56 /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
    pokazivac glava se nalazi na adresi adresa_glave. */
58 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci);

60 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj. Azurira
    pokazivac na glavu liste, koji može biti promenjen u slučaju da se
62 obrise stara glava. */
    void obrisi_cvor(Cvor ** adresa_glave, int broj);

64 /* Funkcija brise iz liste sve cvorove koji sadrže dati broj,
    oslanjajući se na činjenicu da je prosledjena lista neopadajuće
66 sortirana. Azurira pokazivac na glavu liste, koji može biti
    promenjen ukoliko se obrise stara glava liste. */
68 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

70 /* Funkcija prikazuje vrednosti cvorova liste počev od glave ka kraju
    liste, razdvojene zarezima i uokvirene zagradama. */
72 void ispisi_listu(Cvor * glava);

74 /* Funkcija prikazuje cvorove liste počev od kraja ka glavi liste. */
76 void ispisi_listu_unazad(Cvor * glava);

78 #endif

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include "lista.h"

4
Cvor *napravi_cvor(int broj)
6 {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    8     if (novi == NULL)
        return NULL;

    10     novi->vrednost = broj;
    12     novi->sledeci = NULL;
    return novi;
14 }

16 void oslobodi_listu(Cvor ** adresa_glave)
{
    18     Cvor *pomocni = NULL;

    20     /* Ako lista nije prazna, onda treba osloboditi memoriju. */
    while (*adresa_glave != NULL) {
    22         /* Potrebno je prvo zapamtiti adresu sledećeg cvora i onda
            osloboditi memoriju cvora koji predstavlja glavu liste. */
    24         pomocni = (*adresa_glave)->sledeci;
            free(*adresa_glave);
    26         /* Sledeći cvor je nova glava liste. */
            *adresa_glave = pomocni;
    }
}

```

```
28     }
29 }
30
31 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
32 {
33     Cvor *novi = napravi_cvor(broj);
34     if (novi == NULL)
35         return 1;
36
37     /* Sledbenik novog cvora je glava stare liste */
38     novi->sledeci = *adresa_glave;
39     /* Ako stara lista nije bila prazna, onda prethodni od glave treba
40        da bude nov cvor. */
41     if (*adresa_glave != NULL)
42         (*adresa_glave)->prethodni = novi;
43     /* Novi cvor je nova glava liste. */
44     *adresa_glave = novi;
45
46     return 0;
47 }
48
49 Cvor *pronadji_poslednji(Cvor * glava)
50 {
51     /* U praznoj listi nema ni poslednjeg cvora i vraca se NULL. */
52     if (glava == NULL)
53         return NULL;
54
55     /* Sve dok glava pokazuje na cvor koji ima sledeceg, pokazivac
56        glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
57        ce pokazivati na cvor liste koji nema sledeceg, tj. na poslednji
58        cvor liste i vraca se vrednost pokazivaca glava.
59
60        Pokazivac glava je argument funkcije i njegove promene nece se
61        odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
62     while (glava->sledeci != NULL)
63         glava = glava->sledeci;
64
65     return glava;
66 }
67
68 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
69 {
70     Cvor *novi = napravi_cvor(broj);
71     if (novi == NULL)
72         return 1;
73
74     /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
75        ujedno i cela lista. Azurira se vrednost na koju pokazuje
76        adresa_glave i tako se azurira i pokazivacka promenljiva u
77        pozivajucoj funkciji. */
78     if (*adresa_glave == NULL) {
79         *adresa_glave = novi;
```

```
80     return 0;
81 }
82
83 /* Kako lista nije prazna, pronalazi se poslednji cvor i novi cvor
84    se dodaje na kraj liste kao sledbenik poslednjeg. */
85 Cvor *poslednji = pronadji_poslednji(*adresa_glave);
86 poslednji->sledeci = novi;
87 novi->prethodni = poslednji;
88
89     return 0;
90 }
91
92 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
93 {
94     /* U praznoj listi nema takvog mesta i vraca se NULL. */
95     if (glava == NULL)
96         return NULL;
97
98     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
99        pokazivala na cvor ciji je sledeci ili ne postoji ili ima
100        vrednost vecu ili jednaku vrednosti novog cvora.
101
102        Zbog izracunavanja izraza u C-u prvi deo konjukcije mora biti
103        provera da li se doslo do poslednjeg cvora liste pre nego sto se
104        proveru vrednost u sledecem cvoru, jer u slucaju poslednjeg,
105        sledeci ne postoji, pa ni njegova vrednost. */
106     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
107         glava = glava->sledeci;
108
109     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
110        poslednjeg cvora ili, ranije, na cvoru ciji sledeci ima vrednost
111        vecu od broj. */
112     return glava;
113 }
114
115 void dodaj_iza(Cvor * tekuci, Cvor * novi)
116 {
117     novi->sledeci = tekuci->sledeci;
118     novi->prethodni = tekuci;
119
120     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
121        a potom i tekuci dobija novog sledeceg postavljajanjem pokazivaca
122        na ispravne adrese. */
123     if (tekuci->sledeci != NULL)
124         tekuci->sledeci->prethodni = novi;
125     tekuci->sledeci = novi;
126 }
127
128 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
129 {
130     /* Ako je lista prazna, glava nove liste je novi cvor. */
131     if (*adresa_glave == NULL) {
```

```

132     Cvor *novi = napravi_cvor(broj);
133     if (novi == NULL)
134         return 1;
135     *adresa_glave = novi;
136     return 0;
137 }
138
139 /* Ukoliko je vrednost glave liste veca ili jednaka od nove
140    vrednosti onda nov cvor treba staviti na pocetak liste. */
141 if ((*adresa_glave)->vrednost >= broj) {
142     dodaj_na_pocetak_liste(adresa_glave, broj);
143     return 0;
144 }
145
146 Cvor *novi = napravi_cvor(broj);
147 if (novi == NULL)
148     return 1;
149
150 /* Pronazi se cvor iza koga treba uvezati nov cvor. */
151 Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
152 dodaj_iza(pomocni, novi);
153
154 return 0;
155 }
156
157 Cvor *pretrazi_listu(Cvor * glava, int broj)
158 {
159     for (; glava != NULL; glava = glava->sledeci)
160         if (glava->vrednost == broj)
161             return glava;
162
163     /* Nema trazenog broja u listi i vraca se NULL. */
164     return NULL;
165 }
166
167 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
168 {
169     /* U uslovu ostanka u petlji, bitan je redosled u konjukciji. */
170     for (; glava != NULL && glava->vrednost <= broj;
171          glava = glava->sledeci)
172         if (glava->vrednost == broj)
173             return glava;
174
175     /* Nema trazenog broja u listi i bice vrateno NULL. */
176     return NULL;
177 }
178
179 /* Kod dvostruko povezane liste brisanje cvora na koji pokazuje
180    tekuci moze se lako uraditi jer sadrzi pokazivace na svog
181    sledbenika i prethodnika u listi. */
182 void obrisi_tekuci(Cvor ** adresa_glave, Cvor * tekuci)
183 {

```

```
184  /* Ako je tekuci NULL pokazivac, nema sta da se brise. */
185  if (tekuci == NULL)
186      return;

188  /* Ako postoji prethodnik od tekućeg, onda se postavlja da njegov
189     sledeci bude sledeci od tekućeg. */
190  if (tekuci->prethodni != NULL)
191      tekuci->prethodni->sledeci = tekuci->sledeci;

192
193  /* Ako postoji sledbenik tekućeg, onda njegov prethodnik treba da
194     bude prethodnik tekućeg. */
195  if (tekuci->sledeci != NULL)
196      tekuci->sledeci->prethodni = tekuci->prethodni;

198  /* Ako je glava cvor koji se brise, nova glava liste bice sledbenik
199     stare glave. */
200  if (tekuci == *adresa_glave)
201      *adresa_glave = tekuci->sledeci;

202
203  /* Oslobadja se dinamički alociran prostor za cvor tekuci. */
204  free(tekuci);
205  }

206  void obrisi_cvor(Cvor ** adresa_glave, int broj)
207  {
208      Cvor *tekuci = *adresa_glave;

209
210      while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
211          obrisi_tekuci(adresa_glave, tekuci);
212  }

213
214  void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
215  {
216      Cvor *tekuci = *adresa_glave;

217
218      while ((tekuci =
219              pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
220          obrisi_tekuci(adresa_glave, tekuci);
221  }

222
223  void ispisi_listu(Cvor * glava)
224  {
225
226      /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
227         jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
228         na glavu liste iz pozivajuće funkcije. */
229      putchar('[');
230      for (; glava != NULL; glava = glava->sledeci) {
231          printf("%d", glava->vrednost);
232          if (glava->sledeci != NULL)
233              printf(", ");
234      }
```

```

236     printf("]\n");
237 }
238
239 void ispisi_listu_unazad(Cvor * glava)
240 {
241     putchar('[');
242     if (glava == NULL) {
243         printf("]\n");
244         return;
245     }
246
247     glava = pronadji_poslednji(glava);
248
249     for (; glava != NULL; glava = glava->prethodni) {
250         printf("%d", glava->vrednost);
251         if (glava->prethodni != NULL)
252             printf(", ");
253     }
254     printf("]\n");
255 }

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "lista.h"
4
5  /* 1) Glavni program */
6  int main()
7  {
8      /* Lista je prazna na pocetku. */
9      Cvor *glava = NULL;
10     Cvor *trazeni = NULL;
11     int broj;
12
13     /* Testiranje dodavanja novog broja na pocetak liste. */
14     printf("Unesite brojeve: (za kraj CTRL+D)\n");
15     while (scanf("%d", &broj) > 0) {
16         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17            memorije za nov cvor. Memoriju alociranu za cvorove liste
18            treba osloboditi pre napustanja programa. */
19         if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
20             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21             oslobodi_listu(&glava);
22             exit(EXIT_FAILURE);
23         }
24         printf("\tLista: ");
25         ispisi_listu(glava);
26     }
27
28     printf("\nUnesite broj koji se trazi u listi: ");
29     scanf("%d", &broj);
30
31     trazeni = pretrazi_listu(glava, broj);

```

```
    if (trazeni == NULL)
33     printf("Broj %d se ne nalazi u listi!\n", broj);
    else
35     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

37     printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(glava);

39     oslobodi_listu(&glava);

41     return 0;
43 }
```

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

5  /* 2) Glavni program */
   int main()
7  {
    Cvor *glava = NULL;
9     int broj;

11     /* Testiranje dodavanja novog broja na kraj liste. */
    printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
13     while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
15         memorije za nov cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa. */
17         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
19             oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
21         }
        printf("\tLista: ");
23         ispisi_listu(glava);
    }

25     printf("\nUnesite broj koji se brise iz liste: ");
27     scanf("%d", &broj);

29     /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
        procitanom sa ulaza. */
31     obrisi_cvor(&glava, broj);

33     printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

35     printf("\nLista ispisana u nazad: ");
37     ispisi_listu_unazad(glava);

39     oslobodi_listu(&glava);
```



```
41     return 0;
    }
```

```

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

/* 3) Glavni program */
int main()
{
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;
    int broj;

    /* Testira se dodavanje u listu tako da ona bude neopadajuće
       uredjena */
    printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
    while (scanf("%d", &broj) > 0) {
        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
           memorije za nov cvor. Memoriju alociranu za cvorove liste
           treba osloboditi pre napustanja programa. */
        if (dodaj_sortirano(&glava, broj) == 1) {
            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
            oslobodi_listu(&glava);
            exit(EXIT_FAILURE);
        }
        printf("\tLista: ");
        ispisi_listu(glava);
    }

    printf("\nUnosite broj koji se trazi u listi: ");
    scanf("%d", &broj);

    trazeni = pretrazi_listu(glava, broj);
    if (trazeni == NULL)
        printf("Broj %d se ne nalazi u listi!\n", broj);
    else
        printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

    printf("\nUnosite broj koji se brise iz liste: ");
    scanf("%d", &broj);

    /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
       procitanom sa ulaza. */
    obrisi_cvor_sortirane_liste(&glava, broj);

    printf("Lista nakon brisanja: ");
    ispisi_listu(glava);

    printf("\nLista ispisana u nazad: ");
    ispisi_listu_unazad(glava);
}
```

```
50     oslobodi_listu(&glava);
52     return 0;
}
```

Rešenje 4.4

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   typedef struct cvor {
5     char zagrada;
       struct cvor *sledeci;
7 } Cvor;

9 int main()
{
11     Cvor *stek = NULL;
       FILE *ulaz = NULL;
13     char c;
       Cvor *pomocni = NULL;
15
       ulaz = fopen("izraz.txt", "r");
17     if (ulaz == NULL) {
           fprintf(stderr,
19                 "Greska prilikom otvaranja datoteke izraz.txt!\n");
           exit(EXIT_FAILURE);
21     }

23     while ((c = fgetc(ulaz)) != EOF) {
           /* Ako je učitana otvorena zagrada, stavlja se na stek. */
25         if (c == '(' || c == '{' || c == '[') {
               pomocni = (Cvor *) malloc(sizeof(Cvor));
27               if (pomocni == NULL) {
                   fprintf(stderr, "Greska prilikom alokacije memorije!\n");
29                   return 1;
               }
               pomocni->zagrada = c;
               pomocni->sledeci = stek;
31               stek = pomocni;
33         }

35         /* Ako je učitana zatvorena zagrada, proverava se da li je stek
           prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
           otvorena zagrada. */
37         else {
39             if (c == ')' || c == '}' || c == ']') {
                   if (stek != NULL && ((stek->zagrada == '(' && c == ')')
41                       || (stek->zagrada == '{' && c == '}')
                           || (stek->zagrada == '[' && c == ']')))
                       {
```

```

43         /* Sa vrha steka se uklanja otvorena zagrada */
           pomocni = stek->sledeci;
45         free(stek);
           stek = pomocni;
47     } else {
           /* Zgrade u izrazu nisu ispravno uparene. */
49         break;
       }
51     }
   }
53 }
/* Procitana je cela datoteka. Zatvaramo je. */
55 fclose(ulaz);

57 /* Ako je stek prazan i procitana je cela datoteka, zgrade su
   ispravno uparene, u suprotnom, nisu. */
59 if (stek == NULL && c == EOF)
   printf("Zgrade su ispravno uparene.\n");
61 else {
   printf("Zgrade nisu ispravno uparene.\n");
63   /* U slucaju neispravnog uparivanja treba osloboditi memoriju
      koja je ostala zauzeta stekom. */
65   while (stek != NULL) {
       pomocni = stek->sledeci;
67       free(stek);
       stek = pomocni;
69   }
   }
71
73 return 0;
}

```

Rešenje 4.5

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include <ctype.h>

6 #define MAX 100

8 #define OTVORENA 1
#define ZATVORENA 2
10

#define VAN_ETIKETE 0
12 #define PROCITANO_MANJE 1
#define U_ETIKETI 2
14

/* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
16 pokazivac na sledeci cvor. */
typedef struct cvor {

```

```
18     char etiketa[MAX];
19     struct cvor *sledeci;
20 } Cvor;

22 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
23    adresu ili NULL ako alokacija nije bila uspesna. */
24 Cvor *napravi_cvor(char *etiketa)
25 {
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;

30     /* Inicijalizacija polja u novom cvoru */
31     if (strlen(etiketa) >= MAX) {
32         fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
33         etiketa[MAX - 1] = '\0';
34     }
35     strcpy(novi->etiketa, etiketa);
36     novi->sledeci = NULL;
37     return novi;
38 }

40 /* Funkcija oslobadja memoriju zauzetu stekom. */
41 void oslobodi_stek(Cvor ** adresa_vrha)
42 {
43     Cvor *pomocni;
44     while (*adresa_vrha != NULL) {
45         pomocni = *adresa_vrha;
46         *adresa_vrha = (*adresa_vrha)->sledeci;
47         free(pomocni);
48     }
49 }

50
51 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
52    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
53    zauzeta memorija za listu ciji se pokazivac vrh nalazi na adresi
54    adresa_vrha. */
55 void prover_i_alokaciju(Cvor ** adresa_vrha, Cvor * novi)
56 {
57     if (novi == NULL) {
58         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
59         oslobodi_stek(adresa_vrha);
60         exit(EXIT_FAILURE);
61     }
62 }

63
64 /* Funkcija postavlja na vrh steka novu etiketu. */
65 void potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
66 {
67     Cvor *novi = napravi_cvor(etiketa);
68     prover_i_alokaciju(adresa_vrha, novi);
69     novi->sledeci = *adresa_vrha;
```

```
70  *adresa_vrha = novi;
71  }
72
73  /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
74  pokazivac razlicit od NULL, tada u niz karaktera na koji on
75  pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
76  suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
77  samim tim nije bilo moguće skinuti vrednost sa steka) ili 1 u
78  suprotnom. */
79  int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
80  {
81      Cvor *pomocni;
82
83      /* Pokusaj skidanja vrednost sa vrha praznog steka rezultuje
84      greskom i vraca se 0. */
85      if (*adresa_vrha == NULL)
86          return 0;
87
88      /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
89      adresu kopira etiketa sa vrha steka. */
90      if (etiketa != NULL)
91          strcpy(etiketa, (*adresa_vrha)->etiketa);
92
93      /* Element sa vrha steka se uklanja. */
94      pomocni = *adresa_vrha;
95      *adresa_vrha = (*adresa_vrha)->sledeci;
96      free(pomocni);
97
98      return 1;
99  }
100
101  /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
102  steka. Ukoliko je stek prazan, vraca NULL. */
103  char *vrh_steka(Cvor * vrh)
104  {
105      if (vrh == NULL)
106          return NULL;
107      return vrh->etiketa;
108  }
109
110  /* Funkcija prikazuje stek pocev od vrha prema dnu. */
111  void prikazi_stek(Cvor * vrh)
112  {
113      for (; vrh != NULL; vrh = vrh->sledeci)
114          printf("<%=s>\n", vrh->etiketa);
115  }
116
117  /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
118  etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
119  Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
120  procita etiketa. Vraca OTVORENA, ako je procitana otvorena
121  etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa. */
```

```
122 int uzmi_etiketu(FILE * f, char *etiketa)
123 {
124     int c;
125     int i = 0;
126     /* Stanje predstavlja informaciju dokle se stalo sa citanjem
127     etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
128     nije zapoceto citanje. */
129     /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
130     OTVORENA ili ZATVORENA. */
131     int stanje = VAN_ETIKETE;
132     int tip;
133
134     /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
135     to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
136     samo malim slovima. */
137     while ((c = fgetc(f)) != EOF) {
138         switch (stanje) {
139             case VAN_ETIKETE:
140                 if (c == '<')
141                     stanje = PROCITANO_MANJE;
142                 break;
143             case PROCITANO_MANJE:
144                 if (c == '/') {
145                     /* Cita se zatvorena etiketa. */
146                     tip = ZATVORENA;
147                 } else {
148                     if (isalpha(c)) {
149                         /* Cita se otvorena etiketa */
150                         tip = OTVORENA;
151                         etiketa[i++] = tolower(c);
152                     }
153                 }
154                 /* Od sada se cita etiketa i zato se menja stanje. */
155                 stanje = U_ETIKETI;
156                 break;
157             case U_ETIKETI:
158                 if (isalpha(c) && i < MAX - 1) {
159                     /* Ako je procitani karakter slovo i nije premasena
160                     dozvoljena duzina etikete, procitani karakter se smanjuje
161                     i smesta u etiketu. */
162                     etiketa[i++] = tolower(c);
163                 } else {
164                     /* Inace, staje se sa citanjem etikete. Korektno se završava
165                     niska koja sadrzi procitanu etiketu i vraća se njen tip.
166                     */
167                     etiketa[i] = '\\0';
168                     return tip;
169                 }
170                 break;
171         }
172     }
173     /* Doslo se do kraja datoteke pre nego sto je procitana naredna
```

```

    etiketa i vraca se EOF. */
174 return EOF;
}
176
177 int main(int argc, char **argv)
178 {
    /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
180    nije procitana. */
    Cvor *vrh = NULL;
182    char etiketa[MAX];
    int tip;
184    int uparene = 1;
    FILE *f = NULL;
186
    /* Ime datoteke se preuzima iz komandne linije. */
188    if (argc < 2) {
        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
190        exit(0);
    }
192
    /* Datoteka se otvara za citanje. */
194    if ((f = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
196                argv[1]);
        exit(1);
198    }

200    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
    while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
202        /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
        etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
204        potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
        koje nemaju sadrzaj (npr link). Zbog jednostavnosti
206        pretpostavlja se da njih nema u HTML dokumentu. */
        if (tip == OTVORENA) {
208            if (strcmp(etiketa, "br") != 0
                && strcmp(etiketa, "hr") != 0
210                && strcmp(etiketa, "meta") != 0)
                potisni_na_stek(&vrh, etiketa);
212        }
        /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
214        u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
        uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
216        je taj uslov ispunjen, skida se sa steka, jer je upravo
        zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
218        nisu pravilno uparene. */
        else if (tip == ZATVORENA) {
220            if (vrh_steka(vrh) != NULL
                && strcmp(vrh_steka(vrh), etiketa) == 0)
                skini_sa_steka(&vrh, NULL);
222            else {
224                printf("Etikete nisu pravilno uparene\n");
            }
        }
    }
}

```

```
printf("(nadjena je etiketa </%s>", etiketa);
226 if (vrh_steka(vrh) != NULL)
    printf(", a poslednja otvorena je </%s>\n", vrh_steka(vrh))
    ;
228 else
    printf(" koja nije otvorena)\n");
230 uparene = 0;
    break;
232 }
    }
234 }
/* Završeno je citanje datoteke i zatvara se. */
236 fclose(f);

/* Ako do sada nije pronadjeno pogresno uparivanje, stek bi trebalo
da bude prazan. Ukoliko nije, tada postoje etikete koje su
ostale otvorene. */
240 if (uparene) {
242     if (vrh_steka(vrh) == NULL)
        printf("Etikete su pravilno uparene!\n");
244     else {
        printf("Etikete nisu pravilno uparene\n");
246         printf("(etiketa </%s> nije zatvorena)\n", vrh_steka(vrh));
        /* Oslobadja se memorija zauzeta stekom. */
248         oslobodi_stek(&vrh);
    }
250 }
    return 0;
252 }
```

Rešenje 4.6

```
1 #ifndef _RED_H
#define _RED_H
3
#include <stdio.h>
5 #include <stdlib.h>

7 #define MAX 1000
#define JMBG_DUZINA 14
9
/* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11 opis njegovog zahteva. */
typedef struct {
13     char jmbg[JMBG_DUZINA];
    char opis[MAX];
15 } Zahtev;

17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
    korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
```



```

    Zahtev nalog;
21     struct cvor *sledeci;
    } Cvor;
23
    /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25     poslate adrese i vraca adresu novog cvora ili NULL ako je doslo do
    greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27     treba smestiti u nov cvor zbog smestanja manjeg podatka na
    sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29     bajtovima(B) u odnosu na strukturu Zahtev. */
    Cvor *napravi_cvor(Zahtev * zahtev);
31
    /* Funkcija prazni red, oslobadjajuci memoriju koji je red zauzeo. */
33     void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);
35
    /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
37     zauzeta memorija za listu ciji se pokazivac pocetak se nalazi na
    adresi adresa_pocetka i prekida program. */
39     void prover_i_alokaciju(Cvor ** adresa_pocetka,
                             Cvor ** adresa_kraja, Cvor * novi);
41
    /* Funkcija dodaje na kraj reda novi zahtev. */
43     void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                      Zahtev * zahtev);
45
    /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
47     pokazivac razlicit od NULL, tada se u strukturu na koju on
    pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
49     suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
    suprotnom. */
51     int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                       Zahtev * zahtev);
53
    /* Funkcija vraca pokazivac na strukturu koji sadrzi zahtev korisnika
    na pocetku reda. Ukoliko je red prazan, vraca NULL. */
55     Zahtev *pocetak_reda(Cvor * pocetak);
57
    /* Funkcija prikazuje sadrzaj reda. */
59     void prikazi_red(Cvor * pocetak);
61 #endif

1 #include "red.h"

3 Cvor *napravi_cvor(Zahtev * zahtev)
{
5     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
7         return NULL;

9     novi->nalog = *zahtev;

```

```
    novi->sledeci = NULL;
11     return novi;
12 }
13
14 void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
15 {
16     Cvor *pomocni = NULL;
17
18     while (*pocetak != NULL) {
19         pomocni = *pocetak;
20         *pocetak = (*pocetak)->sledeci;
21         free(pomocni);
22     }
23     *kraj = NULL;
24 }
25
26 void prover_i_alokaciju(Cvor ** adresa_pocetka,
27                        Cvor ** adresa_kraja, Cvor * novi)
28 {
29     if (novi == NULL) {
30         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
31         oslobodi_red(adresa_pocetka, adresa_kraja);
32         exit(EXIT_FAILURE);
33     }
34 }
35
36 void dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
37                 Zahtev * zahtev)
38 {
39     Cvor *novi = napravi_cvor(zahtev);
40     prover_i_alokaciju(adresa_pocetka, adresa_kraja, novi);
41
42     /* U red se uvek dodaje na kraj, ali zbog postojanja pokazivaca na
43        kraj, dodavanje na kraj je podjednako efikasno kao dodavanje na
44        pocetak. */
45     if (*adresa_kraja != NULL) {
46         (*adresa_kraja)->sledeci = novi;
47         *adresa_kraja = novi;
48     } else {
49         /* Ako je red bio ranije prazan */
50         *adresa_pocetka = novi;
51         *adresa_kraja = novi;
52     }
53 }
54
55 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
56                  Zahtev * zahtev)
57 {
58     Cvor *pomocni = NULL;
59
60     if (*adresa_pocetka == NULL)
61         return 0;
```

```

63     if (zahtev != NULL)
        *zahtev = (*adresa_pocetka)->nalog;
65
        pomocni = *adresa_pocetka;
67     *adresa_pocetka = (*adresa_pocetka)->sledeci;
        free(pomocni);
69
        if (*adresa_pocetka == NULL)
71            *adresa_kraja = NULL;
73
        return 1;
    }
75
    Zahtev *pocetak_reda(Cvor * pocetak)
77    {
        if (pocetak == NULL)
79            return NULL;
81
        return &(pocetak->nalog);
    }
83
    void prikazi_red(Cvor * pocetak)
85    {
        for (; pocetak != NULL; pocetak = pocetak->sledeci)
87            printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
89
        printf("\n");
    }

```

```

#include <stdio.h>
2  #include <stdlib.h>
   #include <string.h>
4  #include "red.h"

6  #define VREME_ZA_PAUZU 5

8  int main(int argc, char **argv)
   {
10     /* Red je prazan. */
        Cvor *pocetak = NULL, *kraj = NULL;
12     Zahtev nov_zahtev;
        Zahtev *sledeci = NULL;
14     char odgovor[3];
        int broj_usluzenih = 0;
16     FILE *izlaz = fopen("izvestaj.txt", "a");

18     if (izlaz == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
20         exit(EXIT_FAILURE);
    }
22

```

```
24  /* Sluzbenik evidentira korisnicke zahteve unosnjem njihovog JMBG
    broja i opisa potrebne usluge. */
    printf("Sluzbenik evidentira korisnicke zahteve:\n");

26

28  /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
    JMBG broja procitao i da bi fgets nakon toga procitala ispravan
    red sa opisom zahteva. */
    printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
    while (scanf("%s", nov_zahtev.jmbg) != EOF) {
32        getchar();
        printf("\tOpis problema: ");
34        fgets(nov_zahtev.opis, MAX - 1, stdin);
        /* Ako je poslednji karakter nov red, eliminiše se. */
36        if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
            nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';
38        dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
        printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
40    }

42  /* Datoteka više nije potrebna i treba je zatvoriti. */
    fclose(izlaz);

44

46  /* Dokle god ima korisnika u redu, treba ih uslužiti. */
    while (1) {
        sledeci = pocetak_reda(pocetak);
48        /* Ako nema nikog više u redu, prekida se petlja. */
        if (sledeci == NULL)
50            break;

52        printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
        printf("i zahtevom: %s\n", sledeci->opis);

54        skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

56        broj_usluzenih++;

58        printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
        scanf("%s", odgovor);

60

62        if (strcmp(odgovor, "Da") == 0)
            dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
64        else
            fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
66                    nov_zahtev.opis);

68        if (broj_usluzenih == VREME_ZA_PAUZU) {
            printf("\nDa li je kraj smene? [Da/Ne] ");
70            scanf("%s", odgovor);

72            if (strcmp(odgovor, "Da") == 0)
                break;
74            else
```

```

    broj_usluzenih = 0;
76 }
77 }
78
79 /*****
80  Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:
81
82  while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
83      printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
84             nov_zahtev.jmbg);
85      printf("sa zahtevom: %s\n", nov_zahtev.opis);
86      broj_usluzenih++;
87
88      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
89      scanf("%s", odgovor);
90      if (strcmp(odgovor, "Da") == 0)
91          dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
92      else
93          fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
94                  nov_zahtev.jmbg, nov_zahtev.opis);
95
96      if (broj_usluzenih == VREME_ZA_PAUZU) {
97          printf("\nDa li je kraj smene? [Da/Ne] ");
98          scanf("%s", odgovor);
99          if (strcmp(odgovor, "Da") == 0)
100              break;
101          else
102              broj_usluzenih = 0;
103      }
104  }
105  *****/
106
107  /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
108     neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
109     koju zauzima red sa neobradjenim zahtevima korisnika. */
110  oslobodi_red(&pocetak, &kraj);
111
112  return 0;
113 }

```

Rešenje 4.7

```

1  #include<stdio.h>
2  #include<string.h>
3  #include<stdlib.h>
4  #define MAX_DUZINA 20
5
6  typedef struct _Cvor {
7      unsigned broj_pojavljivanja;
8      char etiketa[20];
9      struct _Cvor *sledeci;

```

```
11 } Cvor;

12 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
13    ili NULL ako alokacija nije uspesno izvrшена. */
14 Cvor *napravi_cvor(unsigned br, char *etiketa)
15 {
16     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
17     if (novi == NULL)
18         return NULL;
19
20     novi->broj_pojavljivanja = br;
21     strcpy(novi->etiketa, etiketa);
22     novi->sledeci = NULL;
23     return novi;
24 }
25
26 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste. */
27 void oslobodi_listu(Cvor ** adresa_glave)
28 {
29     Cvor *pomocni = NULL;
30
31     while (*adresa_glave != NULL) {
32         pomocni = (*adresa_glave)->sledeci;
33         free(*adresa_glave);
34         *adresa_glave = pomocni;
35     }
36 }
37
38 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
39    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
40    zauzeta memorija za listu ciji pokazivac glava se nalazi na adresi
41    adresa_glave i prekida program. */
42 void prover_a_lokacije(Cvor * novi, Cvor ** adresa_glave)
43 {
44     if (novi == NULL) {
45         fprintf(stderr, "malloc() greska u funkciji napravi_cvor()!\n");
46         oslobodi_listu(adresa_glave);
47         exit(EXIT_FAILURE);
48     }
49 }
50
51 /* Funkcija dodaje novi cvor na pocetak liste. */
52 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
53                             char *etiketa)
54 {
55     Cvor *novi = napravi_cvor(br, etiketa);
56     prover_a_lokacije(novi, adresa_glave);
57     novi->sledeci = *adresa_glave;
58     *adresa_glave = novi;
59 }
60
61 /* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
```

```

        NULL ako takav cvor ne postoji. */
63 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
{
65     Cvor *tekuci;
    for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
67         if (strcmp(tekuci->etiketa, etiketa) == 0)
            return tekuci;
69     return NULL;
}

71 /* Funkcija ispisuje sadrzaj liste */
73 void ispisi_listu(Cvor * glava)
{
75     for (; glava != NULL; glava = glava->sledeci)
        printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
77 }

79 /* Glavni program */
int main(int argc, char **argv)
{
81     if (argc != 2) {
83         fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out datoteka.html!\n");
        ;
85         exit(EXIT_FAILURE);
    }

87     /* Otvaramo datoteku za citanje */
89     FILE *in = NULL;
    in = fopen(argv[1], "r");
91     if (in == NULL) {
        fprintf(stderr,
93         "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
        exit(EXIT_FAILURE);
95     }

97     char c;
    int i = 0;
99     char procitana[MAX_DUZINA];
    Cvor *glava = NULL;
    Cvor *trazeni = NULL;

101     while ((c = fgetc(in)) != EOF) {

103         if (c == '<') {
            /* Cita se zatvorena etiketa. */
105             if ((c = fgetc(in)) == '/') {
                i = 0;
107                 while ((c = fgetc(in)) != '>')
                    procitana[i++] = c;
109             }
            /* Cita se otvorena etiketa. */
111

```

```
113     else {
114         i = 0;
115         procitana[i++] = c;
116         while ((c = fgetc(in)) != ' ' && c != '>')
117             procitana[i++] = c;
118     }
119     procitana[i] = '\0';

121     /* Trazi se ucitana etiketa medju postojećim cvorovima liste.
122        Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
123        sa brojem pojavljivanja 1, inace uvecava se broj
124        pojavljivanja etikete. */
125     trazeni = pretrazi_listu(glava, procitana);
126     if (trazeni == NULL)
127         dodaj_na_pocetak_liste(&glava, 1, procitana);
128     else
129         trazeni->broj_pojavljivanja++;
130 }
131 }

133 fclose(in);

135 ispisi_listu(glava);
136 oslobodi_listu(&glava);

137 return 0;
138 }
```

Rešenje 4.8

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

4
#define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21

8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
   studentu. */
10 typedef struct _Cvor {
    char broj_indeksa[MAX_INDEKS];
12    char ime[MAX_IME_PREZIME];
    char prezime[MAX_IME_PREZIME];
14    struct _Cvor *sledeci;
} Cvor;

16
/* Funkcija kreira, inicijalizuje cvor liste i vraca pokazivac na nov
18    cvor ili NULL ukoliko alokacija nije prosla. */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
```



```
22     if (novi == NULL)
23         return NULL;
24     /* Inicijalizacija polja novog cvora */
25     strcpy(novi->broj_indeksa, broj_indeksa);
26     strcpy(novi->ime, ime);
27     strcpy(novi->prezime, prezime);
28     novi->sledeci = NULL;
29     return novi;
30 }

32 /* Funkcija oslobadja memoriju zauzetu za cvorove liste. */
33 void oslobodi_listu(Cvor ** adresa_glave)
34 {
35     if (*adresa_glave == NULL)
36         return;
37     /* Rep liste se oslobadja rekurzivnim pozivom. */
38     oslobodi_listu(&(*adresa_glave)->sledeci);
39     /* Potom se oslobadja i glava liste. */
40     free(*adresa_glave);
41     *adresa_glave = NULL;
42 }

44 /* Funkcija proverava da li je novi NULL pokazivac, i ukoliko jeste
45    oslobadja celu listu ciji se pokazivac glava nalazi na adresi
46    adresa_glave i prekida program. */
47 void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
48 {
49     if (novi == NULL) {
50         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
51         oslobodi_listu(adresa_glave);
52         exit(EXIT_FAILURE);
53     }
54 }

56 /* Funkcija dodaje novi cvor na pocetak liste. */
57 void dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
58                             char *ime, char *prezime)
59 {
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
61     prover_i_alokaciju(adresa_glave, novi);
62     novi->sledeci = *adresa_glave;
63     *adresa_glave = novi;
64 }

66 /* Funkcija ispisuje sadrzaj cvorova liste. */
67 void ispisi_listu(Cvor * glava)
68 {
69     for (; glava != NULL; glava = glava->sledeci)
70         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
71               glava->prezime);
72 }
```

```

74  /* Funkcija vraća cvor koji kao vrednost sadrži traženu etiketu, u
    suprotnom vraća NULL. */
76  Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
    {
78      if (glava == NULL)
          return NULL;
80      if (!strcmp(glava->broj_indeksa, broj_indeksa))
          return glava;
82      return pretrazi_listu(glava->sledeci, broj_indeksa);
    }

84
86  int main(int argc, char **argv)
    {
88      /* Argumenti komandne linije su neophodni jer se iz komandne linije
        dobija ime datoteke sa informacijama o studentima. */
90      if (argc != 2) {
          fprintf(stderr,
92              "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
          exit(EXIT_FAILURE);
      }

94      /* Otvaranje datoteke za citanje */
96      FILE *in = NULL;
          in = fopen(argv[1], "r");
98      if (in == NULL) {
          fprintf(stderr,
100              "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
          exit(EXIT_FAILURE);
102      }

104      /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
        listu */
106      char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
          char broj_indeksa[MAX_INDEKS];
108      Cvor *glava = NULL;
          Cvor *trazeni = NULL;

110
112      /* Ucitavanje vrednosti u listu */
          while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
              dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime);
114
116      /* Datoteka vise nije potrebna i zatvara se. */
          fclose(in);

118      /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
          while (scanf("%s", broj_indeksa) != EOF) {
120          trazeni = pretrazi_listu(glava, broj_indeksa);
              if (trazeni == NULL)
122                  printf("ne\n");
              else
124                  printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
          }
    }

```

```

126      /* Oslobadja se memorija zauzeta za cvorove liste. */
128      oslobodi_listu(&glava);

130      return 0;
  }

```

Rešenje 4.9

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include "601/lista.h"

5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
   na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7  pokazivaca postojećih cvorova listi. */
Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
9  {
   /* Pokazivac na glavu rezultujuće liste. */
11  Cvor *rezultujuca = NULL;
   /* Tekuci je pokazivac na pokazivac kome sledecem treba promeniti
13  vrednosti. Inicijalizuje se na adresu pokazivaca rezultujuca jer
   prvo treba odrediti glavu rezultujuće liste. */
15  Cvor **tekuci = &rezultujuca;

17  /* Ako su obe liste prazne, rezultat je isto prazna lista. */
   if (*adresa_glave_1 == NULL && *adresa_glave_2 == NULL)
19     return NULL;

21  /* Ako je prva lista prazna, rezultat je druga lista. */
   if (*adresa_glave_1 == NULL)
23     return *adresa_glave_2;

25  /* Ako je druga lista prazna, rezultat je prva lista. */
   if (*adresa_glave_2 == NULL)
27     return *adresa_glave_1;

29  /* Sve dok u obe liste ima cvorova, azurira se vrednost pokazivaca
   na koji tekuci pokazuje. U prvoj iteraciji tekuci pokazuje na
31  pokazivac rezultujuca i ovako se pokazivac rezultujuca usmerava
   da pokazuje na pocetak nove liste, tj. na cvor sa vrednoscu
33  manjeg od brojeva sadrzanih u cvorovima na koje vode pokazivaci
   na adresama adresa_glave_1 i adresa_glave_2. U svim ostalim
35  iteracijama to isto se dogadja samo pokazivacu na koji tekuci u
   tom trenutku pokazuje. */
37  while (*adresa_glave_1 != NULL && *adresa_glave_2 != NULL) {
   if ((*adresa_glave_1)->vrednost < (*adresa_glave_2)->vrednost) {
39     /* Pokazivac na koji tekuci pokazuje dobija vrednosti
       pokazivaca koji se nalazi na adresa_glave_1. Time sledbenik
41     poslednjeg uvezanog cvora postaje cvor koji je aktuelna
       glava prve liste. */

```

```
43     *tekuci = *adresa_glave_1;
44     /* Pomera se glava prve liste na sledeci cvor prve liste.
45
46     Ova promena bice vidljiva i van funkcije jer se direktno
47     menja promenljiva koja se nalazi na adresi adresa_glave_1.
48 */
49     *adresa_glave_1 = (*adresa_glave_1)->sledeci;
50 } else {
51     /* Sledbenik poslednjeg uvezanog cvora bice cvor koji je
52     aktuelna glava druge liste. */
53     *tekuci = *adresa_glave_2;
54     /* Pomera se glava druge liste na sledeci cvor druge liste */
55     *adresa_glave_2 = (*adresa_glave_2)->sledeci;
56 }
57 /* Tekuci se pomera na pokazivac sledeci od poslednjeg uvezanog,
58 jer je upravo to pokazivac koji treba da bude azuriran u
59 sledecoj iteraciji petlje. */
60 tekuci = &(*tekuci)->sledeci);
61 }
62
63 /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
64 rezultujucu listu treba nadovezati ostatak druge liste. Tako
65 sledbenik poslednjeg uvezanog cvora treba da bude ostatak druge
66 liste. */
67 if (*adresa_glave_1 == NULL)
68     *tekuci = *adresa_glave_2;
69 else {
70     if (*adresa_glave_2 == NULL)
71         *tekuci = *adresa_glave_1;
72 }
73
74 return rezultujuca;
75 }
76
77 /* Druga verzija prethodne funkcije koja ne pristupa pokazivacima
78 preko adresa vec direktno. Ne salju joj se adrese, vec vrednosti
79 pokazivaca na glave listi. */
80 Cvor *objedini_v2(Cvor * lista1, Cvor * lista2)
81 {
82     Cvor *rezultujuca = NULL;
83     Cvor *tekuci = NULL;
84
85     /* Ako su obe liste prazne i rezultat je prazna lista. */
86     if (lista1 == NULL && lista2 == NULL)
87         return NULL;
88
89     /* Ako je prva lista prazna, rezultat je druga lista. */
90     if (lista1 == NULL)
91         return lista2;
92
93     /* Ako je druga lista prazna, rezultat je prva lista. */
94     if (lista2 == NULL)
```

```
    return lista1;
95
/* Rezultujuca pokazuje na pocetak nove liste, tj. na cvor sa
97 vrednoscu manjeg od brojeva sadrzanih u cvorovima na koje
pokazuju lista1 i lista2. */
99 if (lista1->vrednost < lista2->vrednost) {
    rezultujuca = lista1;
101 lista1 = lista1->sledeci;
} else {
103 rezultujuca = lista2;
    lista2 = lista2->sledeci;
105 }
tekuci = rezultujuca;
107
/* Kako rezultujuca pokazuje na pocetak nove liste i ne sme joj se
109 menjati vrednost, koristi se pokazivac tekuci koji trenutno
sadrzi adresu promenljive rezultujuca. U svakoj iteraciji
111 petlje, dobijace adekvatnog sledbenika tako da i nova lista bude
uredjena neopadajuce i pomerace se na adresu sledeceg. */
113 while (lista1 != NULL && lista2 != NULL) {
    if (lista1->vrednost < lista2->vrednost) {
115 tekuci->sledeci = lista1;
        lista1 = lista1->sledeci;
117 } else {
        tekuci->sledeci = lista2;
119 lista2 = lista2->sledeci;
    }
121 tekuci = tekuci->sledeci;
}
123
/* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
125 rezultujucu listu treba nadovezati ostatak druge liste. */
if (lista1 == NULL)
127 tekuci->sledeci = lista2;
else
129 tekuci->sledeci = lista1;

131 return rezultujuca;
}
133
/* Glavni program */
135 int main(int argc, char **argv)
{
137 /* Argumenti komandne linije su neophodni. */
    if (argc != 3) {
139 fprintf(stderr,
        "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
141 exit(EXIT_FAILURE);
    }
143
/* Otvaramo datoteke sa elementima obe liste. */
145 FILE *in1 = NULL;
```

```

147     in1 = fopen(argv[1], "r");
148     if (in1 == NULL) {
149         fprintf(stderr,
150             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
151         exit(EXIT_FAILURE);
152     }
153
154     FILE *in2 = NULL;
155     in2 = fopen(argv[2], "r");
156     if (in2 == NULL) {
157         fprintf(stderr,
158             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
159         exit(EXIT_FAILURE);
160     }
161
162     int broj;
163     Cvor *lista1 = NULL;
164     Cvor *lista2 = NULL;
165     Cvor *rezultat = NULL;
166
167     /* Ucitavanje listi */
168     while (fscanf(in1, "%d", &broj) != EOF)
169         dodaj_na_kraj_liste(&lista1, broj);
170     while (fscanf(in2, "%d", &broj) != EOF)
171         dodaj_na_kraj_liste(&lista2, broj);
172
173     /* Pokazivac rezultat ce pokazivati na glavu liste koja se dobila
174        objedinjavanjem listi */
175     rezultat = objedini(&lista1, &lista2);
176
177     /******
178     Poziv druge verzije prethodne funkcije
179
180     rezultat = objedini_v2(lista1, lista2);
181     *****/
182
183     /* Ispis rezultujuce liste. */
184     ispisi_listu(rezultat);
185
186     /* Kako je lista rezultat dobijena prevezivanjem cvorova polaznih
187        listi, njenim oslobadjanjem bice oslobodjena sva zauzeta
188        memorija. */
189     oslobodi_listu(&rezultat);
190
191     fclose(in1);
192     fclose(in2);
193     return 0;
194 }

```

Rešenje 4.14

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

4  /* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
6  typedef struct cvor {
8      int broj;
9      struct cvor *levo;
10     struct cvor *desno;
11 } Cvor;

12 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj)
15 {
16     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
17     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
18     if (novi == NULL)
19         return NULL;
20
21     /* Inicijalizuju se polja novog cvora. */
22     novi->broj = broj;
23     novi->levo = NULL;
24     novi->desno = NULL;
25
26     /* Vraca se adresa novog cvora. */
27     return novi;
28 }

30 /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
32 void proveri_alokaciju(Cvor * novi_cvor)
33 {
34     /* Ukoliko je cvor neuspesno kreiran */
35     if (novi_cvor == NULL) {
36
37         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
           */
38         fprintf(stderr, "Malloc greska za novi cvor!\n");
39         exit(EXIT_FAILURE);
40     }
41 }

44 /* c) Funkcija koja dodaje zadati broj u stablo */
46 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
47 {
48     /* Ako je stablo prazno */
49     if (*adresa_korena == NULL) {
50
51         /* Kreira se novi cvor */
```

```
52     Cvor *novi = napravi_cvor(broj);
    prover_i_alokaciju(novi);
54
    /* I proglašava se korenom stabla */
56     *adresa_korena = novi;
    return;
58 }

60 /* U suprotnom traži se odgovarajuća pozicija za zadati broj: */

62 /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*adresa_korena)->broj)
64
        /* Broj se dodaje u levo podstablo */
66         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
68     else
        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
69         dodaje u desno podstablo */
70         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
72 }

74 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
    Cvor *pretrazi_stablo(Cvor * koren, int broj)
76 {
78     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
    if (koren == NULL)
80         return NULL;
82
    /* Ako je trazena vrednost sadržana u korenu */
    if (koren->broj == broj) {
84
        /* Prekidamo pretragu */
86         return koren;
    }
88
    /* Inace, ako je broj manji od vrednosti sadržane u korenu */
90     if (broj < koren->broj)

        /* Pretraga se nastavlja u levom podstablu */
92         return pretrazi_stablo(koren->levo, broj);
94
    else
        /* U suprotnom, pretraga se nastavlja u desnom podstablu */
96         return pretrazi_stablo(koren->desno, broj);
98 }

100 /* e) Funkcija pronalazi cvor koji sadrži najmanju vrednost u stablu */
    Cvor *pronadji_najmanji(Cvor * koren)
102 {
```



```
104  /* Ako je stablo prazno, prekida se pretraga */
106  if (koren == NULL)
108      return NULL;

110  /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
112     levo od njega */

114  /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
116     najmanju vrednost */
118  if (koren->levo == NULL)
120      return koren;

122  /* Inace, pretragu treba nastaviti u levom podstablu */
124  return pronadji_najmanji(koren->levo);
126 }

128 /* f) Funkcija pronalazi cvor koji sadrzi najveću vrednost u stablu
130     */
132 Cvor *pronadji_najveci(Cvor * koren)
134 {
136     /* Ako je stablo prazno, prekida se pretraga */
138     if (koren == NULL)
140         return NULL;

142     /* Vrednosti koje su veće od vrednosti u korenu stabla nalaze se
144        desno od njega */

146     /* Ako je koren cvor koji nema desno podstablo, onda on sadrži
148        najveću vrednost */
150     if (koren->desno == NULL)
152         return koren;

154     /* Inace, pretragu treba nastaviti u desnom podstablu */
156     return pronadji_najveci(koren->desno);
158 }

160 /* g) Funkcija koja briše cvor stabla koji sadrži zadati broj */
162 void obrisi_element(Cvor ** adresa_korena, int broj)
164 {
166     Cvor *pomocni_cvor = NULL;

168     /* Ako je stablo prazno, brisanje nije primenljivo */
170     if (*adresa_korena == NULL)
172         return;

174     /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
176        stabla, ona se eventualno nalazi u levom podstablu, pa treba
178        rekursivno primeniti postupak na levo podstablo. Koren ovako
180        modifikovanog stabla je nepromenjen. */
182     if (broj < (*adresa_korena)->broj) {
184         obrisi_element(&(*adresa_korena)->levo, broj);
186     }
```

```
154     return;
155 }
156
157 /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
158 stabla, ona se eventualno nalazi u desnom podstablu pa treba
159 rekursivno primeniti postupak na desno podstablo. Koren ovako
160 modifikovanog stabla je nepromenjen. */
161 if ((*adresa_korena)->broj < broj) {
162     obrisi_element(&(*adresa_korena)->desno, broj);
163     return;
164 }
165
166 /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
167 jednaka broju koji se brise (tj. slucaj kada treba obrisati
168 koren) */
169
170 /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
171 prazno stablo (vraca se NULL) */
172 if ((*adresa_korena)->levo == NULL
173     && (*adresa_korena)->desno == NULL) {
174     free(*adresa_korena);
175     *adresa_korena = NULL;
176     return;
177 }
178
179 /* Ako koren ima samo levog sina, tada se brisanje vrši tako sto se
180 brise koren, a novi koren postaje levi sin */
181 if ((*adresa_korena)->levo != NULL
182     && (*adresa_korena)->desno == NULL) {
183     pomocni_cvor = (*adresa_korena)->levo;
184     free(*adresa_korena);
185     *adresa_korena = pomocni_cvor;
186     return;
187 }
188
189 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako sto
190 se brise koren, a novi koren postaje desni sin */
191 if ((*adresa_korena)->desno != NULL
192     && (*adresa_korena)->levo == NULL) {
193     pomocni_cvor = (*adresa_korena)->desno;
194     free(*adresa_korena);
195     *adresa_korena = pomocni_cvor;
196     return;
197 }
198
199 /* Slucaj kada koren ima oba sina - najpre se potrazi sledbenik
200 korena (u smislu poretka) u stablu. To je upravo po vrednosti
201 najmanji cvor u desnom podstablu. On se moze pronaci npr.
202 funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
203 vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
204 broj koji se brise). Zatim se prosto rekursivno pozove funkcija
    za brisanje na desno podstablo. S obzirom da u njemu treba
```

```

206     obrisati najmanji element, a on zasigurno ima najviše jednog
208     potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
        od jednostavnijih načina koji su gore opisani. */
210     pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
        (*adresa_korena)->broj = pomocni_cvor->broj;
212     pomocni_cvor->broj = broj;
        obris_element(&(*adresa_korena)->desno, broj);
    }

214
216     /* h) Funkcija ispisuje stablo u infiksnoj notaciji (Levo postablo -
        Koren - Desno podstablo) */
218     void ispisi_stablo_infiksno(Cvor * koren)
    {
220         /* Ako stablo nije prazno */
        if (koren != NULL) {

222             /* Prvo se ispisuju svi cvorovi levo od korena */
            ispisi_stablo_infiksno(koren->levo);

224
            /* Zatim se ispisuje vrednost u korenu */
226             printf("%d ", koren->broj);

            /* Na kraju se ispisuju cvorovi desno od korena */
228             ispisi_stablo_infiksno(koren->desno);
230         }
    }

232
234     /* i) Funkcija ispisuje stablo u prefiksnoj notaciji (Koren - Levo
        podstablo - Desno podstablo) */
236     void ispisi_stablo_prefiksno(Cvor * koren)
    {
238         /* Ako stablo nije prazno */
        if (koren != NULL) {

240             /* Prvo se ispisuje vrednost u korenu */
            printf("%d ", koren->broj);

242
            /* Zatim se ispisuju svi cvorovi levo od korena */
244             ispisi_stablo_prefiksno(koren->levo);

            /* Na kraju se ispisuju svi cvorovi desno od korena */
246             ispisi_stablo_prefiksno(koren->desno);
248         }
    }

250
252     /* j) Funkcija ispisuje stablo postfiksnoj notaciji (Levo podstablo
        - Desno postablo - Koren) */
254     void ispisi_stablo_postfiksno(Cvor * koren)
    {

256         /* Ako stablo nije prazno */
        if (koren != NULL) {

```

```

258      /* Prvo se ispisuju svi cvorovi levo od korena */
260      ispisi_stablo_postfiksno(koren->levo);

262      /* Zatim se ispisuju svi cvorovi desno od korena */
264      ispisi_stablo_postfiksno(koren->desno);

266      /* Na kraju se ispisuje vrednost u korenu */
      printf("%d ", koren->broj);
  }
268 }

270 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
void oslobodi_stablo(Cvor ** adresa_korena)
272 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
274     if (*adresa_korena == NULL)
        return;

276     /* Inace ... */
278     /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);

280     /* Oslobadja se memorija zauzeta desnim podstablom */
282     oslobodi_stablo(&(*adresa_korena)->desno);

284     /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);

286     /* Proglasava se stablo praznim */
288     *adresa_korena = NULL;
}

290 int main()
292 {
    Cvor *koren;
294     int n;
    Cvor *trazeni_cvor;

296     /* Proglasava se stablo praznim */
298     koren = NULL;

300     /* Dodaju se vrednosti u stablo */
    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
302     while (scanf("%d", &n) != EOF) {
        dodaj_u_stablo(&koren, n);
304     }

306     /* Generisu se trazeni ispisi: */
    printf("\nInfiksni ispis: ");
308     ispisi_stablo_infiksno(koren);
    printf("\nPrefiksni ispis: ");

```

```

310     ispisi_stablo_prefiksno(koren);
311     printf("\nPostfiksni ispis: ");
312     ispisi_stablo_postfiksno(koren);

313
314     /* Demonstrira se rad funkcije za pretragu */
315     printf("\nTraži se broj: ");
316     scanf("%d", &n);
317     trazen_i_cvor = pretrazi_stablo(koren, n);
318     if (trazen_i_cvor == NULL)
319         printf("Broj se ne nalazi u stablu!\n");
320
321     else
322         printf("Broj se nalazi u stablu!\n");

323
324     /* Demonstrira se rad funkcije za brisanje */
325     printf("Briše se broj: ");
326     scanf("%d", &n);
327     obrisi_element(&koren, n);
328     printf("Rezultujuće stablo: ");
329     ispisi_stablo_infiksno(koren);
330     printf("\n");

331
332     /* Oslobadja se memorija zauzeta stablom */
333     oslobodi_stablo(&koren);
334     return 0;
}

```

Rešenje 4.15

```

#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include <ctype.h>

6 #define MAX 50

8 /* Struktura kojom se opisuje cvor stabla: sadrži rec, njen broj
   pojavljivanja i redom pokazuje na levo i desno podstablo */
10 typedef struct cvor {
11     char *rec;
12     int broj;
13     struct cvor *levo;
14     struct cvor *desno;
15 } Cvor;

16
17 /* Funkcija koja kreira novi cvor stabla */
18 Cvor *napravi_cvor(char *rec)
19 {
20     /* Alocira se memorija za novi cvor i proverava se uspešnost
       alokacije. */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));

```

```
24     if (novi_cvor == NULL)
        return NULL;

26     /* Alocira se memorija za zadatu rec: potrebno je rezervisati
        memoriju za svaki karakter reci ukljucujuci i terminirajucu nulu
        */
28     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
30     if (novi_cvor->rec == NULL) {
        free(novi_cvor);
32     return NULL;
    }

34     /* Inicijalizuju se polja u novom cvoru */
36     strcpy(novi_cvor->rec, rec);
    novi_cvor->brojac = 1;
38     novi_cvor->levo = NULL;
    novi_cvor->desno = NULL;

40     /* Vraca se adresa novog cvora */
42     return novi_cvor;
}

44     /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
46 void proveri_alokaciju(Cvor * novi_cvor)
{
48     /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
50         /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
        */
52         fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
54     }
}

56     /* Funkcija koja dodaje novu rec u stablo. */
58 void dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
{
60     /* Ako je stablo prazno */
    if (*adresa_korena == NULL) {
62         /* Kreira se cvor koji sadrzi zadatu rec */
        Cvor *novi = napravi_cvor(rec);
64         proveri_alokaciju(novi);

        /* I proglašava se korenom stabla */
        *adresa_korena = novi;
68         return;
    }

70     /* U suprotnom se trazi odgovarajuca pozicija za novu rec */

72     /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
74     levo podstablo */
```

```

76     if (strcmp(rec, (*adresa_korena)->rec) < 0)
        dodaj_u_stablo(&(*adresa_korena)->levo, rec);

78     else
        /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
80        desno podstablo */
        if (strcmp(rec, (*adresa_korena)->rec) > 0)
82            dodaj_u_stablo(&(*adresa_korena)->desno, rec);

84     else
        /* Ako je rec jednaka reci u korenu, uvecava se njen broj
86        pojavljivanja */
        (*adresa_korena)->brojac++;
88 }

90 /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
92 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
94     if (*adresa_korena == NULL)
        return;

96     /* Inace ... */
98     /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);

100     /* Oslobadja se memorija zauzeta desnim podstablom */
102     oslobodi_stablo(&(*adresa_korena)->desno);

104     /* Oslobadja se memorija zauzeta korenom */
    free((*adresa_korena)->rec);
106     free(*adresa_korena);

108     /* Stablo se proglašava praznim */
    *adresa_korena = NULL;
110 }

112 /* Funkcija koja pronalazi cvor koji sadrži najfrekventniju rec (rec
    sa najvećim brojem pojavljivanja) */
114 Cvor *nadj_i_najfrekventniju_rec(Cvor * koren)
    {
116         Cvor *max, *max_levo, *max_desno;

118         /* Ako je stablo prazno, prekida se sa pretragom */
        if (koren == NULL)
120            return NULL;

122         /* Pronalazi se najfrekventnija rec u levom podstablu */
        max_levo = nadj_i_najfrekventniju_rec(koren->levo);
124
        /* Pronalazi se najfrekventnija rec u desnom podstablu */
126         max_desno = nadj_i_najfrekventniju_rec(koren->desno);

```

```
128  /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
    podstabla, korena i desnog podstabla */
130  max = koren;
    if (max_levo != NULL && max_levo->brojac > max->brojac)
132      max = max_levo;
    if (max_desno != NULL && max_desno->brojac > max->brojac)
134      max = max_desno;

136  /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
    return max;
138 }

140 /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
    pracene brojem pojavljivanja */
142 void prikazi_stablo(Cvor * koren)
{
144     /* Ako je stablo prazno, završava se sa ispisom */
    if (koren == NULL)
146         return;

148     /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz levog
        podstabla */
    prikazi_stablo(koren->levo);

152     /* Zatim rec iz korena */
    printf("%s: %d\n", koren->rec, koren->brojac);

154     /* I nastavlja se sa ispisom reci iz desnog podstabla */
    prikazi_stablo(koren->desno);
156 }

158 /* Funkcija učitava sledeću rec iz zadate datoteke f i upisuje je u
    niz rec. Maksimalna dužina reci je određena argumentom max.
    Funkcija vraća EOF ako u datoteci nema više reci ili 0 u
    suprotnom. Rec je niz malih ili velikih slova. */
162 int procitaj_rec(FILE * f, char rec[], int max)
{
164     /* Karakter koji se čita */
    int c;

166     /* Indeks pozicije na koju se smesta procitani karakter */
    int i = 0;

170     /* Sve dok ima mesta za još jedan karakter u nizu i dokle se god
        nije stiglo do kraja datoteke... */
    while (i < max - 1 && (c = fgetc(f)) != EOF) {
172         /* Proverava se da li je procitani karakter slovo */
        if (isalpha(c))
174             /* Ako jeste, smesta se u niz - pritom se vrši konverzija u
                mala slova jer program treba da bude neosetljiv na razliku
                između malih i velikih slova */
                rec[i++] = tolower(c);
            else
                continue;
176     }
    rec[i] = '\0';
178 }
```



```
180     rec[i++] = tolower(c);
182     else
183         /* Ako nije, proverava se da li je procitano barem jedno slovo
184            nove reci */
185         /* Ako jeste, prekida se sa citanjem */
186         if (i > 0)
187             break;
188
189         /* U suprotnom se ide na sledecu iteraciju */
190     }
191
192     /* Dodaje se na rec terminirajuca nula */
193     rec[i] = '\0';
194
195     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
196     return i > 0 ? 0 : EOF;
197 }
198
199 int main(int argc, char **argv)
200 {
201     Cvor *koren = NULL, *max;
202     FILE *f;
203     char rec[MAX];
204
205     /* Provera da li je navedeno ime datoteke prilikom pokretanja
206        programa */
207     if (argc < 2) {
208         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
209         exit(EXIT_FAILURE);
210     }
211
212     /* Priprema datoteke za citanje */
213     if ((f = fopen(argv[1], "r")) == NULL) {
214         fprintf(stderr, "fopen() greska pri otvaranju %s\n", argv[1]);
215         exit(EXIT_FAILURE);
216     }
217
218     /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
219        pretrage. */
220     while (procitaj_rec(f, rec, MAX) != EOF)
221         dodaj_u_stablo(&koren, rec);
222
223     /* Posto je citanjem reci zavrшено, zatvara se datoteka */
224     fclose(f);
225
226     /* Prikazuju se sve reci iz teksta i brojevi njihovih
227        pojavljivanja. */
228     prikazi_stablo(koren);
229
230     /* Pronalazi se najfrekventnija rec */
231     max = nadji_najfrekventniju_rec(koren);
```

```
232  /* Ako takve reci nema... */
    if (max == NULL)
234
        /* Ispisuje se odgovarajuće obavestjenje */
236        printf("U tekstu nema reci!\n");
238    else
        /* Inace, ispisuje se broj pojavljivanja reci */
240        printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
                max->rec, max->brojac);
242
        /* Oslobadja se dinamicki alociran prostor za stablo */
244        oslobodi_stablo(&koren);
246    return 0;
}
```

Rešenje 4.16

```
#include <stdio.h>
2  #include <stdlib.h>
    #include <string.h>
4  #include <ctype.h>

6  #define MAX_IME_DATOTEKE 50
    #define MAX_CIFARA 13
8  #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime, broj
    telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
    char ime_i_prezime[MAX_IME_I_PREZIME];
14    char telefon[MAX_CIFARA];
    struct cvor *levo;
16    struct cvor *desno;
} Cvor;
18

/* Funkcija koja kreira novi cvora stabla */
20 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
22    /* Alocira se memorija za novi cvor i proverava se uspesnost
        alokacije. */
24    Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
    if (novi_cvor == NULL)
26        return NULL;

28    /* Inicijalizuju se polja novog cvora */
    strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
30    strcpy(novi_cvor->telefon, telefon);
    novi_cvor->levo = NULL;
```

```
32     novi_cvor->desno = NULL;

34     /* Vraca se adresa novog cvora */
    return novi_cvor;
36 }

38 /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
void prover_i_alokaciju(Cvor * novi_cvor)
40 {
    /* Ukoliko je cvor neuspesno kreiran */
42     if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
        */
44         fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
46     }
48 }

50 /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo. */
void dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
52                  char *telefon)
54 {
    /* Ako je stablo prazno */
56     if (*adresa_korena == NULL) {
        /* Kreira se novi cvor */
58         Cvor *novi = napravi_cvor(ime_i_prezime, telefon);
        prover_i_alokaciju(novi);
60
        /* I proglašava se korenom stabla */
62         *adresa_korena = novi;
        return;
64     }

66     /* U suprotnom trazi se odgovarajuca pozicija za novi unos. Kako
    pretragu treba vrsiti po imenu i prezimenu, stablo treba da bude
    pretrazivacko po ovom polju */
68

70     /* Ako je zadato ime i prezime leksikografski manje od imena i
    prezimena sadržanog u korenu, podaci se dodaju u levo podstablo
    */
72     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
        < 0)
74         dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime, telefon);

76     else
        /* Ako je zadato ime i prezime leksikografski vece od imena i
        prezimena sadržanog u korenu, podaci se dodaju u desno
        podstablo */
78
80     if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
        dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime, telefon);
82 }
```

```
84 /* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
86 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
88     if (*adresa_korena == NULL)
        return;

90     /* Inace ... */
92     /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);

94     /* Oslobadja se memorija zauzeta desnim podstablom */
96     oslobodi_stablo(&(*adresa_korena)->desno);

98     /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);

100     /* Stablo se proglašava praznim */
102     *adresa_korena = NULL;
}

104 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
106 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
    koristiti za proveru da li je stablo lepo kreirano ili ne */
108 void prikazi_stablo(Cvor * koren)
{
110     /* Ako je stablo prazno, završava se sa ispisom */
    if (koren == NULL)
112         return;

114     /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
        podstabla */
116     prikazi_stablo(koren->levo);

118     /* Zatim se ispisuju podaci iz korena */
    printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

120     /* I nastavlja se sa ispisom podataka iz desnog podstabla */
122     prikazi_stablo(koren->desno);
}

124 /* Funkcija učitava sledeći kontakt iz zadate datoteke i upisuje ime
126     i prezime i broj telefona u odgovarajuće nizove. Maksimalna dužina
    imena i prezimena određena je konstantom MAX_IME_PREZIME, a
128     maksimalna dužina broja telefona konstantom MAX_CIFARA. Funkcija
    vraća EOF ako nema više kontakata ili 0 u suprotnom. */
130 int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
{
132     /* Karakter koji se cita */
    int c;

134
```

```

136  /* Indeks pozicije na koju se smesta procitani karakter */
137  int i = 0;

138  /* Linije datoteke koje se obradjuju su formata Ime Prezime
139     BrojTelefona */

140  /* Preskacu se eventualne praznine sa pocetka linije datoteke */
141  while ((c = fgetc(f)) != EOF && isspace(c));

142  /* Prvo procitano slovo upisuje se u ime i prezime */
143  if (!feof(f))
144      ime_i_prezime[i++] = c;

145  /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
146     cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
147     Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
148     procitanog karaktera i da se slucajno ne dodje do kraja datoteke
149     */
150  while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
151      if (!isdigit(c))
152          ime_i_prezime[i++] = c;

153      else if (i > 0)
154          break;
155  }

156  /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
157     blanko karaktera */
158  ime_i_prezime[--i] = '\0';

159  /* I pocinje se sa citanjem broja telefona */
160  i = 0;

161  /* Upisuje se cifra koja je vec procitana */
162  telefon[i++] = c;

163  /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
164     karaktera cije prisustvo nije dozvoljeno u broju telefona */
165  while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
166      if (c == '/' || c == '-' || isdigit(c))
167          telefon[i++] = c;
168      else
169          break;

170  /* Upisuje se terminirajuca nula */
171  telefon[i] = '\0';

172  /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
173  return !feof(f) ? 0 : EOF;
174 }

175 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
176  */

```

```
188  */
189  Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
190  {
191      /* Ako je imenik prazan, završava se sa pretragom */
192      if (koren == NULL)
193          return NULL;
194
195      /* Ako je traženo ime i prezime sadržano u korenu, također se
196         završava sa pretragom */
197      if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
198          return koren;
199
200      /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
201         korenu pretraga se nastavlja levo */
202      if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
203          return pretrazi_imenik(koren->levo, ime_i_prezime);
204
205      else
206          /* u suprotnom, pretraga se nastavlja desno */
207          return pretrazi_imenik(koren->desno, ime_i_prezime);
208  }
209
210  int main(int argc, char **argv)
211  {
212      char ime_datoteke[MAX_IME_DATOTEKE];
213      Cvor *koren = NULL;
214      Cvor *trazeni;
215      FILE *f;
216      char ime_i_prezime[MAX_IME_I_PREZIME];
217      char telefon[MAX_CIFARA];
218      char c;
219      int i;
220
221      /* Učitava se ime datoteke i vrsi se njena priprema za citanje */
222      printf("Unesite ime datoteke: ");
223      scanf("%s", ime_datoteke);
224      if ((f = fopen(ime_datoteke, "r")) == NULL) {
225          fprintf(stderr, "Greska prilikom otvaranja datoteke
226          %s!\n", ime_datoteke);
227          exit(EXIT_FAILURE);
228      }
229
230      /* Podaci se citaju iz datoteke i smestaju u binarno stablo
231         pretrage. */
232      while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
233          dodaj_u_stablo(&koren, ime_i_prezime, telefon);
234
235      /* Zatvara se datoteka */
236      fclose(f);
237
238      /* Omogućava se pretraga imenika */
239      while (1) {
```

```

240     /* Ucitavaja se ime i prezime */
    printf("Unesite ime i prezime: ");
    i = 0;
242     while ((c = getchar()) != '\n')
        ime_i_prezime[i++] = c;
244     ime_i_prezime[i] = '\0';

246     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
        funkcionalnost */
248     if (strcmp(ime_i_prezime, "KRAJ") == 0)
        break;

250     /* Inace se ispisuje rezultat pretrage */
252     trazeni = pretrazi_imenik(koren, ime_i_prezime);
    if (trazeni == NULL)
254         printf("Broj nije u imeniku!\n");
    else
256         printf("Broj je: %s \n", trazeni->telefon);
}

258     /* Oslobadja se memorija zauzeta imenikom */
260     oslobodi_stablo(&koren);

262     return 0;
}

```

Rešenje 4.17

```

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

5  #define MAX 51

7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
   studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9  jezika i redom pokazivace na levo i desno podstablo */
typedef struct cvor_stabla {
11     char ime[MAX];
    char prezime[MAX];
13     double uspeh;
    double matematika;
15     double jezik;
    struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
} Cvor;

19

/* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
    double matematika, double jezik)
23 {

```

```
25  /* Alocira se memorija za novi cvor i proverava se uspesnost
    alocacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27  if (novi == NULL)
    return NULL;
29
    /* Inicijalizuju se polja strukture */
31  strcpy(novi->ime, ime);
    strcpy(novi->prezime, prezime);
33  novi->uspeh = uspeh;
    novi->matematika = matematika;
35  novi->jezik = jezik;
    novi->levo = NULL;
37  novi->desno = NULL;

39  /* Vraca se adresa kreiranog cvora */
    return novi;
41 }

43 /* Funkcija kojom se proverava uspesnost alociranja memorije */
void proveri_alokaciju(Cvor * novi_cvor)
45 {
    /* Ukoliko je cvor neuspesno kreiran */
47  if (novi_cvor == NULL) {
    /* Ispisuje se odgovarajuca poruka i prekida se izvorsavanje
    programa */
49  fprintf(stderr, "Malloc greska za novi cvor!\n");
    exit(EXIT_FAILURE);
51  }
53 }

55 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
void oslobodi_stablo(Cvor ** koren)
57 {
    /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
59  if (*koren == NULL)
    return;
61
    /* Inace ... */
63  /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*koren)->levo);
65
    /* Oslobadja se memorija zauzeta desnim podstablom */
67  oslobodi_stablo(&(*koren)->desno);

69  /* Oslobadja se memorija zauzeta korenom */
    free(*koren);
71
    /* Stablo se proglašava praznim */
73  *koren = NULL;
    }
75
```



```

77  /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo */
    void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                        double uspeh, double matematika, double jezik)
79  {
    /* Ako je stablo prazno */
81    if (*koren == NULL) {
        /* Kreira se novi cvor */
83        Cvor *novi = napravi_cvor(ime, prezime, uspeh, matematika, jezik)
        ;
        prover_i_alokaciju(novi);

85        /* I proglašava se korenom stabla */
87        *koren = novi;

89        return;
    }

91    /* Inace, dodaje se cvor u stablo tako da bude sortirano po ukupnom
93    broju poena */
    if (uspeh + matematika + jezik >
95        (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
        dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
97        matematika, jezik);
    else
99        dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
        matematika, jezik);
101 }

103 /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
    polozili jednaka 0 ispisuju se informacije o uenicima koji nisu
105 polozili prijemni, a ako je vrednost argumenta razlicita od nule,
    ispisuju se informacije o uenicima koji su polozili prijemni */
107 void stampaj(Cvor * koren, int polozili)
    {
109        /* Stablo je prazno - prekida se sa ispisom */
        if (koren == NULL)
111            return;

113        /* Stampaju se informacije iz levog podstabla */
        stampaj(koren->levo, polozili);

115        /* Stampaju se informacije iz korenog cvora */
117        if (polozili && koren->matematika + koren->jezik >= 10)
            printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
119                koren->prezime, koren->uspeh, koren->matematika,
                koren->jezik,
121                koren->uspeh + koren->matematika + koren->jezik);
        else if (!polozili && koren->matematika + koren->jezik < 10)
123            printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
                koren->prezime, koren->uspeh, koren->matematika,
                koren->jezik,
125                koren->uspeh + koren->matematika + koren->jezik);
    }

```

```

127      /* Stampaju se informacije iz desnog podstabla */
129      stampaj(koren->desno, položili);
131  }
133  /* Funkcija koja određuje koliko studenata nije položilo prijemni
134     ispit */
135  int nisu_položili(Cvor * koren)
136  {
137      /* Ako je stablo prazno, broj onih koji nisu položili je 0 */
138      if (koren == NULL)
139          return 0;
140
141      /* Pretraga se vrši i u levom i u desnom podstablu - ako uslov za
142         polaganje nije ispunjen za koreni cvor, broj studenata se
143         uvecava za 1 */
144      if (koren->matematika + koren->jezik < 10)
145          return 1 + nisu_položili(koren->levo) +
146              nisu_položili(koren->desno);
147
148      return nisu_položili(koren->levo) + nisu_položili(koren->desno);
149  }
150
151  int main(int argc, char **argv)
152  {
153      FILE *in;
154      Cvor *koren;
155      char ime[MAX], prezime[MAX];
156      double uspeh, matematika, jezik;
157
158      /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
159      in = fopen("prijemni.txt", "r");
160      if (in == NULL) {
161          fprintf(stderr, "Greska prilikom otvaranja datoteke!\n");
162          exit(EXIT_FAILURE);
163      }
164
165      /* Citanje podataka i dodavanje u stablo */
166      koren = NULL;
167      while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
168                  &matematika, &jezik) != EOF) {
169          dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik);
170      }
171
172      /* Zatvaranje datoteke */
173      fclose(in);
174
175      /* Stampaju se prvo podaci o ucenicima koji su položili prijemni */
176      stampaj(koren, 1);
177
178      /* Linij se iscrtava samo ako postoje učenici koji nisu položili
179         prijemni */

```

```

179     if (nisu_položili(koren) != 0)
180         printf("-----\n");
181
182     /* Stampaju se podaci o ucenicima koji nisu položili prijemni */
183     stampaj(koren, 0);
184
185     /* Oslobadja se memorija zauzeta stablom */
186     oslobodi_stablo(&koren);
187
188     return 0;
189 }

```

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 rođenja i redom pokazuje 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;
}

```

```
38  /* Funkcija koja proverava uspesnost alokacije */
40  void proveri_alokaciju(Cvor * novi_cvor)
41  {
42      /* Ako memorija nije uspesno alocirana */
43      if (novi_cvor == NULL) {
44          /* Ispisuje se poruka i prekida se sa izvršavanjem programa */
45          fprintf(stderr, "Malloc greska za novi cvor!\n");
46          exit(EXIT_FAILURE);
47      }
48  }

49  /* Funkcija koja oslobadja memoriju zauzetu stablom */
50  void oslobodi_stablo(Cvor ** koren)
51  {
52      /* Stablo je prazno */
53      if (*koren == NULL)
54          return;
55
56      /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
57      if ((*koren)->levo)
58          oslobodi_stablo(&(*koren)->levo);
59
60      /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
61      if ((*koren)->desno)
62          oslobodi_stablo(&(*koren)->desno);
63
64      /* Oslobadja se memorija zauzeta korenom */
65      free(*koren);
66
67      /* Proglasava se stablo praznim */
68      *koren = NULL;
69  }

70
71  /* Funkcija koja dodaje novi cvor u stablo - stablo treba da bude
72     uredjeno po datumu - prvo po mesecu, a zatim po danu */
73  void dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
74                     int dan, int mesec)
75  {
76      /* Ako je stablo prazno */
77      if (*koren == NULL) {
78
79          /* Kreira se novi cvor */
80          Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
81          proveri_alokaciju(novi_cvor);
82
83          /* I proglasava se korenom */
84          *koren = novi_cvor;
85
86          return;
87      }
88  }
```

```

90  /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
    meseca */
92  if (mesec < (*koren)->mesec)
    dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
94  else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
    dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
96  else
    dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
98  }

100 /* Funkcija vrši pretragu stabla i vraća cvor sa traženim datumom.
    */
    Cvor *pretrazi(Cvor * koren, int dan, int mesec)
102 {
    /* Stablo je prazno, obustavlja se pretraga */
104     if (koren == NULL)
        return NULL;

106     /* Ako je traženi datum u korenu */
108     if (koren->dan == dan && koren->mesec == mesec)
        return koren;

110     /* Ako je mesec traženog datuma manji od meseca sadržanog u korenu
    ili ako su meseci isti ali je dan traženog datuma manji od
    aktuelnog datuma, pretražuje se levo podstablo - pre toga se
    svakako proverava da li leva grana postoji - ako ne postoji
    treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
    */
116     if (mesec < koren->mesec
        || (mesec == koren->mesec && dan < koren->dan)) {
118         if (koren->levo == NULL)
            return koren;
120         else
            return pretrazi(koren->levo, dan, mesec);
122     }

124     /* Inace se nastavlja pretraga u desnom delu */
    return pretrazi(koren->desno, dan, mesec);
126 }

128 /* Funkcija koja pronalazi najmanji datum u stablu */
    Cvor *pronadji_najmanji_datum(Cvor * koren)
130 {
    /* Stablo je prazno, obustavlja se pretraga */
132     if (koren == NULL)
        return NULL;

134     /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
    sadrzi najmanji datum */
136     if (koren->levo == NULL)
        return koren;
138     else

```

```
140     /* Inace, trazimo manji datum u levom podstablu */
141     return pronadji_najmanji_datum(koren->levo);
142 }
143
144 /* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
145    */
146 void datum_u_nisku(int dan, int mesec, char datum[])
147 {
148     if (dan < 10) {
149         datum[0] = '0';
150         datum[1] = dan + '0';
151     } else {
152         datum[0] = dan / 10 + '0';
153         datum[1] = dan % 10 + '0';
154     }
155     datum[2] = '.';
156
157     if (mesec < 10) {
158         datum[3] = '0';
159         datum[4] = mesec + '0';
160     } else {
161         datum[3] = mesec / 10 + '0';
162         datum[4] = mesec % 10 + '0';
163     }
164     datum[5] = '.';
165     datum[6] = '\\0';
166 }
167
168 int main(int argc, char **argv)
169 {
170     FILE *in;
171     Cvor *koren;
172     Cvor *slavljenik;
173     char ime[MAX_NISKA], prezime[MAX_NISKA];
174     int dan, mesec;
175     char datum[7];
176
177     /* Provera da li je zadato ime ulazne datoteke */
178     if (argc < 2) {
179         /* Ako nije, ispisuje se poruka i prekida se sa izvršavanjem
180            programa */
181         printf("Nedostaje ime ulazne datoteke!\\n");
182         return 0;
183     }
184
185     /* Inace, priprema se datoteka za citanje */
186     in = fopen(argv[1], "r");
187     if (in == NULL) {
188         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\\n",
189            argv[1]);
190         exit(EXIT_FAILURE);
191     }
```

```
192  /* I stablo se popunjava podacima */
193  koren = NULL;
194  while (fscanf
195      (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
196      dodaj_u_stablo(&koren, ime, prezime, dan, mesec);

198  /* Datoteka se zatvara */
199  fclose(in);

200
201  /* Omogucuje se pretraga podataka */
202  while (1) {

203
204      /* Ucitava se novi datum */
205      printf("Unesite datum: ");
206      if (scanf("%d.%d.", &dan, &mesec) == EOF)
207          break;
208
209      /* Pretrazuje se stablo */
210      slavljenik = pretrazi(koren, dan, mesec);

211
212      /* Ispisuju se pronadjeni podaci */

213
214      /* Ako slavljenik nije pronadjen, to moze znaci da: */
215      /* 1. drvo je prazno */
216      if (slavljenik == NULL && koren == NULL) {
217          printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
218          continue;
219      }
220      /* 2. posle datuma koji je unesen, nema podataka u stablu - u
221      ovom slucaju se pretraga vrsi pocevsi od naredne godine i
222      ispisuje se najmanji datum */
223      if (slavljenik == NULL) {
224          slavljenik = pronadji_najmanji_datum(koren);
225          datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
226          printf("Slavljenik: %s %s %s\n", slavljenik->ime,
227              slavljenik->prezime, datum);
228          continue;
229      }

230
231      /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
232      /* 1. Pronadjeni su tacni podaci */
233      if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
234          printf("Slavljenik: %s %s\n", slavljenik->ime,
235              slavljenik->prezime);
236          continue;
237      }

238
239      /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
240      datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
241      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
242          slavljenik->prezime, datum);
```

```
244     }
245     /* Oslobadja se memorija zauzeta stablom */
246     oslobodi_stablo(&koren);
247
248     return 0;
249 }
```

Rešenje 4.19

```
1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1
3
4  /* Struktura kojom se predstavlja cvor binarnog pretraživackog stabla */
5  typedef struct cvor {
6      int broj;
7      struct cvor *levo, *desno;
8  } Cvor;
9
10 /* b) Funkcija koja alokira memoriju za novi cvor stabla,
11     inicijalizuje polja strukture i vraća pokazivac na novi cvor */
12 Cvor *napravi_cvor(int broj);
13
14 /* Funkcija koja proverava uspesnost kreiranja novog cvora stabla */
15 void prover_i_alokaciju(Cvor * novi_cvor);
16
17 /* Funkcija koja dodaje zadati broj u stablo */
18 void dodaj_u_stablo(Cvor ** adresa_korena, int broj);
19
20 /* Funkcija koja proverava da li se zadati broj nalazi u stablu */
21 Cvor *pretrazi_stablo(Cvor * koren, int broj);
22
23 /* Funkcija koja pronalazi cvor koji sadrži najmanju vrednost u stablu */
24 Cvor *pronadji_najmanji(Cvor * koren);
25
26 /* Funkcija koja pronalazi cvor koji sadrži najveću vrednost u stablu */
27 Cvor *pronadji_najveci(Cvor * koren);
28
29 /* Funkcija koja briše cvor stabla koji sadrži zadati broj. */
30 void obrisi_element(Cvor ** adresa_korena, int broj);
31
32 /* Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo podstablo
33     - Koren - Desno podstablo) */
34 void prikazi_stablo(Cvor * koren);
35
36 /* Funkcija koja oslobadja memoriju zauzetu stablom */
37 void oslobodi_stablo(Cvor ** adresa_korena);
38
```



```

42 #endif

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

Cvor *napravi_cvor(int broj)
{
    /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
        return NULL;
    /* Inicijalizuju se polja novog cvora. */
    novi->broj = broj;
    novi->levo = NULL;
    novi->desno = NULL;
    /* Vraca se adresa novog cvora. */
    return novi;
}

void prover_i_alokaciju(Cvor * novi_cvor)
{
    /* Ukoliko je cvor neuspesno kreiran */
    if (novi_cvor == NULL) {
        /* Ispisuje se odgovarajuca poruka i prekida izvršavanje programa
           */
        fprintf(stderr, "Malloc greska za novi cvor!\n");
        exit(EXIT_FAILURE);
    }
}

void dodaj_u_stablo(Cvor ** koren, int broj)
{
    /* Ako je stablo prazno */
    if (*koren == NULL) {
        /* Kreira se novi cvor */
        Cvor *novi = napravi_cvor(broj);
        prover_i_alokaciju(novi);
        /* I proglašava se korenom stabla */
        *koren = novi;
        return;
    }
    /* U suprotnom se trazi odgovarajuca pozicija za zadati broj: */

    /* Ako je zadata vrednost manja od vrednosti korena */
    if (broj < (*koren)->broj)
        /* Broj se dodaje u levo podstablo */
        dodaj_u_stablo(&(*koren)->levo, broj);
    else
        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
           dodaje u desno podstablo */

```

```

52     dodaj_u_stablo(&(*koren)->desno, broj);
53 }
54 Cvor *pretrazi_stablo(Cvor * koren, int broj)
55 {
56     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
57     if (koren == NULL)
58         return NULL;
59     /* Ako je trazena vrednost sadrzana u korenu */
60     if (koren->broj == broj) {
61         /* Prekida se pretraga */
62         return koren;
63     }
64     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
65     if (broj < koren->broj)
66         /* Pretraga se nastavlja u levom podstablu */
67         return pretrazi_stablo(koren->levo, broj);
68     else
69         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
70         return pretrazi_stablo(koren->desno, broj);
71 }
72
73 Cvor *pronadji_najmanji(Cvor * koren)
74 {
75     /* Ako je stablo prazno, prekida se pretraga */
76     if (koren == NULL)
77         return NULL;
78
79     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
80        levo od njega */
81
82     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
83        najmanju vrednost */
84     if (koren->levo == NULL)
85         return koren;
86
87     /* Inace, pretragu treba nastaviti u levom podstablu */
88     return pronadji_najmanji(koren->levo);
89 }
90
91 Cvor *pronadji_najveci(Cvor * koren)
92 {
93     /* Ako je stablo prazno, prekida se pretraga */
94     if (koren == NULL)
95         return NULL;
96
97     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
98        desno od njega */
99
100    /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
101       najveću vrednost */
102    if (koren->desno == NULL)

```

```

    return koren;
104
    /* Inace, pretragu treba nastaviti u desnom podstablu */
106    return pronadji_najveci(koren->desno);
}
108
void obrisi_element(Cvor ** adresa_korena, int broj)
110 {
    Cvor *pomocni_cvor = NULL;
112
    /* Ako je stablo prazno, brisanje nije primenljivo */
114    if (*adresa_korena == NULL)
        return;
116
    /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
118       stabla, ona se eventualno nalazi u levom podstablu, pa treba
       rekurzivno primeniti postupak na levo podstablo. Koren ovako
120       modifikovanog stabla je nepromenjen. */
    if (broj < (*adresa_korena)->broj) {
122        obrisi_element(&(*adresa_korena)->levo, broj);
        return;
124    }

126    /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
       stabla, ona se eventualno nalazi u desnom podstablu pa treba
128       rekurzivno primeniti postupak na desno podstablo. Koren ovako
       modifikovanog stabla je nepromenjen. */
    if ((*adresa_korena)->broj < broj) {
130        obrisi_element(&(*adresa_korena)->desno, broj);
        return;
132    }
134

    /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
136       jednaka broju koji se brise (tj. slucaj kada treba obrisati
       koren) */
138

    /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
140       prazno stablo (vraca se NULL) */
    if ((*adresa_korena)->levo == NULL
142        && (*adresa_korena)->desno == NULL) {
        free(*adresa_korena);
144        *adresa_korena = NULL;
        return;
146    }

148    /* Ako koren ima samo levog sina, tada se brisanje vrši tako sto se
       brise koren, a novi koren postaje levi sin */
    if ((*adresa_korena)->levo != NULL
150        && (*adresa_korena)->desno == NULL) {
        pomocni_cvor = (*adresa_korena)->levo;
152        free(*adresa_korena);
        *adresa_korena = pomocni_cvor;
154

```

```

156     return;
157 }
158 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako što
159    se briše koren, a novi koren postaje desni sin */
160 if ((*adresa_korena)->desno != NULL
161     && (*adresa_korena)->levo == NULL) {
162     pomocni_cvor = (*adresa_korena)->desno;
163     free(*adresa_korena);
164     *adresa_korena = pomocni_cvor;
165     return;
166 }
167
168 /* Slučaj kada koren ima oba sina - najpre se potraži sledbenik
169    korena (u smislu poretka) u stablu. To je upravo po vrednosti
170    najmanji cvor u desnom podstablu. On se može pronaći npr.
171    funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
172    vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
173    broj koji se briše). Zatim se prosto rekurzivno pozove funkcija
174    za brisanje na desno podstablu. S obzirom da u njemu treba
175    obrisati najmanji element, a on zasigurno ima najviše jednog
176    potomka, jasno je da će njegovo brisanje biti obavljeno na jedan
177    od jednostavnijih načina koji su gore opisani. */
178 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
179 (*adresa_korena)->broj = pomocni_cvor->broj;
180 pomocni_cvor->broj = broj;
181 obriši_element(&(*adresa_korena)->desno, broj);
182 }
183
184 void prikazi_stablo(Cvor * koren)
185 {
186     /* Ako je stablo prazno, prekida se ispis */
187     if (koren == NULL)
188         return;
189
190     /* Prvo se ispisuju svi cvorovi levo od korena */
191     prikazi_stablo(koren->levo);
192
193     /* Zatim se ispisuje vrednost u korenu */
194     printf("%d ", koren->broj);
195
196     /* Na kraju se ispisuju svi cvorovi desno od korena */
197     prikazi_stablo(koren->desno);
198 }
199
200 void oslobodi_stablo(Cvor ** koren)
201 {
202     /* Ako je stablo prazno, nepotrebno je oslobađati memoriju */
203     if (*koren == NULL)
204         return;
205     /* Inace ... */
206     /* Oslobađja se memorija zauzeta levim podstablom */

```

```

    if ((*koren)->levo)
208     oslobodi_stablo(&(*koren)->levo);
    /* Oslobadja se memorija zauzetu desnim podstablom */
210     if ((*koren)->desno)
        oslobodi_stablo(&(*koren)->desno);
212     /* Oslobadja se memorija zauzeta korenom */
        free(*koren);
214     /* Proglasava se stablo praznim */
        *koren = NULL;
216 }

```

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima - pogledati uvodni
    zadatak ove glave */
6  #include "stabla.h"

8
10 /* Funkcija koja proverava da li su dva stabla koja sadrze cele
    brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
    odnosno 0 ako nisu */
12 int identitet(Cvor * koren1, Cvor * koren2)
{
14     /* Ako su oba stabla prazna, jednaka su */
    if (koren1 == NULL && koren2 == NULL)
16         return 1;

18     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednaka */
    if (koren1 == NULL || koren2 == NULL)
20         return 0;

22     /* Ako su oba stabla neprazna i u korenu se nalaze razlicite
        vrednosti, moze se zakljuciti da se razlikuju */
24     if (koren1->broj != koren2->broj)
        return 0;

26     /* Inace, proverava se da li vazi i jednakost levih i desnih
        podstabala */
28     return (identitet(koren1->levo, koren2->levo)
        && identitet(koren1->desno, koren2->desno));
30 }

32
34 int main()
{
36     int broj;
    Cvor *koren1, *koren2;

38     /* Ucitavaju se elementi prvog stabla */
    koren1 = NULL;
40     printf("Prvo stablo: ");
    scanf("%d", &broj);

```

```
42 while (broj != 0) {
    dodaj_u_stablo(&koren1, broj);
44     scanf("%d", &broj);
}

46 /* Ucitavaju se elementi drugog stabla */
48 koren2 = NULL;
printf("Drugo stablo: ");
50 scanf("%d", &broj);
while (broj != 0) {
52     dodaj_u_stablo(&koren2, broj);
    scanf("%d", &broj);
54 }

56 /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
    njen rezultat. */
58 if (identitet(koren1, koren2))
    printf("Stabla jesu identicna.\n");
60 else
    printf("Stabla nisu identicna.\n");
62
64 /* Oslobadja se memorija zauzeta stablima */
oslobodi_stablo(&koren1);
oslobodi_stablo(&koren2);
66
68 return 0;
}
```

Rešenje 4.20

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

4 /* Uklucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6
/* Funkcija kreira novo stablo identicno stablu koje je dato korenom.
8 */
void kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
10 {
    /* Izlaz iz rekurzije */
12     if (koren == NULL) {
        *duplikat = NULL;
14         return;
    }

16     /* Duplira se koren stabla i postavlja da bude koren novog stabla
        */
18     *duplikat = napravi_cvor(koren->broj);
    proveri_alokaciju(*duplikat);
20 }
```

```

22  /* Rekurzivno se duplira levo podstablo i njegova adresa se cuva u
    pokazivacu na levo podstablo korena duplikata. */
    kopiraj_stablo(koren->levo, &(*duplikat)->levo);
24
    /* Rekurzivno se duplira desno podstablo i njegova adresa se cuva u
    pokazivacu na desno podstablo korena duplikata. */
    kopiraj_stablo(koren->desno, &(*duplikat)->desno);
26
28 }

30 /* Funkcija izracunava uniju dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
32 void kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
    {
34     /* Ako drugo stablo nije prazno */
    if (koren2 != NULL) {
36         /* Dodaje se njegov koren u prvo stablo */
        dodaj_u_stablo(adresa_korena1, koren2->broj);
38
        /* Rekurzivno se racuna unija levog i desnog podstabla drugog
        stabla sa prvim stablom */
        kreiraj_uniju(adresa_korena1, koren2->levo);
        kreiraj_uniju(adresa_korena1, koren2->desno);
40
    }
42
44 }

46 /* Funkcija izracunava presek dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
48 void kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
    {
50     /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
    if (*adresa_korena1 == NULL)
52         return;

54     /* 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 */
56     kreiraj_presek(&(*adresa_korena1)->levo, koren2);
    kreiraj_presek(&(*adresa_korena1)->desno, koren2);
58
60     /* 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)
62         obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
64
66 }

68 /* Funkcija izracunava razliku dva stabla - rezultujuce stablo se
    dobija modifikacijom prvog stabla */
    void kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
70     {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
    if (*adresa_korena1 == NULL)
72

```

```
    return;

74
/* Inace... */
76
/* Kreira se razlika levog i desnog podstabla sa drugim stablom,
   tj. iz levog i desnog podstabla prvog stabla se brisu svi oni
78   elementi koji postoje i u drugom stablu */
kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
80
kreiraj_razliku(&(*adresa_korena1)->desno, koren2);

82
/* Ako se koren prvog stabla nalazi i u drugom stablu tada se isti
   uklanja iz prvog stabla */
84
if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
    obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
86
}

88
int main()
{
90
    Cvor *koren1;
    Cvor *koren2;
92
    Cvor *pomocni = NULL;
    int n;

94

    /* Ucitavaju se elementi prvog stabla */
96
    koren1 = NULL;
    printf("Prvo stablo: ");
98
    while (scanf("%d", &n) != EOF) {
        dodaj_u_stablo(&koren1, n);
100
    }

102

    /* Ucitavaju se elementi drugog stabla */
    koren2 = NULL;
104
    printf("Drugo stablo: ");
    while (scanf("%d", &n) != EOF) {
106
        dodaj_u_stablo(&koren2, n);
    }

108

    /* Kreira se unija stabala: prvo se napravi kopija prvog stabla
       kako bi se isto moglo iskoristiti i za preostale operacije */
110
    kopiraj_stablo(koren1, &pomocni);
    kreiraj_uniju(&pomocni, koren2);
112
    printf("Unija: ");
    prikazi_stablo(pomocni);
114
    putchar('\n');

116

    /* Oslobadja se stablo sa rezultatom operacije */
118
    oslobodi_stablo(&pomocni);

120

    /* Kreira se presek stabala: prvo se napravi kopija prvog stabla
       kako bi se isto moglo iskoristiti i za preostale operacije */
122
    kopiraj_stablo(koren1, &pomocni);
    kreiraj_presek(&pomocni, koren2);
124
    printf("Presek: ");
```



```

126 prikazi_stablo(pomocni);
    putchar('\n');

128 /* Oslobadja se stablo sa rezultatom operacije */
    oslobodi_stablo(&pomocni);

130
132 /* Kreira se razlika stabala: prvo se napravi kopija prvog stabla
    kako bi se isto moglo iskoristiti i za preostale operacije; */
    kopiraj_stablo(koren1, &pomocni);
134 kreiraj_razliku(&pomocni, koren2);
    printf("Razlika: ");
136 prikazi_stablo(pomocni);
    putchar('\n');

138
140 /* Oslobadja se stablo sa rezultatom operacije */
    oslobodi_stablo(&pomocni);

142 /* Oslobadjaju se i polazna stabla */
    oslobodi_stablo(&koren2);
144 oslobodi_stablo(&koren1);

146 return 0;
}

```

Rešenje 4.21

```

1  #include <stdio.h>
    #include <stdlib.h>
3
    /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
7
    #define MAX 50
9
    /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
    cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11 su smestene u niz. */
    int kreiraj_niz(Cvor * koren, int a[])
13 {
    int r, s;
15
    /* Stablo je prazno - u niz je smesteno 0 elemenata */
17 if (koren == NULL)
    return 0;
19
    /* Dodaju se u niz elementi iz levog podstabla */
21 r = kreiraj_niz(koren->levo, a);

23 /* Tekuca vrednost promenljive r je broj elemenata koji su upisani
    u niz i na osnovu nje se moze odrediti indeks novog elementa */
25

```

```

27  /* Smesta se vrednost iz korena */
    a[r] = koren->broj;

29  /* Dodaju se elementi iz desnog podstabla */
    s = kreiraj_niz(koren->desno, a + r + 1);

31

33  /* Racuna se indeks na koji treba smestiti naredni element */
    return r + s + 1;
}

35

37  /* Funkcija sortira niz tako sto najpre elemente niza smesti u
    stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva u
    desno.

39

41  Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
    sto je to slucaj sa ostalim opisanim algoritmima sortiranja jer se
    sortiranje vrši u pomocnoj dinamičkoj strukturi, a ne razmenom
    elemenata niza. */
43  void sortiraj(int a[], int n)
44  {
45      int i;
46      Cvor *koren;

49      /* Kreira se stablo smestanjem elemenata iz niza u stablo */
      koren = NULL;
      for (i = 0; i < n; i++)
          dodaj_u_stablo(&koren, a[i]);

53

55      /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
          a */
      kreiraj_niz(koren, a);

57

59      /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
          */
      oslobodi_stablo(&koren);
  }

61

63  int main()
64  {
65      int a[MAX];
66      int n, i;

67      /* Ucitavaju se dimenzija i elementi niza */
      printf("n: ");
      scanf("%d", &n);
      if (n < 0 || n > MAX) {
91          printf("Greska: pogresna dimenzija niza!\n");
          return 0;
93      }

95      printf("a: ");
      for (i = 0; i < n; i++)

```

```

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

79     /* Poziva se funkcija za sortiranje */
    sortiraj(a, n);

81     /* Ispisuje se rezultat */
83     for (i = 0; i < n; i++)
        printf("%d ", a[i]);
85     printf("\n");

87     return 0;
}

```

Rešenje 4.22

```

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

3

   /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  /* a) Funkcija koja izracunava broj cvorova stabla */
   int broj_cvorova(Cvor * koren)
9  {
   /* Ako je stablo prazno, broj cvorova je nula */
11     if (koren == NULL)
        return 0;

13     /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15     levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
        zato sto treba racunati i koren */
17     return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
}

19

   /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
   {
23     /* Ako je stablo prazno, broj listova je nula */
        if (koren == NULL)
25         return 0;

27     /* Proverava se da li je tekuci cvor list */
        if (koren->levo == NULL && koren->desno == NULL)
29         /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
            uvecati broj listova za 1 */
31         return 1;

33     /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
        */
35     return broj_listova(koren->levo) + broj_listova(koren->desno);
}

```

```
37 /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
39 void pozitivni_listovi(Cvor * koren)
40 {
41     /* Slučaj kada je stablo prazno */
42     if (koren == NULL)
43         return;
44
45     /* Ako je cvor list i sadrži pozitivnu vrednost */
46     if (koren->levo == NULL && koren->desno == NULL && koren->broj > 0)
47         /* Stampa se */
48         printf("%d ", koren->broj);
49
50     /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
51     pozitivni_listovi(koren->levo);
52     pozitivni_listovi(koren->desno);
53 }
54
55 /* d) Funkcija koja izracunava zbir cvorova stabla */
56 int zbir_svih_cvorova(Cvor * koren)
57 {
58     /* Ako je stablo prazno, zbir cvorova je 0 */
59     if (koren == NULL)
60         return 0;
61
62     /* Inace, zbir cvorova stabla izracunava se kao zbir korena i svih
63     elemenata u podstablima */
64     return koren->broj + zbir_svih_cvorova(koren->levo) +
65            zbir_svih_cvorova(koren->desno);
66 }
67
68 /* e) Funkcija koja izracunava najveći element stabla */
69 Cvor *najveci_element(Cvor * koren)
70 {
71     /* Ako je stablo prazno, obustavlja se pretraga */
72     if (koren == NULL)
73         return NULL;
74
75     /* Zbog prirode pretraživackog stabla, vrednosti veće od korena se
76     nalaze u desnom podstablu */
77
78     /* Ako desnog podstabla nema */
79     if (koren->desno == NULL)
80         /* Najveća vrednost je koren */
81         return koren;
82
83     /* Inace, najveća vrednost se traži desno */
84     return najveci_element(koren->desno);
85 }
86
87 /* f) Funkcija koja izracunava dubinu stabla */
88 int dubina_stabla(Cvor * koren)
```

```

89 {
90     /* Dubina praznog stabla je 0 */
91     if (koren == NULL)
92         return 0;
93
94     /* Izracunava se dubina levog podstabla */
95     int dubina_levo = dubina_stabla(koren->levo);
96
97     /* Izracunava se dubina desnog podstabla */
98     int dubina_desno = dubina_stabla(koren->desno);
99
100    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
101       jer se racuna i koren */
102    return dubina_levo >
103           dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
104 }
105
106 /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
107 int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
108 {
109     /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazene
110        nivoa */
111
112     /* Ako nema vise cvorova, nema spustanja niz stablo */
113     if (koren == NULL)
114         return 0;
115
116     /* Ako se stiglo do trazene nivoa, vraca se 1 - to ce kasnije zbog
117        rekurzivnih poziva uvecati broj cvorova za 1 */
118     if (i == 0)
119         return 1;
120
121     /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
122        */
123     return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
124         + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
125 }
126
127 /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
128 void ispis_nivo(Cvor * koren, int i)
129 {
130     /* Ideja je slicna ideji iz prethodne funkcije */
131
132     /* Nema vise cvorova, nema spustanja kroz stablo */
133     if (koren == NULL)
134         return;
135
136     /* Ako se stiglo do trazene nivoa - ispisuje se vrednost */
137     if (i == 0) {
138         printf("%d ", koren->broj);
139         return;
140     }

```

```
141  /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
    desnom podstablu */
143  ispis_nivo(koren->levo, i - 1);
    ispis_nivo(koren->desno, i - 1);
145  }

147  /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
    stabla */
Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
149  {
    /* Ako je stablo prazno, obustavlja se pretraga */
151    if (koren == NULL)
        return NULL;

153    /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
155    if (i == 0)
        return koren;

157    /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
159    Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);

161    /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
    Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);

163    /* Trazi se i vraća maksimum izracunatih vrednosti */
165    if (a == NULL && b == NULL)
        return NULL;
    if (a == NULL)
167        return b;
    if (b == NULL)
169        return a;
171    return a->broj > b->broj ? a : b;
}

173

175  /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
177  {
    /* Ako je stablo prazno, zbir je nula */
179    if (koren == NULL)
        return 0;

181    /* Ako se stiglo do trazenog nivoa, vraća se vrednost */
    if (i == 0)
183        return koren->broj;

185    /* Inace, spustanje se nastavlja za jedan nivo nize i traže se sume
    iz levog i desnog podstabla */
187    return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
        + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
189  }

191
```

```

193  /* 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)
195  {
      /* Ako je stablo prazno, zbir je nula */
197      if (koren == NULL)
          return 0;

199      /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
      prirode pretrazivackog stabla treba obici i levo i desno
      podstablo */
201      if (koren->broj <= x)
          return koren->broj + zbir_manjih_od_x(koren->levo, x) +
203              zbir_manjih_od_x(koren->desno, x);

205      /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
      medju njima jedino moze biti onih koje zadovoljavaju uslov */
207      return zbir_manjih_od_x(koren->levo, x);
209  }

211  int main(int argc, char **argv)
212  {
213      /* Analiza argumenata komandne linije */
214      if (argc != 3) {
215          fprintf(stderr,
216              "Greska! Program se poziva sa: ./a.out nivo
              broj_za_pretragu\n");
217          exit(EXIT_FAILURE);
218      }
219      int i = atoi(argv[1]);
220      int x = atoi(argv[2]);

221      /* Kreira se stablo */
222      Cvor *koren = NULL;
223      int broj;
224      while (scanf("%d", &broj) != EOF)
225          dodaj_u_stablo(&koren, broj);

226      /* ispisuju se rezultati rada funkcija */
227      printf("Broj cvorova: %d\n", broj_cvorova(koren));
228      printf("Broj listova: %d\n", broj_listova(koren));
229      printf("Pozitivni listovi: ");
230      pozitivni_listovi(koren);
231      printf("\n");
232      printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
233      if (najveci_element(koren) == NULL)
234          printf("Najveci element: ne postoji\n");
235      else
236          printf("Najveci element: %d\n", najveci_element(koren)->broj);
237      printf("Dubina stabla: %d\n", dubina_stabla(koren));
238  }

```

```
243 printf("Broj cvorova na %d. nivou: %d\n", i,  
        broj_cvorova_na_itom_nivou(koren, i));  
245 printf("Elementi na %d. nivou: ", i);  
    ispis_nivo(koren, i);  
247 printf("\n");  
    if (najveci_element_na_itom_nivou(koren, i) == NULL)  
249         printf("Nema elemenata na %d. nivou!\n", i);  
    else  
251         printf("Maksimalni element na %d. nivou: %d\n", i,  
                najveci_element_na_itom_nivou(koren, i)->broj);  
253  
    printf("Zbir elemenata na %d. nivou: %d\n", i,  
255         zbir_cvorova_na_itom_nivou(koren, i));  
    printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,  
257         zbir_manjih_od_x(koren, x));  
  
259 /* Oslobadja se memorija zauzeta stablom */  
    oslobodi_stablo(&koren);  
261  
    return 0;  
263 }
```

Rešenje 4.23

```
#include<stdio.h>  
2 #include<stdlib.h>  
  
4 /* Ukljucuje se biblioteka za rad sa stablima */  
#include "stabla.h"  
6  
/* Funkcija koja izracunava dubinu stabla */  
8 int dubina_stabla(Cvor * koren)  
{  
10     /* Dubina praznog stabla je 0 */  
    if (koren == NULL)  
12         return 0;  
  
14     /* Izracunava se dubina levog podstabla */  
    int dubina_levo = dubina_stabla(koren->levo);  
16  
    /* Izracunava se dubina desnog podstabla */  
18     int dubina_desno = dubina_stabla(koren->desno);  
  
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje  
        jer se racuna i koren */  
22     return dubina_levo >  
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;  
24 }  
  
26 /* Funkcija koja ispisuje sve elemente na i-tom nivou */  
void ispisi_nivo(Cvor * koren, int i)
```



```

28 {
30     /* Ideja je slicna ideji iz prethodne funkcije */
31     /* Nema vise cvorova, nema spustanja niz stablo */
32     if (koren == NULL)
33         return;
34
35     /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
36     if (i == 0) {
37         printf("%d ", koren->broj);
38         return;
39     }
40     /* Inace, vrshi se spustanje za jedan nivo nize i u levom i u desnom
41         podstablu */
42     ispisi_nivo(koren->levo, i - 1);
43     ispisi_nivo(koren->desno, i - 1);
44 }
45
46 /* Funkcija koja ispisuje stablo po nivoima */
47 void ispisi_stablo_po_nivoima(Cvor * koren)
48 {
49     int i;
50
51     /* Prvo se izracunava dubina stabla */
52     int dubina;
53     dubina = dubina_stabla(koren);
54
55     /* Ispisuje se nivo po nivo stabla */
56     for (i = 0; i < dubina; i++) {
57         printf("%d. nivo: ", i);
58         ispisi_nivo(koren, i);
59         printf("\n");
60     }
61 }
62
63 int main(int argc, char **argv)
64 {
65     Cvor *koren;
66     int broj;
67
68     /* Citaju se vrednosti sa ulaza i dodaju se u stablo */
69     koren = NULL;
70     while (scanf("%d", &broj) != EOF) {
71         dodaj_u_stablo(&koren, broj);
72     }
73
74     /* Ispisuje se stablo po nivoima */
75     ispisi_stablo_po_nivoima(koren);
76
77     /* Oslobadja se memorija zauzeta stablom */
78     oslobodi_stablo(&koren);
79
80     return 0;

```

80 | }

Rešenje 4.25

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  /* Uključuje se biblioteka za rad sa stablima */
5  #include "stabla.h"
6
7  /* Funkcija koja izračunava dubinu stabla */
8  int dubina_stabla(Cvor * koren)
9  {
10     /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
12         return 0;
13
14     /* Izračunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izračunava se dubina desnog podstabla */
18     int dubina_desno = dubina_stabla(koren->desno);
19
20     /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
21     jer se racuna i koren */
22     return dubina_levo >
23         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24 }
25
26 /* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
27 stablo */
28 int avl(Cvor * koren)
29 {
30     int dubina_levo, dubina_desno;
31
32     /* Ako je stablo prazno, zaustavlja se brojanje */
33     if (koren == NULL) {
34         return 0;
35     }
36
37     /* Izračunava se dubina levog podstabla korena */
38     dubina_levo = dubina_stabla(koren->levo);
39
40     /* Izračunava se dubina desnog podstabla korena */
41     dubina_desno = dubina_stabla(koren->desno);
42
43     /* Ako je uslov za AVL stablo ispunjen */
44     if (abs(dubina_desno - dubina_levo) <= 1) {
45         /* Racuna se broj AVL cvorova u levom i desnom podstablu i
46         uvecava za jedan iz razloga sto koren ispunjava uslov */
47         return 1 + avl(koren->levo) + avl(koren->desno);
48     }
49 }
```

```

    } else {
49      /* Inace, racuna se samo broj AVL cvorova u podstablima */
      return avl(koren->levo) + avl(koren->desno);
51    }
  }
53
55  int main(int argc, char **argv)
  {
    Cvor *koren;
57    int broj;

59    /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo */
    koren = NULL;
61    while (scanf("%d", &broj) != EOF) {
      dodaj_u_stablo(&koren, broj);
63    }

65    /* Racuna se i ispisuje broj AVL cvorova */
    printf("%d\n", avl(koren));
67

69    /* Oslobadja se memorija zauzeta stablom */
    oslobodi_stablo(&koren);

71    return 0;
  }

```

Rešenje 4.26

```

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

4  /* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

6

8  /* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
   brojeva heap. Ideja koja ce biti implementirana u osnovi ima
   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
10  desnog podstabla - ako je vrednost u korenu veca od izracunatih
   vrednosti uoceni fragment stabla zadovoljava uslov za heap. Zato
12  ce funkcija vratiti maksimalne vrednosti iz uocenog podstabala ili
   vrednost -1 ukoliko se zakljuci da stablo nije heap. */
14  int heap(Cvor * koren)
  {
16    int max_levo, max_desno;

18    /* Prazno sablo je heap - kao rezultat se vraca 0 kao najmanji
       pozitivan broj */
20    if (koren == NULL) {
      return 0;
22    }

    /* Ukoliko je stablo list... */

```

```
24  if (koren->levo == NULL && koren->desno == NULL) {
    /* Vraca se njegova vrednost */
26  return koren->broj;
    }
28  /* Inace... */

30  /* Proverava se svojstvo za levo podstablo. */
    max_levo = heap(koren->levo);
32
34  /* Proverava se svojstvo za desno podstablo. */
    max_desno = heap(koren->desno);

36  /* Ako levo ili desno podstablo uocenog cvora nije heap, onda nije
    ni celo stablo. */
38  if (max_levo == -1 || max_desno == -1) {
    return -1;
40  }

42  /* U suprotnom proverava se da li svojstvo vazi za uoceni cvor. */
    if (koren->broj > max_levo && koren->broj > max_desno) {
44      /* Ako vazi, vraca se vrednost korena */
        return koren->broj;
46  }

48  /* U suprotnom zakljucuje se da stablo nije heap */
    return -1;
50 }

52 int main(int argc, char **argv)
{
54     Cvor *koren;
    int heap_indikator;
56
58     /* Kreira se stablo koje sadrzi brojeve 100 19 36 17 3 25 1 2 7 */
    koren = NULL;
    koren = napravi_cvor(100);
60    koren->levo = napravi_cvor(19);
    koren->levo->levo = napravi_cvor(17);
62    koren->levo->levo->levo = napravi_cvor(2);
    koren->levo->levo->desno = napravi_cvor(7);
64    koren->levo->desno = napravi_cvor(3);
    koren->desno = napravi_cvor(36);
66    koren->desno->levo = napravi_cvor(25);
    koren->desno->desno = napravi_cvor(1);
68

70    /* Poziva se funkcija kojom se proverava da li je stablo heap */
    heap_indikator = heap(koren);

72    /* Ispisuje se rezultat */
    if (heap_indikator == -1) {
74        printf("Zadato tablo nije heap\n");
    } else {
```

```
76     printf("Zadato stablo je heap!\n");  
77 }  
78  
79 /* Oslobadja se memorija zauzeta stablom. */  
80 oslobodi_stablo(&koren);  
81  
82 return 0;  
83 }
```


Glava 5

Ispitni rokovi

5.1 Programiranje 2, praktični deo ispita, jun 2015.

Zadatak 5.1 Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost -1 i prekinuti izvršavanje programa.

Primer 1

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

```
ULAZ.TXT
5
Programiranje
Matematika
12345
dInAmiCnArEc
Ispit
```

```
INTERAKCIJA PROGRAMA:
Ispit
Matematika
Programiranje
```

Primer 2

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

```
ULAZ.TXT
2
maksimalano
poena
```

```
INTERAKCIJA PROGRAMA:
```

Primer 3

```

Poziv: ./a.out ulaz.txt
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA PROGRAMA:
-1

```

Primer 4

```

Poziv: ./a.out
INTERAKCIJA PROGRAMA:
-1

```

Zadatak 5.2 Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na n -tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj n , a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test 1

```

ULAZ:
2 8 10 3 6 14 13 7 4 0
IZLAZ:
20

```

Test 2

```

ULAZ:
0 50 14 5 2 4 56 8 52 7 1 0
IZLAZ:
50

```

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A , a zatim i elementi matrice A . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

Test 1

```

ULAZ:
4 5
1 2 3 4 5
-1 2 -3 4 -5
-5 -4 -3 -2 1
-1 0 0 0 0
IZLAZ:
0

```

Test 2

```

ULAZ:
2 3
0 0 -5
1 2 -4
IZLAZ:
2

```

Test 3

```

ULAZ:
-2
IZLAZ:
-1

```


5.2 Programiranje 2, praktični deo ispita, jul 2015.

Zadatak 5.4 Napisati program koji kao prvi argument komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od 127 karaktera.

Potpis funkcije `strstr`:

```
char *strstr(const char *haystack, const char *needle);
```

Funkcija traži prvo pojavljivanje podniske `needle` u nisci `haystack`, i vraća pokazivač na početak podniske, ili `NULL` ako podniska nije pronađena.

Primer 1

```
POZIV: ./a.out ulaz.txt test
ULAZ.TXT
Ovo je test primer.
U njemu se rec test javlja
vise puta. testtesttest
INTERAKCIJA PROGRAMA:
5
```

Primer 2

```
POZIV: ./a.out
INTERAKCIJA PROGRAMA:
(na stderr) -1
```

Primer 3

```
POZIV: ./a.out ulaz.txt foo
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA PROGRAMA:
(na stderr) -1
```

Primer 4

```
POZIV: ./a.out ulaz.txt .
DATOTEKA ULAZ.TXT JE PRAZNA
INTERAKCIJA PROGRAMA:
0
```

Zadatak 5.5 Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac: $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$, gde je s polu-obim trougla). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Primer 1

```

TROUGLOVI.TXT
4
0 0 0 1.2 1 0
0.3 0.3 0.5 0.5 0.9 1
-2 0 0 0 1
-2 0 0 0 1

INTERAKCIJA PROGRAMA:
2 0 2 2 -1 -1
-2 0 0 0 1
0 0 0 1.2 1 0
0.3 0.3 0.5 0.5 0.9 1

```

Primer 2

```

TROUGLOVI.TXT
3
1.2 3.2 1.1 4.3

INTERAKCIJA PROGRAMA:
-1

```

Primer 3

```

DATOTEKA TROUGLOVI.TXT NE POSTOJI

INTERAKCIJA PROGRAMA:
-1

```

Primer 4

```

TROUGLOVI.TXT
0

INTERAKCIJA PROGRAMA:

```

Zadatak 5.6 Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju `int f3(Cvor *koren, int n)` koja u datom stablu prebrojava čvorove na n -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

Test 1

```

ULAZ:
1 5 3 6 1 4 7 9
IZLAZ:
1

```

Test 2

```

ULAZ:
2 5 3 6 1 0 4 7 9
IZLAZ:
2

```

Test 3

```

ULAZ:
0 4 2 5
IZLAZ:
0

```

Test 4

```

ULAZ:
3
IZLAZ:
0

```

Test 5

```

ULAZ:
-1 4 5 1 7
IZLAZ:
0

```

5.3 Programiranje 2, praktični deo ispita, septembar 2015.

Test 1

ULAZ:	6 1
IZLAZ:	3

Test 2

ULAZ:	15 3
IZLAZ:	3758096385

Test 3

ULAZ:	31 100
IZLAZ:	4026531841

Test 4

ULAZ:	4 0
IZLAZ:	4

Test 5

ULAZ:	0 5
IZLAZ:	0

Zadatak 5.8 Napisati funkciju `void dopuni_listu(Cvor** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

```

Test 1
ULAZ:
  1 2 3 4 5
IZLAZ:
  1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00

```

Test 2

ULAZ:	12
IZLAZ:	12.00

Test 3

ULAZ:	<i>prazna lista</i>
IZLAZ:	

Test 4

ULAZ:	13.3	15.8
IZLAZ:	13.30	14.55

319

5 Ispitni rokovi

naka). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati -". Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 4 1 2 3 4 2 1 4 3 3 4 2 1 4 3 1 2 IZLAZ: 10	ULAZ: 3 1 1 1 1 1 1 1 1 1 IZLAZ: 3	ULAZ: 2 1 1 2 2 IZLAZ: -
<i>Test 4</i>	<i>Test 5</i>	<i>Test 6</i>
ULAZ: 2 1 2 1 2 IZLAZ: -	ULAZ: 1 5 IZLAZ: 5	ULAZ: 0 IZLAZ: -1

5.4 Rešenja

Rešenje 5.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define MAX 50
5
6 void greska()
7 {
8     printf("-1\n");
9     exit(EXIT_FAILURE);
10 }
11
12 int main(int argc, char *argv[])
13 {
14     FILE *ulaz;
15     char **linije;
16     int i, j, n;
17
18     /* Proverava argumenata komandne linije. */
19     if (argc != 2) {
20         greska();
21     }
22 }
```

```
22     greska();
23 }
24
25 /* Otvaranje datoteke cije ime je navedeno kao argument komandne
26    linije neposredno nakon imena programa koji se poziva. */
27 ulaz = fopen(argv[1], "r");
28 if (ulaz == NULL) {
29     greska();
30 }
31
32 /* Ucitavanje broja linija. */
33 fscanf(ulaz, "%d", &n);
34
35 /* Alociranje memorije na osnovu ucitanog broja linija. */
36 linije = (char **) malloc(n * sizeof(char *));
37 if (linije == NULL) {
38     greska();
39 }
40 for (i = 0; i < n; i++) {
41     linije[i] = malloc(MAX * sizeof(char));
42     if (linije[i] == NULL) {
43         for (j = 0; j < i; j++) {
44             free(linije[j]);
45         }
46         free(linije);
47         greska();
48     }
49 }
50
51 /* Ucitavanje svih n linija iz datoteke. */
52 for (i = 0; i < n; i++) {
53     fscanf(ulaz, "%s", linije[i]);
54 }
55
56 /* Ispisivanje u odgovarajucem poretку ucitane linije koje
57    zadovoljavaju kriterijum. */
58 for (i = n - 1; i >= 0; i--) {
59     if (isupper(linije[i][0])) {
60         printf("%s\n", linije[i]);
61     }
62 }
63
64 /* Oslobadjanje memorije koja je dinamički alocirana. */
65 for (i = 0; i < n; i++) {
66     free(linije[i]);
67 }
68
69 free(linije);
70
71 /* Zatvaranje datoteku. */
72 fclose(ulaz);
```

```
74     return 0;
    }
```

Rešenje 5.2

```
2  #ifndef __STABLA_H__
   #define __STABLA_H__ 1

4  /* Struktura kojom se predstavlja Cvor stabla */
   typedef struct dcvor {
6      int broj;
       struct dcvor *levo, *desno;
8  } Cvor;

10 /* Funkcija alokira prostor za novi Cvor stabla, inicijalizuje polja
    strukture i vraća pokazivac na nov Cvor */
12 Cvor *napravi_cvor(int b);

14 /* Funkcija oslobadja dinamički alokirani prostor za stablo Nakon
    oslobađanja se u pozivajućoj funkciji koren postavlja NULL, jer
16 je stablo prazno */
   void oslobodi_stablo(Cvor ** adresa_korena);

18 /* Funkcija proverava da li je novi Cvor ispravno alokiran, i nakon
    toga prekida program */
20 void prover_i_alokaciju(Cvor * novi);

22 /* Funkcija dodaje nov Cvor u stablo i azurira vrednost korena stabla
    u pozivajućoj funkciji. */
24 void dodaj_u_stablo(Cvor ** adresa_korena, int broj);
26 #endif
```

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

5  Cvor *napravi_cvor(int b)
   {
7      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
       if (novi == NULL)
9          return NULL;

11     /* Inicijalizacija polja novog Cvora */
       novi->broj = b;
13     novi->levo = NULL;
       novi->desno = NULL;

15     return novi;
17 }
```

```

19 void oslobodi_stablo(Cvor ** adresa_korena)
20 {
21     /* Prazno stablo i nema sta da se oslobadja */
22     if (*adresa_korena == NULL)
23         return;
24
25     /* Rekurzivno se oslobadja najpre levo, a onda i desno podstablo */
26     if ((*adresa_korena)->levo)
27         oslobodi_stablo(&(*adresa_korena)->levo);
28     if ((*adresa_korena)->desno)
29         oslobodi_stablo(&(*adresa_korena)->desno);
30
31     free(*adresa_korena);
32     *adresa_korena = NULL;
33 }
34
35 void prover_i_alokaciju(Cvor * novi)
36 {
37     if (novi == NULL) {
38         fprintf(stderr, "Malloc greska za nov cvor!\n");
39         exit(EXIT_FAILURE);
40     }
41 }
42
43 void dodaj_u_stablo(Cvor ** adresa_korena, int broj)
44 {
45     /* Postojece stablo je prazno */
46     if (*adresa_korena == NULL) {
47         Cvor *novi = napravi_cvor(broj);
48         prover_i_alokaciju(novi);
49         /* Kreirani Cvor novi ce biti od sada koren stabla */
50         *adresa_korena = novi;
51         return;
52     }
53
54     /* Brojevi se smestaju u uredjeno binarno stablo, pa ako je broj
55        koji se ubacuje manji od broja koji je u korenu onda se dodaje u
56        levo podstablo. */
57     if (broj < (*adresa_korena)->broj)
58         dodaj_u_stablo(&(*adresa_korena)->levo, broj);
59     /* Ako je broj manji ili jednak od broja koji je u korenu stabla,
60        dodaje se nov Cvor desno od korena. */
61     else
62         dodaj_u_stablo(&(*adresa_korena)->desno, broj);
63 }

```

```

1  #include <stdio.h>
2  #include "stabla.h"
3
4  int sumirajN(Cvor * koren, int n)
5  {

```

```
7     if (koren == NULL)
8         return 0;
9
10    if (n == 0)
11        return koren->broj;
12
13    return sumirajN(koren->levo, n - 1) + sumirajN(koren->desno, n - 1)
14        ;
15 }
16
17 int main()
18 {
19     Cvor *koren = NULL;
20     int n;
21     int nivo;
22
23     scanf("%d", &nivo);
24
25     while (1) {
26         scanf("%d", &n);
27         /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
28         if (n == 0)
29             break;
30
31         /* Ako nije, dodaje se procitani broj u stablo. */
32         dodaj_u_stablo(&koren, n);
33     }
34
35     /* Ispisuje se rezultat rada trazene funkcije */
36     printf("%d\n", sumirajN(koren, nivo));
37
38     /* Oslobadja se memorija */
39     oslobodi_stablo(&koren);
40
41     return 0;
42 }
```

Rešenje 5.3

```
1 #include <stdio.h>
2 #define MAX 50
3
4 int main()
5 {
6     int m[MAX][MAX];
7     int v, k;
8     int i, j;
9     int max_broj_negativnih, max_indeks_kolone;
10    int broj_negativnih;
11
12    /* Ucitavanje dimenzije matrice */
```



```
scanf("%d", &v);
14 if (v < 0 || v > MAX) {
    fprintf(stderr, "-1\n");
16     return 0;
}

18 scanf("%d", &k);
20 if (k < 0 || k > MAX) {
    fprintf(stderr, "-1\n");
22     return 0;
}

24 /* Ucitavanje elemenata matrice */
26 for (i = 0; i < v; i++) {
    for (j = 0; j < k; j++) {
28         scanf("%d", &m[i][j]);
    }
30 }

32 /* Pronalazenje kolone koja sadrzi najveći broj negativnih
    elemenata */
34 max_indeks_kolone = 0;

36 max_broj_negativnih = 0;
38 for (i = 0; i < v; i++) {
    if (m[i][0] < 0) {
40         max_broj_negativnih++;
    }

42 }

44 for (j = 0; j < k; j++) {
    broj_negativnih = 0;
46     for (i = 0; i < v; i++) {
        if (m[i][j] < 0) {
48             broj_negativnih++;
        }
50         if (broj_negativnih > max_broj_negativnih) {
            max_indeks_kolone = j;
52         }
    }
54 }

56 /* Ispisivanje traženog rezultata */
58 printf("%d\n", max_indeks_kolone);

60 return 0;
}
```

Rešenje 5.4

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAX 128
5
6 int main(int argc, char **argv)
7 {
8     FILE *f;
9     int brojac = 0;
10    char linija[MAX], *p;
11
12    if (argc != 3) {
13        fprintf(stderr, "-1\n");
14        exit(EXIT_FAILURE);
15    }
16
17    /* Otvaranje datoteke ciji je naziv zadat kao argument komandne
18       linije */
19    if ((f = fopen(argv[1], "r")) == NULL) {
20        fprintf(stderr, "-1\n");
21        exit(EXIT_FAILURE);
22    }
23
24    while (fgets(linija, MAX, f) != NULL) {
25        p = linija;
26        while (1) {
27            p = strstr(p, argv[2]);
28            if (p == NULL)
29                break;
30            brojac++;
31            p = p + strlen(argv[2]);
32        }
33    }
34
35    fclose(f);
36
37    printf("%d\n", brojac);
38
39    return 0;
40 }
```

Rešenje 5.5

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 /* Struktura trougao */
6 typedef struct _trougao {
7     double xa, ya, xb, yb, xc, yc;
```

```

9   } trougao;

11  /* Funkcija racuna duzinu duzi */
12  double duzina(double x1, double y1, double x2, double y2)
13  {
14      return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
15  }

17  /* Funkcija racuna povrsinu trougla */
18  double povrsina(trougao t)
19  {
20      double a = duzina(t.xb, t.yb, t.xc, t.yc);
21      double b = duzina(t.xa, t.ya, t.xc, t.yc);
22      double c = duzina(t.xa, t.ya, t.xb, t.yb);
23      double s = (a + b + c) / 2;
24      return sqrt(s * (s - a) * (s - b) * (s - c));
25  }

27  /* Funkcija racuna poredi dva trougla, napisana tako da se moze
28     proslediti funkciji qsort */
29  int poredi(const void *a, const void *b)
30  {
31      trougao x = *(trougao *) a;
32      trougao y = *(trougao *) b;
33      double xp = povrsina(x);
34      double yp = povrsina(y);
35      if (xp < yp)
36          return 1;
37      if (xp > yp)
38          return -1;
39      return 0;
40  }

42  int main()
43  {
44      FILE *f;
45      int n, i;
46      trougao *niz;

47      if ((f = fopen("trouglovi.txt", "r")) == NULL) {
48          fprintf(stderr, "-1\n");
49          exit(EXIT_FAILURE);
50      }

51      if (fscanf(f, "%d", &n) != 1) {
52          fprintf(stderr, "-1\n");
53          exit(EXIT_FAILURE);
54      }

55      if ((niz = malloc(n * sizeof(trougao))) == NULL) {
56          fprintf(stderr, "-1\n");
57          exit(EXIT_FAILURE);
58      }

```

```

    }

61
    for (i = 0; i < n; i++) {
63        if (fscanf(f, "%lf%lf%lf%lf%lf",
                    &niz[i].xa, &niz[i].ya,
65                    &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
        {
            fprintf(stderr, "-1\n");
67            exit(EXIT_FAILURE);
        }
69    }

    qsort(niz, n, sizeof(trougao), &poredi);

71
    for (i = 0; i < n; i++)
73        printf("%g %g %g %g %g %g\n",
75            niz[i].xa, niz[i].ya,
            niz[i].xb, niz[i].yb, niz[i].xc, niz[i].yc);
77
    free(niz);
79    fclose(f);

81    return 0;
}

```

Rešenje 5.6

```

1  #ifndef __STABLA_H__
2  #define __STABLA_H__ 1

4  /* Struktura koja predstavlja cvor stabla, sadrzi vrednost koja se
   cuva i pokazivace na levo i desno podstablo. */
6  typedef struct cvor {
    int vrednost;
8    struct cvor *levi;
    struct cvor *desni;
10 } Cvor;

12 /* Pomocna funkcija za kreiranje cvora. Cvor se kreira dinamicki,
   funkcijom malloc(). U slucaju greske program se prekida i ispisuje
14   se poruka o gresci. U slucaju uspeha inicijalizuje se vrednost
   datim brojem, a pokazivaci na podstabla se inicijalizuju na NULL.
16   Funkcija vraca adresu novokreiranog cvora */
   Cvor *napravi_cvor(int broj);
18

20 /* Funkcija dodaje novi cvor u stablo sa datim korenom. Ukoliko broj
   vec postoji u stablu, ne radi nista. Cvor se kreira funkcijom
   napravi_cvor(). */
22 void dodaj_u_stablo(Cvor ** koren, int broj);

24 /* Funkcija prikazuje stablo s leva u desno (tj. prikazuje elemente u

```

```

    rastucem poretku) */
26 void prikazi_stablo(Cvor * koren);

28 /* Funkcija ucitava stablo sa standardnog ulaza do kraja ulaza i
    vraca pokazican na njegov koren */
30 Cvor *ucitaj_stablo();

32 /* Funkcija oslobadja prostor koji je alociran za cvorove stabla. */
    void oslobodi_stablo(Cvor ** koren);
34
    #endif
}

1 #include <stdio.h>
    #include <stdlib.h>
3 #include "stabla.h"

5 Cvor *napravi_cvor(int broj)
    {
7     /* Dinamicki kreiramo cvor */
        Cvor *novi = (Cvor *) malloc(sizeof(Cvor));

9
        /* U slucaju greske ... */
11     if (novi == NULL) {
            fprintf(stderr, "-1\n");
13         exit(1);
        }

15
        /* Inicijalizacija */
17     novi->vrednost = broj;
        novi->levi = NULL;
19     novi->desni = NULL;

21
        /* Vracamo adresu novog cvora */
        return novi;
23     }

25 void dodaj_u_stablo(Cvor ** koren, int broj)
    {
27     /* Izlaz iz rekurzije: ako je stablo bilo prazno, novi koren je
        upravo novi cvor */
29     if (*koren == NULL) {
            *koren = napravi_cvor(broj);
31         return;
        }

33
        /* Ako je stablo neprazno, i koren sadrzi manju vrednost od datog
        broja, broj se umece u desno podstablo, rekurzivnim pozivom */
35     if ((*koren)->vrednost < broj)
        dodaj_u_stablo(&(*koren)->desni, broj);
37
        /* Ako je stablo neprazno, i koren sadrzi vecu vrednost od datog
        broja, broj se umece u levo podstablo, rekurzivnim pozivom */
39     else if ((*koren)->vrednost > broj)

```

```

41     dodaj_u_stablo(&(*koren)->levi, broj);
42 }
43
44 void prikazi_stablo(Cvor * koren)
45 {
46     /* Izlaz iz rekurzije */
47     if (koren == NULL)
48         return;
49
50     prikazi_stablo(koren->levi);
51     printf("%d ", koren->vrednost);
52     prikazi_stablo(koren->desni);
53 }
54
55 Cvor *ucitaj_stablo()
56 {
57     Cvor *koren = NULL;
58     int x;
59     while (scanf("%d", &x) == 1)
60         dodaj_u_stablo(&koren, x);
61     return koren;
62 }
63
64 void oslobodi_stablo(Cvor ** koren)
65 {
66     /* Izlaz iz rekurzije */
67     if (*koren == NULL)
68         return;
69
70     oslobodi_stablo(&(*koren)->levi);
71     oslobodi_stablo(&(*koren)->desni);
72     free(*koren);
73
74     *koren = NULL;
75 }

```

```

1  #include <stdio.h>
2  #include "stabla.h"
3
4  int f3(Cvor * koren, int n)
5  {
6      if (koren == NULL || n < 0)
7          return 0;
8      if (n == 0) {
9          if (koren->levi == NULL && koren->desni != NULL)
10             return 1;
11         if (koren->levi != NULL && koren->desni == NULL)
12             return 1;
13         return 0;
14     }
15     return f3(koren->levi, n - 1) + f3(koren->desni, n - 1);
16 }

```

```

17 int main()
19 {
    Cvor *koren;
21     int n;

23     scanf("%d", &n);
    koren = ucitaj_stablo();

25     printf("%d\n", f3(koren, n));

27     oslobodi_stablo(&koren);

29     return 0;
31 }

```

Rešenje 5.7

```

#include <stdio.h>

2 unsigned int Rotiraj(unsigned int x, unsigned int n)
4 {
    int i;
6     unsigned int maska = 1;

8     /* Formiranje maske sa n jedinica na kraju 000...00001111 */
    for (i = 1; i < n; i++)
10         maska = (maska << 1) | 1;

12     return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
14 }

16 int main()
18 {
    unsigned int x, n;

20     scanf("%u%u", &x, &n);

22     printf("%u\n", Rotiraj(x, n));

24     return 0;
}

```

Rešenje 5.8

```

1 #ifndef __LISTE_H__
2 #define __LISTE_H__ 1

4 /* Struktura koja predstavlja cvor liste */

```

```

typedef struct cvor {
6     double vrednost;
    struct cvor *sledeci;
8 } Cvor;

10 /* Pomocna funkcija koja kreira cvor. */
Cvor *napravi_cvor(double broj);

12
14 /* Funkcija oslobadja dinamiciku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

16
18 /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
    zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
    adresa_glave. */
20 void proveri_alokaciju(Cvor ** adresa_glave, Cvor * novi);
22
24 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
Cvor *pronadji_poslednji(Cvor * glava);
26
28 /* Funkcija dodaje novi cvor na kraj liste. */
void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj);

30 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
void ispisi_listu(Cvor * glava);
32
34 /* Funkcija koja dopunjuje listu na nacin opisan u zadatku */
void dopuni_listu(Cvor ** adresa_glave);

36 #endif

```

```

#include <stdio.h>
2 #include <stdlib.h>
#include "liste.h"

4
6 /* Pomocna funkcija koja kreira cvor. */
Cvor *napravi_cvor(double broj)
{
8     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
    if (novi == NULL)
10         return NULL;

12     /* inicijalizacija polja u novom cvoru */
    novi->vrednost = broj;
    novi->sledeci = NULL;
14

16     return novi;
18 }

```



```
20  /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
    ciji se pocetni cvor nalazi na adresi adresa_glave. */
22  void oslobodi_listu(Cvor ** adresa_glave)
    {
24      Cvor *pomocni = NULL;

        while (*adresa_glave != NULL) {
26          pomocni = (*adresa_glave)->sledeci;
          free(*adresa_glave);
28          *adresa_glave = pomocni;
        }
30  }

32  /* Funkcija proverava uspesnost alokacije memorije za cvor novi i
    ukoliko alokacija nije bila uspesna, oslobadja se sva prethodno
34  zauzeta memorija za listu cija pocetni cvor se nalazi na adresi
    adresa_glave. */
36  void prover_i_alokaciju(Cvor ** adresa_glave, Cvor * novi)
    {
38      /* Ukoliko je novi NULL */
        if (novi == NULL) {
40          fprintf(stderr, "Neuspela alokacija za nov cvor\n");
          oslobodi_listu(adresa_glave);
42          exit(EXIT_FAILURE);
        }
44  }

46  /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
    ili NULL kao je lista prazna */
48  Cvor *pronadji_poslednji(Cvor * glava)
    {
50      /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
        funkcija vraca NULL. */
52      if (glava == NULL)
          return NULL;

54      while (glava->sledeci != NULL)
56          glava = glava->sledeci;

58      return glava;
    }

60  /* Funkcija dodaje novi cvor na kraj liste. */
62  void dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
    {
64      Cvor *novi = napravi_cvor(broj);
        prover_i_alokaciju(adresa_glave, novi);

66      if (*adresa_glave == NULL) {
68          *adresa_glave = novi;
          return;
70  }
```

```

72  Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
74  }

76  /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
    */
    void ispisi_listu(Cvor * glava)
78  {
        for (; glava != NULL; glava = glava->sledeci)
80            printf("%.2lf ", glava->vrednost);

82        putchar('\n');
    }

84  /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka */
86  void dopuni_listu(Cvor ** adresa_glave)
    {
88        Cvor *tekuci;
        Cvor *novi;
        double aritmeticka_sredina;
90        if (*adresa_glave == NULL || (*adresa_glave)->sledeci == NULL)
92            return;

94        tekuci = *adresa_glave;
        while (tekuci->sledeci != NULL) {
96            aritmeticka_sredina =
                ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
98            novi = napravi_cvor(aritmeticka_sredina);
            prover_i_alokaciju(adresa_glave, novi);

100            novi->sledeci = tekuci->sledeci;
102            tekuci->sledeci = novi;
            tekuci = tekuci->sledeci;
104            tekuci = tekuci->sledeci;
        }

106        return;
108    }

```

```

#include <stdio.h>
2  #include "liste.h"

4  int main()
    {
6        Cvor *glava = NULL;
        double broj;

8        /* Ucitavanje se vrsi do kraja ulaza. Elementi se dodaju na kraj
10        liste! */
        while (scanf("%lf", &broj) > 0)
12            dodaj_na_kraj_liste(&glava, broj);

```

```
14     dopuni_listu(&glava);
16     ispisi_listu(glava);
18     oslobodi_listu(&glava);
20     return 0;
}
```

Rešenje 5.9

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

/* Funkcija proverava da li je magican kvadrat koji joj se
   prosledjuje kao argument. Ukoliko jeste magican funkcija vraca 1,
   inace 0. */
int magicni_kvadrat(int **M, int n)
{
    int i, j;
    int zbir = 0, zbir_pom;

    for (j = 0; j < n; j++)
        zbir += M[0][j];

    for (i = 1; i < n; i++) {
        zbir_pom = 0;
        for (j = 0; j < n; j++)
            zbir_pom += M[i][j];
        if (zbir_pom != zbir)
            return 0;
    }

    for (j = 0; j < n; j++) {
        zbir_pom = 0;
        for (i = 0; i < n; i++)
            zbir_pom += M[i][j];
        if (zbir_pom != zbir)
            return 0;
    }
    return 1;
}

int main()
{
    int n, i, j;
    int **matrica = NULL;
    int zbir = 0;

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

```
40     if (n <= 0) {
42         printf("-1\n");
43         exit(EXIT_FAILURE);
44     }

46     matrica = (int **) malloc(n * sizeof(int *));
47     if (matrica == NULL) {
48         printf("-1\n");
49         exit(EXIT_FAILURE);
50     }

52     for (i = 0; i < n; i++) {
53         matrica[i] = (int *) malloc(n * sizeof(int));

54         if (matrica[i] == NULL) {
56             fprintf(stderr, "-1\n");
57             for (j = 0; j < i; j++)
58                 free(matrica[j]);

60             free(matrica);
61             exit(EXIT_FAILURE);
62         }
63     }

64     for (i = 0; i < n; i++)
65         for (j = 0; j < n; j++)
66             scanf("%d", &matrica[i][j]);

68     if (magicni_kvadrat(matrica, n)) {
70         for (i = 0; i < n; i++)
71             zbir += matrica[0][i];
72         printf("%d\n", zbir);
73     } else
74         printf("-\n");

76     for (j = 0; j < n; j++)
77         free(matrica[j]);

78     free(matrica);

80     return 0;
82 }
```