

Univerzitet u Beogradu
Matematički fakultet

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

**PROGRAMIRANJE 2
Zbirka zadataka sa rešenjima**

**Beograd
2016.**

Autori:

dr Milena Vujošević Janičić, docent na Matematičkom fakultetu u Beogradu

dr Jelena Graovac, docent na Matematičkom fakultetu u Beogradu

Ana Spasić, asistent na Matematičkom fakultetu u Beogradu

Mirko Spasić, asistent na Matematičkom fakultetu u Beogradu

Andželka Zečević, asistent na Matematičkom fakultetu u Beogradu

Nina Radojičić, asistent na Matematičkom fakultetu u Beogradu

PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu

Studentski trg 16, 11000 Beograd

Za izdavača: *prof. dr Zoran Rakić*, dekan

Recenzenti:

dr Gordana Pavlović-Lažetić, redovni profesor na Matematičkom fakultetu u Beogradu

dr Dragan Urošević, naučni savetnik na Matematičkom institutu SANU

Obrada teksta, crteži i korice: *autori*

Štampa: Skripta internacional, Beograd

Tiraž: 150

СИР Каталогизација у публикацији

Народна библиотека Србије, Београд

©2015. Milena Vujošević Janičić, Jelena Graovac, Ana Spasić, Mirko Spasić, Andželka Zečević, Nina Radojičić

Ovo delo zaštićeno je licencom Creative Commons CC BY-NC-ND 4.0 (Attribution-NonCommercial-NoDerivatives 4.0 International License). Detalji licence mogu se videti na veb-adresi <http://creativecommons.org/licenses/by-nc-nd/4.0/>. Dozvoljeno je umnožavanje, distribucija i javno saopštavanje dela, pod uslovom da se navedu imena autora. Upotreba dela u komercijalne svrhe nije dozvoljena. Prerada, preoblikovanje i upotreba dela u sklopu nekog drugog nije dozvoljena.



Sadržaj

1 Uvodni zadaci	2
1.1 Podela koda po datotekama	2
1.2 Algoritmi za rad sa bitovima	6
1.3 Rekurzija	12
1.4 Rešenja	21
2 Pokazivači	70
2.1 Pokazivačka aritmetika	70
2.2 Višedimenzionalni nizovi	74
2.3 Dinamička alokacija memorije	78
2.4 Pokazivači na funkcije	85
2.5 Rešenja	87
3 Algoritmi pretrage i sortiranja	128
3.1 Algoritmi pretrage	128
3.2 Algoritmi sortiranja	134
3.3 Bibliotečke funkcije pretrage i sortiranja	144
3.4 Rešenja	149
4 Dinamičke strukture podataka	221
4.1 Liste	221
4.2 Stabla	232
4.3 Rešenja	241
5 Ispitni rokovi	336
5.1 Praktični deo ispita, jun 2015	336
5.2 Praktični deo ispita, jul 2015	338
5.3 Praktični deo ispita, septembar 2015	340
5.4 Praktični deo ispita, oktobar 2015	342
5.5 Praktični deo ispita, januar 2016	342

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 alociranoj 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 održanih ispita. Elektronska verzija zbirke, dostupna je (besplatno) u okviru strane kursa www.programiranje2.matf.bg.ac.rs, a tu je dostupan i radni rezervorijum elektronskih verzija rešenja zadataka.

U prvom poglavlju zbirke obrađene su uvodne teme koje obuhvataju osnovne tehnike koje se koriste u rešavanju svih ostalih zadataka u zbirci: podela koda po datotekama i rekurzivni pristup rešavanju problema. Takođe, u okviru ovog poglavlja dati su i osnovni algoritmi za rad sa bitovima. Drugo poglavlje je posvećeno pokazivačima, pokazivačkoj aritmetici, višedimenzionim nizovima, dinamičkoj alokaciji memorije i radu sa pokazivačima na funkcije. Treće poglavlje obrađuje algoritme pretrage i sortiranja, a četvrto dinamičke strukture podataka: liste i stabla. U poslednjem poglavlju dati su zadaci sa ispitnih rokova u toku jedne školske godine. Većina zadataka je rešena, teži zadaci su obeleženi sa zvezdicom, a posebno teški zadaci sa dve zvezdice.

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. Zbog toga smatramo da je naš osnovni doprinos što smo objedinili, precizno formulisali, rešili i detaljno iskomentarisali sve najvažnije zadatke koji su potreбni za uspešno savlađivanje koncepcata koji se obrađuju u okviru kursa.

Zahvaljujemo recenzentima, Gordani Pavlović Lažetić i Draganu Uroševiću, na veoma pažljivom čitanju rukopisa i na brojnim korisnim sugestijama. Takođe, zahvaljujemo studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli u obličavanju ovog materijala.

Svi komentari i sugestije na zadatke i rešenja zbirke su dobrodošli i osećajte se slobodno da ih pošaljete elektronskom poštom bilo kome od autora¹.

Autori

¹Adrese autora su: milena, jgraovac, aspasic, mirko, nina, andjelkaz sa nastavkom @matf.bg.ac.rs

1

Uvodni zadaci

1.1 Podela koda po datotekama

Zadatak 1.1 Napisati program za rad sa kompleksnim brojevima.

- (a) Definisati strukturu `KompleksanBroj` koja opisuje kompleksan broj zadat njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj `z` sa standardnog ulaza.
- (c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` koja ispisuje kompleksan broj `z` na standardni izlaz u odgovarajućem formatu (npr. broj čiji je realni deo 2, a imaginarni -3 treba ispisati kao $(2 - 3i)$).
- (d) Napisati funkciju `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja `z`.
- (e) Napisati funkciju `float imaginaran_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja `z`.
- (f) Napisati funkciju `float moduo(KompleksanBroj z)` koja vraća moduo kompleksnog broja `z`.
- (g) Napisati funkciju `KompleksanBroj konjugovan(KompleksanBroj z)` koja vraća konjugovano-kompleksni broj broja `z`.
- (h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća zbir dva kompleksna broja `z1` i `z2`.

1.1 Podela koda po datotekama

- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća razliku dva kompleksna broja z_1 i z_2 .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća proizvod dva kompleksna broja z_1 i z_2 .
- (k) Napisati funkciju `float argument(KompleksanBroj z)` koja vraća argument kompleksnog broja z .

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza uneti dva kompleksna broja z_1 i z_2 a zatim ispisati realni deo, imaginarni deo, moduo, konjugovano-kompleksan broj i argument broja koji se dobija kao zbir, razlika ili proizvod brojeva z_1 i z_2 u zavisnosti od znaka ($+$, $-$, $*$) koji se unosi sa standardnog ulaza.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite realni i imaginarni deo kompleksnog broja: 1 -3  
(1.00 - 3.00 i)  
Unesite realni i imaginarni deo kompleksnog broja: -1 4  
(-1.00 + 4.00 i)  
Unesite znak (+,-,*): -  
(1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)  
Realni_deo: 2  
Imaginarni_deo: -7.000000  
Moduo: 7.280110  
Konjugovano kompleksan broj: (2.00 + 7.00 i)  
Argument kompleksnog broja: - 1.292497
```

[Rešenje 1.1]

Zadatak 1.2 Uraditi prethodni zadatak tako da su sve napisane funkcije za rad sa kompleksnim brojevima zajedno sa definicijom strukture `KompleksanBroj` izdvojene u posebnu biblioteku. Napisati program koji testira ovu biblioteku. Sa standardnog ulaza uneti kompleksan broj a zatim na standardni izlaz ispisati njegov polarni oblik.

Primer 1

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

[Rešenje 1.2]

Zadatak 1.3 Napisati biblioteku za rad sa polinomima.

- (a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20 koji je zadat nizom svojih koeficijenata tako da se na i-toj poziciji u nizu nalazi koeficijent uz i-ti stepen polinoma.
- (b) Napisati funkciju `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz, od najvišeg ka najnižem stepenu. Ipisati samo koeficijente koji su različiti od nule.
- (c) Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom najpre uneti stepen a zatim njegove koeficijente.
- (d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` koja vraća vrednosti polinoma `p` u dатој тачки `x` користећи Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja vraća zbir dva polinoma `p` и `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja vraća proizvod dva polinoma `p` и `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja vraća izvod polinoma `p`.
- (h) Napisati funkciju `Polinom n_izvod(const Polinom * p, int n)` koja vraća `n`-ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati polinome `p` i `q` a zatim ih ispisati na standardni izlaz u odgovarajućem formatu. Izračunati i ispisati zbir `z` i proizvod `r` unetih polinoma `p` i `q`. Sa standardnog ulaza učitati realni broj `x` a zatim na standardni izlaz ispisati vrednost polinoma `z` u tački `x` zaokruženu na dve decimale. Na kraju, sa standardnog ulaza učitati broj `n` i na izlaz ispisati `n`-ti izvod polinoma `r`.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite polinom p (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):  
3 1.2 3.5 2.1 4.2  
Unesite polinom q (prvo stepen, pa zatim koeficijente od najveceg stepena do nultog):  
2 2.1 0 -3.9  
Zbir polinoma je polinom z:  
1.20x^3+5.60x^2+2.10x+0.30  
Prozvod polinoma je polinom r:  
2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38  
Unesite tacku u kojoj racunate vrednost polinoma z:  
0  
Vrednost polinoma z u tacki 0.00 je 0.30  
Unesite izvod polinoma koji zelite:  
3  
3. izvod polinoma r je: 151.20x^2+176.40x-1.62
```

[Rešenje 1.3]

Zadatak 1.4 Napisati biblioteku za rad sa razlomcima.

- (a) Definisati strukturu `Razlomak` koja opisuje razlomak.
- (b) Napisati funkciju `Razlomak ucitaj()` za učitavanje razlomka.
- (c) Napisati funkciju `void ispisi(const Razlomak * r)` koja ispisuje razlomak `r`.
- (d) Napisati funkciju `int brojilac(const Razlomak * r)` koja vraćaja brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraćaja imenilac razlomka `r`.
- (f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.
- (g) Napisati funkciju `double reciprocna_vrednost(const Razlomak * r)` koja vraća recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skrati(const Razlomak * r)` koja vraća skraćenu vrednost datog razlomka `r`.
- (i) Napisati funkciju `Razlomak saberi(const Razlomak * r1, const Razlomak * r2)` koja vraća zbir dva razlomka `r1` i `r2`.
- (j) Napisati funkciju `Razlomak oduzmi(const Razlomak * r1, const Razlomak * r2)` koja vraća razliku dva razlomka `r1` i `r2`.
- (k) Napisati funkciju `Razlomak pomnozi(const Razlomak * r1, const Razlomak * r2)` koja vraća proizvod dva razlomka `r1` i `r2`.
- (l) Napisati funkciju `Razlomak podeli(const Razlomak * r1, const Razlomak * r2)` koja vraća količnik dva razlomka `r1` i `r2`.

Napisati program koji testira prethodne funkcije. Sa standardnog ulaza učiti dva razlomka `r1` i `r2`. Na standardni izlaz ispisati skraćene vrednosti zbiru, razlike, proizvoda i količnika razlomaka `r1` i recipročne vrednosti razlomka `r2`.

Primer 1

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

1.2 Algoritmi za rad sa bitovima

Zadatak 1.5 Napisati biblioteku `stampanje_bitova` za rad sa bitovima. Biblioteka treba da sadrži funkcije `stampanje_bitova`, `stampanje_bitova_short` i `stampanje_bitova_char` za štampanje bitova u binarnom zapisu celog broja tipa `int`, `short` i `char`, koji se zadaje kao argument funkcije. Napisati program koji testira napisanu biblioteku. Sa standardnog ulaza učitati u heksadekadnom formatu cele brojeve tipa `int`, `short` i `char` i na standardni izlaz ispisati njihovu binarnu reprezentaciju.

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite broj tipa int: 0x4f4f4f4f  
|| Binarna reprezentacija: 0100111010011110100111101001111  
|| Unesite broj tipa short: 0x4f4f  
|| Binarna reprezentacija: 010011101001111  
|| Unesite broj tipa char: 0x4f  
|| Binarna reprezentacija: 01001111
```

[Rešenje 1.5]

Zadatak 1.6 Napisati funkcije `prebroj_bitove_1` i `prebroj_bitove_2` koje vraćaju broj jedinica u binarnom zapisu označenog celog broja x koji se zadaje kao argument funkcije. Prebrojavanje bitova ostvariti na dva načina:

- (a) formiranjem odgovarajuće maske i njenim pomeranjem (funkcija `prebroj_bitove_1`)
- (b) formiranjem odgovarajuće maske i pomeranjem promenljive x (funkcija `prebroj_bitove_2`).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati ceo broj u heksadekasnom formatu i na standardni izlaz ispisati broj jedinica u binarnom zapisu učitanog broja pozivom jedne od dve napisane funkcije. Izbor funkcije izvršiti na osnovu opcije ('1' ili '2') koju korisnik programa unese na

1.2 Algoritmi za rad sa bitovima

ulazu. Ukoliko korisnik ne uneše ispravnu vrednost za redni broj funkcije, prekini izvršavanje programa i ispisati odgovarajuću poruku na standardni izlaz za greške.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0x7F  
Unesite redni broj funkcije: 1  
Poziva se funkcija prebroj_bitove_1  
Broj jedinica u zapisu je 7
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: -0x7F  
Unesite redni broj funkcije: 2  
Poziva se funkcija prebroj_bitove_2  
Broj jedinica u zapisu je 26
```

Primer 3

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0x00FF00FF  
Unesite redni broj funkcije: 2  
Poziva se funkcija prebroj_bitove_2  
Broj jedinica u zapisu je 16
```

Primer 4

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj: 0x00FF00FF  
Unesite redni broj funkcije: 3  
IZLAZ ZA GREŠKU:  
Neodgovarajuci redni broj funkcije!
```

[Rešenje 1.6]

Zadatak 1.7 Napisati funkcije `unsigned najveci(unsigned x)` i `unsigned najmanji(unsigned x)` koje vraćaju najveći, odnosno najmanji neoznačen ceo broj koji se može zapisati istim binarnim ciframa kao broj `x`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati neoznačen ceo broj u heksadekadnom formatu a zatim ispisati binarnu reprezentaciju najvećeg i najmanjeg broja koji se može zapisati istim binarnim ciframa kao učitani broj.

Test 1

```
ULAZ:  
0x7F  
IZLAZ:  
Najveci:  
11111110000000000000000000000000  
Najmanji:  
00000000000000000000000000000001111111
```

Test 2

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

Test 3

```
ULAZ:  
0x00FF00FF  
IZLAZ:  
Najveci:  
11111111111110000000000000000000  
Najmanji:  
00000000000000001111111111111111
```

Test 4

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

[Rešenje 1.7]

Zadatak 1.8 Napisati funkcije za rad sa bitovima. NAPOMENA: *Bit najmanje težine je krajnji desni bit i njegova pozicija se označava nultom dok se pozicije ostalih bitova uvećavaju za jedan, sa desna na levo.*

- Napisati funkciju `unsigned postavi_0(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se n bitova datog broja x , počevši od pozicije p , postave na 0.
- Napisati funkciju `unsigned postavi_1(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se n bitova datog broja x , počevši od pozicije p , postave na 1.
- Napisati funkciju `unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)` koja vraća broj u kome se n bitova najmanje težine poklapa sa n bitova broja x počevši od pozicije p , dok su mu ostali bitovi postavljeni na 0.
- Napisati funkciju `unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p, unsigned y)` koja vraća broj koji se dobija upisivanjem poslednjih n bitova najmanje težine broja y u broj x , počevši od pozicije p .
- Napisati funkciju `unsigned invertuj(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija invertovanjem n bitova broja x počevši od pozicije p .

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve x, n, p, y koji se unose sa standardnog ulaza. Na standardni izlaz ispisati binarne reprezenatacije brojeva x i y , a zatim i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite neoznacen ceo broj x: 235
Unesite neoznacen ceo broj n: 9
Unesite neoznacen ceo broj p: 24
Unesite neoznacen ceo broj y: 127
x =      235                      = 0000000000000000000000000000000011101011
postavi_0( 235,      9,      24)    = 0000000000000000000000000000000011101011

x =      235                      = 0000000000000000000000000000000011101011
postavi_1( 235,      9,      24)    = 00000000111111100000000011101011

x =      235                      = 0000000000000000000000000000000011101011
vrati_bitove( 235,      9,      24) = 000000000000000000000000000000000000000000000000000000000

x =      235                      = 0000000000000000000000000000000011101011
y =      127                      = 000000000000000000000000000000001111111
postavi_1_n_bitove( 235,  9,  24,  127) = 00000000111111100000000011101011

x =      235                      = 0000000000000000000000000000000011101011
invertuj( 235,      9,      24)    = 00000000111111100000000011101011
```

Primer 2

```

|| INTERAKCIJA SA PROGRAMOM:
Unesite neoznacen ceo broj x: 2882398951
Unesite neoznacen ceo broj n: 5
Unesite neoznacen ceo broj p: 10
Unesite neoznacen ceo broj y: 35156526
x = 2882398951                      = 1010101111001101110101011100111
postavi_0(2882398951,      5,      10)    = 1010101111001101110100000100111

x = 2882398951                      = 1010101111001101110101011100111
postavi_1(2882398951,      5,      10)    = 101010111100110111011111100111

x = 2882398951                      = 1010101111001101110101011100111
vrati_bitove(2882398951,      5,      10)  = 00000000000000000000000000001011

x = 2882398951                      = 1010101111001101110101011100111
y = 35156526                         = 0000001000011000011100100010110
postavi_1_n_bitove(2882398951, 5, 10, 35156526) = 1010101111001101110101110100111

x = 2882398951                      = 1010101111001101110101011100111
invertuj(2882398951,      5,      10)    = 1010101111001101110110100100111

```

[Rešenje 1.8]

Zadatak 1.9 Pod rotiranjem bitova uлево подразумева се померавање свих битова за једну позицију улево, с тим што се бит са позиције највеће тежине помера на позицију најмање тежине. Аналогно, ротирање битова удесно подразумева померавање свих битова за једну позицију удесно, с тим што се бит са позиције најмање тежине помера на позицију највеће тежине.

- Napisati функцију `unsigned rotiraj_ulevo(unsigned x, unsigned n)` која враћа број који се добија ротирањем `n` пута улево датог целог неозначеног броја `x`.
- Napisati функцију `unsigned rotiraj_udesno(unsigned x, unsigned n)` која враћа број који се добија ротирањем `n` пута удесно датог целог неозначеног броја `x`.
- Napisati функцију `int rotiraj_udesno_oznaceni(int x, unsigned n)` која враћа број који се добија ротирањем `n` пута удесно датог целог броја `x`.

Napisati програм који са стандардног улаза учијава неозначеноце бројеве `x` и `n` који се уносе у хексадекасном формату, татим испијаве бинарну представу вредности добијене позивом три претходно написане функције са аргументима `x` и `n`, а на крају испијаве бинарну представу вредности добијене позивом функције `rotiraj_udesno_oznaceni` за аргументе `-x` и `n`.

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:
|| Unesite neoznacen ceo broj x: ba11a7
|| Unesite neoznacen ceo broj n: 5
|| x = 0000000010110100001000110100111
|| rotiraj_ulevo(ba11a7, 5)      = 00010111010000100011010011100000
|| rotiraj_udesno(ba11a7, 5)     = 00111000000001011101000010001101
|| rotiraj_udesno_oznaceni(ba11a7, 5) = 00111000000001011101000010001101
|| rotiraj_udesno_oznaceni(-ba11a7, 5) = 110001111111010001011101110010
```

[Rešenje 1.9]

Zadatak 1.10 Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća ceo broj čiji binarni zapis predstavlja sliku u ogledalu binarnog zapisa broja `x`. Napisati program koji testira datu funkciju za broj koji se sa standardnog ulaza zadaje u heksadekadnom formatu. Najpre ispisati binarnu reprezentaciju unetog broja, a zatim i binarnu reprezentaciju broja dobijenog kao njegova slika u ogledalu.

Test 1

```
|| ULAZ:
|| 255
|| IZLAZ:
|| 0000000000000000000000001001010101
|| 101010100100000000000000000000000000000
```

Test 2

```
|| ULAZ:
|| -15
|| IZLAZ:
|| 11111111111111111111111111111101011
|| 1101011111111111111111111111111111111111
```

[Rešenje 1.10]

Zadatak 1.11 Napisati funkciju `int broj_01(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
```

[Rešenje 1.11]

Zadatak 1.12 Napisati funkciju `int broj_parova(unsigned int x)` koja vraća broj pojava dve uzastopne jedinice u binarnom zapisu celog neoznačenog broja `x`. Napisati program koji tu funkciju testira za broj koji se zadaje sa

1.2 Algoritmi za rad sa bitovima

standardnog ulaza. NAPOMENA: *Tri uzastopne jedinice sadrže dve uzastopne jedinice dva puta.*

Test 1

ULAZ:	11
IZLAZ:	1

Test 2

ULAZ:	1024
IZLAZ:	0

Test 3

ULAZ:	2147377146
IZLAZ:	22

[Rešenje 1.12]

Zadatak 1.13 Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama i i j . Pozicije i i j učitati kao parametre komandne linije. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore $+$, $-$, $/$, $*$, $\%$.

Primer 1

POKRETANJE:	./a.out 1 2
INTERAKCIJA SA PROGRAMOM:	
ULAZ:	11
IZLAZ:	13

Primer 2

POKRETANJE:	./a.out 1 2
INTERAKCIJA SA PROGRAMOM:	
ULAZ:	1024
IZLAZ:	1024

Primer 2

POKRETANJE:	./a.out 12 12
INTERAKCIJA SA PROGRAMOM:	
ULAZ:	12345
IZLAZ:	12345

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

Test 1

ULAZ:	11
IZLAZ:	0000000B

Test 2

ULAZ:	1024
IZLAZ:	00000400

Test 3

ULAZ:	12345
IZLAZ:	00003039

[Rešenje 1.14]

Zadatak 1.15

Napisati funkciju koja za data dva neoznačena broja x i y invertuje one bitove u broju x koji se poklapaju sa odgovarajućim bitovima u broju y . Ostali

bitovi treba da ostanu nepromenjeni. Napisati program koji testira tu funkciju za brojeve koji se zadaju sa standardnog ulaza.

Test 1

ULAZ:	123 10
IZLAZ:	4294967285

Test 2

ULAZ:	3251 0
IZLAZ:	4294967295

Test 3

ULAZ:	12541 1024
IZLAZ:	4294966271

Zadatak 1.16 Napisati funkciju koja vraća broj petica u oktalmom zapisu neoznačenog celog broja x . Napisati program koji testira tu funkciju za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

Test 1

ULAZ:	123
IZLAZ:	0

Test 2

ULAZ:	3245
IZLAZ:	2

Test 3

ULAZ:	100328
IZLAZ:	1

1.3 Rekurzija

Zadatak 1.17 Napisati rekurzivnu funkciju koja izračunava x^k , za dati ceo broj x i prirodan broj k

- (a) tako da rešenje bude linearne složenosti,
- (b) tako da rešenje bude logaritamske složenosti.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1' ili '2'), ceo broj x i prirodan broj k , a zatim na standarni izlaz ispisati rezultat primene izabrane funkcije na unete brojeve. Ukoliko se na ulazu unese pogrešan redni broj funkcije, ispisati odgovarajuću poruku o grešci na standardni izlaz i prekinuti izvršavanje programa.

Primer 1

INTERAKCIJA SA PROGRAMOM:	
Unesite redni broj funkcije (1/2):	
1	
Unesite broj x:	2
Unesite broj k:	10
1024	

Primer 2

INTERAKCIJA SA PROGRAMOM:	
Unesite redni broj funkcije (1/2):	
2	
Unesite broj x:	9
Unesite broj k:	4
6561	

[Rešenje 1.17]

Zadatak 1.18 Koristeći uzajamnu (posrednu) rekurziju napisati:

- (a) funkciju `unsigned paran(unsigned n)` koja proverava da li je broj cifara broja x paran i vraća 1 ako jeste, a 0 inače;
- (b) i funkciju `unsigned neparan(unsigned n)` koja proverava da li je broj cifara broja x neparan i vraća 1 ako jeste, a 0 inače.

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

Test 1

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

Test 2

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

[Rešenje 1.18]

Zadatak 1.19 Napisati repno-rekurzivnu funkciju koja izračunava faktorijel broja n . Napisati program koji testira napisanu funkciju za proizvoljan broj n ($n \leq 12$) unet sa standardnog ulaza. NAPOMENA: *Gornja vrednost za n je postavljena na 12 zbog ograničenja veličine broja koji može da stane u promenljivu tipa `int` i činjenice da niz faktorijela brzo raste.*

Primer 1

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

Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:
  Unesite n (<= 12):  0
  0! = 1
```

[Rešenje 1.19]

Zadatak 1.20 Napisati funkciju koja vraća n -ti element u nizu Fibonačijevih brojeva. Elementi niza Fibonačijevih brojeva F izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2).$$

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati prirodan broj n i na standardni izlaz ispisati rezultat primene napisane funkcije na prirodan broj n .

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite koji clan niza se racuna: 5  
|| F(5) = 5
```

Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesite koji clan niza se racuna: 8  
|| F(8) = 21
```

Zadatak 1.21 Elementi niza F izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

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

Napisati funkciju koja računa n -ti element u nizu F

- (a) iterativno,
- (b) tako da funkcija bude rekurzivna i da koristi navedene rekurentne relacije,
- (c) tako da funkcija bude rekurzivna ali da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1', '2', '3'), vrednosti koeficijenata a i b i prirodan broj n . Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim podacima, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa. NAPOMENA: *Niz F definisan na ovaj način predstavlja uopštenje Fibonačijevih brojeva.*

Primer 1

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

Primer 2

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

[Rešenje 1.21]

Zadatak 1.22 Napisati rekurzivnu funkciju koja sabira dekadne cifre datog celog broja x . Napisati program koji testira ovu funkciju za broj koji se unosi sa standardnog ulaza.

Test 1

ULAZ:	123
IZLAZ:	6

Test 2

ULAZ:	23156
IZLAZ:	17

Test 3

ULAZ:	1432
IZLAZ:	10

Test 4

ULAZ:	1
IZLAZ:	1

Test 5

ULAZ:	0
IZLAZ:	0

[Rešenje 1.22]

Zadatak 1.23 Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva

- (a) sabirajući elemente počev od početka niza ka kraju niza,
- (b) sabirajući elemente počev od kraja niza ka početku niza.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije ('1' ili '2'), zatim dimenziju n ($0 < n \leq 100$) celobrojnog niza, a potom i elemente niza. Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim nizom, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa.

Primer 1

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

Primer 2

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

[Rešenje 1.23]

Zadatak 1.24 Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. Elementi niza se unose sve do kraja ulaza (EOF). Prepostaviti da niz neće imati više od 256 elemenata.

Test 1

```
|| ULAZ:
|| 3 2 1 4 21
|| IZLAZ:
|| 21
```

Test 2

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

Test 3

```
|| ULAZ:
|| 1 11 3 5 8 1
|| IZLAZ:
|| 11
```

[Rešenje 1.24]

Zadatak 1.25 Napisati rekurzivnu funkciju koja izračunava skalarni proizvod dva vektora celih brojeva. Napisati program koji testira ovu funkciju za nizove (vektore) koji se unose sa standardnog ulaza. Prvo treba uneti dimenziju nizova, a zatim i njihove elemente. Na standardni izlaz ispisati skalarni proizvod unetih nizova. Pretpostaviti da nizovi neće imati više od 256 elemenata.

Primer 1

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

Primer 2

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

[Rešenje 1.25]

Zadatak 1.26 Napisati rekurzivnu funkciju koja vraća broj pojavljivanja elementa x u nizu a dužine n . Napisati program koji testira ovu funkciju za broj x i niz a koji se unose sa standardnog ulaza. Prvo se unosi x , a zatim elementi niza sve do kraja ulaza. Prepostaviti da nizovi neće imati više od 256 elemenata.

Primer 1

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

Primer 2

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

Primer 3

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

[Rešenje 1.26]

Zadatak 1.27 Napisati rekurzivnu funkciju kojom se proverava da li su tri data cela broja uzastopni članovi datog celobrojnog niza. Sa standardnog ulaza učitati tri broja, a zatim elemente niza sve do kraja ulaza. Na standardni izlaz ispisati rezultat primene funkcije nad učitanim podacima. Pretpostaviti da neće biti uneto više od 256 brojeva.

Primer 1

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

Primer 2

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

[Rešenje 1.27]

Zadatak 1.28 Napisati rekurzivnu funkciju `int prebroj(int x)` koja vraća broj bitova postavljenih na 1 u binarnoj reprezentaciji broja `x`. Napisati program koji testira napisanu funkciju za broj koji se učitava sa standardnog ulaza u heksadekadnom formatu.

Test 1

ULAZ:
0x7F
IZLAZ:
7

Test 2

ULAZ:
0x00FF00FF
IZLAZ:
16

Test 3

ULAZ:
0xFFFFFFFF
IZLAZ:
32

[Rešenje 1.28]

Zadatak 1.29 Napisati rekurzivnu funkciju koja štampa bitovsku reprezentaciju neoznačenog celog broja, i program koji je testira za vrednost koja se zadaje sa standardnog ulaza.

Test 1

ULAZ:
10
IZLAZ:
001010

Test 2

ULAZ:
0
IZLAZ:
00

Zadatak 1.30 Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. UPUT-

STVO: *Binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.*

Test 1

	ULAZ:
	5
	IZLAZ:
	5

Test 2

	ULAZ:
	125
	IZLAZ:
	7

Test 3

	ULAZ:
	8
	IZLAZ:
	1

[Rešenje 1.30]

Zadatak 1.31 Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. UPUTSTVO: *Binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.*

Test 1

	ULAZ:
	5
	IZLAZ:
	5

Test 2

	ULAZ:
	16
	IZLAZ:
	1

Test 3

	ULAZ:
	18
	IZLAZ:
	2

[Rešenje 1.31]

Zadatak 1.32 Napisati rekurzivnu funkciju `int palindrom(char s[], int n)` koja ispituje da li je data niska `s` palindrom. Napisati program koji testira ovu funkciju za nisku koja se zadaje sa standardnog ulaza. Pretpostaviti da niska neće imati više od 31 karaktera.

Test 1

	ULAZ:
	a
	IZLAZ:
	da

Test 2

	ULAZ:
	aa
	IZLAZ:
	da

Test 3

	ULAZ:
	aba
	IZLAZ:
	da

Test 4

	ULAZ:
	programiranje
	IZLAZ:
	ne

Test 5

	ULAZ:
	anavolimilovana
	IZLAZ:
	da

[Rešenje 1.32]

* **Zadatak 1.33** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa $\{1, 2, \dots, n\}$. Napisati program koji testira napisanu funkciju za proizvoljan prirodan broj n ($n \leq 15$) unet sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: <code>2</code> IZLAZ: <code>1 2</code> <code>2 1</code>	ULAZ: <code>3</code> IZLAZ: <code>1 2 3</code> <code>1 3 2</code> <code>2 1 3</code> <code>2 3 1</code> <code>3 1 2</code> <code>3 2 1</code>	ULAZ: <code>-5</code> Duzina permutacije mora biti broj iz intervala <code>[0, 15]!</code>

[Rešenje 1.33]

* **Zadatak 1.34** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbiru dve vrednosti koje su u susedna dva polja iznad. Izuzetak su jedinice na krajevima. Vrednosti brojeva Paskalovog trougla odgovaraju binomnim koeficijentima tj. vrednost polja (n, k) , gde je n redni broj hipotenuze, a k redni broj elemenata 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.

$$\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 1 & & 1 & & \\
 & 1 & & 2 & & 1 & \\
 1 & & 3 & & 3 & & 1 \\
 1 & & 4 & & 6 & & 4 & 1 \\
 1 & & 5 & & 10 & & 10 & 5 & 1
 \end{array}$$

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

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

Test 1

ULAZ:	5 3
IZLAZ:	
	1 1 1 1 2 1 1 3 3 1 1 4 6 10 10 4 5 1 1 1 5 10 10 4 5 1 1 8

Test 2

ULAZ:	6 5
IZLAZ:	
	1 1 1 1 2 1 1 3 3 1 1 4 6 10 10 4 5 1 1 1 5 10 15 20 15 6 1 32

[Rešenje 1.34]

* **Zadatak 1.35** Napisati rekurzivnu funkciju koja prikazuje sve varijacije sa ponavljanjem dužine n skupa $\{a, b\}$, i program koji je testira, za n koje se unosi sa standardnog ulaza.

Test 1

ULAZ:	2
IZLAZ:	
	a a a b b a b b

Test 2

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

* **Zadatak 1.36 Hanojske kule:** Data su tri vertikalna štapa. Na jednom od njih se nalazi n diskova poluprečnika 1, 2, 3, ... do n , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je prenesti diskove sa jednog na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg. Preostali štap koristiti kao pomoći štap prilikom premeštanja.

Napisati program koji za proizvoljnu vrednost n , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

* **Zadatak 1.37 Modifikacija Hanojskih kula:** Data su četiri vertikalna štapa. Na jednom se nalazi n diskova poluprečnika 1, 2, 3, ... do n , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala tri štapa su prazna. Potrebno je prenesti diskove na drugi štap tako da budu u istom redosledu, prenestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk

preko manjeg. Preostala dva štapa koristiti kao pomoćne štapove 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 <stdlib.h>
3 #include <math.h>
4
5 /* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
6   imaginarni deo kompleksnog broja */
7 typedef struct {
8     float real;
9     float imag;
10 } KompleksanBroj;
11
12 /* Funkcija ucitava sa standardnog ulaza realan i imaginarni deo
13   kompleksnog broja i smesta ih u strukturu cija je adresa argument
14   funkcije */
15 void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
17     /* Ucitavanje vrednosti sa standardnog ulaza */
18     printf("Unesite realni i imaginarni deo kompleksnog broja: ");
19     scanf("%f", &z->real);
20     scanf("%f", &z->imag);
21 }
22
23 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
24   obliku (x + i y). Ovoj funkciji se argument prenosi po vrednosti
25   jer se u samoj funkciji ne menja njegova vrednost */
26 void ispisi_kompleksan_broj(KompleksanBroj z)
27 {
28     /* Zapocinje se sa ispisom */
29     printf("(");
30
31     /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja
32       razlicit od nule: tada se realni deo ispisuje na standardni
33       izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
34       imaginarni deo pozitivan ili negativan, a potom i apsolutna
35       vrednost imaginarnog dela kompleksnog broja 2) realni deo
36       kompleksnog broja je nula: tada se samo ispisuje imaginarni deo,
37       s tim sto se ukoliko su obe dela nula ispisuje samo 0, bez
38       decimalnih mesta */

```

```

40   if (z.real != 0) {
41     printf("%.2f", z.real);
42
43     if (z.imag > 0)
44       printf(" + %.2f i", z.imag);
45     else if (z.imag < 0)
46       printf(" - %.2f i", -z.imag);
47   } else {
48     if (z.imag == 0)
49       printf("0");
50     else
51       printf("%.2f i", z.imag);
52   }
53
54   /* Zavrsava se sa ispisom */
55   printf("\n");
56 }
57
58 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
59 float realan_deo(KompleksanBroj z)
60 {
61   return z.real;
62 }
63
64 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
65 float imaginaran_deo(KompleksanBroj z)
66 {
67   return z.imag;
68 }
69
70 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
71 float moduo(KompleksanBroj z)
72 {
73   return sqrt(z.real * z.real + z.imag * z.imag);
74 }
75
76 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
77  odgovara kompleksnom broju argumentu */
78 KompleksanBroj konjugovan(KompleksanBroj z)
79 {
80   /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
81    imaginarnom delu kompleksnog broja */
82
83   KompleksanBroj z1 = z;
84
85   z1.imag *= -1;
86
87   return z1;
88 }
89
90 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru

```

```

92     argumenata funkcije */
93     KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
94     {
95         /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
96            broj ciji je realan deo zbir realnih delova kompleksnih brojeva
97            z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
98            brojeva z1 i z2 */
99
100    KompleksanBroj z = z1;
101
102    z.real += z2.real;
103    z.imag += z2.imag;
104
105    return z;
106 }
107
108 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
109   argumenata funkcije */
110 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
111 {
112     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
113        broj ciji je realan deo razlika realnih delova kompleksnih
114        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
115        kompleksnih brojeva z1 i z2 */
116
117    KompleksanBroj z = z1;
118
119    z.real -= z2.real;
120    z.imag -= z2.imag;
121
122    return z;
123 }
124
125 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
126   argumenata funkcije */
127 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
128 {
129     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
130        broj ciji se realan i imaginaran deo racunaju po formuli za
131        mnozenje kompleksnih brojeva z1 i z2 */
132
133    KompleksanBroj z;
134
135    z.real = z1.real * z2.real - z1.imag * z2.imag;
136    z.imag = z1.real * z2.imag + z1.imag * z2.real;
137
138    return z;
139 }
140
141 /* Funkcija vraca argument zadatog kompleksnog broja */
142 float argument(KompleksanBroj z)
143 {

```

```

144  /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
145   iz biblioteke math.h */
146
148 } // main()
149
150 int main()
151 {
152     char c;
153
154     /* Deklaracija 3 promenljive tipa KompleksanBroj */
155     KompleksanBroj z1, z2, z;
156
157     /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
158      ispisivanje na standardni izlaz */
159     ucitaj_kompleksan_broj(&z1);
160     ispisi_kompleksan_broj(z1);
161     printf("\n");
162
163     /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
164      na standardni izlaz */
165     ucitaj_kompleksan_broj(&z2);
166     ispisi_kompleksan_broj(z2);
167     printf("\n");
168
169     /* Ucitavanje i provera znaka na osnovu koga korisnik bira
170      aritmeticku operaciju koja ce se izvrsiti nad kompleksnim
171      brojevima */
172     getchar();
173     printf("Unesite znak (+,-,*): ");
174     scanf("%c", &c);
175     if (c != '+' && c != '-' && c != '*') {
176         printf("Greska: nedozvoljena vrednost operatora!\n");
177         exit(EXIT_FAILURE);
178     }
179
180     /* Analizira se uneti operator */
181     if (c == '+') {
182         /* Racuna se zbir */
183         z = saberi(z1, z2);
184     } else if (c == '-') {
185         /* Racuna se razlika */
186         z = oduzmi(z1, z2);
187     } else {
188         /* Racuna se proizvod */
189         z = mnozi(z1, z2);
190     }
191
192     /* Ispisuje se rezultat */
193     ispisi_kompleksan_broj(z1);
194     printf(" %c ", c);
195     ispisi_kompleksan_broj(z2);

```

```

196     printf(" = ");
198     ispisi_kompleksan_broj(z);
200
201     /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
202        broja */
203     printf("\nRealni_deo: %.f\nImaginarni_deo: %f\nModuo: %f\n",
204           realan_deo(z), imaginaran_deo(z), moduo(z));
205
206     /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
207        kompleksnog broja */
208     printf("Konjugovano kompleksan broj: ");
209     ispisi_kompleksan_broj(konjugovan(z));
210     printf("\n");
211
212     /* Testira se funkcija koja racuna argument kompleksnog broja */
213     printf("Argument kompleksnog broja: %f\n", argument(z));
214
215     exit(EXIT_SUCCESS);
216 }
```

Rešenje 1.2

kompleksan_broj.h

```

1  /* Zaglavlje kompleksan_broj.h sadrzi definiciju tipa KompleksanBroj
2   i
3   deklaracije funkcija za rad sa kompleksnim brojevima. Zaglavlje
4   nikada ne treba da sadrzi definicije funkcija. Da bi neki program
5   mogao da koristi ove brojeve i funkcije iz ove biblioteke,
6   neophodno je da ukljuci ovo zaglavlje. */
7
8  /* Ovim preprocesorskim direktivama se zaključava zaglavlje i
9   onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
10  posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi u
11  narednoj preprocesorskoj define direktivi. */
12 #ifndef _KOMPLEKSAN_BROJ_H
13 #define _KOMPLEKSAN_BROJ_H
14
15 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
16  koje se koriste u definicijama funkcija navedenim u
17  kompleksan_broj.c */
18 #include <stdio.h>
19 #include <math.h>
20
21 /* Struktura KompleksanBroj */
22 typedef struct {
23     float real;
24     float imag;
25 } KompleksanBroj;
```

```

25 /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
26 definisane u kompleksan_broj.c */
27
28 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
29 kompleksnog broja i smesta ih u strukturu cija je adresa argument
30 funkcije */
31 void ucitaj_kompleksan_broj(KompleksanBroj * z);
32
33 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
34 obliku (x + i y) */
35 void ispisi_kompleksan_broj(KompleksanBroj z);
36
37 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
38 float realan_deo(KompleksanBroj z);
39
40 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
41 float imaginaran_deo(KompleksanBroj z);
42
43 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
44 float moduo(KompleksanBroj z);
45
46 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
47 odgovara kompleksnom broju argumentu */
48 KompleksanBroj konjugovan(KompleksanBroj z);
49
50 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
51 argumenata funkcije */
52 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
53
54 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
55 argumenata funkcije */
56 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
57
58 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
59 argumenata funkcije */
60 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
61
62 /* Funkcija vraca argument zadatog kompleksnog broja */
63 float argument(KompleksanBroj z);
64
65 /* Kraj zakljucanog dela */
66 #endif

```

kompleksan_broj.c

```

1 /* Uključuje se zaglavlj za rad sa kompleksnim brojevima, jer je
2 neophodno da bude poznata definicija tipa KompleksanBroj. Takodje,
3 time su uključena zaglavla standardne biblioteke koja su navedena
4 u kompleksan_broj.h */
5 #include "kompleksan_broj.h"
6

```

```

8 void ucitaj_kompleksan_broj(KompleksanBroj * z)
{
10 /* Ucitavanje vrednosti sa standardnog ulaza */
11 printf("Unesite realan i imaginaran deo kompleksnog broja: ");
12 scanf("%f", &z->real);
13 scanf("%f", &z->imag);
14 }

16 void ispisi_kompleksan_broj(KompleksanBroj z)
{
17 /* Zapocinje se sa ispisom */
18 printf("(");
20
21 /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja
22 razlicit od nule: tada se realni deo ispisuje na standardni
23 izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
24 imaginarni deo pozitivan ili negativan, a potom i apsolutna
25 vrednost imaginarnog dela kompleksnog broja 2) realni deo
26 kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
27 s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
28 decimalnih mesta */

29 if (z.real != 0) {
30     printf("%.2f", z.real);
31
32     if (z.imag > 0)
33         printf(" + %.2f i", z.imag);
34     else if (z.imag < 0)
35         printf(" - %.2f i", -z.imag);
36     } else {
37         if (z.imag == 0)
38             printf("0");
39         else
40             printf("%.2f i", z.imag);
41     }
42
43 /* Zavrsava se sa ispisom */
44 printf(")");
45 }

48 float realan_deo(KompleksanBroj z)
{
49 /* Vraca se vrednost realnog dela kompleksnog broja */
50     return z.real;
51 }

54 float imaginaran_deo(KompleksanBroj z)
{
55 /* Vraca se vrednost imaginarnog dela kompleksnog broja */
56     return z.imag;
57 }

```

```

60 float moduo(KompleksanBroj z)
61 {
62     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
63     return sqrt(z.real * z.real + z.imag * z.imag);
64 }
65
66 KompleksanBroj konjugovan(KompleksanBroj z)
67 {
68     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
69      se promeni znak imaginarnom delu kompleksnog broja */
70     KompleksanBroj z1 = z;
71     z1.imag *= -1;
72     return z1;
73 }
74
75 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76 {
77     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
78      broj ciji je realan deo zbir realnih delova kompleksnih brojeva
79      z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
80      brojeva z1 i z2 */
81     KompleksanBroj z = z1;
82
83     z.real += z2.real;
84     z.imag += z2.imag;
85
86     return z;
87 }
88
89 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
90 {
91     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
92      broj ciji je realan deo razlika realnih delova kompleksnih
93      brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
94      kompleksnih brojeva z1 i z2 */
95     KompleksanBroj z = z1;
96     z.real -= z2.real;
97     z.imag -= z2.imag;
98     return z;
99 }
100
101 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
102 {
103     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
104      broj ciji se realan i imaginaran deo racunaju po formuli za
105      mnozenje kompleksnih brojeva z1 i z2 */
106     KompleksanBroj z;
107
108     z.real = z1.real * z2.real - z1.imag * z2.imag;
109     z.imag = z1.real * z2.imag + z1.imag * z2.real;
110 }
```

```

112     return z;
113 }
114 float argument(KompleksanBroj z)
115 {
116     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
117      iz biblioteke math.h */
118     return atan2(z.imag, z.real);
119 }
```

main.c

```

1 ****
2 Ovaj program koristi korektno definisaniu biblioteku kompleksnih
3 brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
4      kompleksnog broja
5 i popis deklaracija podrzanih funkcija, a u kompleksan_broj.c se
6      nalaze
7 njihove definicije.
8
9 Kompilacija programa se najjednostavnije postize naredbom
10 gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c
11
12 Kompilacija se moze uraditi i na sledeci nacin:
13 gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
14 gcc -Wall -c -o main.o main.c
15 gcc -lm -o kompleksan_broj kompleksan_broj.o main.o
16
17 Napomena: Prethodne komande se koriste kada se sva tri navedena
18 dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
19 kompleksan_broj.c kompleksan_broj.h) nalazi u direktorijumu sa imenom
20      header_dir
21 prevodjenje se vrsti dodavanjem opcije opcije -I header_dir
22 gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c main
23 .c
24 ****
25
26 #include <stdio.h>
27 /* Uključuje se zaglavljne neophodno za rad sa kompleksnim brojevima
28 */
29 #include "kompleksan_broj.h"
30
31 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
32      polarni oblik */
33 int main()
34 {
35     KompleksanBroj z;
36
37     /* Ucitavamo kompleksan broj */
38     ucitaj_kompleksan_broj(&z);
39 }
```

```

35  /* Ispisujemo njegov polarni oblik */
37  printf("Polarni oblik kompleksnog broja je %.2f * e^i * %.2f\n",
38      moduo(z), argument(z));
39
40  return 0;
41 }

```

Rešenje 1.3

polinom.h

```

1 #ifndef _POLINOM_H
2 #define _POLINOM_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 /* Maksimalni stepen polinoma */
8 #define MAKS_STEPEN 20
9
10
11 /* Polinomi se predstavljaju strukturu koja cuva koeficijente
12    (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
13 typedef struct {
14     double koef[MAKS_STEPEN + 1];
15     int stepen;
16 } Polinom;
17
18 /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
19    obliku. Polinom se prenosi po adresi da bi se usteđela memorija:
20    ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
21    se nalazi polinom koji ispisujemo */
22 void ispisi(const Polinom * p);
23
24 /* Funkcija koja ucitava polinom sa tastature */
25 Polinom ucitaj();
26
27 /* Funkcija racuna i vraca vrednost polinoma p u tacki x Hornerovim
28    algoritmom */
29 double izracunaj(const Polinom * p, double x);
30
31 /* Funkcija koja sabira dva polinoma */
32 Polinom saberi(const Polinom * p, const Polinom * q);
33
34 /* Funkcija koja mnozi dva polinoma p i q */
35 Polinom pomnozi(const Polinom * p, const Polinom * q);
36
37 /* Funkcija koja racuna izvod polinoma p */
38 Polinom izvod(const Polinom * p);

```

```

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

```

polinom.c

```

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

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

        if (p->koef[i]) {
            /* Polinom nije nula polinom, cim je neki od koeficijenata
               razlicit od nule */
            nulaPolinom = 0;
            if (p->koef[i] >= 0 && i != p->stepen)
                putchar('+');
            if (i > 1)
                printf("%.2fx^%d", p->koef[i], i);
            else if (i == 1)
                printf("%.2fx", p->koef[i]);
            else
                printf("%.2f", p->koef[i]);
        }
    }
    /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada se
       ispisuje nula. */
    if(nulaPolinom)
        printf("0");
    putchar('\n');
}

Polinom ucitaj()
{
    int i;
    Polinom p;

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

```

```

46  /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
47   dozvoljenog opsega */
48   while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
49     printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
50     scanf("%d", &p.stepen);
51   }
52
53  /* Unose se koeficijenti polinoma */
54  for (i = p.stepen; i >= 0; i--)
55    scanf("%lf", &p.koef[i]);
56
57  /* Vraca se procitani polinom */
58  return p;
59 }

60 double izracunaj(const Polinom * p, double x)
61 {
62  /* Rezultat se na pocetku inicializuje na nulu, a potom se u
63   svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
64   vrednost odgovarajuceg koeficijenta */
65
66  /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
67    $x^4+2x^3+3x^2+2x+1 = (((x+2)*x + 3)*x + 2)*x + 1$  */
68
69  double rezultat = 0;
70  int i = p->stepen;
71  for (; i >= 0; i--)
72    rezultat = rezultat * x + p->koef[i];
73  return rezultat;
74 }

75 Polinom saberi(const Polinom * p, const Polinom * q)
76 {
77  Polinom rez;
78  int i;
79
80  /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
81   stepenom */
82  rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
83
84  /* Racunaju se svi koeficijenti rezultujuceg polinoma tako sto se
85   sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
86   sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
87   od stepena nekog od polaznih polinoma podrazumeva se da je
88   koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
89   polinoma */
90  for (i = 0; i <= rez.stepen; i++)
91    rez.koef[i] =
92      (i > p->stepen ? 0 : p->koef[i]) +
93      (i > q->stepen ? 0 : q->koef[i]);
94
95  /* Vraca se dobijeni polinom */
96

```

```

    return rez;
98 }
100 Polinom pomnozi(const Polinom * p, const Polinom * q)
102 {
103     int i, j;
104     Polinom r;

105     /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
106      */
107     r.stepen = p->stepen + q->stepen;
108     if (r.stepen > MAKS_STEPEN) {
109         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
110         exit(EXIT_FAILURE);
111     }

112     /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na nulu
113      */
114     for (i = 0; i <= r.stepen; i++)
115         r.koef[i] = 0;

116     /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
117      za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
118     for (i = 0; i <= p->stepen; i++)
119         for (j = 0; j <= q->stepen; j++)
120             r.koef[i + j] += p->koef[i] * q->koef[j];

121     /* Vraca se dobijeni polinom */
122     return r;
123 }

124 Polinom izvod(const Polinom * p)
125 {
126     int i;
127     Polinom r;

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

133         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
134          polaznog polinoma */
135         for (i = 0; i <= r.stepen; i++)
136             r.koef[i] = (i + 1) * p->koef[i + 1];
137     } else
138         r.koef[0] = r.stepen = 0;

139     /* Vraca se dobijeni polinom */
140     return r;
141 }
```

```

148 }
149 Polinom n_izvod(const Polinom * p, int n)
150 {
151     int i;
152     Polinom r;
153
154     /* Provera da li je n nenegativna vrednost */
155     if (n < 0) {
156         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
157         exit(EXIT_FAILURE);
158     }
159
160     /* Nulti izvod je bas taj polinom */
161     if (n == 0)
162         return *p;
163
164     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove funkcija
165      za racunanje prvog izvoda polinoma */
166     r = izvod(p);
167     for (i = 1; i < n; i++)
168         r = izvod(&r);
169
170     /* Vraca se dobijeni polinom */
171     return r;
172 }
```

main.c

```

1 #include <stdio.h>
2 #include "polinom.h"
3
4 int main(int argc, char **argv)
5 {
6     Polinom p, q, z, r;
7     double x;
8     int n;
9
10    /* Unos polinoma p */
11    printf
12        ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
13         najveceg stepena do nultog):\n");
14    p = ucitaj();
15
16    /* Ispis polinoma p */
17    ispisi(&p);
18
19    /* Unos polinoma q */
20    printf
21        ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
22         od najveceg stepena do nultog):\n");
```

```

1     q = ucitaj();
2
3     /* Polinomi se sabiraju i ispisuje se izracunati zbir */
4     z = saberi(&p, &q);
5     printf("Zbir polinoma je polinom z:\n");
6     ispisi(&z);
7
8     /* Polinomi se mnoze i ispisuje se izracunati prozvod */
9     r = pomnozi(&p, &q);
10    printf("Prozvod polinoma je polinom r:\n");
11    ispisi(&r);
12
13    /* Ispisuje se vrednost polinoma u unetoj tacki */
14    printf("Unesite tacku u kojoj racunate vrednost polinoma z:\n");
15    scanf("%lf", &x);
16    printf("Vrednost polinoma z u tacki %.2f je %.2f\n", x, izracunaj(&z, x));
17
18
19    /* Racuna se n-ti izvoda polinoma i ispisuje se dobijeni polinoma */
20    /*
21    printf("Unesite izvod polinoma koji zelite:\n");
22    scanf("%d", &n);
23    r = n_izvod(&r, n);
24    printf("%d. izvod polinoma r je: ", n);
25    ispisi(&r);
26
27    exit(EXIT_SUCCESS);
28 }

```

Rešenje 1.5

stampanje_bitova.h

```

1 #ifndef _STAMPANJE_BITOVA_H
2 #define _STAMPANJE_BITOVA_H
3
4 #include <stdio.h>
5
6 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
7   celog broja u memoriji. Bitove koji predstavljaju binarnu
8   reprezentaciju broja treba ispisati sa leva na desno, tj. od bita
9   najvece tezine ka bitu najmanje tezine */
10 void stampaj_bitove(unsigned x);
11
12 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
13   celog broja tipa 'short' u memoriji. */
14 void stampaj_bitove_short(short x);
15
16 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju

```

```

17     karaktera u memoriji. */
void stampaj_bitove_char(char x);
19 #endif

```

stampanje_bitova.c

```

2 #include <stdio.h>
3 #include "stampanje_bitova.h"
4
5 void stampaj_bitove(unsigned x)
6 {
7     /* Broj bitova celog broja */
8     unsigned velicina = sizeof(unsigned) * 8;
9
10    /* Maska koja se koristi za "ocitavanje" bitova celog broja */
11    unsigned maska;
12
13    /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
14       na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
15       mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
16       se jedini bit jedinica pomera udesno, kako bi se odredio naredni
17       bit broja x koji je argument funkcije. Zatim se odgovarajuca
18       cifra, ('0' ili '1'), ispisuje na standardnom izlazu. Neophodno
19       je da promenljiva maska bude deklarisana kao neoznacen ceo broj
20       kako bi se pomeranjem u desno vrsilo logicko pomeranje
21       (popunjavanje nulama), a ne aritmeticko pomeranje (popunjavanje
22       znakom broja). */
23    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
24        putchar(x & maska ? '1' : '0');
25
26    putchar('\n');
27}
28
29 void stampaj_bitove_short(short x)
30 {
31     /* Broj bitova celog broja tipa short */
32     unsigned velicina = sizeof(short) * 8;
33
34     /* Maska koja se koristi za "ocitavanje" bitova broja tipa short */
35     unsigned short maska;
36
37     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
38         putchar(x & maska ? '1' : '0');
39
40     putchar('\n');
41}
42
43 void stampaj_bitove_char(char x)
44 {
45     /* Broj bitova karaktera */

```

```

46     unsigned velicina = sizeof(char) * 8;
48
49     /* Maska koja se koristi za "ocitavanje" bitova jednog karaktera */
50     unsigned char maska;
51
52     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
53         putchar(x & maska ? '1' : '0');
54
55     putchar('\n');
56 }
```

main.c

```

2 #include <stdio.h>
3 #include "stampanje_bitova.h"
4
5 int main()
6 {
7     int broj_int;
8     short broj_short;
9     char broj_char;
10
11     printf("Unesite broj tipa int: ");
12     /* Ucitava se broj sa ulaza */
13     scanf("%x", &broj_int);
14
15     /* I ispisuje se njegova binarna reprezentacija */
16     printf("Binarna reprezentacija: ");
17     stampaj_bitove(broj_int);
18
19     printf("Unesite broj tipa short: ");
20     /* Ucitava se broj sa ulaza */
21     scanf("%x", &broj_short);
22
23     /* I ispisuje se njegova binarna reprezentacija */
24     printf("Binarna reprezentacija: ");
25     stampaj_bitove_short(broj_short);
26
27     printf("Unesite broj tipa char: ");
28     /* Ucitava se broj sa ulaza */
29     scanf("%x", &broj_char);
30
31     /* I ispisuje se njegova binarna reprezentacija */
32     printf("Binarna reprezentacija: ");
33     stampaj_bitove_char(broj_char);
34
35     return 0;
36 }
```

Rešenje 1.6

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
5    kreiranjem odgovarajuce maske i njenim pomeranjem */
6 int prebroj_bitove_1(int x)
7 {
8     int br = 0;
9     unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;
10
11    /* Formiranje se maska cija binarna reprezentacija izgleda
12       100000...0000000, koja sluzi za citavanje bita najvece tezine.
13       U svakoj iteraciji maska se pomera u desno za 1 mesto, i
14       citavamo sledeci bit. Petlja se završava kada vise nema
15       jedinica tj. kada maska postane nula. */
16    unsigned maska = 1 << broj_pomeranja;
17    for (; maska != 0; maska >>= 1)
18        x & maska ? br++ : 1;
19
20    return br;
21 }
22
23 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
24    formiranjem odgovarajuce maske i pomeranjem promenljive x */
25 int prebroj_bitove_2(int x)
26 {
27     int br = 0;
28     unsigned broj_pomeranja = sizeof(int) * 8 - 1;
29
30     /* Kako je argument funkcije ozначен ceo broj x naredba x>>=1 bi
31        vrsila aritmeticko pomeranje u desno, tj. popunjavanje bita
32        najvece tezine bitom znaka. U tom slučaju nikad ne bi bio
33        ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
34        petlji. Zbog toga se koristi pomeranje broja x uлево i maska
35        koja citava bit najvece tezine. */
36
37     unsigned maska = 1 << broj_pomeranja;
38     for (; x != 0; x <<= 1)
39         x & maska ? br++ : 1;
40
41     return br;
42 }
43
44 int main()
45 {
46     int x, i;
47
48     /* Ucitava se broj sa ulaza */
49     printf("Unesite broj:\n");
50     scanf("%x", &x);

```

```

51  /* Dozvoljava se korisniku da bira na koji nacin ce biti izracunat
53   broj jedinica u zapisu broja */
54   printf("Unesite redni broj funkcije:\n");
55   scanf("%d", &i);

57  /* Ispisuje se rezultat */
58  if (i == 1){
59    printf("Poziva se funkcija prebroj_bitove_1\n");
60    printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
61  }else if (i == 2){
62    printf("Poziva se funkcija prebroj_bitove_2\n");
63    printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
64  }else {
65    fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
66    exit(EXIT_FAILURE);
67  }

68  exit(EXIT_SUCCESS);
69 }

```

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

```

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

```

```

27    }
29    /* Vraca se dobijena vrednost */
31    return rezultat;
32 }
33 /* Funkcija vraca najmanji neoznaceni broj sastavljen od istih bitova
   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
34 unsigned najmanji(unsigned x)
35 {
36    /* Rezultat se inicijalizuje vrednoscu 0 */
37    unsigned rezultat = 0;
38
39    /* Promenljiva x se pomera u desno sve dok postoje jedinice u
   njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
   razlicita od nule). */
40    for (; x != 0; x >>= 1) {
41        /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
           detektuje na poziciji najmanje tezine u binarnoj
           reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
           sa desna u rezultat */
42        if (x & 1) {
43            rezultat <= 1;
44            rezultat |= 1;
45        }
46    }
47
48    /* Vraca se dobijena vrednost */
49    return rezultat;
50 }
51
52 int main()
53 {
54    int broj;
55
56    /* Ucitava se broj sa ulaza */
57    scanf("%x", &broj);
58
59    /* Ispisuju se, redom, najveci i najmanji broj formirani od bitova
       unetog broja */
60    printf("Najveci:\n");
61    stampaj_bitove(najveci(broj));
62
63    printf("Najmanji:\n");
64    stampaj_bitove(najmanji(broj));
65
66    return 0;
67 }

```

Rešenje 1.8NAPOMENA: *Rešenje koristi biblioteku za štampanje bitova*

iz zadatka 1.5.

```

2 #include <stdio.h>
3 #include "stampanje_bitova.h"
4
5 /* Funckija postavlja na nulu n bitova pocev od pozicije p. */
6 unsigned postavi_0(unsigned x, unsigned n, unsigned p)
7 {
8     //*****
9     // Formira se maska cija binarna reprezentacija ima n bitova
10    // postavljenih na 0 pocev od pozicije p, dok su svi ostali
11    // postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
12    // 1111 1111 1111 1111 1111 1000 0011 1111
13    // To se postize na sledeci nacin:
14    // ~0          1111 1111 1111 1111 1111 1111 1111 1111
15    // (~0 << n)      1111 1111 1111 1111 1111 1111 1110 0000
16    // ~(~0 << n)      0000 0000 0000 0000 0000 0000 0001 1111
17    // (~(~0 << n) << (p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000
18    // ~(~(~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
19    // *****
20    // unsigned maska = ~(~(~0 << n) << (p - n + 1));
21
22    return x & maska;
23 }
24
25 /* Funckija postavlja na jedinicu n bitova pocev od pozicije p. */
26 unsigned postavi_1(unsigned x, unsigned n, unsigned p)
27 {
28
29     //*****
30     // Formira se maska kod koje je samo n bitova pocev od pocev od
31     // pozicije p jednako 1, a ostali su 0.
32     // Na primer, za n=5 i p=10 formira se maska oblika
33     // 0000 0000 0000 0000 0000 0111 1100 0000
34     // *****
35     // unsigned maska = ~(~0 << n) << (p - n + 1);
36
37     return x | maska;
38 }
39
40 /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
41 predstavlja n bitova pocev od pozicije p u binarnoj
42 reprezentaciji broja x. */
43 unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
44 {
45
46     //*****
47     // Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
48     // Na primer, za n=5
49     // 0000 0000 0000 0000 0000 0001 1111
50     // *****

```

```

52     unsigned maska = ~(~0 << n);
54
55     /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
56      polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
57      zeljenih n i funkcija vraca tako dobijenu vrednost */
58     return maska & (x >> (p - n + 1));
59 }
60
61
62 /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
63    postavljeni na vrednosti n bitova najmanje tezine binarne
64    reprezentacije broja y */
65 unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
66                             unsigned y)
67 {
68     /* Kreira se maska kod koje su poslednjih n bitova 1, a
69      ostali su 0. */
70     unsigned poslednjih_n_1 = ~(~0 << n);
71
72     /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima n
73      bitova postavljenih na 0 pocevsi od pozicije p, dok su
74      ostali bitovi 1. */
75     unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));
76
77     /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se u
78      binarnoj reprezentaciji vrednosti promenljive x postavi na 0 n
79      bitova na pozicijama pocev od p */
80     unsigned x_postavi_0 = x & srednjih_n_0;
81
82     /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
83      binarne reprezentacije vrednosti promenljive y cijih je n bitova
84      najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
85      bitovi su nule. Sa (y & poslednjih_n_1) postave na 0 svi bitovi
86      osim najnizih n */
87     unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);
88
89     return x_postavi_0 ^ y_pomeri_srednje;
90 }
91
92 /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
93   njih n */
94 unsigned invertuj(unsigned x, unsigned n, unsigned p)
95 {
96     /* Formira se maska sa n jedinica pocev od pozicije p. */
97     unsigned maska = ~(~0 << n) << (p - n + 1);
98
99     /* Operator ekskluzivno ili invertuje sve bitove gde je
100    odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
101    return maska ^ x;
102 }

```

```

102 int main()
103 {
104     unsigned x, p, n, y;
105
106     /* Ucitavaju se vrednosti sa standardnog ulaza */
107     printf("Unesite neoznacen ceo broj x:\n");
108     scanf("%u", &x);
109     printf("Unesite neoznacen ceo broj n:\n");
110     scanf("%u", &n);
111     printf("Unesite neoznacen ceo broj p:\n");
112     scanf("%u", &p);
113     printf("Unesite neoznacen ceo broj y:\n");
114     scanf("%u", &y);
115
116     /* Ispisuju se binarne reprezentacije broja x i broja koji se
117      dobije kada se primeni funkcija postavi_0 za x, n i p*/
118     printf("x = %10u %36s = ", x, "");
119     stampaj_bitove(x);
120     printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
121     stampaj_bitove( postavi_0(x, n, p));
122     printf("\n");
123
124     /* Ispisuju se binarne reprezentacije broja x i broja koji se
125      dobije kada se primeni funkcija postavi_1 za x, n i p*/
126     printf("x = %10u %36s = ", x, "");
127     stampaj_bitove(x);
128     printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
129     stampaj_bitove( postavi_1(x, n, p));
130     printf("\n");
131
132     /* Ispisuju se binarne reprezentacije broja x i broja koji se
133      dobije kada se primeni funkcija vrati_bitove za x, n i p*/
134     printf("x = %10u %36s = ", x, "");
135     stampaj_bitove(x);
136     printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
137     stampaj_bitove( vrati_bitove(x, n, p));
138     printf("\n");
139
140     /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
141      dobije kada se primeni funkcija postavi_1_n_bitova za x, n i p*/
142     printf("x = %10u %36s = ", x, "");
143     stampaj_bitove(x);
144     printf("y = %10u %36s = ", y, "");
145     stampaj_bitove(y);
146     printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
147     stampaj_bitove( postavi_1_n_bitova(x, n, p, y));
148     printf("\n");
149
150     /* Ispisuju se binarne reprezentacije broja x i broja koji se
151      dobije kada se primeni funkcija invertuj za x, n i p*/
152     printf("x = %10u %36s = ", x, "");
153     stampaj_bitove(x);

```

```

154     printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
155     stampaj_bitove( invertuj(x, n, p));
156
157     return 0;
158 }
```

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

```

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

/* Funkcija ceo broj x rotira u levo za n mesta. */
unsigned rotiraj_ulevo(int x, unsigned n)
{
    unsigned bit_najvece_tezine;

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

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

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

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

    /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
       iteraciji se određuje bit na poziciji najmanje tezine broja x,
       zatim tako određeni bit se pomera u levo tako da bit na
       poziciji najmanje tezine dodje do pozicije najveće tezine.
       Zatim, nakon pomeranja binarne reprezentacije trenutne vrednosti
```

```

42     promenljive x za 1 u desno, bit na poziciji najvece tezine se
43     postaljva na vrednost vec zapamcenog bita koji je bio na
44     poziciji najmanje tezine. */
45     for (i = 0; i < n; i++) {
46         bit_najmanje_tezine = x & 1;
47         x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
48     }
49
50     /* Vraca se dobijena vrednost */
51     return x;
52 }
53
54 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
55    argument funkcije x označeni ceo broj */
56 int rotiraj_udesno_oznaceni(int x, unsigned n)
57 {
58     unsigned bit_najmanje_tezine;
59     int i;
60
61     /* U svakoj iteraciji se određuje bit na poziciji najmanje tezine
62        i smesta u promenljivu bit_najmanje_tezine. Kako je x označen
63        ceo broj, tada se prilikom pomeranja u desno vrši aritmeticko
64        pomeranje i cuva se znak broja. Dakle, razlikuju se dva slučaja
65        u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
66        izvrši pre petlje, s obzirom da rotiranjem u desno na poziciju
67        najveće tezine može doći i 0 i 1, nezavisno od početnog znaka
68        broja smestenog u promenljivu x. */
69     for (i = 0; i < n; i++) {
70         bit_najmanje_tezine = x & 1;
71
72         if (x < 0)
73             *****
74             Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
75             na poziciji najveće tezine. Na primer ako je x
76             1010 1011 1100 1101 1110 0001 0010 001b
77             (sa b je označen ili 1 ili 0 na poziciji najmanje tezine)
78             Onda je sadržaj promenljive bit_najmanje_tezine:
79             0000 0000 0000 0000 0000 0000 000b
80             Nakon siftovanja sadržaja promenljive x za 1 u desno
81             1101 0101 1110 0110 1111 0000 1001 0001
82             Kako bi umesto 1 na poziciji najveće tezine u trenutnoj binarnoj
83             reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
84             poziciju najveće tezine jer bi se time dobile 0, a u ovom slučaju
85             su potrebne jedinice zbog bitovskog & zato se prvo vrši
86             komplementiranje, a zatim pomeranje
87             ~bit_najmanje_tezine << (sizeof(int)*8 -1)
88             B000 0000 0000 0000 0000 0000 0000
89             gde B označava -b.
90             Potom se ponovo vrši komplementiranje kako bi se b nalazilo na
91             poziciji najveće tezine i sve jedinice na ostalim pozicijama
92             ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
93             b111 1111 1111 1111 1111 1111 1111

```

```

94   ****
95   x = (x >> 1) & ~(~bit_najmanje_tezine << (sizeof(int) * 8 - 1))
96   ;
97   else
98     x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
99   }
100
101 /* Vraca se dobijena vrednost */
102 return x;
103 }
104
105 int main()
106 {
107   unsigned x, n;
108
109   /* Ucitavaju se vrednosti sa standardnog ulaza */
110   printf("Unesite neoznacen ceo broj x:");
111   scanf("%x", &x);
112   printf("Unesite neoznacen ceo broj n:");
113   scanf("%x", &n);
114
115   /* Ispisuje se binarna reprezentacija broja x */
116   printf("x = ");
117   stampaj_bitove(x);
118
119   /* Testira se rad napisanih funkcija */
120   printf("rotiraj_ulevo(%10x,%10u) %10s= ", x, n, "");
121   stampaj_bitove(rotiraj_ulevo(x, n));
122
123   printf("rotiraj_udesno(%10x,%10u) %9s= ", x, n, "");
124   stampaj_bitove(rotiraj_udesno(x, n));
125
126   printf("rotiraj_udesno_oznaceni(%10x,%10u) = ", x, n);
127   stampaj_bitove(rotiraj_udesno_oznaceni(x, n));
128
129   printf("rotiraj_udesno_oznaceni(%10x,%10u) = ", -x, n);
130   stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));
131
132   return 0;
133 }
```

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

```

1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 /* Funkcija vraca vrednost cija je binarna reprezentacija slika u
5    ogledalu binarne reprezentacije broja x. */
6 unsigned ogledalo(unsigned x)
```

```

7  {
9    unsigned najnizi_bit;
11   unsigned rezultat = 0;
13
14   int i;
15   /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuce
16      vrednosti broja x se odreduje i pamti u promenljivoj
17      najnizi_bit, nakon cega se na promenljivu x primeni pomeranje u
18      desno */
19   for (i = 0; i < sizeof(x) * 8; i++) {
20     najnizi_bit = x & 1;
21     x >>= 1;
22     /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
23        postavljeni bitovi dobijaju vecu poziciju. Novi bit se
24        postavlja na najnizu poziciju */
25     rezultat <= 1;
26     rezultat |= najnizi_bit;
27   }
28
29   /* Vraca se dobijena vrednost */
30   return rezultat;
31 }
32
33 int main()
34 {
35   int broj;
36
37   /* Ucitava se broj sa ulaza */
38   scanf("%x", &broj);
39
40   /* Ispisuje se njegova binarna reprezentacija */
41   stampaj_bitove(broj);
42
43   /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
44      funkcije ogledalo */
45   stampaj_bitove(ogledalo(broj));
46
47   return 0;
48 }
```

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 broj_01(unsigned int n)
6 {
7
8   int broj_nula, broj_jedinica;
9   unsigned int maska;
```

```

10    broj_nula = 0;
12    broj_jedinica = 0;

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

18    /* Cilj je proći kroz sve bitove broja x, zato se maska u svakoj
19       iteraciji pomera u desno pa će jedini bit koji je postavljen na
20       1 biti na svim pozicijama u binarnoj reprezentaciji maske */
21    while (maska != 0) {

22        /* Provera da li se na poziciji koju određuje maska nalazi 0 ili
23           1 i uveća se odgovarajući brojac */
24        if (n & maska) {
25            broj_jedinica++;
26        } else {
27            broj_nula++;
28        }

29        /* Pomera se maska u desnu stranu */
30        maska = maska >> 1;
31    }

32    /* Ako je broj jedinica veći od broja nula funkcija vraca 1, u
33       suprotnom vraca 0 */
34    return (broj_jedinica > broj_nula) ? 1 : 0;
35}

36    int main()
37 {
38        unsigned int n;

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

41        /* Ispisuje se rezultat */
42        printf("%d\n", broj_01(n));

43        return 0;
44}

```

Rešenje 1.12

```

1 #include <stdio.h>
2
3 /* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju u
4    binarnom zapisu celog čneoznaenog broja x */
5    int broj_parova(unsigned int x)

```

```

6  {
8      int broj_parova;
10     unsigned int maska;
12     /* Vrednost promenljive koja predstavlja broj parova se
13        inicijalizuje na 0 */
14     broj_parova = 0;
16     /* Postavlja se maska tako da moze da procitamo da li su dva
17        najmanja bita u zapisu broja x 11 */
18     /* Binarna reprezentacija broja 3 je 000....00011 */
19     maska = 3;
20
21     while (x != 0) {
22         /* Provera da li se na najmanjim pozicijama broj x nalazi 11 par
23            */
24         if ((x & maska) == maska) {
25             broj_parova++;
26         }
27
28         /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
29            proveravao sledeci par bitova. Pomeranjem u desno bit najvece
30            tezine se popunjava nulom jer je x neoznacen broj */
31         x = x >> 1;
32     }
33
34     /* Vraca se dobijena vrednost */
35     return broj_parova;
36 }
37
38 int main()
39 {
40     unsigned int x;
41
42     /* Ucitava se broj sa ulaza */
43     scanf("%u", &x);
44
45     /* Ispisuje se rezultat */
46     printf("%d\n", broj_parova(x));
47
48     return 0;
49 }
```

Rešenje 1.14

```

1 #include <stdio.h>
2
3 /* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 + 1 jer
```

```

4    su za svaku heksadekadnu cifru potrebne 4 binarne cifre i jedna
5    dodatna pozicija za terminirajući nulu.
6    Prethodni izraz se može zapisati kao sizeof(unsigned int)*2+1. */
#define MAKS_DUZINA sizeof(unsigned int)*2 +1
8
10   /* Funkcija za neoznacen broj x formira nisku s koja sadrži njegov
11      heksadekadni zapis */
12  void prevod(unsigned int x, char s[])
13 {
14
15     int i;
16     unsigned int maska;
17     int vrednost;
18
19     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovaračući
20        maska za citanje 4 uzastopne cifre */
21     maska = 15;
22
23     *****
24     Broj se posmatra od pozicije najmanje tezine ka poziciji najveće
25     tezine. Na primer za broj cijela je binarna reprezentacija
26     000000000110100010000111010101
27     u prvom koraku se citaju bitovi izdvojeni sa <...>:
28     00000000011010001000011101<0101>
29     u drugom koraku:
30     00000000011010001000011<1101>0101
31     u trećem koraku:
32     0000000001101000100<0011>11010101 i tako redom...
33
34     Indeks i označava poziciju na koju se smesta vrednost.
35     *****
36     for (i = MAKS_DUZINA - 2; i >= 0; i--) {
37       /* Vrednost izdvojene cifre */
38       vrednost = x & maska;
39
40       /* Ako je vrednost iz opsega od 0 do 9 odgovarajući karakter se
41          dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
42          10 do 15 odgovarajući karakter se dobija tako što se prvo
43          oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na takoj
44          dobijenoj vrednosti doda ASCII kod 'A' (time se dobija
45          odgovarajuće slovo 'A', 'B', ... 'F') */
46       if (vrednost < 10) {
47         s[i] = vrednost + '0';
48       } else {
49         s[i] = vrednost - 10 + 'A';
50       }
51
52       /* Primenljiva x se pomera za 4 bita u desnu stranu i time se u
53          narednoj iteraciji posmatraju sledeća 4 bita */
54       x = x >> 4;
55     }

```

```

56  /* Upisuje se terminirajuća nula */
57  s[MAKS_DUZINA - 1] = '\0';
58 }

60 int main()
{
61
62     unsigned int x;
63     char s[MAKS_DUZINA];
64
65     /* Ucitava se broj sa ulaza */
66     scanf("%u", &x);
67
68     /* Poziva se funkcija za prevodjenje */
69     prevod(x, s);
70
71     /* I stampa se dobijena niska */
72     printf("%s\n", s);
73
74     return 0;
75 }
```

Rešenje 1.17

```

#include <stdio.h>
1
/************************************************************
2 Resenje linearne slozenosti:
3     x^0 = 1
4     x^k = x * x^(k-1)
5 ************************************************************/
6 int stepen(int x, int k)
7 {
8     if (k == 0)
9         return 1;
10
11    return x * stepen(x, k - 1);
12
13    /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
14}
15
16/************************************************************
17 Resenje logaritamske slozenosti:
18     x^0 = 1;
19     x^k = x * (x^2)^(k/2) , za neparno k
20     x^k = (x^2)^(k/2) , za parno k
21
22 Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
23 se doslo do rezultata, i stoga je efikasnije.
24 ************************************************************/
25 int stepen_2(int x, int k)
26 {
27     if (k == 0)
28         return 1;
```

```

30  /* Ako je stepen paran */
31  if ((k % 2) == 0)
32      return stepen_2(x * x, k / 2);
33
34  /* Inace (ukoliko je stepen neparan) */
35  return x * stepen_2(x * x, k / 2);
36 }
37
38 int main()
39 {
40     int x, k, ind;
41
42     /* Ucitava se redni broj funkcije koja ce se primeniti */
43     printf("Unesite redni broj funkcije (1/2):\n");
44     scanf("%d", &ind);
45
46     /* Ucitavaju se vrednosti za x i k */
47     printf("Unesite broj x:\n");
48     scanf("%d%d", &x);
49     printf("Unesite broj k:\n");
50     scanf("%d%d", &k);
51
52     /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
53     if (x == 1)
54         printf("%d\n", stepen(x, k));
55     else if (x == 2)
56         printf("%d\n", stepen_2(x, k));
57     else {
58         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
59         exit(EXIT_FAILURE);
60     }
61     return 0;
62 }
```

Rešenje 1.18

```

#include <stdio.h>
2
/* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
4  (posrednu) rekurziju */
6
/* Deklaracija funkcije neparan mora da bude navedena jer se ta
8  funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
  definicije. Funkcija je mogla biti deklarisana i u telu funkcije
  paran. */
10 unsigned neparan(unsigned n);
12
/* Funkcija vraca 1 ako broj n ima paran broj cifara, inace vraca 0
   */
14 unsigned paran(unsigned n)
```

```

14 {
15     if (n <= 9)
16         return 0;
17     else
18         return neparan(n / 10);
19 }
20
21 /* Funkcija vraca 1 ako broj n ima neparan broj cifara, inace vraca
22 0 */
23 unsigned neparan(unsigned n)
24 {
25     if (n <= 9)
26         return 1;
27     else
28         return paran(n / 10);
29 }
30
31
32 int main()
33 {
34     int n;
35
36     /* Ucitava se ceo broj */
37     scanf("%d", &n);
38
39     /* Ispisuje se rezultat */
40     printf("Uneti broj ima %sparan broj cifara.\n",
41           (paran(n) == 1 ? "" : "ne"));
42
43     return 0;
44 }
```

Rešenje 1.19

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
5    rekurziju. Result je argument u kome se akumulira do tada
6    izracunatu vrednost faktorijela. Kada dodje do izlaza iz rekurzije
7    iz rekurzije potrebno je da vratimo result. */
8 int faktorijel_repn(int n, int result)
9 {
10     if (n == 0)
11         return result;
12
13     return faktorijel_repn(n - 1, n * result);
14 }
15
16 /* U sledecim funkcijama je prikazan postupak oslobadjanja od
17    repne rekurzije koja postoji u funkciji faktorijel_repn. */
```

```

19 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
21 naredbama kojima se vrednost argumenta funkcije postavlja na
22 vrednost koja bi se prosledjivala rekurzivnom pozivu i navodjenjem
23 goto naredbe za vracanje na pocetak tela funkcije. */
24 int faktorijel_repna_v1(int n, int result)
25 {
26     pocetak:
27         if (n == 0)
28             return result;
29
30         result = n * result;
31         n = n - 1;
32         goto pocetak;
33 }
34
35 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
36 praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
37 iterativno resenje bez bezuslovnih skokova */
38 int faktorijel_repna_v2(int n, int result)
39 {
40     while (n != 0) {
41         result = n * result;
42         n = n - 1;
43     }
44
45     return result;
46 }
47
48 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
49 broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
50 ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde radi
51 udobnosti korisnika, jer je sasvim prirodno da za faktorijel
52 zahteva samo 1 parametar. Funkcija faktorijel izracunava  $n!$ , tako
53 sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
54 faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
55 funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
56 poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
57 pozivom funkcije faktorijel_repna_v2. */
58 int faktorijel(int n)
59 {
60     return faktorijel_repna(n, 1);
61 }
62
63 int main()
64 {
65     int n;
66
67     /* Ucitava se ceo broj */
68     printf("Unesite n (<= 12): ");
69     scanf("%d", &n);
70     if (n > 12) {
71
72         /* Nece da se izvede */
73         goto pocetak;
74     }
75
76     /* Ispisuje rezultat */
77     printf("Faktorijel (%d) je %d\n", n, result);
78
79     /* Isključuje se iz ciklusa */
80     exit(0);
81 }

```

```

71     fprintf(stderr, "Greska: nedozvoljena vrednost za n!\n");
72     exit(EXIT_FAILURE);
73 }
74
75 /* Ispisuje se rezultat */
76 printf("%d! = %d\n", n, faktorijel(n));
77
78 exit(EXIT_SUCCESS);
79 }
```

Rešenje 1.21

```

1 #include <stdio.h>
2
3 /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
4 int f_iterativna(int n, int a, int b)
5 {
6     int i;
7     int f_0 = 0;
8     int f_1 = 1;
9     int tmp;
10
11    if (n == 0)
12        return 0;
13
14    for (i = 2; i <= n; i++) {
15        tmp = a * f_1 + b * f_0;
16        f_0 = f_1;
17        f_1 = tmp;
18    }
19
20    return f_1;
21 }
22
23 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
24 int f_rekurzivna(int n, int a, int b)
25 {
26     /* Izlaz iz rekurzije */
27     if (n == 0 || n == 1)
28         return n;
29
30     /* Rekurzivni pozivi */
31     return a * f_rekurzivna(n - 1, a, b) +
32            b * f_rekurzivna(n - 2, a, b);
33 }
34
35 /* NAPOMENA: U slučaju da se rekurzijom problem svodi na više manjih
36 podproblema koji se mogu preklapati, postoji opasnost da se
37 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
38 Npr. F(20) = a*F(19) + b*F(18), a F(19) = a*F(18) + b*F(17), tj.
39 */
```

```

41     problem F(18) se resava dva puta! Problemi manjih
42     dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
43     problem je kombinacija rekurzije sa dinamickim programiranjem.
44     Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
45     memoriji (obicno u nizovima ili matricama), odakle se koriste ako
46     tokom resavanja ponovo budu potrebni.

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

53 /* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
54    verzija */
55 int f_napredna(int n, int a, int b)
{
56     /* Niz koji cuva resenja pod problema. Kompajler inicijalizuje
57        staticke promenljive na podrazumevane vrednosti. Stoga, elemente
58        celobrojnog niza inicijalizuje na 0 */
59     static int f[20];

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

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

67     /* Rekursivni pozivi */
68     return f[n] =
69         a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
70 }

72 int main()
73 {
74     int n, a, b, ind;

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

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

84     /* Ucitava se broj n */
85 }

```

```

93     printf("Unesite koji clan niza se racuna:\n");
94     scanf("%d", &n);

95     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
96      funkcije f_iterativna, f_rekursivna ili f_napredna */
97     if (ind == 0)
98         printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
99     else if (ind == 1)
100        printf("F(%d) = %d\n", n, f_rekursivna(n, a, b));
101    else
102        printf("F(%d) = %d\n", n, f_napredna(n, a, b));
103
104    return 0;
105 }
```

Rešenje 1.22

```

# include <stdio.h>
2
/* Funkcija određuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
{
6     /* Izlazak iz rekurzije: ako je broj jednocifern */
8     if (x < 10)
9         return x;

10    /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
11       poslednje cifre + poslednja cifra tog broja */
12    return zbir_cifara(x / 10) + x % 10;
}
14
16 int main()
{
17     unsigned int x;
18
19     /* Ucitava se ceo broj */
20     scanf("%u", &x);

22     /* Ispisuje se zbir cifara ucitanog broja */
23     printf("%d\n", zbir_cifara(x));
24
25     return 0;
26 }
```

Rešenje 1.23

```

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

```

4 #define MAKS_DIM 100

6 /* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
7  suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
8  element niza. */
9 int suma_niza_1(int *a, int n)
10 {
11     if (n <= 0)
12         return 0;
13
14     return suma_niza_1(a, n - 1) + a[n - 1];
15 }

16 /* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
17  jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
18  elementa niza i sume preostalih n-1 elemenata. */
19 int suma_niza_2(int *a, int n)
20 {
21     if (n <= 0)
22         return 0;
23
24     return a[0] + suma_niza_2(a + 1, n - 1);
25 }

26 int main()
27 {
28     int a[MAKS_DIM];
29     int n, i = 0, ind;
30
31     /* Ucitava se redni broj funkcije */
32     printf("Unesite redni broj funkcije (1 ili 2):\n");
33     scanf("%d", &ind);
34
35     /* Ucitava se broj elemenata niza */
36     printf("Unesite dimenziju niza:");
37     scanf("%d", &n);
38
39     /* Ucitava se n elemenata niza. */
40     printf("Unesite elemente niza:");
41     for (i = 0; i < n; i++)
42         scanf("%d", &a[i]);
43
44     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
45      funkcije suma_niza_1, ondosno suma_niza_2 */
46     if (ind == 1)
47         printf("Suma elemenata je %d\n", suma_niza_1(a, n));
48     else if (ind == 2)
49         printf("Suma elemenata je %d\n", suma_niza_2(a, n));
50     else{
51         fprintf(stderr, "Neodgovarajuci redni broj funkcije!\n");
52         exit(EXIT_FAILURE);
53     }
54 }
```

```

56     exit(EXIT_SUCCESS);
58 }

```

Rešenje 1.24

```

#include <stdio.h>
#define MAKS_DIM 256

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

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

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

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

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

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

    return 0;
}

```

Rešenje 1.25

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Funkcija koja izracunava skalarni proizvod dva data vektora */
5 int skalarno(int a[], int b[], int n)
6 {
7     /* Izlazak iz rekurzije: vektori su duzine 0 */
8     if (n == 0)
9         return 0;
10
11    /* Na osnovu resenja problema dimenzije n-1, resava se problem
12       dimenzije n primenom definicije skalarnog proizvoda
13       a*b = a[0]*b[0] + a[1]*b[1] +...+ a[n-2]*a[n-2] + a[n-1]*a[n-1]
14       Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
15       na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
16       polazna dva vektora otklanjanjem poslednjih elemenata, doda
17       proizvod poslednja dva elementa polaznih vektora. */
18    else
19        return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
20 }
21
22 int main()
23 {
24     int i, a[MAKS_DIM], b[MAKS_DIM], n;
25
26     /* Unosi se dimenzija nizova */
27     printf("Unesite dimenziju nizova:");
28     scanf("%d", &n);
29
30     /* Provera da li je dimenzija niza odgovarajuca */
31     if (n < 0 || n > MAKS_DIM) {
32         printf("Dimenzija mora biti prirodan broj <= %d!\n", MAKS_DIM);
33         return 0;
34     }
35
36     /* A zatim i elementi nizova */
37     printf("Unesite elemente prvog niza:");
38     for (i = 0; i < n; i++)
39         scanf("%d", &a[i]);
40
41     printf("Unesite elemente drugog niza:");
42     for (i = 0; i < n; i++)
43         scanf("%d", &b[i]);
44
45     /* Ispisuje se rezultat skalarnog proizvoda dva ucitana niza */
46     printf("Skalarni proizvod je %d\n", skalarno(a, b, n));
47
48     return 0;
49 }
```

Rešenje 1.26

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Funkcija koja racuna broj pojavljivanja elementa x u nizu a duzine
   n */
5 int br_pojave(int x, int a[], int n)
6 {
7     /* Izlazak iz rekurzije: za niz duzine jedan broj pojava broja x u
       nizu je 1 ukoliko je jedini element a[0] bas x ili 0 inace */
8     if (n == 1)
9         return a[0] == x ? 1 : 0;
10
11    /* U promenljivu bp se smesta broj pojave broja x u prvih n-1
      elemenata niza a. Ukupan broj pojavljivanja broja x u celom nizu
      a je jednak bp uvecanom za jedan ukoliko je se na poziciji n-1 u
      nizu a nalazi broj x */
12    int bp = br_pojave(x, a, n - 1);
13    return a[n - 1] == x ? 1 + bp : bp;
14 }
15
16 int main()
17 {
18     int x, a[MAKS_DIM];
19     int n, i = 0;
20
21     /* Ucitava se ceo broj */
22     printf("Unesite ceo broj:");
23     scanf("%d", &x);
24
25     /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u niz.
       Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
       ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
       i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
26     printf("Unesite elemente niza:");
27     i = 0;
28     while (scanf("%d", &a[i]) != EOF) {
29         i++;
30         if (i == MAKS_DIM)
31             break;
32     }
33     n = i;
34
35     /* Ispisuje se broj pojavljivanja */
36     printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
37
38     return 0;
39 }
```

Rešenje 1.27

```

1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Funkcija koja proverava da li su tri zadata broja uzastopni
   clanovi niza */
5 int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
6 {
7     /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i vraca
        se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi niza */
8     if (n < 3)
9         return 0;
10
11    /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
       dovoljno je da su oni poslednja tri clana niza ili da se oni
       rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
12    return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
13        || tri_uzastopna_clana(x, y, z, a, n - 1);
14}
15
16 int main()
17 {
18     int x, y, z, a[MAKS_DIM];
19     int n;
20
21     /* Ucitavaju se tri cela broja za koje se ispituje da li su
       uzastopni clanovi niza */
22     printf("Unesite tri cela broja:");
23     scanf("%d%d%d", &x, &y, &z);
24
25     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u niz.
       Promenljiva i predstavlja indeks tekuceg broja. U niz se ne moze
       ucitati vise od MAKS_DIM brojeva, pa se u slucaju da promenljiva
       i dostigne vrednost MAKS_DIM prekida unos novih brojeva. */
26     printf("Unesite elemente niza:");
27     int i = 0;
28     while (scanf("%d", &a[i]) != EOF) {
29         i++;
30         if (i == MAKS_DIM)
31             break;
32     }
33     n = i;
34
35     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
       se odgovarajuca poruka */
36     if (tri_uzastopna_clana(x, y, z, a, n))
37         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
38     else
39         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
40
41     return 0;
42 }
```

```
| }
```

Rešenje 1.28

```
#include <stdio.h>
2
/* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
{
6     /* Izlaz iz rekurzije */
8     if (x == 0)
9         return 0;

10    /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
11       postavljen na 1. Koriscenjem odgovarajuce maske proverava se
12       vrednost bita na poziciji najvece tezine i na osnovu toga se
13       razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
14       broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
15       x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
16       na poziciji najvece tezine jedinica, rezultat dobijen
17       rekurzivnim pozivom funkcije za x<<1 treba uvecati za jedan.
18       Za rekurzivni poziv se salje vrednost koja se dobija kada se x
19       pomeri u levo. Napomena: argument funkcije x je ozначен ceo
20       broj, usled cega se ne koristi pomeranje udesno, jer funkciji
21       moze biti prosledjen i negativan broj. Iz tog razloga,
22       odlucujemo se da proveramo najvisi, umesto najnizeg bita */
23    if (x & (1 << (sizeof(x) * 8 - 1)))
24        return 1 + prebroj(x << 1);
25    else
26        return prebroj(x << 1);
27    //*****
28    Krace zapisano
29    return ((x & (1 << (sizeof(x)*8-1))) ? 1 : 0) + prebroj(x << 1);
30    *****/
31}
32
int main()
33{
34    int x;
35
36    /* Ucitava se ceo broj */
37    scanf("%x", &x);
38
39    /* Ispisuje se rezultat */
40    printf("%d\n", prebroj(x));
41
42    return 0;
43}
```

Rešenje 1.30

```

1 #include <stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre u broju
4 */
5 int maks_oktalna_cifra(unsigned x)
6 {
7     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8      vrednost najveće oktalne cifre u broju 0 */
9     if (x == 0)
10        return 0;
11
12     /* Odredjivanje poslednje oktalne cifre u broju */
13     int poslednja_cifra = x & 7;
14
15     /* Odredjivanje maksimalne oktalne cifre u broju kada se iz njega
16      izbriše poslednja oktalna cifra */
17     int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);
18
19     return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
20           : maks_bez_poslednje_cifre;
21 }
22
23 int main()
24 {
25     unsigned x;
26
27     /* Ucitava se neoznacen ceo broj */
28     scanf("%u", &x);
29
30     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
31     printf("%d\n", maks_oktalna_cifra(x));
32
33     return 0;
34 }
```

Rešenje 1.31

```

1 #include <stdio.h>
2
3 /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre u
4   broju */
5 int maks_heksadekadna_cifra(unsigned x)
6 {
7     /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8      vrednost najveće heksadekadne cifre u broju 0 */
9     if (x == 0)
10        return 0;
```

```

12  /* Odredjivanje poslednje heksadekadne cifre u broju */
13  int poslednja_cifra = x & 15;
14
15  /* Odredjivanje maksimalne heksadekadne cifre broja kada se iz
16  njega izbriše poslednja heksadekadna cifra */
17  int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);
18
19  return poslednja_cifra > maks_bez_poslednje_cifre ? poslednja_cifra
20    : maks_bez_poslednje_cifre;
21 }
22
23 int main()
24 {
25     unsigned x;
26
27     /* Ucitava se neoznacen ceo broj */
28     scanf("%u", &x);
29
30     /* Ispisuje se vrednost najvece heksadekadne cifre unetog broja */
31     printf("%d\n", maks_heksadekadna_cifra(x));
32
33     return 0;
34 }
```

Rešenje 1.32

```

1 #include <stdio.h>
2 #include <string.h>
3
4 /* Niska moze imati najvise 31 karaktera + 1 za terminalnu nulu */
5 #define MAKS_DIM 32
6
7 /* Funkcija ispituje da li je zadata niska duzine n palindrom */
8 int palindrom(char s[], int n)
9 {
10     /* Izlaz iz rekurzije - trivijalno, niska duzine 0 ili 1 je
11     palindrom */
12     if ((n == 1) || (n == 0))
13         return 1;
14
15     /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
16     poslednji karakter i da je palindrom niska koja nastaje kada se
17     polaznoj nisci otklone prvi i poslednji karakter */
18     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
19 }
20
21 int main()
22 {
23     char s[MAKS_DIM];
24     int n;
```

```

26  /* Ucitavanje niske sa standardnog ulaza */
27  scanf("%s", s);
28
29  /* Odredjuje se duzina niske */
30  n = strlen(s);
31
32  /* Ispisuje se poruka da li je niska palindrom ili nije */
33  if (palindrom(s, n))
34      printf("da\n");
35  else
36      printf("ne\n");
37
38  return 0;
}

```

Rešenje 1.33

```

#include <stdio.h>
2 #include <stdlib.h>
3 #define MAKS_DUZINA_PERMUTACIJE 15
4
5 /* Funkcija koja ispisuje elemente niza a duzine n */
6 void ispisi_niz(int a[], int n)
7 {
8     int i;
9
10    for (i = 1; i <= n; i++)
11        printf("%d ", a[i]);
12    printf("\n");
13 }
14
15 /* Funkcija koja proverava da li se broj x nalazi u nizu a duzine n
16 */
17 int koriscen(int a[], int n, int x)
18 {
19     int i;
20
21     /* Obilaze se svi elementi niza */
22     for (i = 1; i <= n; i++)
23         /* Ukoliko se nađe na traženu vrednost, pretraga se prekida */
24         if (a[i] == x)
25             return 1;
26
27     /* Zaključuje se da broj nije pronadjen */
28     return 0;
}
29
30 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija
31 kao argument niz a[] u koji se smesta permutacija, broj m označava
32 da se u okviru tog poziva funkcije na m-tu poziciju u permutaciji
smesta jedan od preostalih celih brojeva, n je veličina skupa koji

```

```

34     se permutuje. Funkciju se inicialno poziva sa argumentom m = 1,
35     jer formiranje permutacije pocinje od pozicije broj 1. Stoga, a[0]
36     se ne koristi. */
37 void permutacija(int a[], int m, int n)
38 {
39     int i;
40
41     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
42        premasila velicinu skupa, onda se svi brojevi vec nalaze u
43        permutaciji i ispisuje se permutacija. */
44     if (m > n) {
45         ispisi_niz(a, n);
46         return;
47     }
48
49     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
50        mesto u nizu (broj koji se do sada nije pojavio u permutaciji).
51        Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
52        ovako postavljenom pocetku permutacije. Kada se to zavrsi, vrsti
53        se provera da li postoji jos neki broj koji moze da se stavi na
54        m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
55        funkcija zavrsava sa radom. Ukoliko takav broj postoji, onda se
56        ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
57        ali sada sa drugacijom postavljenim prefiksom. */
58     for (i = 1; i <= n; i++) {
59         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
60            pozicije, onda se on postavlja na poziciju m i poziva se
61            ponovo funkcija da dopuni ostatak permutacije posle upisivanja
62            i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
63            se nije pojavio do sada u permutaciji. */
64         if (!koriscen(a, m - 1, i)) {
65             a[m] = i;
66             permutacija(a, m + 1, n);
67         }
68     }
69 }
70
71 int main(void)
72 {
73     int n;
74     int a[MAKS_DUZINA_PERMUTACIJE+1];
75
76     /* Ucitava se broja n i provera se da li je u odgovarajucem opsegu
77        */
78     scanf("%d", &n);
79     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
80         fprintf(stderr,
81                 "Duzina permutacije mora biti broj iz intervala [0, %d]!\n"
82                 "%",
83                 MAKS_DUZINA_PERMUTACIJE);
84         exit(EXIT_FAILURE);
85     }

```

```

84  /* Ispisuju se permutacije duzine n */
85  permutacija(a, 1, n);
86
87  exit(EXIT_SUCCESS);
88 }

```

Rešenje 1.34

```

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

4 /* Rekurzivna funkcija za racunanje binomnog koeficijenta */
5 int binomni_koeficijent(int n, int k)
6 {
7     /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0.
8      Ukoliko je k strog izmedju 0 i n, onda se koristi formula
9      bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
10     koja se moze izvesti iz definicije binomnog koeficijenta */
11     return (0 < k && k < n) ?
12         binomni_koeficijent(n - 1, k - 1) + binomni_koeficijent(n - 1,
13                                         k) : 1;
14 }

16 **** Iterativno izracunavanje datog binomnog koeficijenta
17
18 int binomni_koeficijent (int n, int k) {
19     int i, j, b;
20
21     for (b=i=1, j=n; i<=k; b =b * j-- / i++);
22
23     return b;
24 }
25
26 **** Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
27 resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
28 ****
29
30 /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
31 zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
32 ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
33 veca od sume elemenata prethodne hipotenuze. */
34 int suma_elemenata_hipotenuze(int n)
35 {
36     return n > 0 ? 2 * suma_elemenata_hipotenuze(n - 1) : 1;
37 }
38
39 int main()
40 {
41     int n, k, i, d, r;

```

```
44  /* Ucitavaju se brojevi d i r */
45  scanf("%d %d", &d, &r);
46
47  /* Ispisuje se Paskalov trougao */
48  putchar('\n');
49  for (n = 0; n <= d; n++) {
50      for (i = 0; i < d - n; i++)
51          printf(" ");
52      for (k = 0; k <= n; k++)
53          printf("%4d", binomni_koeficijent(n, k));
54      putchar('\n');
55  }
56
57  /* Provera da li je r nenegativan */
58  if (r < 0) {
59      fprintf(stderr,
60              "Redni broj hipotenuze mora biti veci ili jednak od 0!\n");
61      exit(EXIT_FAILURE);
62  }
63
64  /* Ispisuje se suma elemenata hipotenuze */
65  printf("%d\n", suma_elemenata_hipotenuze(r));
66
67  exit(EXIT_SUCCESS);
68 }
```

2

Pokazivači

2.1 Pokazivačka aritmetika

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

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

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

Primer 1

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

Primer 2

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

[Rešenje 2.1]

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

2.1 Pokazivačka aritmetika

- (a) funkciju `zbir` koja izračunava zbir elemenata niza,
- (b) funkciju `proizvod` koja izračunava proizvod elemenata niza,
- (c) funkciju `min_element` koja izračunava najmanji elemenat niza,
- (d) funkciju `max_element` koja izračunava najveći elemenat niza.

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

Primer 1

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

Primer 2

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

[Rešenje 2.2]

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

Primer 1

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

Primer 2

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

Primer 3

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

Primer 4

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

[Rešenje 2.3]

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

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

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

Primer 1

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

INTERAKCIJA SA PROGRAMOM:
Broj argumenata komandne linije je 5.
Kako zelite da ispisete argументе,
koriscenjem indeksне ili pokazivачке
sintakсе (I ili P)? I
Argumenti komandne linije su:
0 ./a.out
1 prvi
2 .
3 treci
4 -4
Pocetna slova argumenata komandne linije:
. p 2 t -
```

Primer 2

```
POKRETANJE: ./a.out

INTERAKCIJA SA PROGRAMOM:
Broj argumenata komandne linije je 1.
Kako zelite da ispisete argументе,
koriscenjem indeksне или pokazivачке
sintаксе (I или P)? P
Argumenti komandne linije su:
0 ./a.out
Pocetna slova argumenata komandne linije:
.
```

[Rešenje 2.4]

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

Primer 1

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

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

Primer 2

```
POKRETANJE: ./a.out

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

[Rešenje 2.5]

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

Primer 1

```
POKRETANJE: ./a.out ulaz.txt 1  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima  
reci koje imaju 1 karakter  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci ciji je broj karaktera 1 je 3.
```

Primer 2

```
POKRETANJE: ./a.out ulaz.txt  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima  
reci koje imaju 1 karakter  
  
INTERAKCIJA SA PROGRAMOM:  
Greska: Nedovoljan broj argumenata  
komandne linije.  
Program se poziva sa  
. ./a.out ime_dat br_karaktera.
```

Primer 3

```
POKRETANJE: ./a.out ulaz.txt 2  
  
DATOTEKA ULAZ.TXT NE POSTOJI  
  
INTERAKCIJA SA PROGRAMOM:  
Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

[Rešenje 2.6]

Zadatak 2.7 Napisati program koji kao prvi argument komandne linije prihvata putanju do datoteke za koju treba proveriti koliko reči ima zadati sufiks (ili prefiks), koji se zadaje kao drugi argument komandne linije. Smatratи 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

```
POKRETANJE: ./a.out ulaz.txt ke -s  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima reci  
koje se zavrsavaju na ke  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci koje se zavrsavaju na ke je 2.
```

Primer 2

```
POKRETANJE: ./a.out ulaz.txt sa -p  
  
ULAZ.TXT  
Ovo je sadrzaj datoteke i u njoj ima reci  
koje pocinju sa sa  
  
INTERAKCIJA SA PROGRAMOM:  
Broj reci koje pocinju na sa je 3.
```

Primer 3

```
POKRETANJE: ./a.out ulaz.txt sa -p
DATOTEKA ULAZ.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje
datoteke ulaz.txt.
```

Primer 4

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

[Rešenje 2.7]

2.2 Višedimenzioni nizovi

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

- 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 broj vrsta (ili kolona) kvadratne matrice n ($0 < n \leq 100$), a zatim i elemente matrice. Na standardni izlaz ispisati učitanu matricu, a zatim trag, euklidsku normu i vandijagonalnu normu učitane matrice.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj vrsta matrice: 3
Unesite elemente matrice, vrstu po vrstu:
1 -2 3
4 -5 6
7 -8 9
Trag matrice je 5.
Euklidска норма матрице је 16.88.
Вандижагонална норма матрице је 11.
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite dimenziju matrice: 0
Greska: neodgovarajuća dimenzija matrice.
```

[Rešenje 2.8]

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

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

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

Primer 1

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

[Rešenje 2.9]

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

- Napisati funkciju koja proverava da li je relacija zadata matricom refleksivna.
- Napisati funkciju koja proverava da li je relacija zadata matricom simetrična.
- Napisati funkciju koja proverava da li je relacija zadata matricom tranzitivna.

- (d) Napisati funkciju koja određuje refleksivno zatvorene relacije (najmanju refleksivnu relaciju koja je nadskup date).
- (e) Napisati funkciju koja određuje simetrično zatvorene relacije (najmanju simetričnu relaciju koja je nadskup date).
- (f) Napisati funkciju koja određuje refleksivno-tranzitivno zatvorene relacije (najmanju refleksivnu i tranzitivnu relaciju koja sadrži datu). NAPOMENA: Koristiti Varšalov algoritam.

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

Primer 1

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

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

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

[Rešenje 2.10]

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

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

- (b) Napisati funkciju koja određuje indeks kolone koja sadrži najmanji element matrice.
- (c) Napisati funkciju koja određuje indeks vrste koja sadrži najveći element matrice.
- (d) Napisati funkciju koja određuje broj negativnih elemenata matrice.

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

Primer 1

```
POKRETANJE: ./a.out 3
```

INTERAKCIJA SA PROGRAMOM:

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

Primer 2

```
POKRETANJE: ./a.out 4
```

INTERAKCIJA SA PROGRAMOM:

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

[Rešenje 2.11]

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

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
```

```
Unesite broj vrsta matrice: 4
Unesite elemente matrice, vrstu po vrstu:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Matrica je ortonormirana.
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
```

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

[Rešenje 2.12]

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

- Napisati funkciju koja učitava elemente matrice sa standardnog ulaza
- Napisati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice, u smeru kretanja kazaljke na satu.

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

Primer 1

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

Primer 2

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

[Rešenje 2.13]

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

Primer 1

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

2.3 Dinamička alokacija memorije

2.3 Dinamička alokacija memorije

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

Primer 1

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

Primer 2

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

[Rešenje 2.15]

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

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

Od korisnika sa ulaza tražiti da izabere način realokacije memorije.

Primer 1

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

Primer 2

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

[Rešenje 2.16]

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

Primer 1

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

Primer 2

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

[Rešenje 2.17]

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

Primer 1

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

Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj vrsta i broj kolona matrice:  
2 2  
Unesite elemente matrice, vrstu po vrstu:  
-0.1 -0.2  
-0.3 -0.4  
Trag unete matrice je -0.50.
```

[Rešenje 2.18]

Zadatak 2.19 Napisati biblioteku za rad sa celobrojnim matricama.

- Napisati funkciju `int **alociraj_matricu(int n, int m)` koja dinamički alocira memoriju potrebnu za matricu dimenzija $n \times m$.
- Napisati funkciju `int **alociraj_kvadratnu_matricu(int n)` koja alocira memoriju za kvadratnu matricu dimenzije n .
- Napisati funkciju `int **dealociraj_matricu(int **A, int n)` koja dealocira memoriju matrice sa n vrsta. Povratna vrednost ove funkcije treba da bude “prazna” matrica.
- Napisati funkciju `void ucitaj_matricu(int **A, int n, int m)` koja učitava već alociranu matricu dimenzija $n \times m$ sa standardnog ulaza.
- Napisati funkciju `void ucitaj_kvadratnu_matricu(int **A, int n)` koja učitava već alociranu kvadratnu matricu dimenzije $n \times n$ sa standardnog ulaza.
- Napisati funkciju `void ispisi_matricu(int **A, int n, int m)` koja ispisuje matricu dimenzija $n \times m$ na standardnom izlazu.
- Napisati funkciju `void ispisi_kvadratnu_matricu(int **A, int n)` koja ispisuje kvadratnu matricu dimenzije $n \times n$ na standardnom izlazu.
- Napisati funkciju `int ucitaj_matricu_iz_datoteke(int **A, int n, int m, FILE * f)` koja učitava već alociranu matricu dimenzija $n \times m$ iz već otvorene datoteke f . U slučaju neuspešnog učitavanja vratiti vrednost različitu od 0.

2.3 Dinamička alokacija memorije

- (i) Napisati funkciju `int ucitaj_kvadratnu_matricu_iz_datoteke(int **A, int n, FILE * f)` koja učitava već alociranu kvadratnu matricu dimenzije $n \times n$ iz već otvorene datoteke f . U slučaju neuspjelog učitavanja vratiti vrednost različitu od 0.
- (j) Napisati funkciju `int upisi_matricu_u_datoteku(int **A, int n, int m, FILE * f)` koja upisuje matricu dimenzija $n \times m$ u već otvorenu datoteku f . U slučaju neuspjelog upisivanja vratiti vrednost različitu od 0.
- (k) Napisati funkciju `int upisi_kvadratnu_matricu_u_datoteku(int **A, int n, FILE * f)` koja upisuje kvadratnu matricu dimenzije $n \times n$ u već otvorenu datoteku f . U slučaju neuspjelog upisivanja vratiti vrednost različitu od 0.

Napisati programe koji testiraju napisanu biblioteku.

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

Primer 1

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

Primer 2

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

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre>POKRETANJE: ./a.out ulaz.txt</pre> <pre>ULAZ.TXT 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</pre> <pre>IZLAZ: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</pre>	<pre>POKRETANJE: ./a.out ulaz.txt</pre> <pre>ULAZ.TXT dimenzija: 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</pre> <pre>IZLAZ: Neispravan pocetak fajla</pre>	<pre>POKRETANJE: ./a.out</pre> <pre>IZLAZ: Koriscenje programa: ./a.out datoteka</pre>

[Rešenje 2.19]

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

Primer 1

<pre>INTERAKCIJA SA PROGRAMOM: Unesite broj vrsta i broj kolona matrice: 2 3 Unesite elemente matrice, vrstu po vrstu: 1 -2 3 -4 5 -6 Elementi ispod glavne dijagonale matrice: 1 -4 5</pre>
--

[Rešenje 2.20]

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

Primer 1

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

Primer 2

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

Zadatak 2.22 Data je realna kvadratna matrica dimenzije $n \times n$.

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

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

Primer 1

```
||| POKRETANJE: ./a.out matrica.txt  
|||  
||| MATRICA.TXT  
||| 3  
||| 1.1 -2.2 3.3  
||| -4.4 5.5 -6.6  
||| 7.7 -8.8 9.9
```

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

[Rešenje 2.22]

Zadatak 2.23 Napisati program koji na osnovu dve realne matrice dimenzija $m \times n$ formira matricu dimenzije $2 \cdot m \times n$ tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matrice su zapisane u datoteci **matrice.txt**. U prvom redu se nalaze dimenzije matrica m i n , u narednih m redova se nalaze vrste prve matrice, a u narednih m redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz.

Primer 1

```
POKRETANJE: ./a.out matrice.txt
MATRICE.TXT
3
1.1 -2.2 3.3
-4.4 5.5 -6.6
7.7 -8.8 9.9
-1.1 2.2 -3.3
4.4 -5.5 6.6
-7.7 8.8 -9.9
INTERAKCIJA SA PROGRAMOM:
1.1 -2.2 3.3
-1.1 2.2 -3.3
-4.4 5.5 -6.6
4.4 -5.5 6.6
7.7 -8.8 9.9
-7.7 8.8 -9.9
```

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

Primer 1

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

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite elemente niza, nulu za kraj:
-5 -2 -4 -1 0
Trazena matrica je:
-5 -2 -4 -1
-2 -4 -1 -5
-4 -1 -5 -2
-1 -5 -2 -4
```

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

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

Primer 1

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

```
INTERAKCIJA SA PROGRAMOM:
Petru ukupno nedostaje 7 sličica.
Reprezentacija za koju je sakupio
najmanji broj sličica je Brazil.
```

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

Primer 1

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

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

Primer 2

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

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

2.4 Pokazivači na funkcije

Zadatak 2.27 Napisati program koji tabelarno štampa vrednosti projizvoljne realne funkcije sa jednim realnim argumentom, odnosno izračunava i ispisuje vrednosti date funkcije u n ekvidistantnih tačaka na intervalu $[a, b]$. Realni brojevi a i b ($a < b$) kao i ceo broj n ($n \geq 2$) se učitavaju sa standardnog ulaza. Ime funkcije se zadaje kao argument komandne linije (`sin`, `cos`, `tan`, `atan`, `acos`, `asin`, `exp`, `log`, `log10`, `sqrt`, `floor`, `ceil`, `sqr`).

Primer 1

```
POKRETANJE: ./a.out sin
```

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

Primer 2

```
POKRETANJE: ./a.out cos
```

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

[Rešenje 2.27]

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

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

Primer 1

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

Primer 2

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

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

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

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

Primer 1

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

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

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

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

Primer 1

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

Primer 2

```
||| INTERAKCIJA SA PROGRAMOM:
||| Unesite ime funkcije, n, a i b:
||| tan 5000 -4.1 -2.3
||| Vrednost integrala je -0.147640.
```

2.5 Rešenja

Rešenje 2.1

```

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
            pokazuje pokazivac "poslednji". Ovaj pokazivac se zatim
            umanjuje za jedan, sto za posledicu ima da pokazivac
```

```

        "poslednji" sada pokazuje na element koji mu prethodi u nizu
    */
    *poslednji-- = t;
}

/***********************
 Drugi nacin za obrtanje niza

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

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

    /* Broj elemenata niza a */
    int n;

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

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

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

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

    obrni_niz_v1(a, n);

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

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

    obrni_niz_v2(a, n);
}

```

```

89   printf("Nakon ponovnog obrtanja elemenata, niz je:\n");
91
92     for (p = a; p - a < n; p++)
93       printf("%d ", *p);
94     printf("\n");
95
96     exit(EXIT_SUCCESS);
97 }
```

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; 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 }
27
28 /* Funkcija izracunava minimalni element niza */
29 double min(double *a, int n)
30 {
31   /* Na pocetku, minimalni element je prvi element */
32   double min = *a;
33   int i;
34
35   /* Ispituje se da li se medju ostalim elementima niza nalazi
36   minimalni */
37   for (i = 1; i < n; i++)
38     if (*(a + i) < min)
39       min = *(a + i);
```

```

40     return min;
42 }
44 /* Funkcija izracunava maksimalni element niza */
45 double max(double *a, int n)
46 {
47     /* Na pocetku, maksimalni element je prvi element */
48     double max = *a;
49
50     /* Ispituje se da li se medju ostalim elementima niza nalazi
51      maksimalni */
52     for (a++, n--; n > 0; a++, n--)
53         if (*a > max)
54             max = *a;
55
56     return max;
57 }
58
59
60 int main()
61 {
62     double a[MAX];
63     int n, i;
64
65     printf("Unesite dimenziju niza: ");
66     scanf("%d", &n);
67
68     /* Proverava se da li je doslo do prekoracenja ogranicenja
69      dimenzije */
70     if (n <= 0 || n > MAX) {
71         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
72         exit(EXIT_FAILURE);
73     }
74
75     printf("Unesite elemente niza:\n");
76     for (i = 0; i < n; i++)
77         scanf("%lf", a + i);
78
79     /* Vrsi se testiranje definisanih funkcija */
80     printf("Zbir elemenata niza je %5.3f.\n", zbir(a, n));
81     printf("Proizvod elemenata niza je %5.3f.\n", proizvod(a, n));
82     printf("Minimalni element niza je %5.3f.\n", min(a, n));
83     printf("Maksimalni element niza je %5.3f.\n", max(a, n));
84
85     exit(EXIT_SUCCESS);
86 }
```

Rešenje 2.3

```

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

5 /* Funkcija povecava za jedan sve elemente u prvoj polovini niza a
6 smanjuje za jedan sve elemente u drugoj polovini niza. Ukoliko niz
7 ima neparan broj elemenata, srednji element ostaje nepromenjen */
8 void povecaj_smanji(int *a, int n)
9 {
10     int *prvi = a;
11     int *poslednji = a + n - 1;

13     while (prvi < poslednji) {

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

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

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

25         /* Pokazivac poslednji se pomera na prethodni element */
26         poslednji--;
27     }

29     *****
30     Drugi nacin:
31     while (prvi < poslednji) {
32         (*prvi)++;
33         (*poslednji)--;
34     }
35     *****
36 }
37
38 int main()
39 {
40     int a[MAX];
41     int n;
42     int *p;
43
44     printf("Unesite dimenziju niza: ");
45     scanf("%d", &n);

46     /* Proverava se da li je doslo do prekoracenja ogranicenja
47         dimenziije */
48     if (n <= 0 || n > MAX) {
49         fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
50         exit(EXIT_FAILURE);
51     }

```

```

53     }
54
55     printf("Unesite elemente niza:\n");
56     for (p = a; p - a < n; p++)
57         scanf("%d", p);
58
59     povecaj_smanji(a, n);
60
61     printf("Transformisan niz je:\n");
62     for (p = a; p - a < n; p++)
63         printf("%d ", *p);
64     printf("\n");
65
66     exit(EXIT_SUCCESS);
}

```

Rešenje 2.4

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     int i;
6     char tip_ispisa;
7
8     printf("Broj argumenata komandne linije je %d.\n", argc);
9
10    printf("Kako zelite da ispisete argumente, koriscenjem"
11          " indeksne ili pokazivacke sintakse (I ili P)? ");
12    scanf("%c", &tip_ispisa);
13
14    printf("Argumenti komandne linije su:\n");
15    if (tip_ispisa == 'I') {
16        /* Ispisuju se argumenti komandne linije koriscenjem indeksne
17           sintakse */
18        for (i = 0; i < argc; i++)
19            printf("%d %s\n", i, argv[i]);
20    } else if (tip_ispisa == 'P') {
21        /* Ispisuju se argumenti komandne linije koriscenjem pokazivacke
22           sintakse */
23        i = argc;
24        for (; argc > 0; argc--)
25            printf("%d %s\n", i - argc, *argv++);
26
27        /* Nakon ove petlje "argc" je jednako nuli a "argv" pokazuje na
28           polje u memoriji koje se nalazi iza poslednjeg argumenta
29           komandne linije. Kako je u promenljivoj "i" sacuvana vrednost
30           broja argumenta komandne linije to sada moze ponovo da se
31           postavi "argv" da pokazuje na multi argument komandne linije
32        */
33        argv = argv - i;
34
35    }
36
37    return 0;
38 }

```

```

33     argc = i;
34 }
35
36 printf("Pocetna slova argumenata komandne linije:\n");
37 if (tip_ispisa == 'I') {
38     /* koristeci indeksnu sintaksu */
39     for (i = 0; i < argc; i++)
40         printf("%c ", argv[i][0]);
41     printf("\n");
42 } else if (tip_ispisa == 'P') {
43     /* koristeci pokazivacku sintaksu */
44     for (i = 0; i < argc; i++)
45         printf("%c ", **argv++);
46     printf("\n");
47 }
48
49 return 0;
50 }
```

Rešenje 2.5

```

1 #include <stdio.h>
2 #include <string.h>
3 #define MAX 100
4
5 /* Funkcija ispituje da li je niska palindrom, odnosno da li se isto
6    cita spreda i odpozadi */
7 int palindrom(char *niska)
8 {
9     int i, j;
10    for (i = 0, j = strlen(niska) - 1; i < j; i++, j--)
11        if (*(niska + i) != *(niska + j))
12            return 0;
13    return 1;
14 }
15
16 int main(int argc, char **argv)
17 {
18     int i, n = 0;
19
20     /* Nulti argument komandne linije je ime izvrsnog programa */
21     for (i = 1; i < argc; i++)
22         if (palindrom(*(argv + i)))
23             n++;
24
25     printf
26         ("Broj argumenata komandne linije koji su palindromi je %d.\n",
27          n);
28     return 0;
29 }
```

Rešenje 2.6

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_KARAKTERA 100
5
6 /* Implementacija funkcije strlen() iz standardne biblioteke */
7 int duzina(char *s)
8 {
9     int i;
10    for (i = 0; *(s + i); i++);
11    return i;
12 }
13
14 int main(int argc, char **argv)
15 {
16     char rec[MAX_KARAKTERA+1];
17     int br = 0, n;
18     FILE *in;
19
20     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
21      */
22     if (argc < 3) {
23         printf("Greska: ");
24         printf("Nedovoljan broj argumenata komandne linije.\n");
25         printf("Program se poziva sa %s ime_dat br_karaktera.\n",
26                argv[0]);
27         exit(EXIT_FAILURE);
28     }
29
30     /* Otvara se datoteka sa imenom koje se zadaje kao prvi argument
31      komandne linije. */
32     in = fopen(*(argv + 1), "r");
33     if (in == NULL) {
34         fprintf(stderr, "Greska: ");
35         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
36         exit(EXIT_FAILURE);
37     }
38
39     n = atoi(*(argv + 2));
40
41     /* Broje se reci cija je duzina jednaka broju zadatom drugim
42      argumentom komandne linije */
43     while (fscanf(in, "%s", rec) != EOF)
44     {
45         if (duzina(rec) == n)
46             br++;
47     }
48
49     printf("Broj reci ciji je broj karaktera %d je %d.\n", n, br);
50
51     /* Zatvara se datoteka */
52     fclose(in);

```

```

50     exit(EXIT_SUCCESS);
51 }

```

Rešenje 2.7

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_KARAKTERA 100
5
6 /* Implementacija funkcije strcpy() iz standardne biblioteke */
7 void kopiranje_niske(char *dest, char *src)
8 {
9     int i;
10    for (i = 0; *(src + i); i++)
11        *(dest + i) = *(src + i);
12 }
13
14 /* Implementacija funkcije strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
16 {
17     int i;
18     for (i = 0; *(s + i) == *(t + i); i++)
19         if (*(s + i) == '\0')
20             return 0;
21     return *(s + i) - *(t + i);
22 }
23
24 /* Implementacija funkcije strlen() iz standardne biblioteke */
25 int duzina_niske(char *s)
26 {
27     int i;
28     for (i = 0; *(s + i); i++);
29     return i;
30 }
31
32 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
33    sufiks niske zadate prvi argumentom funkcije */
34 int sufiks_niske(char *niska, char *sufiks)
35 {
36     int duzina_sufiksa = duzina_niske(sufiks);
37     int duzina_niske_pom = duzina_niske(niska);
38     if (duzina_sufiksa <= duzina_niske_pom &&
39         poredjenje_niski(niska + duzina_niske_pom -
40                           duzina_sufiksa, sufiks) == 0)
41         return 1;
42     return 0;
43 }
44
45 /* Funkcija ispituje da li je niska zadata drugim argumentom funkcije
46    niske_sufiks zadate prvi argumentom funkcije */
47 int niske_sufiks(char *niska, char *sufiks)
48 {
49     int duzina_niske = duzina_niske(niska);
50     int duzina_sufiksa = duzina_niske(sufiks);
51     if (duzina_niske <= duzina_sufiksa &&
52         poredjenje_niski(niska + duzina_niske - duzina_sufiksa, sufiks) == 0)
53         return 1;
54     return 0;
55 }

```

```

46     prefiks niske zadate prvi argumentom funkcije */
47 int prefiks_niske(char *niska, char *prefiks)
48 {
49     int i;
50     int duzina_prefiksa = duzina_niske(prefiks);
51     int duzina_niske_pom = duzina_niske(niska);
52     if (duzina_prefiksa <= duzina_niske_pom) {
53         for (i = 0; i < duzina_prefiksa; i++)
54             if (*(prefiks + i) != *(niska + i))
55                 return 0;
56     return 1;
57 } else
58     return 0;
59 }
60
61 int main(int argc, char **argv)
62 {
63     /* Ukoliko korisnik nije uneo trazene argumete, prijavljuje se
64      greska */
65     if (argc < 4) {
66         printf("Greska: ");
67         printf("Nedovoljan broj argumenata komandne linije.\n");
68         printf("Program se poziva sa %s ime_dat suf/pref -s/-p.\n",
69                argv[0]);
70         exit(EXIT_FAILURE);
71     }
72
73     FILE *in;
74     int br = 0;
75     char rec[MAX_KARAKTERA+1];
76
77     in = fopen(*(argv + 1), "r");
78     if (in == NULL) {
79         fprintf(stderr, "Greska: ");
80         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
81         exit(EXIT_FAILURE);
82     }
83
84     /* Provera se opcija kojom je pozvan program a zatim se ucitavaju
85      reci iz datoteke i broji se koliko njih zadovoljava trazeni
86      uslov */
87     if (!(poredjenje_niski(*(argv + 3), "-s"))) {
88         while (fscanf(in, "%s", rec) != EOF)
89             br += sufiks_niske(rec, *(argv + 2));
90         printf("Broj reci koje se zavrsavaju na %s je %d.\n", *(argv + 2),
91               br);
92     } else if (!(poredjenje_niski(*(argv + 3), "-p"))) {
93         while (fscanf(in, "%s", rec) != EOF)
94             br += prefiks_niske(rec, *(argv + 2));
95         printf("Broj reci koje pocinju na %s je %d.\n", *(argv + 2), br);
96     }

```

```

98     fclose(in);
100    exit(EXIT_SUCCESS);
}

```

Rešenje 2.8

```

1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 #define MAX 100
6
7 /* Funkcija izracunava trag matrice */
8 int trag(int M[][][MAX], int n)
9 {
10     int trag = 0, i;
11     for (i = 0; i < n; i++)
12         trag += M[i][i];
13     return trag;
14 }
15
16 /* Funkcija izracunava euklidsku normu matrice */
17 double euklidska_norma(int M[][][MAX], int n)
18 {
19     double norma = 0.0;
20     int i, j;
21
22     for (i = 0; i < n; i++)
23         for (j = 0; j < n; j++)
24             norma += M[i][j] * M[i][j];
25
26     return sqrt(norma);
27 }
28
29 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
30 int gornja_vandijagonalna_norma(int M[][][MAX], int n)
31 {
32     int norma = 0;
33     int i, j;
34
35     for (i = 0; i < n; i++) {
36         for (j = i + 1; j < n; j++)
37             norma += abs(M[i][j]);
38     }
39
40     return norma;
41 }
42
43 int main()

```

```

45 {
46     int A[MAX][MAX];
47     int i, j, n;
48
49     printf("Unesite broj vrsta matrice: ");
50     scanf("%d", &n);
51
52     /* Provera prekoracenja dimenzije matrice */
53     if (n > MAX || n <= 0) {
54         fprintf(stderr, "Greska: neodgovarajuca dimenzija matrice.\n");
55         exit(EXIT_FAILURE);
56     }
57
58     printf("Unesite elemente matrice, vrstu po vrstu:\n ");
59     for (i = 0; i < n; i++)
60         for (j = 0; j < n; j++)
61             scanf("%d", &A[i][j]);
62
63     /* Ispis sadrzaja matrice koriscenjem indeksne sintakse */
64     for (i = 0; i < n; i++) {
65         /* Ispis elemenata i-te vrste */
66         for (j = 0; j < n; j++)
67             printf("%d ", A[i][j]);
68         printf("\n");
69     }
70
71     /***** Ispisuju se elemenati matrice koriscenjem pokazivacke sintakse.
72     Kod ovako definisane matrice, elementi su uzastopno smesteni u
73     memoriju, kao na traci. To znaci da su svi elementi prve vrste
74     redom smesteni jedan iza drugog. Odmah iza poslednjeg elementa
75     prve vrste smesten je prvi element druge vrste za kojim slede
76     svi elementi te vrste i tako dalje redom.
77
78     for( i = 0; i < n ; i++) {
79         for ( j=0 ; j<n ; j++)
80             printf("%d ", *(*(A+i)+j));
81         printf("\n");
82     }
83     *****/
84
85     /* Ispisuje se rezultat na standardni izlaz */
86     int tr = trag(A, n);
87     printf("Trag matrice je %d.\n", tr);
88
89     printf("Euklidska norma matrice je %.2f.\n", euklidska_norma(A, n))
90     ;
91     printf("Vandijagonalna norma matrice je = %d.\n",
92           gornja_vandijagonalna_norma(A, n));
93
94     exit(EXIT_SUCCESS);
95 }
```

Rešenje 2.9

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100
5
6 /* Funkcija ucitava elemente kvadratne matrice dimenzije n sa
7 standardnog ulaza */
8 void ucitaj_matricu(int m[][][MAX], int n)
9 {
10     int i, j;
11
12     for (i = 0; i < n; i++)
13         for (j = 0; j < n; j++)
14             scanf("%d", &m[i][j]);
15 }
16
17 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
18 standardni izlaz */
19 void ispisi_matricu(int m[][][MAX], int n)
20 {
21     int i, j;
22
23     for (i = 0; i < n; i++) {
24         for (j = 0; j < n; j++)
25             printf("%d ", m[i][j]);
26         printf("\n");
27     }
28 }
29
30 /* Funkcija proverava da li su zadate kvadratne matrice a i b
31 dimenzije n jednake */
32 int jednake_matrice(int a[][][MAX], int b[][][MAX], int n)
33 {
34     int i, j;
35
36     for (i = 0; i < n; i++)
37         for (j = 0; j < n; j++)
38             if (a[i][j] != b[i][j])
39                 return 0;
40
41     /* Prosla je provjera jednakosti za sve parove elemenata koji su na
42     istim pozicijama. To znači da su matrice jednake */
43     return 1;
44 }
45
46 /* Funkcija izracunava zbir dve kvadratne matice */
47 void saberi(int a[][][MAX], int b[][][MAX], int c[][][MAX], int n)
48 {
49     int i, j;

```

```

51   for (i = 0; i < n; i++)
52     for (j = 0; j < n; j++)
53       c[i][j] = a[i][j] + b[i][j];
54   }
55
56 /* Funkcija izracunava proizvod dve kvadratne matice */
57 void pomnozi(int a[][][MAX], int b[][][MAX], int c[][][MAX], int n)
58 {
59   int i, j, k;
60
61   for (i = 0; i < n; i++)
62     for (j = 0; j < n; j++) {
63       /* Mnozi se i-ta vrsta prve sa j-tom kolonom druge matrice */
64       c[i][j] = 0;
65       for (k = 0; k < n; k++)
66         c[i][j] += a[i][k] * b[k][j];
67     }
68 }
69
70 int main()
71 {
72   /* Matrice ciji se elementi zadaju sa ulaza */
73   int a[MAX][MAX], b[MAX][MAX];
74
75   /* Matrice zbir i proizvoda */
76   int zbir[MAX][MAX], proizvod[MAX][MAX];
77
78   /* Dimenzija matrica */
79   int n;
80
81   printf("Unesite broj vrsta matrica:\n");
82   scanf("%d", &n);
83
84   /* Proverava se da li je doslo do prekoracenja dimenzije */
85   if (n > MAX || n <= 0) {
86     fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
87     fprintf(stderr, "matrica.\n");
88     exit(EXIT_FAILURE);
89   }
90
91   printf("Unesite elemente prve matrice, vrstu po vrstu:\n");
92   ucitaj_maticu(a, n);
93   printf("Unesite elemente druge matrice, vrstu po vrstu:\n");
94   ucitaj_maticu(b, n);
95
96   /* Izracunava se zbir i proizvod matrica */
97   saberi(a, b, zbir, n);
98   pomnozi(a, b, proizvod, n);
99
100  /* Ispisuje se rezultat */
101  if (jednake_matrice(a, b, n) == 1)
102    printf("Matrice su jednake.\n");

```

```

103     else
104         printf("Matrice nisu jednake.\n");
105
106     printf("Zbir matrica je:\n");
107     ispisi_matricu(zbir, n);
108
109     printf("Proizvod matrica je:\n");
110     ispisi_matricu(proizvod, n);
111
112     exit(EXIT_SUCCESS);
113 }
```

Rešenje 2.10

```

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

4 #define MAX 64

6 /* Funkcija proverava da li je relacija refleksivna. Relacija je
8   refleksivna ako je svaki element u relaciji sa sobom, odnosno ako
se u matrici relacije na glavnoj dijagonali nalaze jedinice */
int refleksivnost(int m[][MAX], int n)
10 {
11     int i;
12
13     for (i = 0; i < n; i++) {
14         if (m[i][i] != 1)
15             return 0;
16     }
17
18     return 1;
19 }

20 /* Funkcija određuje refleksivno zatvorene zadate relacije. Ono je
22   određeno matricom koja sadrži sve elemente polazne matrice
dopunjene jedinicama na glavnoj dijagonali */
24 void ref_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
25 {
26     int i, j;
27
28     /* Prepisuju se vrednosti elemenata pocetne matrice */
29     for (i = 0; i < n; i++)
30         for (j = 0; j < n; j++)
31             zatvorenje[i][j] = m[i][j];
32
33     /* Na glavnoj dijagonali se postavljaju jedinice */
34     for (i = 0; i < n; i++)
35         zatvorenje[i][i] = 1;
36 }
```

```

38  /* Funkcija proverava da li je relacija simetricna. Relacija je
40   simetricna ako za svaki par elemenata vazi: ako je element "i" u
42   relaciji sa elementom "j", onda je i element "j" u relaciji sa
44   elementom "i". Ovakve matrice su simetricne u odnosu na glavnu
46   dijagonalu */
48   int simetricnost(int m[][][MAX], int n)
49   {
50     int i, j;
52
54     /* Obilaze se elementi ispod glavne dijagonale matrice i uporedjuju
56      se sa njima simetricnim elementima */
58     for (i = 0; i < n; i++)
59       for (j = 0; j < i; j++)
60         if (m[i][j] != m[j][i])
61           return 0;
63
64     return 1;
65   }
66
68  /* Funkcija odredjuje simetricno zatvorene zadate relacije. Ono je
70   odredjeno matricom koja sadrzi sve elemente polazne matrice
72   dopunjene tako da matrica postane simetricna u odnosu na glavnu
74   dijagonalu */
76  void sim_zatvorene(int m[][][MAX], int n, int zatvorene[][][MAX])
77  {
78    int i, j;
79
80    for (i = 0; i < n; i++)
81      for (j = 0; j < n; j++)
82        zatvorene[i][j] = m[i][j];
83
84    for (i = 0; i < n; i++)
85      for (j = 0; j < n; j++)
86        if (zatvorene[i][j] == 1)
87          zatvorene[j][i] = 1;
88  }
89
90  /* Funkcija proverava da li je relacija tranzitivna. Relacija je
92   tranzitivna ako ispunjava sledece svojstvo: ako je element "i" u
94   relaciji sa elementom "j" i element "j" u relaciji sa elementom
96   "k", onda je i element "i" u relaciji sa elementom "k" */
98  int tranzitivnost(int m[][][MAX], int n)
99  {
100    int i, j, k;
101
102    for (i = 0; i < n; i++)
103      for (j = 0; j < n; j++)
104        /* Ispituje se da li postoji element koji narusava *
105        tranzitivnost */
106        for (k = 0; k < n; k++)
107          if (m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
108            return 0;
109
110    return 1;
111  }

```

```

90         return 0;
92     return 1;
94 }
95
96 /* Funkcija određuje refleksivno-tranzitivno zatvorene zadate
   relacije koriscenjem Varsalovog algoritma */
97 void ref_tran_zatvorene(int m[][][MAX], int n, int zatvorene[][][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            zatvorene[i][j] = m[i][j];
105
106    /* Određuje se reflektivno zatvorene matrice */
107    for (i = 0; i < n; i++)
108        zatvorene[i][i] = 1;
109
110    /* Primenom Varsalovog algoritma određuje se tranzitivno
   zatvorene matrice */
111    for (k = 0; k < n; k++)
112        for (i = 0; i < n; i++)
113            for (j = 0; j < n; j++)
114                if ((zatvorene[i][k] == 1) && (zatvorene[k][j] == 1)
115                    && (zatvorene[i][j] == 0))
116                    zatvorene[i][j] = 1;
117    }
118
119    /* Funkcija ispisuje elemente matrice */
120 void pisi_matricu(int m[][][MAX], int n)
121 {
122     int i, j;
123
124     for (i = 0; i < n; i++) {
125         for (j = 0; j < n; j++)
126             printf("%d ", m[i][j]);
127         printf("\n");
128     }
129 }
130
131 int main(int argc, char *argv[])
132 {
133     FILE *ulaz;
134     int m[MAX][MAX];
135     int pomocna[MAX][MAX];
136     int n, i, j;
137
138     /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
   */

```

```

142     if (argc < 2) {
143         printf("Greska: ");
144         printf("Nedovoljan broj argumenata komandne linije.\n");
145         printf("Program se poziva sa %s ime_dat.\n", argv[0]);
146         exit(EXIT_FAILURE);
147     }
148
149     /* Otvara se datoteka za citanje */
150     ulaz = fopen(argv[1], "r");
151     if (ulaz == NULL) {
152         fprintf(stderr, "Greska: ");
153         fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n", argv[1]);
154         exit(EXIT_FAILURE);
155     }
156
157     /* Ucitava se dimenzija matrice */
158     fscanf(ulaz, "%d", &n);
159
160     /* Proverava se da li je doslo do prekoracenja dimenzije */
161     if (n > MAX || n <= 0) {
162         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
163         fprintf(stderr, "matrice.\n");
164         exit(EXIT_FAILURE);
165     }
166
167     /* Ucitava se element po element matrice */
168     for (i = 0; i < n; i++)
169         for (j = 0; j < n; j++)
170             fscanf(ulaz, "%d", &m[i][j]);
171
172     /* Ispisuje se rezultat */
173     printf("Relacija %s refleksivna.\n",
174           refleksivnost(m, n) == 1 ? "jeste" : "nije");
175
176     printf("Relacija %s simetricna.\n",
177           simetricnost(m, n) == 1 ? "jeste" : "nije");
178
179     printf("Relacija %s tranzitivna.\n",
180           tranzitivnost(m, n) == 1 ? "jeste" : "nije");
181
182     printf("Refleksivno zatvorene relacije:\n");
183     ref_zatvorenje(m, n, pomocna);
184     pisi_matricu(pomocna, n);
185
186     printf("Simetricno zatvorene relacije:\n");
187     sim_zatvorenje(m, n, pomocna);
188     pisi_matricu(pomocna, n);
189
190     printf("Refleksivno-tranzitivno zatvorene relacije:\n");
191     ref_tran_zatvorenje(m, n, pomocna);
192     pisi_matricu(pomocna, n);

```

```

194     /* Zatvara se datoteka */
195     fclose(ulaz);
196
197     exit(EXIT_SUCCESS);
198 }
```

Rešenje 2.11

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
5
6 /* Funkcija izracunava najveci element na sporednoj dijagonali. Za
7  elemente sporedne dijagonale vazi da je zbir indeksa vrste i
8  indeksa kolone jednak n-1 */
9 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++)
29             if (m[i][j] < min) {
30                 min = m[i][j];
31                 indeks_kolone = j;
32             }
33
34     return indeks_kolone;
35 }
36
37 /* Funkcija izracunava indeks vrste najveceg elementa */
38 int indeks_max(int m[][][MAX], int n)
39 {
40     int i, j;
41     int max = m[0][0], indeks_vrste = 0;
42
43     for (i = 0; i < n; i++)
44         for (j = 0; j < n; j++)
45             if (m[i][j] > max) {
46                 max = m[i][j];
47                 indeks_vrste = i;
48             }
49
50     return indeks_vrste;
51 }
```

```

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

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

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

61     return broj_negativnih;
62 }

63 int main(int argc, char *argv[])
64 {
65     int m[MAX][MAX];
66     int n;
67     int i, j;

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

75     /* Ucitava se broj vrsta matrice */
76     n = atoi(argv[1]);

77     if (n > MAX || n <= 0) {
78         fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
79         fprintf(stderr, "matrice.\n");
80         exit(EXIT_FAILURE);
81     }

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

87     printf("Najveci element sporedne dijagonale je %.1f",
88 
```

```

96         max_sporedna_dijagonala(m, n));

98     printf("Indeks kolone sa najmanjim elementom je %d.\n",
100        indeks_min(m, n));
102
104     printf("Indeks vrste sa najvecim elementom je %d.\n",
106        indeks_max(m, n));
108
109     printf("Broj negativnih elemenata matrice je %d.\n",
110        broj_negativnih(m, n));
111
112     exit(EXIT_SUCCESS);
113 }
```

Rešenje 2.12

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
5
6 /* Funkcija ucitava elemente kvadratne matrice sa standardnog ulaza
7   */
8 void ucitaj_matricu(int m[][MAX], int n)
9 {
10    int i, j;
11
12    for (i = 0; i < n; i++)
13        for (j = 0; j < n; j++)
14            scanf("%d", &m[i][j]);
15
16    /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
17   */
18 void ispisi_matricu(int m[][MAX], int n)
19 {
20    int i, j;
21
22    for (i = 0; i < n; i++) {
23        for (j = 0; j < n; j++)
24            printf("%d ", m[i][j]);
25        printf("\n");
26    }
27
28    /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
29    da li je normirana i ortogonalna. Matrica je normirana ako je
30    proizvod svake vrste matrice sa samom sobom jednak jedinici.
31    Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
32    vrste matrice jednak nuli */
33 int ortonormirana(int m[][MAX], int n)
```

```

35    {
36        int i, j, k;
37        int proizvod;
38
39        /* Ispituje se uslov normiranosti */
40        for (i = 0; i < n; i++) {
41            proizvod = 0;
42
43            for (j = 0; j < n; j++)
44                proizvod += m[i][j] * m[i][j];
45
46            if (proizvod != 1)
47                return 0;
48        }
49
50        /* Ispituje se uslov ortogonalnosti */
51        for (i = 0; i < n - 1; i++) {
52            for (j = i + 1; j < n; j++) {
53
54                proizvod = 0;
55
56                for (k = 0; k < n; k++)
57                    proizvod += m[i][k] * m[j][k];
58
59                if (proizvod != 0)
60                    return 0;
61            }
62
63            /* Ako su oba uslova ispunjena, matrica je ortonormirana */
64            return 1;
65        }
66
67    int main()
68    {
69        int A[MAX][MAX];
70        int n;
71
72        printf("Unesite broj vrsta matrice: ");
73        scanf("%d", &n);
74
75        if (n > MAX || n <= 0) {
76            fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
77            fprintf(stderr, "matrice.\n");
78            exit(EXIT_FAILURE);
79        }
80
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");

```

```

87     exit(EXIT_SUCCESS);
}

```

Rešenje 2.13

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 32
5
6 /* Funkcija ucitava elemente kvadratne matrice sa standardnog ulaza
7  */
8 void ucitaj_matricu(int m[][][MAX], int n)
{
9     int i, j;
10
11    for (i = 0; i < n; i++)
12        for (j = 0; j < n; j++)
13            scanf("%d", &m[i][j]);
14
15    /* Funkcija ispisuje elemente kvadratne matrice na standardni izlaz
16     */
17 void ispisi_matricu(int m[][][MAX], int n)
{
18     int i, j;
19
20    for (i = 0; i < n; i++) {
21        for (j = 0; j < n; j++)
22            printf("%d ", m[i][j]);
23        printf("\n");
24    }
25
26    /* Funkcija proverava da li je zadata matrica ortonormirana, odnosno,
27     da li je normirana i ortogonalna. Matrica je normirana ako je
28     proizvod svake vrste matrice sa samom sobom jednak jedinicama.
29     Matrica je ortogonalna, ako je proizvod dve bilo koje razlicite
30     vrste matrice jednak nuli */
31 int ortonormirana(int m[][][MAX], int n)
{
32     int i, j, k;
33     int proizvod;
34
35     /* Ispituje se uslov normiranosti */
36     for (i = 0; i < n; i++) {
37         proizvod = 0;
38
39         for (j = 0; j < n; j++)
40             proizvod += m[i][j] * m[i][j];
41
42         if (proizvod != 1)
43             return 0;
44     }
45
46     for (i = 0; i < n; i++)
47         for (j = i + 1; j < n; j++)
48             if (m[i][j] != 0)
49                 return 0;
50
51     return 1;
52 }

```

```

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

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

53             proizvod = 0;

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

57             if (proizvod != 0)
58                 return 0;
59         }
60     }

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

67 int main()
68 {
69     int A[MAX][MAX];
70     int n;
71
72     printf("Unesite broj vrsta 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     printf("Matrica %s ortonormirana.\n",
84           ortonormirana(A, n) ? "je" : "nije");
85
86     exit(EXIT_SUCCESS);
87 }

```

Rešenje 2.15

```

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

```

```

5   int main()
6   {
7     int *p = NULL;
8     int i, n;
9
10    printf("Unesite dimenziju niza: ");
11    scanf("%d", &n);
12
13    /* Alocira se prostor za n celih brojeva */
14    if ((p = (int *) malloc(sizeof(int) * n)) == NULL) {
15      fprintf(stderr, "malloc(): ");
16      fprintf(stderr, "greska pri alokaciji memorije.\n");
17      exit(EXIT_FAILURE);
18    }
19
20    printf("Unesite elemente niza: ");
21    for (i = 0; i < n; i++)
22      scanf("%d", &p[i]);
23
24    printf("Niz u obrnutom poretku je: ");
25    for (i = n - 1; i >= 0; i--)
26      printf("%d ", p[i]);
27    printf("\n");
28
29    /* Oslobadja se prostor rezervisan funkcijom malloc() */
30    free(p);
31
32    exit(EXIT_SUCCESS);
33  }

```

Rešenje 2.16

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define KORAK 10
4
5 int main()
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     char realokacija;
22
23     /* Inicijalizacija */
24     alocirano = n = 0;
25
26     printf("Unesite zeljeni nacin realokacije (M ili R):\n");
27     scanf("%c", &realokacija);
28
29     printf("Unesite brojeve, nulu za kraj:\n");
30     scanf("%d", &x);
31
32     while (x != 0) {
33         if (n == alocirano) {
34             alocirano = alocirano + KORAK;
35
36             if (realokacija == 'M') {
37                 /* Vrsi se realokacija memorije sa novom velicinom */
38                 b = (int *) malloc(alocirano * sizeof(int));
39
40                 if (b == NULL) {
41                     fprintf(stderr, "malloc(): ");
42                     fprintf(stderr, "greska pri alokaciji memorije.\n");
43                     free(a);
44                     exit(EXIT_FAILURE);
45                 }
46
47                 /* Svih n elemenata koji pocinju na adresi a prepisujemo na
48                  novu adresu b */
49                 for (i = 0; i < n; i++)
50                     b[i] = a[i];
51
52                 free(a);
53
54                 /* Promenljivoj a dodeljuje se adresa pocetka novog, veceg
55                  bloka koji je prilikom alokacije zapamcen u promenljivoj b
56                  */
57                 a = b;
58             } else if (realokacija == 'R') {
59
60                 /* Zbog funkcije realloc je neophodno da i u prvoj iteraciji
61                  "a" bude inicijalizovano na NULL */
62
63                 a = (int *) realloc(a, alocirano * sizeof(int));
64                 if (a == NULL) {
65                     fprintf(stderr, "realloc(): ");
66                     fprintf(stderr, "greska pri alokaciji memorije.\n");
67                     exit(EXIT_FAILURE);
68                 }
69             }
70
71             a[n++] = x;

```

```

72     scanf("%d", &x);
74 }

76 printf("Niz u obrnutom poretku je: ");
77 for (n--; n >= 0; n--)
78     printf("%d ", a[n]);
79 printf("\n");
80
81 /* Oslobadja se dinamicki alocirana memorija */
82 free(a);

83 exit(EXIT_SUCCESS);
84 }

```

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) {
17         fprintf(stderr, "malloc(): ");
18         fprintf(stderr, "greska pri alokaciji memorije.\n");
19         exit(EXIT_FAILURE);
20     }
21
22     /* Kopiraju se i nadovezuju niske karaktera */
23     strcpy(p, s);
24     strcat(p, t);
25
26     return p;
27 }
28
29 int main()
30 {
31     char *s = NULL;
32     char s1[MAX], s2[MAX];
33
34     printf("Unesite dve niske karaktera:\n");

```

```

34     scanf("%s", s1);
35     scanf("%s", s2);

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

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

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

42     /* Izlazi sa uspešnim izlazom */
43     exit(EXIT_SUCCESS);
44 }
```

Rešenje 2.18

```

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

5 int main()
6 {
7     int i, j;

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

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

15    /* Trag matice */
16    double trag = 0;

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

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

23    /* Provera se da li je doslo do greske pri alokaciji */
24    if (A == NULL) {
25        fprintf(stderr, "malloc(): ");
26        fprintf(stderr, "greska pri alokaciji memorije.\n");
27        exit(EXIT_FAILURE);
28    }

31    /* Dinamicki se alocira prostor za elemente u vrstama */
32    for (i = 0; i < n; i++) {
33        A[i] = (double **)malloc(sizeof(double) * m);
34    }
35 }
```

```

35     /* Ukoliko je alokacija neuspesna, pre zavrsetka programa
36      potrebno je oslobiti svih i-1 prethodno alociranih vrsta, i
37      alociran niz pokazivaca */
38      if (A[i] == NULL) {
39          for (j = 0; j < i; j++)
40              free(A[j]);
41          free(A);
42          exit(EXIT_FAILURE);
43      }
44  }
45
46  printf("Unesite elemente matrice, vrstu po vrstu:\n");
47  for (i = 0; i < n; i++)
48      for (j = 0; j < m; j++)
49          scanf("%lf", &A[i][j]);
50
51  /* Izracunava se trag matrice, odnosno suma elemenata na glavnoj
52   * dijagonali */
53  trag = 0.0;
54
55  for (i = 0; i < n; i++)
56      trag += A[i][i];
57
58  printf("Trag unete matrice je %.2f.\n", trag);
59
60  /* Oslobadja se prostor rezervisan za svaku vrstu */
61  for (j = 0; j < n; j++)
62      free(A[j]);
63
64  /* Oslobadja se memorija za niz pokazivaca na vrste */
65  free(A);
66
67  exit(EXIT_SUCCESS);
}

```

Rešenje 2.19

matrica.h

```

1 #ifndef _MATRICA_H_
2 #define _MATRICA_H_ 1
3
4 /* Funkcija dinamicki alocira memoriju za matricu dimenzija n x m */
5 int **alociraj_matricu(int n, int m);
6
7 /* Funkcija dinamicki alocira memoriju za kvadratnu matricu dimenzije
8   n */
9 int **alociraj_kvadratnu_matricu(int n);
10
11 /* Funkcija dealocira memoriju za matricu sa n vrsta */

```

```

13 int **dealociraj_matricu(int **matrica, int n);
14 /* Funkcija ucitava vec alociranu matricu dimenzija n x m sa
15 standardnog ulaza */
16 void ucitaj_matricu(int **matrica, int n, int m);
17 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n sa
18 standardnog ulaza */
19 void ucitaj_kvadratnu_matricu(int **matrica, int n);
20 /* Funkcija ispisuje matricu dimenzija n x m na standardnom izlazu */
21 void ispisi_matricu(int **matrica, int n, int m);
22 /* Funkcija ispisuje kvadratnu matricu dimenzije n na standardnom
23 izlazu */
24 void ispisi_kvadratnu_matricu(int **matrica, int n);
25 /* Funkcija ucitava vec alociranu matricu dimenzija n x m iz datoteke
26 f */
27 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f);
28 ;
29 /* Funkcija ucitava vec alociranu kvadratnu matricu dimenzije n iz
30 datoteke f */
31 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
32 FILE * f);
33 /* Funkcija upisuje matricu dimenzija n x m u datoteku f */
34 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f);
35 /* Funkcija upisuje kvadratnu matricu dimenzije n u datoteku f */
36 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n,
37 FILE * f);
38 ;
39 #endif

```

matrica.c

```

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

5 int **alociraj_matricu(int n, int m)
{
7     int **matrica = NULL;
8     int i, j;
9
10    /* Alocira se prostor za niz vrsta matrice */
11    matrica = (int **) malloc(n * sizeof(int *));
12    /* Ako alokacija nije prosla uspesno, povratna vrednost funkcije ce
13     biti NULL, sto mora biti provereno u main funkciji */

```

```

15     if (matrica == NULL)
16         return NULL;
17
18     /* Alocira se prostor za svaku vrstu matrice */
19     for (i = 0; i < n; i++) {
20         matrica[i] = (int *) malloc(m * sizeof(int));
21         /* Ako alokacija nije prosla uspesno, oslobadaju se svi
22          prethodno alocirani resursi, i povratna vrednost je NULL */
23         if (matrica[i] == NULL) {
24             for (j = 0; j < i; j++)
25                 free(matrica[j]);
26             free(matrica);
27             return NULL;
28         }
29     }
30
31     return matrica;
32 }
33
34 int **alociraj_kvadratnu_matricu(int n)
35 {
36     /* Alociranje matrice dimenzije n x n */
37     return alociraj_matricu(n, n);
38 }
39
40 int **dealociraj_matricu(int **matrica, int n)
41 {
42     int i;
43     /* Oslobadja se prostor rezervisan za svaku vrstu */
44     for (i = 0; i < n; i++)
45         free(matrica[i]);
46     /* Oslobadja se memorija za niz pokazivaca na vrste */
47     free(matrica);
48
49     /* Matrica postaje prazna, tj. nealocirana */
50     return NULL;
51 }
52
53 void ucitaj_matricu(int **matrica, int n, int m)
54 {
55     int i, j;
56     /* Elementi matrice se ucitacaju po vrstama */
57     for (i = 0; i < n; i++)
58         for (j = 0; j < m; j++)
59             scanf("%d", &matrica[i][j]);
60
61     void ucitaj_kvadratnu_matricu(int **matrica, int n)
62     {
63         /* Ucitavanje matrice n x n */
64         ucitaj_matricu(matrica, n, n);
65     }

```

```

67 void ispisi_matricu(int **matrica, int n, int m)
68 {
69     int i, j;
70     /* Ispis po vrstama */
71     for (i = 0; i < n; i++) {
72         for (j = 0; j < m; j++)
73             printf("%d ", matrica[i][j]);
74         printf("\n");
75     }
76 }
77 void ispisi_kvadratnu_matricu(int **matrica, int n)
78 {
79     /* Ispis matrice n x n */
80     ispisi_matricu(matrica, n, n);
81 }
82
83 int ucitaj_matricu_iz_datoteke(int **matrica, int n, int m, FILE * f)
84 {
85     int i, j;
86     /* Elementi matrice se ucitacaju po vrstama */
87     for (i = 0; i < n; i++)
88         for (j = 0; j < m; j++)
89             /* Ako je nemoguce ucitati sledeci element, povratna vrednost
90                funkcije je 1, kao indikator neuspesnog ucitavanja */
91             if (fscanf(f, "%d", &matrica[i][j]) != 1)
92                 return 1;
93
94     /* Uspesno ucitana matrica */
95     return 0;
96 }
97
98 int ucitaj_kvadratnu_matricu_iz_datoteke(int **matrica, int n,
99                                              FILE * f)
100 {
101     /* Ucitavanje matrice n x n iz datoteke */
102     return ucitaj_matricu_iz_datoteke(matrica, n, n, f);
103 }
104
105 int upisi_matricu_u_datoteku(int **matrica, int n, int m, FILE * f)
106 {
107     int i, j;
108     /* Ispis po vrstama */
109     for (i = 0; i < n; i++) {
110         for (j = 0; j < m; j++)
111             /* Ako je nemoguce ispisati sledeci element, povratna vrednost
112                funkcije je 1, kao indikator neuspesnog ispisa */
113             if (fprintf(f, "%d ", matrica[i][j]) <= 0)
114                 return 1;
115             fprintf(f, "\n");
116     }
117 }

```

```

119     /* Uspesno upisana matrica */
120     return 0;
121 }
122 int upisi_kvadratnu_matricu_u_datoteku(int **matrica, int n, FILE * f
123 )
124 {
125     /* Ispis matrice n x n u datoteku */
126     return upisi_matricu_u_datoteku(matrica, n, n, f);
127 }
```

main_a.c

```

2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "matrica.h"
5
6 int main()
7 {
8     int **matrica = NULL;
9     int n, m;
10    FILE *f;
11
12    /* Ucitavanje dimenzije matrice */
13    printf("Unesi broj vrsta matrice: ");
14    scanf("%d", &n);
15    printf("Unesi broj kolona matrice: ");
16    scanf("%d", &m);
17
18    /* Provera dimenzija matrice */
19    if (n <= 0 || m <= 0) {
20        fprintf(stderr, "Neodgovarajce dimenzije matrice\n");
21        exit(EXIT_FAILURE);
22    }
23
24    /* Alokacija matrice i provera alokacije */
25    matrica = alociraj_matricu(n, m);
26    if (matrica == NULL) {
27        fprintf(stderr, "Neuspesna alokacija matrice\n");
28        exit(EXIT_FAILURE);
29    }
30
31    /* Ucitavanje matrice sa standardnog ulaza */
32    printf("Unesi elemente matrice po vrstama:\n");
33    ucitaj_matricu(matrica, n, m);
34
35    /* Otvaranje fajla za upis matrice */
36    if ((f = fopen("matrica.txt", "w")) == NULL) {
37        fprintf(stderr, "fopen() error\n");
38        matrica = dealociraj_matricu(matrica, n);
39        exit(EXIT_FAILURE);
40    }
41
42    /* Upis matrice u fajl */
43    upisi_kvadratnu_matricu_u_datoteku(matrica, n, f);
44
45    /* Zatvaranje fajla */
46    fclose(f);
47
48    /* Slobodovanje memorije */
49    dealociraj_matricu(matrica, n);
50
51    /* Izlazak iz programa */
52    exit(EXIT_SUCCESS);
53 }
```

```

40 }
41
42 /* Upis matrice u fajl */
43 if (upisi_matricu_u_datoteku(matrica, n, m, f) != 0) {
44     fprintf(stderr, "Neuspesno upisivanje matrice u datoteku\n");
45     matrica = dealociraj_matricu(matrica, n);
46     exit(EXIT_FAILURE);
47 }
48
49 /* Zatvaranje fajla */
50 fclose(f);
51
52 /* Dealokacija matrice */
53 matrica = dealociraj_matricu(matrica, n);
54
55 exit(EXIT_SUCCESS);
56 }

```

main_b.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 int main(int argc, char **argv)
6 {
7     int **matrica = NULL;
8     int n;
9     FILE *f;
10
11    /* Provera argumenata komandne linije */
12    if (argc != 2) {
13        fprintf(stderr, "Koriscenje programa: %s datoteka\n", argv[0]);
14        exit(EXIT_FAILURE);
15    }
16
17    /* Otvaranje fajla za citanje */
18    if ((f = fopen(argv[1], "r")) == NULL) {
19        fprintf(stderr, "fopen() error\n");
20        exit(EXIT_FAILURE);
21    }
22
23    /* Ucitavanje dimenzije matrice */
24    if (fscanf(f, "%d", &n) != 1) {
25        fprintf(stderr, "Neispravan pocetak fajla\n");
26        exit(EXIT_FAILURE);
27    }
28
29    /* Provera dimenzije matrice */
30    if (n <= 0) {
31        fprintf(stderr, "Neodgovarajca dimenzija matrice\n");
32    }

```

```

    exit(EXIT_FAILURE);
33 }

35 /* Alokacija matrice i provera alokacije */
36 matrica = alociraj_kvadratnu_matricu(n);
37 if (matrica == NULL) {
38     fprintf(stderr, "Neuspesna alokacija matrice\n");
39     exit(EXIT_FAILURE);
40 }

41 /* Ucitavanje matrice iz datoteke */
42 if (ucitaj_kvadratnu_matricu_iz_datoteke(matrica, n, f) != 0) {
43     fprintf(stderr, "Neuspesno ucitavanje matrice iz datoteke\n");
44     matrica = dealociraj_matricu(matrica, n);
45     exit(EXIT_FAILURE);
46 }

47 /* Zatvaranje fajla */
48 fclose(f);

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

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

56 exit(EXIT_SUCCESS);
59 }

```

Rešenje 2.20

```

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

5 /* Funkcija ispisuje elemente matrice ispod glavne dijagonale */
6 void ispisi_elemente_ispod_dijagonale(int **M, int n, int m)
7 {
8     int i, j;

9     for (i = 0; i < n; i++) {
10         for (j = 0; j <= i; j++)
11             printf("%d ", M[i][j]);
12         printf("\n");
13     }
14 }

15 int main()
16 {
17     int m, n;
18
19 }

```

```

21 int **matrica = NULL;
23 printf("Unesite broj vrsta i broj kolona matrice:\n ");
25 scanf("%d %d", &n, &m);
27 /* Alocira se matrica */
29 matrica = alociraj_matricu(n, m);
31 /* Provera alokacije */
33 if (matrica == NULL) {
35     fprintf(stderr, "Neuspesna alokacija matrice\n");
37     exit(EXIT_FAILURE);
39 }
41 printf("Unesite elemente matrice, vrstu po vrstу:\n");
43 ucitaj_matricu(matrica, n, m);
45 printf("Elementi ispod glavne dijagonale matrice:\n");
47 ispisi_elemlente_ispod_dijagonale(matrica, n, m);
49 /* Oslobođanje memorije */
51 matrica = dealociraj_matricu(matrica, n);
53 exit(EXIT_SUCCESS);
}

```

Rešenje 2.22

```

#include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 /* Funkcija izvrsava trazene transformacije nad matricom */
6 void izmeni(float **a, int n)
7 {
8     int i, j;
9
10    for (i = 0; i < n; i++)
11        for (j = 0; j < n; j++)
12            if (i < j)
13                a[i][j] /= 2;
14            else if (i > j)
15                a[i][j] *= 2;
16
17 /* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
18 sporedne dijagonale. Element se nalazi ispod sporedne dijagonale
20 ukoliko je zbir indeksa vrste i indeksa kolone elementa veci od
21 n-1 */
22 float zbir_ispod_sporedne_dijagonale(float **m, int n)
23 {
24     int i, j;

```

```

26     float zbir = 0;
27
28     for (i = 0; i < n; i++)
29         for (j = n-i; j < n; j++)
30             if (i + j > n - 1)
31                 zbir += fabs(m[i][j]);
32
33     return zbir;
34 }
35
36 /* Funkcija ucitava elemente kvadratne matrice dimenzije n iz zadate
37  datoteke */
38 void ucitaj_matricu(FILE * ulaz, float **m, int n)
39 {
40     int i, j;
41
42     for (i = 0; i < n; i++)
43         for (j = 0; j < n; j++)
44             fscanf(ulaz, "%f", &m[i][j]);
45 }
46
47 /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
48  standardni izlaz */
49 void ispisi_matricu(float **m, int n)
50 {
51     int i, j;
52
53     for (i = 0; i < n; i++) {
54         for (j = 0; j < n; j++)
55             printf("%.2f ", m[i][j]);
56         printf("\n");
57     }
58 }
59
60 /* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
61 float **alociraj_memoriju(int n)
62 {
63     int i, j;
64     float **m;
65
66     m = (float **) malloc(n * sizeof(float *));
67     if (m == NULL) {
68         fprintf(stderr, "malloc(): Neuspela alokacija\n");
69         exit(EXIT_FAILURE);
70     }
71
72     for (i = 0; i < n; i++) {
73         m[i] = (float *) malloc(n * sizeof(float));
74
75         if (m[i] == NULL) {
76             printf("malloc(): neuspela alokacija memorije!\n");
77             for (j = 0; j < i; j++)
78                 free(m[j]);
79             free(m);
80             exit(EXIT_FAILURE);
81         }
82     }
83 }
84
85 /* Funkcija sloboduje memoriju koju je alocirala funkcija
86  alociraj_memoriju */
87 void slobodi_memoriju(float **m, int n)
88 {
89     int i;
90
91     for (i = 0; i < n; i++)
92         free(m[i]);
93
94     free(m);
95 }

```

```

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

```

```

128  /* Poziva se funkcija za transformaciju matrice */
129  izmeni(a, n);
130
132  /* Ispisuje se rezultat */
133  printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
134  printf("je %.2f.\n", zbir);
135
136  printf("Transformisana matrica je:\n");
137  ispisi_matricu(a, n);
138
139  /* Oslobadja se memorija */
140  oslobodi_memoriju(a, n);
141
142  /* Zatvara se datoteka */
143  fclose(ulaz);
144
145  exit(EXIT_SUCCESS);
146 }
```

Rešenje 2.27

```

1 #include <stdio.h>
3 #include <stdlib.h>
5 #include <math.h>
5 #include <string.h>
6
7 /* Funkcija tabela() prihvata granice intervala a i b, broj
9  ekvidistantnih tacaka n, kao i pokazivac f koji pokazuje na
10 funkcijsku koju prihvata double argument, i vraca double vrednost.
11 Za takoj funkciji ispisuju se njene vrednosti u intervalu
12 [a,b] u n ekvidistantnih tacaka intervala */
13 void tabela(double a, double b, int n, double (*fp) (double))
14 {
15     int i;
16     double x;
17
18     printf("-----\n");
19     for (i = 0; i < n; i++) {
20         x = a + i * (b - a) / (n - 1);
21         printf(" | %.5f | %.5f | \n", x, (*fp)(x));
22     }
23     printf("-----\n");
24 }
25
26 double sqr(double a)
27 {
28     return a * a;
29 }
```

```

1 int main(int argc, char *argv[])
2 {
3     double a, b;
4     int n;
5
6     char ime_funkcije[6];
7
8     /* Pokazivac na funkciju koja ima jedan argument tipa double i
9      povratnu vrednost istog tipa */
10    double (*fp) (double);
11
12    /* Ako korisnik nije uneo trazene argumente, prijavljuje se greska
13     */
14    if (argc < 2) {
15        printf("Greska: ");
16        printf("Nedovoljan broj argumenata komandne linije.\n");
17        printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
18               argv[0]);
19        exit(EXIT_FAILURE);
20    }
21
22    /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
23     u
24     komandnoj liniji */
25    strcpy(ime_funkcije, argv[1]);
26
27    /* Inicijalizuje se pokazivac na funkciju koja treba da se tabelira
28     */
29    if (strcmp(ime_funkcije, "sin") == 0)
30        fp = &sin;
31    else if (strcmp(ime_funkcije, "cos") == 0)
32        fp = &cos;
33    else if (strcmp(ime_funkcije, "tan") == 0)
34        fp = &tan;
35    else if (strcmp(ime_funkcije, "atan") == 0)
36        fp = &atan;
37    else if (strcmp(ime_funkcije, "acos") == 0)
38        fp = &acos;
39    else if (strcmp(ime_funkcije, "asin") == 0)
40        fp = &asin;
41    else if (strcmp(ime_funkcije, "exp") == 0)
42        fp = &exp;
43    else if (strcmp(ime_funkcije, "log") == 0)
44        fp = &log;
45    else if (strcmp(ime_funkcije, "log10") == 0)
46        fp = &log10;
47    else if (strcmp(ime_funkcije, "sqrt") == 0)
48        fp = &sqrt;
49    else if (strcmp(ime_funkcije, "floor") == 0)
50        fp = &floor;
51    else if (strcmp(ime_funkcije, "ceil") == 0)
52        fp = &ceil;
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79

```

```
81     else if (strcmp(ime_funkcije, "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("(uključujući krajeve intervala)?\n");
93     scanf("%d", &n);
94
95     /* Mreza mora da uključuje 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_funkcije);
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 }
```

3

Algoritmi pretrage i sortiranja

3.1 Algoritmi pretrage

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

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

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

3.1 Algoritmi pretrage

Test 1

```
|| POKRETANJE: ./a.out 1000000          VREMENA.TXT
      23542
|| IZLAZ:
  Linearna pretraga:
  Element nije u nizu
  Binarna pretraga:
  Element nije u nizu
  Interpolaciona pretraga:
  Element nije u nizu
Dimenzija niza: 1000000
Linearna: 3615091 ns
Binarna: 1536 ns
Interpolaciona: 558 ns
```

Test 2

```
|| POKRETANJE: ./a.out 100000          VREMENA.TXT
      37842
|| IZLAZ:
  Linearna pretraga:
  Element nije u nizu
  Binarna pretraga:
  Element nije u nizu
  Interpolaciona pretraga:
  Element nije u nizu
Dimenzija niza: 1000000
Linearna: 3615091 ns
Binarna: 1536 ns
Interpolaciona: 558 ns
Dimenzija niza: 1000000
Linearna: 360803 ns
Binarna: 1187 ns
Interpolaciona: 628 ns
```

[Rešenje 3.1]

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

Primer 1

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

Primer 2

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

[Rešenje 3.2]

Zadatak 3.3 Napisati program koji preko argumenta komandne linije dobija ime datoteke koja sadrži sortirani spisak studenta po broju indeksa rastuće. Za svakog studenta u jednom redu stoe informacije o indeksu, imenu i prezimenu. Program učitava spisak studenata u niz i traži od korisnika indeks ili prezime studenta čije informacije se potom prikazuju na ekranu. U slučaju više studenata sa istim prezimenom prikazati informacije o prvom takvom. Odabir kriterijuma pretrage se vrši kroz poslednji argument komandne linije, koji može biti **-indeks** ili **-prezime**. U slučaju neuspešnih pretragi, stampati odgovarajuću poruku. Pretrage implementirati u vidu iterativnih funkcija što manje složenoosti. Pretpostaviti da u datoteci neće biti više od 128 studenata i da su imena i prezimena svih kraća od 16 slova.

Primer 1

```
POKRETANJE: ./a.out datoteka.txt -indeks  
  
DATOTEKA.TXT  
20140003 Marina Petrovic  
20140012 Stefan Mitrovic  
20140032 Dejan Popovic  
20140049 Mirko Brankovic  
20140076 Sonja Stevanovic  
20140104 Ivan Popovic  
20140187 Vlada Stankovic  
20140234 Darko Brankovic  
  
INTERAKCIJA SA PROGRAMOM:  
Unesite indeks studenta  
cije informacije zelite:  
20140076  
Indeks: 20140076,  
Ime i prezime: Sonja Stevanovic
```

Primer 2

```
POKRETANJE: ./a.out datoteka.txt -prezime  
  
DATOTEKA.TXT  
20140003 Marina Petrovic  
20140012 Stefan Mitrovic  
20140032 Dejan Popovic  
20140049 Mirko Brankovic  
20140076 Sonja Stevanovic  
20140104 Ivan Popovic  
20140187 Vlada Stankovic  
20140234 Darko Brankovic  
  
INTERAKCIJA SA PROGRAMOM:  
Unesite prezime studenta  
cije informacije zelite:  
Popovic  
Indeks: 20140032,  
Ime i prezime: Dejan Popovic
```

[Rešenje 3.3]

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

[Rešenje 3.4]

Zadatak 3.5 U datoteci koja se zadaje kao prvi argument komandne linije, nalaze se koordinate tačaka. U zavisnosti od prisustva opcija komandne linije (**-x** ili **-y**), pronaći onu koja je najbliža **x**, ili **y** osi, ili koordinatnom početku, ako

3.1 Algoritmi pretrage

nije prisutna nijedna opcija. Pretpostaviti da je broj tačaka u datateci veći od 0 i ne veći od 1024.

Test 1	Test 2	Test 3
POKRETANJE: ./a.out dat.txt -	POKRETANJE: ./a.out dat.txt	POKRETANJE: ./a.out dat.txt -y
DAT.TXT 12 53 2.342 34.1 -0.3 23 -1 23.1 123.5 756.12	DAT.TXT 12 53 2.342 34.1 -0.3 23 -1 2.1 123.5 756.12	DAT.TXT 12 53 2.342 34.1 -0.3 0.23 -1 2.1 123.5 756.12
IZLAZ: -0.3 23	IZLAZ: -1 2.1	IZLAZ: -0.3 0.23

[Rešenje 3.5]

Zadatak 3.6 Napisati funkciju koja određuje nulu funkcije $\cos(x)$ na intervalu $[0, 2]$ metodom polovljenja intervala. Algoritam se završava kada se vrednost kosinusne funkcije razlikuje za najviše 0.001 od nule. UPUTSTVO: *Korisiti algoritam analogan algoritmu binarne pretrage, metod polovljenja intervala.* NAPOMENA: *Ovaj metod se može primeniti na funkciju $\cos(x)$ na intervalu $[0, 2]$ zato što je ona na ovom intervalu neprekidna, i vrednosti funkcije na krajevima intervala su različitog znaka.*

Test 1

IZLAZ:	1.57031
--------	---------

[Rešenje 3.6]

Zadatak 3.7 Napisati funkciju koja metodom polovljenja intervala određuje nulu izabrane funkcije na proizvolnjom intervalu sa tačnošću *epsilon*. Ime funkcije se zadaje kao prvi argument komandne linije, a interval i tačnost se unose sa standardnog ulaza. Pretpostaviti da je izabrana funkcija na tom intervalu neprekidna. UPUTSTVO: *U okviru algoritma pretrage koristiti pokazivač na odgovarajuću funkciju (na primer, kao u zadatku ??).*

3.1 Algoritmi pretrage

Primer 1

```
POKRETANJE: ./a.out cos
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 0 2
Unesite preciznost: 0.001
1.57031
```

Primer 2

```
POKRETANJE: ./a.out sin
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 1 5
Unesite preciznost: 0.00001
3.1416
```

Primer 3

```
POKRETANJE: ./a.out tan
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: -1.1 1
Unesite preciznost: 0.00001
3.8147e-06
```

Primer 4

```
POKRETANJE: ./a.out sin
INTERAKCIJA SA PROGRAMOM:
Unesite krajeve intervala: 1 3
Funkcija sin na intervalu [1, 3]
ne zadovoljava uslove
```

[Rešenje 3.7]

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

Test 1

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

Test 2

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

Test 3

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

[Rešenje 3.8]

Zadatak 3.9 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.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: <code>151 44 5 -12 -13 -15</code>	ULAZ: <code>100 55 15 0 -15 -124 -155 -258 -315 -516</code>	ULAZ: <code>100 15 11 8 7 5 4 3 2</code>
IZLAZ: <code>3</code>	IZLAZ: <code>4</code>	IZLAZ: <code>-1</code>

[Rešenje 3.9]

Zadatak 3.10 Napisati funkciju koja određuje ceo deo logaritma za osnovu 2 datog neoznačenog celog broja koristeći samo bitske i relacione operatore.

- (a) Napisati funkciju linearne složenosti koja određuje logaritam pomeranjem broja udesno.
- (b) Napisati funkciju logaritmske složenosti koja određuje logaritam koristeći binarnu pretragu.

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

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: <code>4</code>	ULAZ: <code>17</code>	ULAZ: <code>1031</code>
IZLAZ: <code>2 2</code>	IZLAZ: <code>4 4</code>	IZLAZ: <code>10 10</code>

[Rešenje 3.10]

*** Zadatak 3.11** U prvom kvadrantu dato je $1 \leq N \leq 10000$ duži svojim koordinatama (duži mogu da se sekut, 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

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesi broj tacaka: 2  
|| Unesi koordinate tacaka:  
|| 2 0 2 1  
|| 1 2 2 2  
|| 26.57
```

Primer 2

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesi broj tacaka: 2  
|| Unesi koordinate tacaka:  
|| 1 0 1 1  
|| 0 1 1 1  
|| 45
```

Primer 3

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Unesi broj tacaka: 3  
|| Unesi koordinate tacaka:  
|| 1 0 1 1  
|| 2 0 2 1  
|| 1 2 2 2  
|| 26.57
```

3.2 Algoritmi sortiranja

Zadatak 3.12 Napraviti biblioteku koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži algoritam sortiranja izborom (engl. *selection sort*), sortiranja spajanjem (engl. *merge sort*), brzog sortiranja (engl. *quick sort*), mehurastog sortiranja (engl. *bubble sort*), sortiranja direktnim umetanjem (engl. *insertion sort*) i sortiranja umetanjem sa inkrementom (engl. *shell sort*). Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na rastuće sortiranim nizovima i na opadajuće sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Moguće opcije kojima se bira algoritam sortiranja su: **-m** za sortiranje spajanjem, **-q** za brzo sortiranje, **-b** za mehurasto, **-i** za sortiranje direktnim umetanjem ili **-s** za sortiranje umetanjem sa inkrementom. U slučaju da nije prisutna ni jedna od ovih opcija, niz sortirati algoritmom sortiranja izborom. Niz koji se sortira generisati neopadajuće ako je prisutna opcija **-r**, nerastuće ako je prisutna opcija **-o** ili potpuno slučajno ako nema nijedne opcije. Vreme meriti programom *time*. Analizirati porast vremena sa porastom dimenzije *n*.

Test 1

```
|| POKRETANJE: time ./a.out  
|| 200000  
||  
|| IZLAZ:  
|| real 0m42.168s  
|| user 0m42.100s  
|| sys 0m0.000s
```

Test 2

```
|| POKRETANJE: time ./a.out  
|| 400000  
||  
|| IZLAZ:  
|| real 2m48.395s  
|| user 2m48.128s  
|| sys 0m0.000s
```

Test 3

```
|| POKRETANJE: time ./a.out  
|| 800000  
||  
|| IZLAZ:  
|| real 11m13.703s  
|| user 11m12.636s  
|| sys 0m0.000s
```

3.2 Algoritmi sortiranja

Test 4	Test 5	Test 6
POKRETANJE: time ./a.out 800000 -r	POKRETANJE: time ./a.out 800000 -q	POKRETANJE: time ./a.out 800000 -m
IZLAZ: real 11m21.533s user 11m20.436s sys 0m0.020s	IZLAZ: real 0m0.159s user 0m0.156s sys 0m0.000s	IZLAZ: real 0m0.137s user 0m0.136s sys 0m0.000s

[Rešenje 3.12]

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

Primer 1	Primer 2	Primer 3
INTERAKCIJA SA PROGRAMOM: Unesite prvu nisku anagram Unesite drugu nisku ramgana jesu	INTERAKCIJA SA PROGRAMOM: Unesite prvu nisku anagram Unesite drugu nisku anagr nisu	INTERAKCIJA SA PROGRAMOM: Unesite prvu nisku test Unesite drugu nisku tset jesu

[Rešenje 3.13]

Zadatak 3.14 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 pronadena dva broja. UPUTSTVO: *Prvo sortirati niz.* NAPOMENA: *Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.*

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

[Rešenje 3.14]

Zadatak 3.15 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*

3.2 Algoritmi sortiranja

niz, a zatim naći najdužu sekvencu jednakih elemenata. NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.

Test 1

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

Test 2

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

Test 3

ULAZ:	123
IZLAZ:	123

[Rešenje 3.15]

Zadatak 3.16 Napisati funkciju koja proverava da li u datom nizu postoje dva elementa čiji zbir je jednak zadatom celom broju. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz. Elementi niza se unose sve do kraja ulaza. Pretpostaviti da u niz neće biti uneto više od 256 brojeva. UPUTSTVO: Prvo sortirati niz. NAPOMENA: Koristiti biblioteku za sortiranje celih brojeva iz zadatka 3.12.

Primer 1

INTERAKCIJA SA PROGRAMOM:	Unesite trazenii zbir: 34
	Unesite elemente niza:
	134 4 1 6 30 23
	da

Primer 2

INTERAKCIJA SA PROGRAMOM:	Unesite trazenii zbir: 12
	Unesite elemente niza:
	53 1 43 3 56 13
	ne

Primer 3

INTERAKCIJA SA PROGRAMOM:	Unesite trazenii zbir: 52
	Unesite elemente niza:
	52
	ne

[Rešenje 3.16]

Zadatak 3.17 Napisati funkciju potpisa int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3, int dim3) koja prima dva sortirana niza, i na osnovu njih pravi novi sortirani niz koji koji sadrži elemente oba niza. Treća dimenzija predstavlja veličinu niza u koji se smešta rezultat. Ako je ona manja od potrebne dužine, funkcija vraća -1 kao indikator neuspeha, inače vraća 0. Napisati zatim program koji testira ovu funkciju. Nizovi se unose sa standardnog ulaza sve dok se ne unese 0 i može se pretpostaviti da će njihove dimenzije biti manje od 256.

Primer 1

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

Primer 2

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

3.2 Algoritmi sortiranja

[Rešenje 3.17]

Zadatak 3.18 Napisati program koji čita sadržaj dveju datoteka od kojih svaka sadrži spisak imena i prezimena studenata iz jedne od dve grupe, rastuće sortiran po imenima, a u slučaju istog imena po prezimenima, i kreira jedinstven spisak studenata sortiranih takođe po istom kriterijumu. Program dobija nazine datoteka iz komandne linije i jedinstveni spisak upisuje u datoteku **ceo-tok.txt**. Prepostaviti da je ime studenta nije duže od 10, a prezime od 15 karaktera.

Test 1

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

PRVI-DEO.TXT
Andrija Petrovic
Anja Ilic
Ivana Markovic
Lazar Micic
Nenad Brankovic
Sofija Filipovic
Uros Milic
Vladimir Savic

DRUGI-DEO.TXT
Aleksandra Cvetic
Bojan Golubovic
Dragan Markovic
Filip Dukic
Ivana Stankovic
Marija Stankovic
Ognjen Peric

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

[Rešenje 3.18]

Zadatak 3.19 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.

3.2 Algoritmi sortiranja

Test 1

```
POKRETANJE: ./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

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

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

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

[Rešenje 3.19]

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

Test 1

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

```
IZLAZ:  
3
```

Test 2

```
BIRACKI-SPISAK.TXT  
Milan Milicevic
```

```
IZLAZ:  
1
```

Test 3

```
DATOTEKA BIRACKI-SPISAK.TXT  
NE POSTOJI
```

```
IZLAZ ZA GREŠKU:  
Neuspesno otvaranje  
datoteke  
biracki-spisak.txt.
```

[Rešenje 3.20]

Zadatak 3.21 Definisati strukturu koja čuva imena, prezimena i godišta dece. Pretpostaviti da su imena i prezimena niske karaktera koje nisu duže od 30 karaktera. Napisati funkciju koja sortira niz dece po godištu, a decu istog godišta sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci čije se ime zadaje kao prvi argument

komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Prepostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 dece.

Test 1

```
|| POKRETANJE: ./a.out in.txt out.txt
|||
|| IN.OUT                                OUT.TXT
|| Petar Petrovic 2007                   Marija Antic 2007
|| Milica Antonic 2008                  Ana Petrovic 2007
|| Ana Petrovic 2007                     Petar Petrovic 2007
|| Ivana Ivanovic 2009                  Milica Antonic 2008
|| Dragana Markovic 2010                 Ivana Ivanovic 2009
|| Marija Antic 2007                     Dragana Markovic 2010
```

Test 2

```
|| POKRETANJE: ./a.out in.txt out.txt
|||
|| IN.OUT                                OUT.TXT
|| Milijana Maric 2009                  Milijana Maric 2009
```

Zadatak 3.22 Napisati funkciju koja sortira niz niski po broju suglasnika u niski. Ukoliko reči imaju isti broj suglasnika tada sortirati ih po dužini niske 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**. Prepostaviti da u nizu nema više od 128 elemenata, kao i da svaka niska sadrži najviše 31 karakter.

Test 1

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

[Rešenje 3.22]

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

Primer 1

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

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

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

UKUPNO: 270.00 dinara.

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

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

UKUPNO: 230.00 dinara.

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

Kraj rada kase!
```

[Rešenje 3.23]

Zadatak 3.24 Napisati program koji iz datoteke **aktivnost.txt** čita podatke o aktivnostima studenata na praktikumima i u datoteke **dat1.txt**, **dat2.txt** i **dat3.txt** upisuje redom tri spiska. Na prvom su studenti sortirani leksikografski po imenu rastuće. Na drugom su sortirani po ukupnom broju urađenih zadataka opadajuće, a ukoliko neki studenti imaju isti broj rešenih zadataka sortiraju se po dužini imena rastuće. Na trećem spisku kriterijum sortiranja je broj časova na kojima su bili opadajuće. Ukoliko neki studenti imaju isti broj časova, sortirati ih opadajuće po broju urađenih zadataka, a ukoliko se i on poklapa sortirati po

prezimenu opadajuće. U datoteci se nalazi ime, prezime studenta, broj časova na kojima je prisustvovao, kao i ukupan broj urađenih zadataka. Pretpostaviti da studenata neće biti više od 500 i da je za ime studenta dovoljno 20, a za prezime 25 karaktera.

Test 1

AKTIVNOSTI.TXT	DAT2.TXT
Aleksandra Cvetic 4 6	Studenti sortirani po broju zadataka opadajuće, pa po duzini imena rastuce:
Bojan Golubovic 4 3	Aleksandra Cvetic 4 6
Dragan Markovic 3 5	Uros Milic 2 5
Ivana Stankovic 3 1	Dragan Markovic 3 5
Marija Stankovic 1 3	Andrija Petrovic 2 5
Ognjen Peric 1 2	Nenad Brankovic 2 4
Uros Milic 2 5	Lazar Micic 1 3
Andrija Petrovic 2 5	Bojan Golubovic 4 3
Anja Ilic 3 1	Marija Stankovic 1 3
Lazar Micic 1 3	Ognjen Peric 1 2
Nenad Brankovic 2 4	Anja Ilic 3 1
	Ivana Stankovic 3 1
DAT1.TXT	DAT3.TXT
Studenti sortirani po imenu leksikografski rastuce:	Studenti sortirani po prisustvu opadajuće, pa po broju zadataka, pa po prezimenima leksikografski opadajuće:
Aleksandra Cvetic 4 6	Aleksandra Cvetic 4 6
Andrija Petrovic 2 5	Bojan Golubovic 4 3
Anja Ilic 3 1	Dragan Markovic 3 5
Bojan Golubovic 4 3	Ivana Stankovic 3 1
Dragan Markovic 3 5	Anja Ilic 3 1
Ivana Stankovic 3 1	Andrija Petrovic 2 5
Lazar Micic 1 3	Uros Milic 2 5
Marija Stankovic 1 3	Nenad Brankovic 2 4
Nenad Brankovic 2 4	Marija Stankovic 1 3
Ognjen Peric 1 2	Lazar Micic 1 3
Uros Milic 2 5	Ognjen Peric 1 2

[Rešenje 3.24]

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

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

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

- nema opcija, sortiranje se vrši po broju gledanja;

3.2 Algoritmi sortiranja

- 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 prepostavki o maksimalnoj dužini imena izvođača i naslova pesme.

Test 1

POKRETANJE: `./a.out`

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

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

Test 2

POKRETANJE: `./a.out -i`

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

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

Test 3

POKRETANJE: `./a.out -n`

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

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

[Rešenje [3.25](#)]

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

Primer 1

INTERAKCIJA SA PROGRAMOM:

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

* **Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze

3.2 Algoritmi sortiranja

najmanja, najveća, itd.... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

3	5	2	1
4	4	1	2
5	—	3	3
1	1	4	4
2	2	—	5

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

Test 1

ULAZ:

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

IZLAZ:

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

Zadatak 3.28 Za zadatu celobrojnu matricu dimenzije $n \times m$ napisati funkciju koja vrši sortiranje vrsta matrice rastuće na osnovu sume elemenata u vrstama. Napisati potom program koji testira ovu funkciju. Sa standardnog ulaza se prvo unose dimenzije matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka ??.*

Test 1

INTERAKCIJA SA PROGRAMOM:

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

Test 2

INTERAKCIJA SA PROGRAMOM:

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

[Rešenje 3.28]

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.29 Za zadatu kvadratnu matricu dimenzije n napisati funkciju koja sortira kolone matrice opadajuće na osnovu vrednosti prvog elementa u koloni. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardnom izlazu. NAPOMENA: *Koristiti biblioteku sa celobrojnim matricama iz zadatka ??.*

Primer 1

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

Primer 2

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

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.30 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 za greške. Niz struktura ima manje od 100 elemenata i uređen je u rastućem leksikografskom poretku po prezimenima. Pretazivanje niza vršiti bibliotečkom funkcijom **bsearch**. Na primer, niz osoba može da bude inicijalizovan na sledeći način:

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

Test 1

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

Test 2

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

Test 3

```
|| ULAZ:  
    X  
  
|| IZLAZ:  
    -1
```

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.31 Napisati program koji ilustruje upotrebu bibliotečkih funkcija za pretraživanje i sortiranje nizova i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva, ne veća od 100, a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu. Na standardnom izlazu ispisati odgovarajuću poruku.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 11  
Uneti elemente niza:  
5 3 1 6 8 90 34 5 3 432 34  
Sortirani niz u rastucem poretku:  
1 3 3 5 5 6 8 34 34 90 432  
Uneti element koji se trazi u nizu: 34  
Binarna pretraga:  
Element je nadjen na poziciji 8  
Linearna pretraga (lfind):  
Element je nadjen na poziciji 7
```

Primer 2

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

[Rešenje 3.31]

Zadatak 3.32 Napisati program koji sa standardnog ulaza učitava dimenziju niza celih brojeva (ne veću od 100), a potom i same elemente niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku prema broju delilaca i tako dobijeni niz odštampati na standardnom izlazu.

Primer 1

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

Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 1  
Uneti elemente niza:  
234  
Sortirani niz u rastucem  
poretku prema broju delilaca  
234
```

Primer 3

```
INTERAKCIJA SA PROGRAMOM:  
Uneti dimenziju niza: 0  
Uneti elemente niza:  
Sortirani niz u rastucem  
poretku prema broju  
delilaca:
```

[Rešenje 3.32]

Zadatak 3.33 Korišćenjem bibliotečke funkcije `qsort` napisati program koji sortira niz niski po sledećim kriterijumima:

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

3.3 Bibliotečke funkcije pretrage i sortiranja

Niske se učitavaju iz datoteke **niske.txt**. Prepostaviti da datoteka ne sadrži više od 1000 niski kao i da je svaka niska dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (**bsearch**) zarad traženja niske unete sa standardnog ulaza, a potom traži istu nisku koristeći funkciju **lfind** u nizu koji je neposredno pre toga sortiran po dužini. Rezultate svih sortiranja i pretraga ispisati na standardnom izlazu.

Primer 1

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

INTERAKCIJA SA PROGRAMOM:
Leksikografski sortirane niske:
aleksandar ana andjela ljubica matej milica milos nikola petar
Uneti traženu nisku: matej
Niska "matej" je pronadjena u nizu na poziciji 4
Niske sortirane po duzini:
ana matej milos petar milica nikola andjela ljubica aleksandar
Niska "matej" je pronadjena u nizu na poziciji 1
```

[Rešenje 3.33]

Zadatak 3.34 Uraditi prethodni zadatak 3.33 sa dinamički alociranim niskama i sortiranjem niza pokazivača, umesto niza niski.

[Rešenje 3.34]

Zadatak 3.35 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. Prepostaviti da neće biti više od 500 studenata i da je za ime i prezime svakog studenta dovoljno po 20 karaktera.

3.3 Bibliotečke funkcije pretrage i sortiranja

Primer 1

```
POKRETANJE: ./a.out kolokvijum.txt  
ULAZNA DATOTEKA (KOLOKVIJUM.TXT):  
Aleksandra Cvetic 15  
Bojan Golubovic 30  
Dragan Markovic 25  
Filip Dukic 20  
Ivana Stankovic 25  
Marija Stankovic 15  
Ognjen Peric 20  
Uros Milic 10  
Andrija Petrovic 0  
Anja Ilic 5  
Ivana Markovic 5  
Lazar Micic 20  
Nenad Brankovic 15  
  
INTERAKCIJA SA PROGRAMOM:  
Studenti sortirani po broju poena  
opadajuće, pa po prezimenu rastuce:  
Bojan Golubovic 30  
Dragan Markovic 25  
Ivana Stankovic 25  
Filip Dukic 20  
Lazar Micic 20  
Ognjen Peric 20  
Nenad Brankovic 15  
Aleksandra Cvetic 15  
Marija Stankovic 15  
Uros Milic 10  
Anja Ilic 5  
Ivana Markovic 5  
Andrija Petrovic 0  
Unesite broj bodova: 20  
Pronadjen je student sa unetim  
brojem bodova: Filip Dukic 20  
Unesite prezime: Markovic  
Pronadjen je student sa unetim  
prezimenom: Dragan Markovic 25
```

[Rešenje 3.35]

Zadatak 3.36 Uraditi zadatak 3.13, ali korišćenjem bibliotečke `qsort` funkcije.

[Rešenje 3.36]

Zadatak 3.37 Napisati program koji sa standardnog ulaza učitava prvo ceo broj n ($n \leq 10$), a zatim niz S od n niski. Maksimalna dužina svake niske je 31 karakter. Sortirati niz S bibliotečkom funkcijom `qsort` i proveriti da li u njemu ima identičnih niski.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj niski: 4  
Unesite niske:  
prog search sort search  
ima
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:  
Unesite broj niski: 3  
Unesite niske:  
test kol ispit  
nema
```

Primer 3

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

[Rešenje 3.37]

Zadatak 3.38 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

3.3 Bibliotečke funkcije pretrage i sortiranja

program koji korišćenjem funkcije `qsort` sortira studente po broju poena opadajuće, ukoliko je prisutna opcija `-p`, ili po nalogu, ukoliko je prisutna opcija `-n`. Studenti se po nalogu sortiraju tako što se sortiraju na osnovu godine, zatim na osnovu smera, i na kraju na osnovu broja indeksa. Sortirane studente upisati u datoteku `izlaz.txt`. Ukoliko je u komandnoj liniji uz opciju `-n` naveden i nalog nekog studenta, funkcijom `bsearch` potražiti i prijaviti broj poena studenta sa tim nalogom.

Test 1

```
POKRETANJE: ./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

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

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

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

[Rešenje 3.38]

Zadatak 3.39 Definisati strukturu `Datum`. Napisati funkciju koja poredi dva datuma hronološki. Potom, napisati i program koji učitava datume iz datoteke koja se zadaje kao prvi argument komandne linije (ne više od 128 datuma), sortira ih pozivajući funkciju `qsort` iz standardne biblioteke i pozivanjem funkcije `bsearch` iz standardne biblioteke proverava da li datumi učitani sa standardnog ulaza postoje među prethodno unetim datumima. Datumi se učitavaju sve do kraja ulaza.

Primer 1

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

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

INTERAKCIJA SA PROGRAMOM:
Unesi sledeći datum: 13.12.2016.
postoji
Unesi sledeći datum: 10.5.2015.
ne postoji
Unesi sledeći datum: 5.2.2009.
postoji
```

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
7 -lrt zbog funkcije clock_gettime() */
8
9 /* Naredne tri funkcije koje vrse pretragu, ukoliko se trazeni
10 element pronadje u nizu, vracaju indeks pozicije na kojoj je
11 element pronadjen. Ovaj indeks je uvek nenegativan. Ako element
12 nije pronadjen u nizu, funkcije vracaju negativnu vrednost -1, kao
13 indikator neuspesne pretrage. */
14
15 /* Linearna pretraga: Funkcija pretrazuje niz a[] celih brojeva
16 duzine n, trazeci u njemu prvo pojavljivanje elementa x. Pretraga
17 se vrsti prostom iteracijom kroz niz. */
18 int linearna_pretraga(int a[], int n, int x)
19 {
20     int i;
21     for (i = 0; i < n; i++)
22         if (a[i] == x)
23             return i;
24     return -1;
25 }
26
27 /* Binarna pretraga: Funkcija trazi u sortiranom nizu a[] duzine n
28 broj x. Pretraga koristi osobinu sortiranosti niza i u svakoj
29 iteraciji polovi interval pretrage. */
30 int binarna_pretraga(int a[], int n, int x)
31 {
32     int levi = 0;
33     int desni = n - 1;
34     int srednji;
35     /* Dokle god je indeks levi levo od indeksa desni */
36     while (levi <= desni) {
37         /* Srednji indeks je njihova aritmeticka sredina */
38         srednji = (levi + desni) / 2;
39         /* Ako je element sa sredisnjim indeksom veci od x, tada se x
40            mora nalaziti u levom delu niza */
41         if (x < a[srednji])
42             desni = srednji - 1;
43         /* Ako je element sa sredisnjim indeksom manji od x, tada se x
44            mora nalaziti u desnom delu niza */
45         else if (x > a[srednji])
46             levi = srednji + 1;

```

```

47     else
49         /* Ako je element sa sredisnjim indeksom jednak x, tada je broj
50            x pronadjen na poziciji srednji */
51         return srednji;
52     }
53     /* Ako element x nije pronadjen, vraca se -1 */
54     return -1;
55 }
56
57 /* Interpolaciona pretraga: Funkcija trazi u sortiranom nizu a[]
58    duzine n broj x. Pretraga koristi osobinu sortiranosti niza i
59    zasniva se na linearnoj interpolaciji vrednosti koja se trazi
60    vrednostima na krajevima prostora pretrage. */
61 int interpolaciona_pretraga(int a[], int n, int x)
62 {
63     int levi = 0;
64     int desni = n - 1;
65     int srednji;
66     /* Dokle god je indeks levi levo od indeksa desni... */
67     while (levi <= desni) {
68         /* Ako je trazeni element manji od pocetnog ili veci od
69            poslednjeg elementa u delu niza a[levi],...,a[desni], tada on
70            nije u tom delu niza. Ova provera je neophodna, da se ne bi
71            dogodilo da se prilikom izracunavanja indeksa srednji izadje
72            izvan opsega indeksa [levi,desni] */
73         if (x < a[levi] || x > a[desni])
74             return -1;
75         /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
76            a[levi] i a[desni] jednaki, tada je jasno da je trazeni broj x
77            jednak ovim vrednostima, pa se vraca indeks levi (ili indeks
78            desni). Ova provera je neophodna, jer bi se u suprotnom
79            prilikom izracunavanja indeksa srednji pojavilo deljenje
80            nulom. */
81         else if (a[levi] == a[desni])
82             return levi;
83         /* Racunanje srednjeg indeksa */
84         srednji =
85             levi +
86             ((int) ((double) (x - a[levi]) / (a[desni] - a[levi])) *
87                 (desni - levi));
88         /* Napomena: Indeks srednji je uvek izmedju levi i desni, ali ce
89            verovatno biti blize trazenoj vrednosti nego da je prosto uvek
90            uzimana aritmiticka sredina indeksa levi i desni. Ovo se moze
91            porediti sa pretragom recnika: ako neko trazi rec na slovo
92            'B', sigurno nece da otvorи recnik na polovini, vec verovatno
93            negde blize pocetku. */
94         /* Ako je element sa indeksom srednji veci od trazenog, tada se
95            trazeni element mora nalaziti u levoj polovini niza */
96         if (x < a[srednji])
97             desni = srednji - 1;
98         /* Ako je element sa indeksom srednji manji od trazenog, tada se
99            trazeni element mora nalaziti u desnoj polovini niza */

```

```

99     else if (x > a[srednji])
100        levi = srednji + 1;
101    else
102      /* Ako je element sa indeksom srednji jednak traženom, onda se
103         pretraga završava na poziciji srednji */
104      return srednji;
105  }
106  /* U slučaju neuspesne pretrage vraca se -1 */
107  return -1;
108 }

109 int main(int argc, char **argv)
110 {
111   int a[MAX];
112   int n, i, x;
113   struct timespec vreme1, vreme2, vreme3, vreme4, vreme5, vreme6;
114   FILE *f;
115   /* Provera argumenata komandne linije */
116   if (argc != 3) {
117     fprintf(stderr,
118             "koriscenje programa: %s dim_niza trazeni_br\n", argv[0])
119     ;
120     exit(EXIT_FAILURE);
121   }

122   /* Dimenzija niza */
123   n = atoi(argv[1]);
124   if (n > MAX || n <= 0) {
125     fprintf(stderr, "Dimenzija niza neodgovarajuća\n");
126     exit(EXIT_FAILURE);
127   }

128   /* Broj koji se trazi */
129   x = atoi(argv[2]);
130   /* Elementi niza se generisu slučajno, tako da je svaki sledeći
131      veci od prethodnog. Funkcija srand() inicijalizuje pocetnu
132      vrednost sa kojom se kreće u izracunavanje sekvence
133      pseudo-slučajnih brojeva. Kako generisani niz ne bi uvek bio
134      isti, ova vrednost se postavlja na tekuce vreme u sekundama od
135      Nove godine 1970, tako da je za svako sledeće pokretanje
136      programa (u vremenskim intervalima vecim od jedne sekunde) ove
137      vrednost drugacija. random()%100 vraca brojeve izmedju 0 i 99 */
138   srand(time(NULL));
139   for (i = 0; i < n; i++)
140     a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
141   /* Linearna pretraga */
142   printf("Linearna pretraga:\n");
143   /* Vreme proteklo od Nove godine 1970 */
144   clock_gettime(CLOCK_REALTIME, &vreme1);
145   i = linearna_pretraga(a, n, x);
146   /* Novo vreme i razlika sa prvim predstavlja vreme utroseno za
147      linearnu pretragu */
148 
```

```

151     clock_gettime(CLOCK_REALTIME, &vreme2);
152     if (i == -1)
153         printf("Element nije u nizu\n");
154     else
155         printf("Element je u nizu na poziciji %d\n", i);
156     /* Binarna pretraga */
157     printf("Binarna pretraga:\n");
158     clock_gettime(CLOCK_REALTIME, &vreme3);
159     i = binarna_pretraga(a, n, x);
160     clock_gettime(CLOCK_REALTIME, &vreme4);
161     if (i == -1)
162         printf("Element nije u nizu\n");
163     else
164         printf("Element je u nizu na poziciji %d\n", i);
165     /* Interpolaciona pretraga */
166     printf("Interpolaciona pretraga:\n");
167     clock_gettime(CLOCK_REALTIME, &vreme5);
168     i = interpolaciona_pretraga(a, n, x);
169     clock_gettime(CLOCK_REALTIME, &vreme6);
170     if (i == -1)
171         printf("Element nije u nizu\n");
172     else
173         printf("Element je u nizu na poziciji %d\n", i);
174     /* Podaci o izvrsavanju programa bivaju upisani u log fajl */
175     if ((f = fopen("vremena.txt", "a")) == NULL) {
176         fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
177         exit(EXIT_FAILURE);
178     }
179
180     fprintf(f, "Dimenzija niza: %d\n", n);
181     fprintf(f, "\tLinearna: %10ld ns\n",
182             (vreme2.tv_sec - vreme1.tv_sec) * 1000000000 +
183             vreme2.tv_nsec - vreme1.tv_nsec);
184     fprintf(f, "\tBinarna: %19ld ns\n",
185             (vreme4.tv_sec - vreme3.tv_sec) * 1000000000 +
186             vreme4.tv_nsec - vreme3.tv_nsec);
187     fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
188             (vreme6.tv_sec - vreme5.tv_sec) * 1000000000 +
189             vreme6.tv_nsec - vreme5.tv_nsec);
190     /* Zatvaranje datoteke */
191     fclose(f);
192     exit(EXIT_SUCCESS);
193 }
```

Rešenje 3.2

```

2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #define MAX 1024
```

```

6 int linearna_pretraga_r1(int a[], int n, int x)
7 {
8     int tmp;
9     /* Izlaz iz rekurzije */
10    if (n <= 0)
11        return -1;
12    /* Ako je prvi element traženi */
13    if (a[0] == x)
14        return 0;
15    /* Pretraga ostatka niza */
16    tmp = linearna_pretraga_r1(a + 1, n - 1, x);
17    return tmp < 0 ? tmp : tmp + 1;
18 }

20 int linearna_pretraga_r2(int a[], int n, int x)
21 {
22     /* Izlaz iz rekurzije */
23     if (n <= 0)
24         return -1;
25     /* Ako je poslednji element traženi */
26     if (a[n - 1] == x)
27         return n - 1;
28     /* Pretraga ostatka niza */
29     return linearna_pretraga_r2(a, n - 1, x);
30 }

32 int binarna_pretraga_r(int a[], int l, int d, int x)
33 {
34     int srednji;
35     if (l > d)
36         return -1;
37     /* Sredisnja pozicija na kojoj se trazi vrednost x */
38     srednji = (l + d) / 2;
39     /* Ako je element na sredisnoj poziciji traženi */
40     if (a[srednji] == x)
41         return srednji;
42     /* Ako je traženi broj veci od broja na sredisnoj poziciji,
43      pretrazuje se desna polovina niza */
44     if (a[srednji] < x)
45         return binarna_pretraga_r(a, srednji + 1, d, x);
46     /* Ako je traženi broj manji od broja na sredisnoj poziciji,
47      pretrazuje se leva polovina niza */
48     else
49         return binarna_pretraga_r(a, l, srednji - 1, x);
50 }

52 int interpolaciona_pretraga_r(int a[], int l, int d, int x)
53 {
54     int p;
55     if (x < a[l] || x > a[d])
56         return -1;

```

```

58     if (a[d] == a[1])
      return 1;
59  /* Pozicija na kojoj se trazi vrednost x */
60  p = 1 + (d - 1) * (x - a[1]) / (a[d] - a[1]);
61  if (a[p] == x)
      return p;
62  if (a[p] < x)
      return interpolaciona_pretraga_r(a, p + 1, d, x);
63  else
      return interpolaciona_pretraga_r(a, 1, p - 1, x);
64 }

65 int main()
66 {
67     int a[MAX];
68     int x;
69     int i, indeks;

70  /* Ucitavanje traženog broja */
71  printf("Unesite traženi broj: ");
72  scanf("%d", &x);

73  /* Ucitavanje elemenata niza sve do kraja ulaza - očekuje se da
     korisnik pritisne CTRL+D za naznaku kraja */
74  i = 0;
75  printf("Unesite sortiran niz elemenata: ");
76  while (i < MAX && scanf("%d", &a[i]) == 1) {
77      if (i > 0 && a[i] < a[i - 1]) {
78          fprintf(stderr,
79                  "Elementi moraju biti uneseni u neopadajućem poretku\n");
80          exit(EXIT_FAILURE);
81      }
82      i++;
83  }

84  /* Linearna pretraga */
85  printf("Linearna pretraga\n");
86  indeks = linearna_pretraga_r1(a, i, x);
87  if (indeks == -1)
88      printf("Element se ne nalazi u nizu.\n");
89  else
90      printf("Pozicija elementa je %d.\n", indeks);

91  /* Binarna pretraga */
92  printf("Binarna pretraga\n");
93  indeks = binarna_pretraga_r(a, 0, i - 1, x);
94  if (indeks == -1)
95      printf("Element se ne nalazi u nizu.\n");
96  else
97      printf("Pozicija elementa je %d.\n", indeks);
98
99
100
101
102
103
104
105
106
107
108

```

```

110  /* Interpolaciona pretraga */
111  printf("Interpolaciona pretraga\n");
112  indeks = interpolaciona_pretraga_r(a, 0, i - 1, x);
113  if (indeks == -1)
114      printf("Element se ne nalazi u nizu.\n");
115  else
116      printf("Pozicija elementa je %d.\n", indeks);
117
118 }
```

Rešenje 3.3

```

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

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

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

26 /* Funkcija trazi u sortiranom nizu studenata a[] duzine n studenta
28    sa indeksom x i vraca indeks pozicije nadjenog clana niza ili -1,
30    ako element nije pronadjen. */
32 int binarna_pretraga(Student a[], int n, long x)
{
34     int levi = 0;
35     int desni = n - 1;
36     int srednji;
37     /* Dokle god je indeks levi levo od indeksa desni */
38     while (levi <= desni) {
39         /* Racuna se srednja pozicija */
40         srednji = (levi + desni) / 2;
41         /* Ako je indeks studenta na toj poziciji veci od trazenog, tada
42            se trazeni indeks mora nalaziti u levoj polovini niza */
43         if (x < a[srednji].indeks)
44             desni = srednji - 1;
45     }
46 }
```

```

40     /* Ako je pak manji od traženog, tada se on mora nalaziti u
41      desnoj polovini niza */
42     else if (x > a[srednji].indeks)
43         levi = srednji + 1;
44     else
45         /* Ako je jednak traženom indeksu x, tada je pronađen student
46          sa traženom indeksom na poziciji srednji */
47         return srednji;
48     /* Ako nije pronađen, vraca se -1 */
49     return -1;
50 }

52 /* Linearnom pretragom niza studenata trazi se prezime x */
53 int linearna_pretraga(Student a[], int n, char x[])
54 {
55     int i;
56     for (i = 0; i < n; i++)
57         /* Poredjenje prezimena i-tog studenta i poslatog x */
58         if (strcmp(a[i].prezime, x) == 0)
59             return i;
60     return -1;
61 }

62 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
63   prosledjuju kao argumenti komandne linije */
64 int main(int argc, char *argv[])
65 {
66     Student dosije[MAX_STUDENATA];
67     FILE *fin = NULL;
68     int i;
69     int br_studenata = 0;
70     long tražen_indeks = 0;
71     char traženo_presime[MAX_DUZINA];
72     int bin_pretraga;

73     /* Provera da li je korisnik prilikom poziva programa prosledio ime
74       datoteke sa informacijama o studentima i opciju pretrage */
75     if (argc != 3) {
76         fprintf(stderr,
77                 "Greska: Program se poziva sa %s ime_datoteke opcija\n",
78                 argv[0]);
79         exit(EXIT_FAILURE);
80     }

81     /* Provera prosledjene opcije */
82     if (strcmp(argv[2], "-indeks") == 0)
83         bin_pretraga = 1;
84     else if (strcmp(argv[2], "-presime") == 0)
85         bin_pretraga = 0;
86     else {
87         fprintf(stderr, "Opcija mora biti -indeks ili -presime\n");
88     }

```

```

    exit(EXIT_FAILURE);
}

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

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

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

/* Pretraga po indeksu */
if (bin_pretraga) {
    /* Unos indeksa koji se binarno trazi u nizu */
    printf("Unesite indeks studenta cije informacije zelite: ");
    scanf("%ld", &trazen_indeks);
    i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
    /* Rezultat binarne pretrage */
    if (i == -1)
        printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
    else
        printf("Indeks: %ld, Ime i prezime: %s %s\n",
               dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
}
/* Pretraga po prezimenu */
else {
    /* Unos prezimena koje se linearно trazi u nizu */
    printf("Unesite prezime studenta cije informacije zelite: ");
    scanf("%s", trazeno_prezime);
    i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
    /* Rezultat linearne pretrage */
    if (i == -1)
        printf("Ne postoji student sa prezimenom %s\n",
               trazeno_prezime);
    else
}

```

```

142     printf("Indeks: %ld, Ime i prezime: %s %s\n",
143            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
144 }
145 exit(EXIT_SUCCESS);
146 }
```

Rešenje 3.4

```

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

4
#define MAX_STUDENATA 128
6 #define MAX_DUZINA 16

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

14 int binarna_pretraga_rekurzivna(Student a[], int levi, int desni,
15                                long x)
16 {
17   /* Ako je pozicija elementa na levom kraju veca od pozicije
18      elementa na desnom kraju dela niza koji se pretrazuje, onda se
19      zapravo pretrazuje prazan deo niza. U praznom delu niza nema
20      trazenog elementa pa se vraca -1 */
21   if (levi > desni)
22     return -1;
23   /* Racunanje pozicije srednjeg elementa */
24   int srednji = (levi + desni) / 2;
25   /* Da li je srednji bas onaj trazeni */
26   if (a[srednji].indeks == x) {
27     return srednji;
28   }
29   /* Ako je trazeni indeks manji od indeksa studenta na srednjoj
30      poziciji, onda se pretraga nastavlja u levoj polovini niza, jer
31      je poznato da je niz sortiran po indeksu u rastucem poretku. */
32   if (x < a[srednji].indeks)
33     return binarna_pretraga_rekurzivna(a, levi, srednji - 1, x);
34   /* Inace ga treba traziti u desnoj polovini */
35   else
36     return binarna_pretraga_rekurzivna(a, srednji + 1, desni, x);
37 }
38
39 int linearna_pretraga_rekurzivna_v2(Student a[], int n, char x[])
40 {
41   /* Ako je niz prazan, vraca se -1 */
42   if (n == 0)
43     return -1;
44 }
```

```

44  /* Kako se trazi prvi student sa traženim prezimenom, pocinje se sa
45   prvim studentom u nizu. */
46  if (strcmp(a[0].prezime, x) == 0)
47    return 0;
48  int i = linearna_pretraga_rekursivna_v2(a + 1, n - 1, x);
49  return i >= 0 ? 1 + i : -1;
50 }

52 int linearna_pretraga_rekursivna(Student a[], int n, char x[])
53 {
54  /* Ako je niz prazan, vraca se -1 */
55  if (n == 0)
56    return -1;
57  /* Ako se trazi poslednji student sa traženim prezimenom, pocinje
58   se sa poslednjim studentom u nizu. */
59  if (strcmp(a[n - 1].prezime, x) == 0)
60    return n - 1;
61  return linearna_pretraga_rekursivna(a, n - 1, x);
62 }

64 /* Main funkcija mora imati argumente jer se ime datoteke i opcija
65  prosledjuju kao argumenti komandne linije */
66 int main(int argc, char *argv[])
67 {
68  Student dosije[MAX_STUDENATA];
69  FILE *fin = NULL;
70  int i;
71  int br_studenata = 0;
72  long tražen_indeks = 0;
73  char traženo_prezime[MAX_DUZINA];
74  int bin_pretraga;

76  /* Provera da li je korisnik prilikom poziva programa prosledio ime
77   datoteke sa informacijama o studentima i opciju pretrage */
78  if (argc != 3) {
79    fprintf(stderr,
80            "Greska: Program se poziva sa %s ime_datoteke opcija\n",
81            argv[0]);
82    exit(EXIT_FAILURE);
83  }

84  /* Provera prosledjene opcije */
85  if (strcmp(argv[2], "-indeks") == 0)
86    bin_pretraga = 1;
87  else if (strcmp(argv[2], "-prezime") == 0)
88    bin_pretraga = 0;
89  else {
90    fprintf(stderr, "Opcija mora biti -indeks ili -prezime\n");
91    exit(EXIT_FAILURE);
92  }

94  /* Otvaranje datoteke */

```

```

96     fin = fopen(argv[1], "r");
98     if (fin == NULL) {
100         fprintf(stderr,
101             "Neuspesno otvaranje datoteke %s za citanje\n", argv[1]);
102         exit(EXIT_FAILURE);
103     }
104
105     /* Citanje se vrši sve dok postoji red sa informacijama o studentu */
106     i = 0;
107     while (1) {
108         if (i == MAX_STUDENATA)
109             break;
110         if (fscanf(
111             (fin, "%ld %s %s", &dosije[i].indeks, dosije[i].ime,
112             dosije[i].prezime) != 3)
113             break;
114         i++;
115     }
116     br_studenata = i;
117
118     /* Nakon citanja, datoteka vise nije neophodna i zatvara se. */
119     fclose(fin);
120
121     /* Pretraga po indeksu */
122     if (bin_pretraga) {
123         /* Unos indeksa koji se binarno trazi u nizu */
124         printf("Unesite indeks studenta cije informacije zelite: ");
125         scanf("%ld", &trazen_indeks);
126         i = binarna_pretraga_rekurzivna(dosije, 0, br_studenata,
127                                         trazen_indeks);
128
129         /* Rezultat binarne pretrage */
130         if (i == -1)
131             printf("Ne postoji student sa indeksom %ld\n", trazen_indeks);
132         else
133             printf("Indeks: %ld, Ime i prezime: %s %s\n",
134                   dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
135     }
136
137     /* Pretraga po prezimenu */
138     else {
139         /* Unos prezimena koje se linearно trazi u nizu */
140         printf("Unesite prezime studenta cije informacije zelite: ");
141         scanf("%s", trazeno_prezime);
142         i = linearna_pretraga_rekurzivna_v2(dosije, br_studenata,
143                                              trazeno_prezime);
144
145         /* Rezultat linearne pretrage */
146         if (i == -1)
147             printf("Ne postoji student sa prezimenom %s\n",
148                   trazeno_prezime);
149         else
150             printf("Indeks: %ld, Ime i prezime: %s %s\n",
151                   dosije[i].indeks, dosije[i].ime, dosije[i].prezime);
152     }

```

```

148     }
149     exit(EXIT_SUCCESS);
150 }
```

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
39     Tacka najbliza;
40     int i;
41     najbliza = t[0];
42     for (i = 1; i < n; i++) {
43         if (fabs(t[i].x) < fabs(najbliza.x)) {
44             najbliza = t[i];
45         }
46     }
47 }
```

```

    }
    return najbliza;
}

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

#define MAX 1024

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

    /* Ocekuje se da korisnik prosledi barem ime izvrsnog programa i
       ime datoteke sa tackama */
    if (argc < 2) {
        fprintf(stderr,
                "koriscenje programa: %s ime_datoteke\n",
                argv[0]);
        exit(EXIT_FAILURE);
    }

    /* Otvaranje datoteke za citanje */
    ulaz = fopen(argv[1], "r");
    if (ulaz == NULL) {
        fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
                argv[1]);
        exit(EXIT_FAILURE);
    }

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

```

```

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

118  /* Stampanje koordinata trazene tacke */
119  printf("%g %g\n", najbliza.x, najbliza.y);
120
121  /* Zatvaranje datoteke */
122  fclose(ulaz);

123  exit(EXIT_SUCCESS);
124 }
```

Rešenje 3.6

```

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

/* Tacnost */
# define EPS 0.001

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

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

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

```

20      /* Ako je absolutna vrednost kosinusa u ovoj tacki manja od
21         zadate 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      /* Stampanje vrednosti trazene tacke */
34      printf("%g\n", s);
35
36      return 0;
}

```

Rešenje 3.7

```

#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>

6 int main(int argc, char **argv)
{
8     double l, d, s, epsilon;

10    char ime_funkcije[6];

12    /* Pokazivac na funkciju koja ima jedan argument tipa double i
13       povratnu vrednost istog tipa */
14    double (*fp) (double);

16    /* Ako korisnik nije uneo argument, prijavljuje se greska */
17    if (argc != 2) {
18        fprintf(stderr, "Greska: ");
19        fprintf(stderr, "Nedovoljan broj argumenata komandne linije.\n");
20        fprintf(stderr,
21                "Program se poziva sa %s ime_funkcije iz math.h.\n",
22                argv[0]);
23        exit(EXIT_FAILURE);
24    }

26    /* Niska ime_funkcije sadrzi ime trazene funkcije koja je navedena
27       u komandnoj liniji */
28    strcpy(ime_funkcije, argv[1]);

```

```

30  /* Inicijalizuje se pokazivac na funkciju koja se tabelira */
31  if (strcmp(ime_funkcije, "sin") == 0)
32    fp = &sin;
33  else if (strcmp(ime_funkcije, "cos") == 0)
34    fp = &cos;
35  else if (strcmp(ime_funkcije, "tan") == 0)
36    fp = &tan;
37  else if (strcmp(ime_funkcije, "atan") == 0)
38    fp = &atan;
39  else if (strcmp(ime_funkcije, "asin") == 0)
40    fp = &asin;
41  else if (strcmp(ime_funkcije, "log") == 0)
42    fp = &log;
43  else if (strcmp(ime_funkcije, "log10") == 0)
44    fp = &log10;
45  else {
46    fprintf(stderr, "Program ne podrzava trazenu funkciju!\n");
47    exit(EXIT_SUCCESS);
48  }

49  printf("Unesite krajeve intervala: ");
50  scanf("%lf %lf", &l, &d);

51  if ((*fp) (l) * (*fp) (d) >= 0) {
52    fprintf(stderr,
53      "Funkcija %s na intervalu [%g, %g] ne zadovoljava uslove\
54      ", ime_funkcije, l, d);
55    exit(EXIT_FAILURE);
56  }

57  printf("Unesite preciznost: ");
58  scanf("%lf", &epsilon);

59  /* Sve dok se ne pronadje trazena vrednost argumenta */
60  while (1) {
61    /* Polovi se interval */
62    s = (l + d) / 2;
63    /* Ako je absolutna vrednost trazene funkcije u ovoj tacki manja
64     od zadate tacnosti, prekida se pretraga */
65    if (fabs((*fp) (s)) < epsilon) {
66      break;
67    }
68    /* Ako je nula u levom delu intervala, nastavlja se pretraga na
69     [l, s] */
70    if ((*fp) (l) * (*fp) (s) < 0)
71      d = s;
72    else
73      /* Inace, na intervalu [s, d] */
74      l = s;
75  }
76
77
78
79
80

```

```

82  /* Stampanje vrednosti trazene tacke */
83  printf("%g\n", s);
84
85  return 0;
86 }
```

Rešenje 3.8

```

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 zavrsena */
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 }
29
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 3.9

```

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
           prethodnik veci ili jednak nuli, pretraga se zavrsava */
16        if (niz[s] < 0 && ((s > 0 && niz[s - 1] >= 0) || s == 0))
17            return s;
18        /* Ako je broj veci ili jednak nuli, pretrazuje se desna polovina
           niza */
19        if (niz[s] >= 0)
20            l = s + 1;
21        /* A inace leva */
22        else
23            d = s - 1;
24    }
25    return -1;
26}
27
28 int main()
29 {
30     int niz[MAX];
31     int n = 0;
32
33     /* Unos niza */
34     while (scanf("%d", &niz[n]) == 1)
35         n++;
36
37     /* Stanpanje rezultata */
38     printf("%d\n", prvi_manji_od_nule(niz, n));
39
40     return 0;
41}
42
43

```

Rešenje 3.10

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 unsigned int logaritam_a(unsigned int x)
5 {
6     /* Izlaz iz rekurzije */
7     if (x == 1)
8         return 0;
9     /* Rekurzivni korak */
10    return 1 + logaritam_a(x >> 1);
11 }
12
13 unsigned int logaritam_b(unsigned int x)
14 {
15     /* Binarnom pretragom se trazi jedinica u binarnom zapisu broja x
16      najvece vaznosti, tj. najlevija. Pretragu se vrsti od pozicije 0
17      do 31 */
18     int d = 0, l = sizeof(unsigned int) * 8 - 1;
19     int s;
20     /* Sve dok je desna granica pretrage desnije od leve */
21     while (d <= l) {
22         /* Racuna se sredisnja pozicija */
23         s = (l + d) / 2;
24         /* Proverava se da li je na toj poziciji trazena jedinic */
25         if ((1 << s) <= x && (1 << (s + 1)) > x)
26             return s;
27         /* Pretraga desne polovine binarnog zapisa */
28         if ((1 << s) > x)
29             l = s - 1;
30         /* Pretraga leve polovine binarnog zapisa */
31         else
32             d = s + 1;
33     }
34     return s;
35 }
36
37 int main()
38 {
39     unsigned int x;
40
41     /* Unos podatka */
42     scanf("%u", &x);
43
44     /* Provera da li je uneti broj pozitivan */
45     if (x == 0) {
46         fprintf(stderr, "Logaritam od nule nije definisan\n");
47         exit(EXIT_FAILURE);
48     }
49
50     /* Ispis povratnih vrednosti funkcija */
51     printf("%u %u\n", logaritam_a(x), logaritam_b(x));

```

```

53     exit(EXIT_SUCCESS);
}

```

Rešenje 3.12

sort.h

```

#ifndef _SORT_H_
#define _SORT_H_ 1

/* Selection sort: Funkcija sortira niz celih brojeva metodom
sortiranja izborom. Ideja algoritma je sledeća: U svakoj
iteraciji pronalazi se najmanji element i prenesti se na pocetak
niza. Dakle, u prvoj iteraciji, pronalazi se najmanji element, i
dovodi na nulto mesto u nizu. U i-toj iteraciji najmanjih i-1
elemenata su već na svojim pozicijama, pa se od elemenata sa
indeksima od i do n-1 trazi najmanji, koji se dovodi na i-tu
poziciju. */
void selection_sort(int a[], int n);

/* Insertion sort: Funkcija sortira niz celih brojeva metodom
sortiranja umetanjem. Ideja algoritma je sledeća: neka je na
pocetku i-te iteracije niz prvih i elemenata
(a[0],a[1],...,a[i-1]) sortirano. U i-toj iteraciji treba element
a[i] umetnuti na pravu poziciju među prvih i elemenata tako da se
dobije niz duzine i+1 koji je sortiran. Ovo se radi tako što se
i-ti element najpre uporedi sa njegovim prvim levim susedom
(a[i-1]). Ako je a[i] veće, tada je on već na pravom mestu, i niz
a[0],a[1],...,a[i] je sortiran, pa se može preci na sledecu
iteraciju. Ako je a[i-1] veće, tada se zamjenjuju a[i] i a[i-1], a
zatim se proverava da li je potrebno dalje potiskivanje elemenata u
levo, poredeći ga sa njegovim novim levim susedom. Ovim uzastopnim
prenestanjem se a[i] umeće na pravo mesto u nizu. */
void insertion_sort(int a[], int n);

/* Bubble sort: Funkcija sortira niz celih brojeva metodom mehurica.
Ideja algoritma je sledeća: prolazi se kroz niz redom poredeći
susedne elemente, i pri tom ih zamjenjujući ako su u pogresnom
poretku. Ovim se najveći element poput mehurica istiskuje na
"površinu", tj. na krajnju desnu poziciju. Nakon toga je potrebno
ovaj postupak ponoviti nad nizom a[0],...,a[n-2], tj. nad prvih
n-1 elemenata niza bez poslednjeg koji je postavljen na pravu
poziciju. Nakon toga se isti postupak ponavlja nad sve kracim i
kracim prefiksima niza, cime se jedan po jedan istiskuju
elementi na svoje prave pozicije. */
void bubble_sort(int a[], int n);

/* Selsort: Ovaj algoritam je jednostavno proširenje sortiranja
umetanjem koje dopušta direktnu razmenu udaljenih elemenata.

```

```

44 Prosirenje se sastoji u tome da se kroz algoritam umetanja prolazi
45 vise puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
46 koji je manji od n (sto omogucuje razmenu udaljenih elemenata) i
47 tako se dobija h-sortiran niz, tj. niz u kome su elementi na
48 rastojanju h sortirani, mada susedni elementi to ne moraju biti. U
49 drugom prolazu kroz isti algoritam sprovodi se isti postupak ali
50 za manji korak h. Sa prolazima se nastavlja sve do koraka h = 1, u
51 kome se dobija potpuno sortirani niz. Izbor pocetne vrednosti za
52 h, i nacina njegovog smanjivanja menja u nekim slucajevima brzinu
53 algoritma, ali bilo koja vrednost ce rezultovati ispravnim
54 sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
55 vrednost 1. */
56 void shell_sort(int a[], int n);
57
58 /* Merge sort: Funkcija sortira niz celih brojeva a[] ucesljavanjem.
59 Sortiranje se vrsti od elementa na poziciji l do onog na poziciji
60 d. Na pocetku, da bi niz bio kompletno sortiran, l mora biti 0, a
61 d je jednako poslednjem validnom indeksu u nizu. Funkcija niz
62 podeli na dve polovine, levu i desnu, koje zatim rekursivno
63 sortira. Od ova dva sortirana podniza, sortiran niz se dobija
64 ucesljavanjem, tj. istovremenim prolaskom kroz oba niza i izborom
65 trenutnog manjeg elementa koji se smesta u pomocni niz. Na kraju
66 algoritma, sortirani elementi su u pomocnom nizu, koji se kopira u
67 originalni niz. */
68 void merge_sort(int a[], int l, int d);
69
70 /* Quick sort: Funkcija sortira deo niza brojeva a izmedju pozicija l
71 i d. Njena ideja sortiranja je izbor jednog elementa niza, koji se
72 naziva pivot, i koji se dovodi na svoje mesto. Posle ovog koraka,
73 svi elementi levo od njega bice manji, a svi desno bice veci od
74 njega. Kako je pivot doveden na svoje mesto, da bi niz bio
75 kompletno sortiran, potrebno je sortirati elemente levo (manje) od
76 njega, i elemente desno (vece). Kako su dimenzije ova dva podniza
77 manje od dimenzije pocetnog niza koji je trebalo sortirati, ovaj
78 deo moze se uraditi rekursivno. */
79 void quick_sort(int a[], int l, int d);
80
81 #endif

```

sort.c

```

1 #include "sort.h"
2
3 #define MAX 1000000
4
5 void selection_sort(int a[], int n)
6 {
7     int i, j;
8     int min;
9     int pom;
10

```

```

12  /* U svakoj iteraciji ove petlje pronađazi se najmanji element
13   medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
14   poziciju i, dok se element na poziciji i premeta na poziciju
15   min, na kojoj se nalazio najmanji od gore navedenih elemenata.
16   */
17   for (i = 0; i < n - 1; i++) {
18     /* Unutrašnja petlja pronađazi poziciju min, na kojoj se nalazi
19      najmanji od elemenata a[i],...,a[n-1]. */
20     min = i;
21     for (j = i + 1; j < n; j++)
22       if (a[j] < a[min])
23         min = j;
24
25     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
26      su (i) i min razliciti, inace je nepotrebno. */
27     if (min != i) {
28       pom = a[i];
29       a[i] = a[min];
30       a[min] = pom;
31     }
32   }
33
34 void insertion_sort(int a[], int n)
35 {
36   int i, j;
37
38   /* Na pocetku iteracije pretpostavlja se da je niz a[0],...,a[i-1]
39    sortiran */
40   for (i = 1; i < n; i++) {
41
42     /* U ovoj petlji se redom potiskuje element a[i] uлево koliko je
43      potrebno, dok ne zauzme pravo mesto, tako da niz a[0],...a[i]
44      bude sortiran. Indeks j je trenutna pozicija na kojoj se
45      element koji se umesti nalazi. Petlja se završava ili kada
46      element dodje do levog kraja (j==0) ili kada se nađe na
47      element a[j-1] koji je manji od a[j]. */
48     int temp = a[i];
49     for (j = i; j > 0 && temp < a[j - 1]; j--)
50       a[j] = a[j - 1];
51     a[j] = temp;
52   }
53
54 void bubble_sort(int a[], int n)
55 {
56   int i, j;
57   int ind;
58
59   for (i = n, ind = 1; i > 1 && ind; i--)
60
61     /* Poput "mehurica" potiskuje se najveci element medju elementima

```

```

62     od a[0] do a[i-1] na poziciju i-1 uporedjujuci susedne
64     elemente niza i potiskujuci veci u desno */
65     for (j = 0, ind = 0; j < i - 1; j++) {
66         if (a[j] > a[j + 1]) {
67             int temp = a[j];
68             a[j] = a[j + 1];
69             a[j + 1] = temp;
70
71             /* Promenljiva ind registruje da je bilo prenestanja. Samo u
72                tom slučaju ima smisla ici na sledeću iteraciju, jer ako
73                nije bilo prenestanja, znaci da su svi elementi vec u
74                dobrom poretku, pa nema potrebe prelaziti na kraci prefiks
75                niza. Algoritam može biti i bez ovoga, sortiranje bi bilo
76                ispravno, ali manje efikasano, jer bi se cesto nepotrebno
77                vrsila mnoga uporedjivanja, kada je vec jasno da je
78                sortiranje završeno. */
79             ind = 1;
80         }
81     }
82 void shell_sort(int a[], int n)
83 {
84     int h = n / 2, i, j;
85     while (h > 0) {
86         /* Insertion sort sa korakom h */
87         for (i = h; i < n; i++) {
88             int temp = a[i];
89             j = i;
90             while (j >= h && a[j - h] > temp) {
91                 a[j] = a[j - h];
92                 j -= h;
93             }
94             a[j] = temp;
95         }
96         h = h / 2;
97     }
98 }
100 void merge_sort(int a[], int l, int d)
101 {
102     int s;
103     static int b[MAX];           /* Pomocni niz */
104     int i, j, k;
105
106     /* Izlaz iz rekurzije */
107     if (l >= d)
108         return;
109
110     /* Odredjivanje srednjeg indeksa */
111     s = (l + d) / 2;
112
113     /* Rekursivni pozivi */

```

```

114     merge_sort(a, l, s);
116     merge_sort(a, s + 1, d);
118
119     /* Inicijalizacija indeksa. Indeks i prolazi kroz levu polovinu
120     niza, dok indeks j prolazi kroz desnu polovinu niza. Indeks k
121     prolazi kroz pomocni niz b[] */
122     i = l;
123     j = s + 1;
124     k = 0;
125
126     /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
127     while (i <= s && j <= d) {
128         if (a[i] < a[j])
129             b[k++] = a[i++];
130         else
131             b[k++] = a[j++];
132     }
133
134     /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive j
135     iz dopustenog opsega u pomocni niz se prepisuje ostatak leve
136     polovine niza */
137     while (i <= s)
138         b[k++] = a[i++];
139
140     /* U slucaju da se prethodna petlja zavrsila izlaskom promenljive i
141     iz dopustenog opsega u pomocni niz se prepisuje ostatak desne
142     polovine niza */
143     while (j <= d)
144         b[k++] = a[j++];
145
146     /* Prepisuje se "ucesljjani" niz u originalni niz */
147     for (k = 0, i = l; i <= d; i++, k++)
148         a[i] = b[k];
149     }
150
151     /* Pomocna funkcija koja menja mesto i-tom i j-tom elementu niza a */
152     void swap(int a[], int i, int j)
153     {
154         int tmp = a[i];
155         a[i] = a[j];
156         a[j] = tmp;
157     }
158
159     void quick_sort(int a[], int l, int d)
160     {
161         int i, pivot_pozicija;
162
163         /* Izlaz iz rekurzije -- prazan niz */
164         if (l >= d)
165             return;
166
167     }

```

```

166  /* Particionisanje niza. Svi elementi na pozicijama levo od
167   pivot_pozicija (izuzev same pozicije 1) su strogo manji od
168   pivota. Kada se pronađe neki element manji od pivota, uvecava
169   se promenljiva pivot_pozicija i na tu poziciju se prenesta
170   nadjeni element. Na kraju će pivot_pozicija zaista biti pozicija
171   na koju treba smestiti pivot, jer će svi elementi levo od te
172   pozicije biti manji a desno biti veci ili jednaki od pivota. */
173   pivot_pozicija = 1;
174   for (i = l + 1; i <= d; i++)
175     if (a[i] < a[l])
176       swap(a, ++pivot_pozicija, i);

177  /* Postavljanje pivota na svoje mesto */
178  swap(a, l, pivot_pozicija);

179  /* Rekurzivno sortiranje elemenata manjih od pivota */
180  quick_sort(a, l, pivot_pozicija - 1);
181  /* Rekurzivno sortiranje elemenata vecih od pivota */
182  quick_sort(a, pivot_pozicija + 1, d);
183 }
}

```

main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "sort.h"

5  /* Maksimalna duzina niza */
6 #define MAX 1000000

7 int main(int argc, char *argv[])
{
8  ****
9   tip_sortiranja == 0 => selectionsort, (podrazumevano)
10  tip_sortiranja == 1 => insertionsort, -i opcija komandne linije
11  tip_sortiranja == 2 => bubblesort,      -b opcija komandne linije
12  tip_sortiranja == 3 => shellsort,      -s opcija komandne linije
13  tip_sortiranja == 4 => mergesort,       -m opcija komandne linije
14  tip_sortiranja == 5 => quicksort,       -q opcija komandne linije
15  ****
16  int tip_sortiranja = 0;
17  ****
18  tip_niza == 0 => slučajno generisani nizovi, (podrazumevano)
19  tip_niza == 1 => rastuce sortirani nizovi,      -r opcija
20  tip_niza == 2 => opadajuće soritrani nizovi, -o opcija
21  ****
22  int tip_niza = 0;

23  /* Dimenzija niza koji se sortira */
24  int dimenzija;

```

```

29     int i;
30     int niz[MAX];
31
32     /* Provera argumenata komandne linije */
33     if (argc < 2) {
34         fprintf(stderr,
35             "Program zahteva bar 2 argumenta komandne linije!\n");
36         exit(EXIT_FAILURE);
37     }
38
39     /* Ocitavanje opcija i argumenata prilikom poziva programa */
40     for (i = 1; i < argc; i++) {
41         /* Ako je u pitanju opcija... */
42         if (argv[i][0] == '-') {
43             switch (argv[i][1]) {
44                 case 'i':
45                     tip_sortiranja = 1;
46                     break;
47                 case 'b':
48                     tip_sortiranja = 2;
49                     break;
50                 case 's':
51                     tip_sortiranja = 3;
52                     break;
53                 case 'm':
54                     tip_sortiranja = 4;
55                     break;
56                 case 'q':
57                     tip_sortiranja = 5;
58                     break;
59                 case 'r':
60                     tip_niza = 1;
61                     break;
62                 case 'o':
63                     tip_niza = 2;
64                     break;
65                 default:
66                     printf("Pogresna opcija -%c\n", argv[i][1]);
67                     return 1;
68                     break;
69             }
70         }
71         /* Ako je u pitanju argument, onda je to duzina niza koji treba
72          da se sortira */
73         else {
74             dimenzija = atoi(argv[i]);
75             if (dimenzija <= 0 || dimenzija > MAX) {
76                 fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
77                 exit(EXIT_FAILURE);
78             }
79         }
80     }

```

```

81  /* Elementi niza se odredjuju slucajno, ali vodeci racuna o tipu
83   niza dobijenom iz komandne linije. srand() funkcija obezbedjuje
85   novi seed za pozivanje rand funkcije, i kako generisani niz ne
86   bi uvek bio isti seed je postavljen na tekuce vreme u sekundama
87   od Nove godine 1970. rand()%100 daje brojeve izmedju 0 i 99 */
88   srand(time(NULL));
89   if (tip_niza == 0)
90     for (i = 0; i < dimenzija; i++)
91       niz[i] = rand();
92   else if (tip_niza == 1)
93     for (i = 0; i < dimenzija; i++)
94       niz[i] = i == 0 ? rand() % 100 : niz[i - 1] + rand() % 100;
95   else
96     for (i = 0; i < dimenzija; i++)
97       niz[i] = i == 0 ? rand() % 100 : niz[i - 1] - rand() % 100;
98
99   /* Ispisivanje elemenata niza */
100  ****
101  Ovaj deo je iskomentarisan jer sledeci ispis ne treba da se nadje
102  na standardnom izlazu. Njegova svrha je samo bila provera da li
103  je niz generisan u skladu sa opcijama komandne linije.
104
105  printf("Niz koji sortiramo je:\n");
106  for (i = 0; i < dimenzija; i++)
107    printf("%d\n", niz[i]);
108  ****
109
110  /* Sortiranje niza na odgovarajuci nacin */
111  if (tip_sortiranja == 0)
112    selection_sort(niz, dimenzija);
113  else if (tip_sortiranja == 1)
114    insertion_sort(niz, dimenzija);
115  else if (tip_sortiranja == 2)
116    bubble_sort(niz, dimenzija);
117  else if (tip_sortiranja == 3)
118    shell_sort(niz, dimenzija);
119  else if (tip_sortiranja == 4)
120    merge_sort(niz, 0, dimenzija - 1);
121  else
122    quick_sort(niz, 0, dimenzija - 1);
123
124  /* Ispis elemenata niza */
125  ****
126  Ovaj deo je iskomentarisan jer vreme potrebno za njegovo
127  izvrsavanje ne bi trebalo da bude ukljuceno u vreme izmereno
128  programom time. Takodje, kako je svrha ovog programa da prikaze
129  vremena razlicitih algoritama sortiranja, dimenzije nizova ce
130  biti, verovatno, ogromne, pa nema smisla imati na izlazu nizove
131  od toliko elemenata. Ovaj deo je koriscen u razvoju programa
132  zarad testiranja korektnosti.

```

```

133     printf("Sortiran niz je:\n");
135     for (i = 0; i < dimenzija; i++)
136         printf("%d\n", niz[i]);
137     ****
138
139     exit(EXIT_SUCCESS);
}

```

Rešenje 3.13

```

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

#define MAX_DIM 128

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

/* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace. */
int anagrami(char s[], char t[])
{
    int i;

    /* Ako dve niske imaju razlicit broj karaktera onda one nisu
     * anagrami */
    if (strlen(s) != strlen(t))
        return 0;

    /* Sortiramo niske */
    selectionSort(s);
    selectionSort(t);

    /* Dve sortirane niske su anagrami ako i samo ako su jednake */
    for (i = 0; s[i] != '\0'; i++)
        if (s[i] != t[i])

```

```

42         return 0;
43     return 1;
44 }
45
46 int main()
47 {
48     char s[MAX_DIM], t[MAX_DIM];
49
50     /* Ucitavanje niski sa ulaza */
51     printf("Unesite prvu nisku: ");
52     scanf("%s", s);
53     printf("Unesite drugu nisku: ");
54     scanf("%s", t);
55
56     /* Poziv funkcije */
57     if (anagrami(s, t))
58         printf("jesu\n");
59     else
60         printf("nisu\n");
61
62     return 0;
63 }
```

Rešenje 3.14

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX 256
4
5 /* Funkcija koja pronalazi najmanje rastojanje izmedju dva broja u
   sortiranom nizu celih brojeva */
6 int najmanje_rastojanje(int a[], int n)
7 {
8     int i, min;
9     min = a[1] - a[0];
10    for (i = 2; i < n; i++)
11        if (a[i] - a[i - 1] < min)
12            min = a[i] - a[i - 1];
13    return min;
14 }
15
16
17 int main()
18 {
19     int i, a[MAX];
20
21     /* Ucitavaju se elementi niza sve do kraja ulaza */
22     i = 0;
23     while (scanf("%d", &a[i]) != EOF)
24         i++;
```

```

27  /* Za sortiranje niza može se koristiti bilo koja od funkcija
28   sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
29   se selection sort. */
30   selection_sort(a, i);
31
32  /* Ispis rezultata */
33  printf("%d\n", najmanje_rastojanje(a, i));
34
35  return 0;
}

```

Rešenje 3.15

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX_DIM 256
4
5 /* Funkcija za određivanje onog elementa sortiranog niza koji se
6   najviše puta pojavio u tom nizu */
7 int najvise_puta(int a[], int n)
8 {
9   int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
10  /* Za i-ti element izračunava se koliko puta se pojavio u nizu */
11  for (i = 0; i < n; i = j) {
12    br_pojava = 1;
13    for (j = i + 1; j < n && a[i] == a[j]; j++)
14      br_pojava++;
15    /* Ispitivanje da li se do tog trenutka i-ti element pojavio
16     najviše puta u nizu */
17    if (br_pojava > max_br_pojava) {
18      max_br_pojava = br_pojava;
19      i_max_pojava = i;
20    }
21  }
22  /* Vraca se element koji se najviše puta pojavio u nizu */
23  return a[i_max_pojava];
24
25
26 int main()
27 {
28   int a[MAX_DIM], i;
29
30   /* Ucitavanje elemenata niza sve do kraja ulaza */
31   i = 0;
32   while (scanf("%d", &a[i]) != EOF)
33     i++;
34
35   /* Za sortiranje niza može se koristiti bilo koja od funkcija
36   sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
37   se merge sort. */
38   merge_sort(a, 0, i - 1);

```

```

39  /* Određuje se broj koji se najviše puta pojavio u nizu */
40  printf("%d\n", najvise puta(a, i));
41
42  return 0;
43 }
```

Rešenje 3.16

```

1 #include <stdio.h>
2 #include "sort.h"
3 #define MAX_DIM 256
4
5 /* Funkcija za binarnu pretragu niza vraca 1 ako se element x nalazi
6  u nizu, a 0 inace. Prepostavlja se da je niz sortiran u rastucem
7  poretku */
8 int binarna_pretraga(int a[], int n, int x)
9 {
10    int levi = 0, desni = n - 1, srednji;
11
12    while (levi <= desni) {
13        srednji = (levi + desni) / 2;
14        if (a[srednji] == x)
15            return 1;
16        else if (a[srednji] > x)
17            desni = srednji - 1;
18        else if (a[srednji] < x)
19            levi = srednji + 1;
20    }
21    return 0;
22}
23
24 int main()
25 {
26    int a[MAX_DIM], n = 0, zbir, i;
27
28    /* Ucitava se traženi zbir */
29    printf("Unesite traženi zbir: ");
30    scanf("%d", &zbir);
31
32    /* Ucitavaju se elementi niza sve do kraja ulaza */
33    i = 0;
34    printf("Unesite elemente niza: ");
35    while (scanf("%d", &a[i]) != EOF)
36        i++;
37    n = i;
38
39    /* Za sortiranje niza može se koristiti bilo koja od funkcija
40     sortiranja iz sort.h. Ilustracije radi, u ovom zadatku koristi
41     se quick sort. */
42    quick_sort(a, 0, n - 1);
43 }
```

```

43     for (i = 0; i < n; i++)
44         /* Za i-ti element niza binarno se pretrazuje da li se u ostatku
45            niza nalazi element koji sabran sa njim ima ucitanu vrednost
46            zbiru */
47         if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
48             printf("da\n");
49             return 0;
50         }
51     printf("ne\n");
52
53     return 0;
54 }
```

Rešenje 3.17

```

1 #include <stdio.h>
2 #define MAX_DIM 256
3
4 int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
5           int dim3)
6 {
7     int i = 0, j = 0, k = 0;
8     /* U slučaju da je dimenzija treceg niza manja od neophodne,
9        funkcija vraca -1 */
10    if (dim3 < dim1 + dim2)
11        return -1;
12
13    /* Vrši se učesljavanje nizova sve dok se ne dodje do kraja jednog
14       od njih */
15    while (i < dim1 && j < dim2) {
16        if (niz1[i] < niz2[j])
17            niz3[k++] = niz1[i++];
18        else
19            niz3[k++] = niz2[j++];
20    }
21    /* Ostatak prvog niza prepisujemo u treći */
22    while (i < dim1)
23        niz3[k++] = niz1[i++];
24
25    /* Ostatak drugog niza prepisujemo u treći */
26    while (j < dim2)
27        niz3[k++] = niz2[j++];
28    return dim1 + dim2;
29}
30
31 int main()
32 {
33     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: ");
39  while (1) {
40      scanf("%d", &niz1[i]);
41      if (niz1[i] == 0)
42          break;
43      i++;
44  }
45  printf("Unesite elemente drugog niza: ");
46  while (1) {
47      scanf("%d", &niz2[j]);
48      if (niz2[j] == 0)
49          break;
50      j++;
51  }
52  /* Poziv trazene funkcije */
53  dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

56  /* Ispis niza */
57  for (k = 0; k < dim3; k++)
58      printf("%d ", niz3[k]);
59  printf("\n");

60  return 0;
61 }
```

Rešenje 3.18

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     FILE *fin1 = NULL, *fin2 = NULL;
8     FILE *fout = NULL;
9     char ime1[11], ime2[11];
10    char prezime1[16], prezime2[16];
11    int kraj1 = 0, kraj2 = 0;
12
13    /* Ako nema dovoljno argumenata komandne linije */
14    if (argc < 3) {
15        fprintf(stderr, "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
16        exit(EXIT_FAILURE);
17    }
18
19    /* Otvaranje datoteke zadate prvim argumentom komandne linije */
20 }
```

```

20     fin1 = fopen(argv[1], "r");
21     if (fin1 == NULL) {
22         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
23         exit(EXIT_FAILURE);
24     }
25
26     /* Otvaranje datoteke zadate drugim argumentom komandne linije */
27     fin2 = fopen(argv[2], "r");
28     if (fin2 == NULL) {
29         fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
30         exit(EXIT_FAILURE);
31     }
32
33     /* Otvaranje datoteke za upis rezultata */
34     fout = fopen("ceo-tok.txt", "w");
35     if (fout == NULL) {
36         fprintf(stderr,
37                 "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n");
38         exit(EXIT_FAILURE);
39     }
40
41     /* Citanje narednog studenta iz prve datoteke */
42     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
43         kraj1 = 1;
44
45     /* Citanje narednog studenta iz druge datoteke */
46     if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
47         kraj2 = 1;
48
49     /* Sve dok nije dostignut kraj neke datoteke */
50     while (!kraj1 && !kraj2) {
51         int tmp = strcmp(ime1, ime2);
52         if (tmp < 0 || (tmp == 0 && strcmp(prezime1, prezime2) < 0)) {
53             /* Ime i prezime iz prve datoteke je leksikografski ranije, i
54                biva upisano u izlaznu datoteku */
55             fprintf(fout, "%s %s\n", ime1, prezime1);
56             /* Citanje narednog studenta iz prve datoteke */
57             if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
58                 kraj1 = 1;
59             } else {
60                 /* Ime i prezime iz druge datoteke je leksikografski ranije, i
61                    biva upisano u izlaznu datoteku */
62                 fprintf(fout, "%s %s\n", ime2, prezime2);
63                 /* Citanje narednog studenta iz druge datoteke */
64                 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
65                     kraj2 = 1;
66             }
67     }
68
69     /* Ako se iz prethodne petlje izaslo zato sto je dostignut kraj
70        druge datoteke, onda ima jos studenata u prvoj datoteci, koje
71        treba prepisati u izlaznu, redom, jer su vec sortirani po imenu.

```

```

72     */
73     while (!kraj1) {
74         fprintf(fout, "%s %s\n", ime1, prezime1);
75         if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
76             kraj1 = 1;
77     }
78
79     /* Ako se iz prve petlje izaslo zato sto je dostignut kraj prve
80      datoteke, onda ima jos studenata u drugoj datoteci, koje treba
81      prepisati u izlaznu, redom, jer su vec sortirani po imenu. */
82     while (!kraj2) {
83         fprintf(fout, "%s %s\n", ime2, prezime2);
84         if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
85             kraj2 = 1;
86     }
87
88     /* Zatvaranje datoteka */
89     fclose(fin1);
90     fclose(fin2);
91     fclose(fout);
92
93     exit(EXIT_SUCCESS);
94 }
```

Rešenje 3.19

```

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
14 /* 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 }
20
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;
```

```

27     Tacka tmp;
28
29     for (i = 0; i < n - 1; i++) {
30         min = i;
31         for (j = i + 1; j < n; j++) {
32             if (rastojanje(t[j]) < rastojanje(t[min])) {
33                 min = j;
34             }
35             if (min != i) {
36                 tmp = t[i];
37                 t[i] = t[min];
38                 t[min] = tmp;
39             }
40         }
41     }
42
43 /* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
44 void sortiraj_po_x(Tacka t[], int n)
45 {
46     int min, i, j;
47     Tacka tmp;
48
49     for (i = 0; i < n - 1; i++) {
50         min = i;
51         for (j = i + 1; j < n; j++) {
52             if (abs(t[j].x) < abs(t[min].x)) {
53                 min = j;
54             }
55         }
56         if (min != i) {
57             tmp = t[i];
58             t[i] = t[min];
59             t[min] = tmp;
60         }
61     }
62 }
63
64 /* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
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) {

```

```

79         tmp = t[i];
80         t[i] = t[min];
81         t[min] = tmp;
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: očekuje 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
117    /* Sve dok se ne stigne do kraja ulazne datoteke, ucitavaju se
118     koordinate tacaka i smestaju na odgovarajuce pozicije odredjene
119     brojacem i. */
120    i = 0;
121    while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
122        i++;
123    }
124
125    /* Ukupan broj procitanih tacaka */
126    n = i;
127
128    /* Analizira se prosledjena opcija. Moguce vrednosti za argv[1] su
129     "-x" ili "-y" ili "-o", pa je argv[1][0] sigurno crtica

```

```

131     (karakter -), a karakter argv[1][1] određuje kriterijum
132     sortiranja */
133     switch (argv[1][1]) {
134     case 'x':
135         /* Sortiranje po vrednosti x koordinate */
136         sortiraj_po_x(tacke, n);
137         break;
138     case 'y':
139         /* Sortiranje po vrednosti y koordinate */
140         sortiraj_po_y(tacke, n);
141         break;
142     case 'o':
143         /* Sortiranje po udaljenosti od koordinatnog pocetka */
144         sortiraj_po_rastojanju(tacke, n);
145         break;
146     }
147
148     /* Upisivanje dobijenog niza u izlaznu datoteku */
149     for (i = 0; i < n; i++) {
150         fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
151     }
152
153     /* Zatvaranje otvorenih datoteka */
154     fclose(ulaz);
155     fclose(izlaz);
156
157     return 0;
}

```

Rešenje 3.20

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 1000
6 #define MAX_DUZINA 16
7
8 /* Struktura koja reprezentuje jednog gradjanina */
9 typedef struct gr {
10     char ime[MAX_DUZINA];
11     char prezime[MAX_DUZINA];
12 } Gradjanin;
13
14 /* Funkcija sortira niz gradjana rastuce po imenima */
15 void sort_ime(Gradjanin a[], int n)
16 {
17     int i, j;
18     int min;
19     Gradjanin pom;

```

```

21   for (i = 0; i < n - 1; i++) {
22     /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
23        najmanji od elemenata a[i].ime,...,a[n-1].ime. */
24     min = i;
25     for (j = i + 1; j < n; j++)
26       if (strcmp(a[j].ime, a[min].ime) < 0)
27         min = j;
28     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
29       su (i) i min razliciti, inace je nepotrebno. */
30     if (min != i) {
31       pom = a[i];
32       a[i] = a[min];
33       a[min] = pom;
34     }
35   }
36
37 /* Funkcija sortira niz gradjana rastuce po prezimenima */
38 void sort_prezime(Gradjanin a[], int n)
39 {
40   int i, j;
41   int min;
42   Gradjanin pom;
43
44   for (i = 0; i < n - 1; i++) {
45     /* Unutrasnja petlja pronalazi poziciju min, na kojoj se nalazi
46        najmanji od elemenata a[i].prezime,...,a[n-1].prezime. */
47     min = i;
48     for (j = i + 1; j < n; j++)
49       if (strcmp(a[j].prezime, a[min].prezime) < 0)
50         min = j;
51     /* Zamena elemenata na pozicijama (i) i min. Ovo se radi samo ako
52       su (i) i min razliciti, inace je nepotrebno. */
53     if (min != i) {
54       pom = a[i];
55       a[i] = a[min];
56       a[min] = pom;
57     }
58   }
59 }
60
61 /* Pretraga niza Gradjana */
62 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
63 {
64   int i;
65   for (i = 0; i < n; i++)
66     if (strcmp(a[i].ime, x->ime) == 0
67       && strcmp(a[i].prezime, x->prezime) == 0)
68       return i;
69   return -1;
70 }
71

```

```

73
74     int main()
75    {
76        Gradjanin spisak1[MAX], spisak2[MAX];
77        int isti_rbr = 0;
78        int i, n;
79        FILE *fp = NULL;
80
81        /* Otvaranje datoteke */
82        if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
83            fprintf(stderr,
84                    "Neunesno otvaranje datoteke biracki-spisak.txt.\n");
85            exit(EXIT_FAILURE);
86        }
87
88        /* Citanje sadrzaja */
89        for (i = 0;
90             fscanf(fp, "%s %s", spisak1[i].ime,
91                     spisak1[i].prezime) != EOF; i++)
92            spisak2[i] = spisak1[i];
93        n = i;
94
95        /* Zatvaranje datoteke */
96        fclose(fp);
97
98        sort_ime(spisak1, n);
99
100       ****
101       Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
102       sortiranih nizova. Koriscen je samo u fazi testiranja programa.
103
104       printf("Biracki spisak [uredjen prema imenima]:\n");
105       for(i=0; i<n; i++)
106           printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
107
108       ****
109       sort_prezime(spisak2, n);
110
111       ****
112       Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
113       sortiranih nizova. Koriscen je samo u fazi testiranja programa.
114
115       printf("Biracki spisak [uredjen prema prezimenima]:\n");
116       for(i=0; i<n; i++)
117           printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
118
119       /* Linearno pretrazivanje nizova */
120       for (i = 0; i < n; i++)
121           if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
122               isti_rbr++;
123

```

```

125 /* Alternativno (efikasnije) resenje */
126 ****
127     for(i=0; i<n ;i++)
128         if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
129             strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
130             isti_rbr++;
131 ****
132
133 /* Ispis rezultata */
134 printf("%d\n", isti_rbr);
135
136 exit(EXIT_SUCCESS);
137 }

```

Rešenje 3.22

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 #define MAX_BR_RECI 128
6 #define MAX_DUZINA_RECI 32
7
8 /* Funkcija koja izracunava broj suglasnika u reci */
9 int broj_suglasnika(char s[])
10 {
11     char c;
12     int i;
13     int suglasnici = 0;
14     /* Prolaz karakter po karakter kroz zadatu nisku */
15     for (i = 0; s[i]; i++) {
16         /* Ako je u pitanju slovo, konvertuje se u veliko da bi bio
17            pokriven slucaj i malih i velikih suglasnika. */
18         if (isalpha(s[i])) {
19             c = toupper(s[i]);
20             /* Ukoliko slovo nije samoglasnik uvecava se broj suglasnika.
21            */
22             if (c != 'A' && c != 'E' && c != 'I' && c != 'O' && c != 'U')
23                 suglasnici++;
24         }
25     /* Vraca se izracunata vrednost */
26     return suglasnici;
27 }
28
29 /* Funkcija koja sortira reci po zadatom kriterijumu. Informacija o
30   duzini reci se mora proslediti zbog pravilnog upravljanja
31   memorijom */
32 void sortiraj_recu(char reci[][MAX_DUZINA_RECI], int n)
33 {
34     int min, i, j, broj_suglasnika_j, broj_suglasnika_min,

```

```

35     duzina_j, duzina_min;
36     char tmp[MAX_DUZINA_REC];
37     for (i = 0; i < n - 1; i++) {
38         min = i;
39         for (j = i; j < n; j++) {
40             /* Prvo se uporedjuje broj suglasnika */
41             broj_suglasnika_j = broj_suglasnika(reci[j]);
42             broj_suglasnika_min = broj_suglasnika(reci[min]);
43             if (broj_suglasnika_j < broj_suglasnika_min)
44                 min = j;
45             else if (broj_suglasnika_j == broj_suglasnika_min) {
46                 /* Zatim, recima koje imaju isti broj suglasnika uporedjuju
47                  se duzine */
48                 duzina_j = strlen(reci[j]);
49                 duzina_min = strlen(reci[min]);
50
51                 if (duzina_j < duzina_min)
52                     min = j;
53                 else
54                     /* Ako reci imaju i isti broj suglasnika i iste duzine,
55                      uporedjuju se leksikografski */
56                     if (duzina_j == duzina_min && strcmp(reci[j], reci[min]) < 0)
57                         min = j;
58                 }
59             }
60             if (min != i) {
61                 strcpy(tmp, reci[min]);
62                 strcpy(reci[min], reci[i]);
63                 strcpy(reci[i], tmp);
64             }
65         }
66     }
67
68     int main()
69     {
70         FILE *ulaz;
71         int i = 0, n;
72
73         /* Niz u koji ce biti smestane reci. Prvi broj označava broj reci,
74          a drugi maksimalnu duzinu pojedinacne reci */
75         char reci[MAX_BR_REC][MAX_DUZINA_REC];
76
77         /* Otvaranje datoteke niske.txt za citanje */
78         ulaz = fopen("niske.txt", "r");
79         if (ulaz == NULL) {
80             fprintf(stderr,
81                     "Greska prilikom otvaranja datoteke niske.txt!\n");
82             return 0;
83         }
84
85         /* Sve dok se moze pročitati sledeća rec */
86         while (fscanf(ulaz, "%s", reci[i]) != EOF) {

```

```

87     /* Proverava se da li ucitan maksimalan broj reci, i ako jeste,
88      prekida se ucitavanje */
89     if (i == MAX_BR_RECI)
90         break;
91     /* Priprema brojaca za narednu iteraciju */
92     i++;
93 }

95     /* n je duzina niza reci i predstavlja poslednju vrednost
96      koriscenog brojaca */
97     n = i;
98     /* Poziv funkcije za sortiranje reci */
99     sortiraj_reci(reci, n);

101    /* Ispis sortiranog niza reci */
102    for (i = 0; i < n; i++) {
103        printf("%s ", reci[i]);
104    }
105    printf("\n");

107    /* Zatvaranje datoteke */
108    fclose(ulaz);

109    return 0;
110}

```

Rešenje 3.23

```

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

#define MAX_ARTIKALA 100000

/* Struktura koja predstavlja jedan artikal */
typedef struct art {
    long kod;
    char naziv[20];
    char proizvodjac[20];
    float cena;
} Artikal;

/* Funkcija koja u nizu artikala binarnom pretragom nalazi onaj sa
   trazenim bar kodom */
int binarna_pretraga(Artikal a[], int n, long x)
{
    int levi = 0;
    int desni = n - 1;

    /* Dokle god je indeks levi levo od indeksa desni */
    while (levi <= desni) {

```

```

24     /* Racuna se sredisnji indeks */
25     int srednji = (levi + desni) / 2;
26     /* Ako je sredisnji element veci od trazenog, tada se trazeni
27      mora nalaziti u levoj polovini niza */
28     if (x < a[srednji].kod)
29         desni = srednji - 1;
30     /* Ako je sredisnji element manji od trazenog, tada se trazeni
31      mora nalaziti u desnoj polovini niza */
32     else if (x > a[srednji].kod)
33         levi = srednji + 1;
34     else
35         /* Ako je sredisnji element jednak trazenom, tada je artikal sa
36          bar kodom x pronadjen na poziciji srednji */
37         return srednji;
38     }
39     /* Ako nije pronadjen artikal za trazenim bar kodom, vraca se -1 */
40     return -1;
41 }
42
43 /* Funkcija koja sortira niz artikala po bar kodovima rastuce */
44 void selection_sort(Artikal a[], int n)
45 {
46     int i, j;
47     int min;
48     Artikal pom;
49
50     for (i = 0; i < n - 1; i++) {
51         min = i;
52         for (j = i + 1; j < n; j++)
53             if (a[j].kod < a[min].kod)
54                 min = j;
55         if (min != i) {
56             pom = a[i];
57             a[i] = a[min];
58             a[min] = pom;
59         }
60     }
61 }
62
63 int main()
64 {
65     Artikal asortiman[MAX_ARTIKALA];
66     long kod;
67     int i, n;
68     float racun;
69
70     FILE *fp = NULL;
71
72     /* Otvaranje datoteke */
73     if ((fp = fopen("artikli.txt", "r")) == NULL) {
74         fprintf(stderr, "Neuspesno otvaranje datoteke artikli.txt.\n");
75         exit(EXIT_FAILURE);
76     }

```

```

76    }

78 /* Ucitavanje artikala */
i = 0;
80 while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
                 asortiman[i].naziv, asortiman[i].proizvodjac,
                 &asortiman[i].cena) == 4)
82     i++;
84
86 /* Zatvaranje datoteke */
fclose(fp);

88 n = i;

90 /* Sortira se celokupan asortiman prodavnice prema kodovima jer ce
92 pri kucanju racuna prodavac unositi kod artikla. Prilikom
94 kucanja svakog racuna pretrazuje se asortiman, da bi se utvrdila
96 cena artikla. Kucanje racuna obuhvata vise pretraga asortimana i
98 cilj je da ta operacija bude sto efikasnija. Zato se koristi
100 algoritam binarne pretrage prilikom pretrazivanja po kodu
102 artikla. Iz tog razloga, potrebno je da asortiman bude sortiran
104 po kodovima i to ce biti uradjeno primenom selection sort
106 algoritma. Sortiranje se vrsti samo jednom na pocetku, ali se
108 zato posle artikli mogu brzo pretrazivati prilikom kucanja
110 proizvoljno puno racuna. Vreme koje se utrosi na sortiranje na
112 pocetku izvrsavanja programa, kasnije se isplati jer se za
brojna trazenja artikla umesto linearne moze koristiti
efikasnija binarna pretraga. */
selection_sort(asortiman, n);

106 /* Ispis stanja u prodavnici */
printf
108     ("Asortiman:\nKOD                      Naziv artikla      Ime
      proizvodjaca      Cena\n");
for (i = 0; i < n; i++)
110     printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
              asortiman[i].naziv, asortiman[i].proizvodjac,
              asortiman[i].cena);

114 kod = 0;
while (1) {
116     printf("-----\n");
118     printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
     printf("- Za nov racun unesite kod artikla!\n\n");
/* Unos bar koda provog artikla sledeceg kupca */
120     if (scanf("%ld", &kod) == EOF)
         break;
/* Trenutni racun novog kupca */
     racun = 0;
/* Za sve artikle trenutnog kupca */
122     while (1) {
         /* Vrsi se njihov pronalazak u nizu */

```

```

128     if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
129         printf("\tGRESKA: Ne postoji proizvod sa traženim kodom!\n");
130     } else {
131         printf("\tTražili ste:\t%s %s %.2f\n",
132               asortiman[i].naziv, asortiman[i].proizvodjac,
133               asortiman[i].cena);
134         /* I dodavanje na ukupan racun */
135         racun += asortiman[i].cena;
136     }
137     /* Unos bar koda sledeceg artikla trenutnog kupca, ili 0 ako on
138      nema vise artikla */
139     printf("Unesite kod artikla [ili 0 za prekid]: \t");
140     scanf("%ld", &kod);
141     if (kod == 0)
142         break;
143     /* Stampanje ukupnog racuna trenutnog kupca */
144     printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
145 }
146
147 printf("Kraj rada kase!\n");
148
149 exit(EXIT_SUCCESS);
150 }
```

Rešenje 3.24

```

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[21];
10    char prezime[26];
11    int prisustvo;
12    int zadaci;
13 } Student;
14
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
23     for (i = 0; i < n - 1; i++) {
24         min = i;
```

```

25     for (j = i + 1; j < n; j++)
26         if (strcmp(niz[j].ime, niz[min].ime) < 0)
27             min = j;
28
29     if (min != i) {
30         pom = niz[min];
31         niz[min] = niz[i];
32         niz[i] = pom;
33     }
34 }
35 }
36
37 /* Funkcija za sortiranje niza struktura po ukupnom broju uradjenih
38  zadataka opadajuce, a ukoliko neki studenti imaju isti broj
39  uradjenih zadataka sortiraju se po duzini imena rastuce. */
40 void sort_zadatke_pa_imena(Student niz[], int n)
41 {
42     int i, j;
43     int max;
44     Student pom;
45     for (i = 0; i < n - 1; i++) {
46         max = i;
47         for (j = i + 1; j < n; j++)
48             if (niz[j].zadaci > niz[max].zadaci)
49                 max = j;
50             else if (niz[j].zadaci == niz[max].zadaci
51                     && strlen(niz[j].ime) < strlen(niz[max].ime))
52                 max = j;
53         if (max != i) {
54             pom = niz[max];
55             niz[max] = niz[i];
56             niz[i] = pom;
57         }
58     }
59 }
60
61 /* Funkcija za sortiranje niza struktura po broju casova na kojima
62  su bili opadajuce. Ukoliko neki studenti imaju isti broj casova,
63  sortiraju se opadajuce po broju uradjenih zadataka, a ukoliko se
64  i po broju zadataka poklapaju, njihovo sortiranje ce biti po
65  prezimenu opadajuce. */
66 void sort_prisustvo_pa_zadatke_pa_prezimena(Student niz[], int n)
67 {
68     int i, j;
69     int max;
70     Student pom;
71     for (i = 0; i < n - 1; i++) {
72         max = i;
73         for (j = i + 1; j < n; j++)
74             if (niz[j].prisustvo > niz[max].prisustvo)
75                 max = j;
76             else if (niz[j].prisustvo == niz[max].prisustvo
77                     && strcmp(niz[j].prezime, niz[max].prezime) < 0)
78                 max = j;
79     }
80 }
```

```

77             && niz[j].zadaci > niz[max].zadaci)
79         max = j;
80     else if (niz[j].prisustvo == niz[max].prisustvo
81             && niz[j].zadaci == niz[max].zadaci
82             && strcmp(niz[j].prezime, niz[max].prezime) > 0)
83         max = j;
84     if (max != i) {
85         pom = niz[max];
86         niz[max] = niz[i];
87         niz[i] = pom;
88     }
89 }
90 }

91 int main(int argc, char *argv[])
{
92     Student praktikum[MAX];
93     int i, br_studenata = 0;
94
95     FILE *fp = NULL;
96
97     /* Otvaranje datoteke za citanje */
98     if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
99         fprintf(stderr, "Neunesno otvaranje datoteke aktivnost.txt.\n");
100        exit(EXIT_FAILURE);
101    }
102
103    /* Ucitavanje sadrzaja */
104    for (i = 0;
105         fscanf(fp, "%s%s%d%d", praktikum[i].ime,
106                 praktikum[i].prezime, &praktikum[i].prisustvo,
107                 &praktikum[i].zadaci) != EOF; i++);
108
109    /* Zatvaranje datoteke */
110    fclose(fp);
111    br_studenata = i;
112
113    /* Kreiranje prvog spiska studenata po prvom kriterijumu */
114    sort_ime_leksikografski(praktikum, br_studenata);
115    /* Otvaranje datoteke za pisanje */
116    if ((fp = fopen("dat1.txt", "w")) == NULL) {
117        fprintf(stderr, "Neunesno otvaranje datoteke dat1.txt.\n");
118        exit(EXIT_FAILURE);
119    }
120    /* Upis niza u datoteku */
121    fprintf
122        (fp, "Studenti sortirani po imenu leksikografski rastuce:\n");
123    for (i = 0; i < br_studenata; i++)
124        fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
125                 praktikum[i].prezime, praktikum[i].prisustvo,
126                 praktikum[i].zadaci);
127
128    /* Zatvaranje datoteke */
129    fclose(fp);

```

```

129  /* Kreiranje drugog spiska studenata po drugom kriterijumu */
131  sort_zadatke_pa_imena(praktikum, br_studenata);
132  /* Otvaranje datoteke za pisanje */
133  if ((fp = fopen("dat2.txt", "w")) == NULL) {
134      fprintf(stderr, "Neunesno otvaranje datoteke dat2.txt.\n");
135      exit(EXIT_FAILURE);
136  }
137  /* Upis niza u datoteku */
138  fprintf(fp, "Studenti sortirani po broju zadataka opadajuce,\n");
139  fprintf(fp, "pa po duzini imena rastuce:\n");
140  for (i = 0; i < br_studenata; i++)
141      fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
142              praktikum[i].prezime, praktikum[i].prisustvo,
143              praktikum[i].zadaci);
144  /* Zatvaranje datoteke */
145  fclose(fp);

146  /* Kreiranje treceg spiska studenata po trećem kriterijumu */
147  sort_prisustvo_pa_zadatke_pa_prezimena(praktikum, br_studenata);
148  /* Otvaranje datoteke za pisanje */
149  if ((fp = fopen("dat3.txt", "w")) == NULL) {
150      fprintf(stderr, "Neunesno otvaranje datoteke dat3.txt.\n");
151      exit(EXIT_FAILURE);
152  }
153  /* Upis niza u datoteku */
154  fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
155  fprintf(fp, "pa po broju zadataka,\n");
156  fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
157  for (i = 0; i < br_studenata; i++)
158      fprintf(fp, "%s %s %d %d\n", praktikum[i].ime,
159              praktikum[i].prezime, praktikum[i].prisustvo,
160              praktikum[i].zadaci);
161  /* Zatvaranje datoteke */
162  fclose(fp);

163  exit(EXIT_SUCCESS);
}

```

Rešenje 3.25

```

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

4 #define KORAK 10
5
6 /* Struktura koja opisuje jednu pesmu */
7 typedef struct {
8     char *izvodjac;
9     char *naslov;
10

```

```

12     int broj_gledanja;
13 } Pesma;

14 /* Funkcija za uporedjivanje pesama po broju gledanosti (potrebna za
15    rad qsort funkcije) */
16 int uporedi_gledanost(const void *pp1, const void *pp2)
17 {
18     Pesma *p1 = (Pesma *) pp1;
19     Pesma *p2 = (Pesma *) pp2;

20     return p2->broj_gledanja - p1->broj_gledanja;
21 }

24 /* Funkcija za uporedjivanje pesama po naslovu (potrebna za rad qsort
25    funkcije) */
26 int uporedi_naslove(const void *pp1, const void *pp2)
27 {
28     Pesma *p1 = (Pesma *) pp1;
29     Pesma *p2 = (Pesma *) pp2;

30     return strcmp(p1->naslov, p2->naslov);
31 }

34 /* Funkcija za uporedjivanje pesama po izvodjacu (potrebna za rad
35    qsort funkcije) */
36 int uporedi_izvodjace(const void *pp1, const void *pp2)
37 {
38     Pesma *p1 = (Pesma *) pp1;
39     Pesma *p2 = (Pesma *) pp2;

40     return strcmp(p1->izvodjac, p2->izvodjac);
41 }

44 int main(int argc, char *argv[])
45 {
46     FILE *ulaz;
47     Pesma *pesme;           /* Pokazivac na deo memorije za
48                           cuvanje pesama */
49     int alocirano_za_pesme; /* Broj mesta alociranih za pesme */
50     int i;                  /* Redni broj pesme cije se
51                           informacije citaju */
52     int n;                  /* Ukupan broj pesama */
53     int j, k;
54     char c;
55     int alocirano;          /* Broj mesta alociranih za propratne
56                           informacije o pesmama */
57     int broj_gledanja;

58     /* Priprema datoteke za citanje */
59     ulaz = fopen("pesme_bez_prepostavki.txt", "r");
60     if (ulaz == NULL) {
61         fprintf(stderr, "Greska pri otvaranju ulazne datoteke!\n");
62 }

```

```

    return 0;
}

/* Citanje informacija o pesmama */
pesme = NULL;
alocirano_za_pesme = 0;
i = 0;

while (1) {
    /* Proverava da li je dostignut kraj datoteke */
    c = fgetc(ulaz);
    if (c == EOF) {
        /* Nema vise sadrzaja za citanje */
        break;
    } else {
        /* Inace, vracamo procitani karakter nazad */
        ungetc(c, ulaz);
    }

    /* Provera da li postoji dovoljno memorije za citanje nove pesme */
    if (alocirano_za_pesme == i) {

        /* Ako ne, ako je potrosena sva alocirana memorija, alocira se
           novih KORAK mesta */
        alocirano_za_pesme += KORAK;
        pesme =
            (Pesma *) realloc(pesme,
                              alocirano_za_pesme * sizeof(Pesma));
    }

    /* Proverava da li je nova memorija uspesno realocirana */
    if (pesme == NULL) {
        /* Ako nije ispisuje se obavestenje */
        fprintf(stderr, "Problem sa alokacijom memorije!\n");
        /* I oslobadjava sva memorija zauzeta do ovog koraka */
        for (k = 0; k < i; k++) {
            free(pesme[k].izvodjac);
            free(pesme[k].naslov);
        }
        free(pesme);
        return 0;
    }
}

/* Ako jeste, nastavlja se sa citanjem pesama ... */
/* Cita se ime izvodjaca */
j = 0;                                /* Pozicija na koju treba smestiti
                                         procitani karakter */
alocirano = 0;                            /* Broj alociranih mesta */
pesme[i].izvodjac = NULL;                /* Memorija za smestanje procitanih
                                         karaktera */

```

```

114
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 izvrsavanjem 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 terminirajuce nule na kraj reci */
151         pesme[i].izvodjac[j] = '\0';
152
153         /* Preskace se karakter - */
154         fgetc(ulaz);
155
156         /* Preskace se razmak */
157         fgetc(ulaz);
158
159         /* Cita se naslov pesme */
160         j = 0;                                /* Pozicija na koju treba smestiti
161                                              procitani karakter */
162         alocirano = 0;                          /* Broj alociranih mesta */
163         pesme[i].naslov = NULL;                /* Memorija za smestanje procitanih
164                                              karaktera */

```

```

166    /* Sve do zareza (koji se nalazi nakon naslova pesme) citaju se
167     karakteri iz datoteke */
168    while ((c = fgetc(ulaz)) != ',') {
169        /* Provera da li postoji dovoljno memorije za smestanje
170         procitanog karaktera */
171        if (j == alocirano) {
172            /* Ako ne, ako je potrosena sva alocirana memorija, alocira
173             se novih KORAK mesta */
174            alocirano += KORAK;
175            pesme[i].naslov =
176                (char *) realloc(pesme[i].naslov,
177                                alocirano * sizeof(char));
178
179            /* Provera da li je nova alokacija uspesna */
180            if (pesme[i].naslov == NULL) {
181                /* Ako nije, oslobadja se sva memorija zauzeta do ovog
182                 koraka */
183                for (k = 0; k < i; k++) {
184                    free(pesme[k].izvodjac);
185                    free(pesme[k].naslov);
186                }
187                free(pesme[i].izvodjac);
188                free(pesme);
189
190                /* I prekida izvrsavanje programa */
191                return 0;
192            }
193            /* Ako postoji dovoljno memorije, smesta se procitani karakter
194             */
195            pesme[i].naslov[j] = c;
196            j++;
197            /* I nastavlja dalje sa citanjem */
198        }
199        /* Upisuje se terminirajuca nula na kraj reci */
200        pesme[i].naslov[j] = '\0';
201
202        /* Preskace se razmak */
203        fgetc(ulaz);
204
205        /* Cita se broj gledanja */
206        broj_gledanja = 0;
207
208        /* Sve do znaka za novi red (kraja linije) citaju se karakteri iz
209         datoteke */
210        while ((c = fgetc(ulaz)) != '\n') {
211            broj_gledanja = broj_gledanja * 10 + (c - '0');
212        }
213        pesme[i].broj_gledanja = broj_gledanja;
214
215        /* Prelazi se na citanje sledece pesme */
216        i++;

```

```

216 }
218 /* Informacija o broju procitanih pesama */
219 n = i;
220 /* Zatvaranje nepotrebne datoteke */
221 fclose(ulaz);
222
223 /* Analiza argumenta komandne linije */
224 if (argc == 1) {
225     /* Nema dodatnih opcija => sortiranje po broju gledanja */
226     qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
227 } else {
228     if (argc == 2 && strcmp(argv[1], "-n") == 0) {
229         /* Sortiranje po naslovu */
230         qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
231     } else {
232         if (argc == 2 && strcmp(argv[1], "-i") == 0) {
233             /* Sortiranje po izvodjacu */
234             qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
235         } else {
236             fprintf(stderr, "Nedozvoljeni argumenti!\n");
237             free(pesme);
238             return 0;
239         }
240     }
241
242 /* Ispis rezultata */
243 for (i = 0; i < n; i++) {
244     printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
245            pesme[i].broj_gledanja);
246 }
247
248 /* Oslobođjanje memorije */
249 for (i = 0; i < n; i++) {
250     free(pesme[i].izvodjac);
251     free(pesme[i].naslov);
252 }
253 free(pesme);
254
255 return 0;
256 }
```

Rešenje 3.28

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 /* Funkcija koja određuje zbir v-te vrste matrice a koja ima m
6   kolona */
```

```

8   int zbir_vrste(int **a, int v, int m)
9   {
10    int i, zbir = 0;
11
12    for (i = 0; i < m; i++) {
13      zbir += a[v][i];
14    }
15    return zbir;
16  }
17
18 /* Funkcija koja sortira vrste matrice (pokazivace na vrste) na
19    osnovu zbirka koriscenjem selection sort algoritma */
20 void sortiraj_vrste(int **a, int n, int m)
21 {
22    int i, j, min;
23
24    for (i = 0; i < n - 1; i++) {
25      min = i;
26      for (j = i + 1; j < n; j++) {
27        if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
28          min = j;
29        }
30      }
31      if (min != i) {
32        int *tmp;
33        tmp = a[i];
34        a[i] = a[min];
35        a[min] = tmp;
36      }
37    }
38
39 int main(int argc, char *argv[])
40 {
41    int **a;
42    int n, m;
43
44    /* Unos dimenzija matrice */
45    printf("Unesite dimenzije matrice: ");
46    scanf("%d %d", &n, &m);
47
48    /* Alokacija memorije */
49    a = alociraj_matricu(n, m);
50    if (a == NULL) {
51      fprintf(stderr, "Neuspesna alokacija matrice\n");
52      exit(EXIT_FAILURE);
53    }
54
55    /* Ucitavanje elemenata matrice */
56    printf("Unesite elemente matrice po vrstama:\n");
57    ucitaj_matricu(a, n, m);
58

```

```

60  /* Poziv funkcije koja sortira vrste matrice prema zbiru */
61  sortiraj_vrste(a, n, m);

62  /* Ispis rezultujuce matrice */
63  printf("Sortirana matrica je:\n");
64  ispisi_matricu(a, n, m);

66  /* Oslobadjanje memorije */
67  a = dealociraj_matricu(a, n);
68

69  return 0;
70 }
```

Rešenje 3.31

```

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 poredi_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     int b1 = *((int *) a);
17     int b2 = *((int *) b);
18
19     /* return b1 - b2; */
20     if (b1 > b2)
21         return 1;
22     else if (b1 < b2)
23         return -1;
24     else
25         return 0;
26 }
27
28 int poredi_int_nerastuce(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 -poredi_int(a, b);
36 }
```

```

37 }
38
39 int main()
40 {
41     size_t n;
42     int i, x;
43     int a[MAX], *p = NULL;
44
45     /* Unos dimenzije */
46     printf("Uneti dimenziju niza: ");
47     scanf("%ld", &n);
48     if (n > MAX)
49         n = MAX;
50
51     /* Unos elementa niza */
52     printf("Uneti elemente niza:\n");
53     for (i = 0; i < n; i++)
54         scanf("%d", &a[i]);
55
56     /* Sortiranje niza celih brojeva */
57     qsort(a, n, sizeof(int), &poredi_int);
58
59     /* Prikaz sortiranog niz */
60     printf("Sortirani niz u rastucem poretku:\n");
61     for (i = 0; i < n; i++)
62         printf("%d ", a[i]);
63     putchar('\n');
64
65     /* Pretrazivanje niza */
66     /* Vrednost koja ce biti trazena u nizu */
67     printf("Uneti element koji se trazi u nizu: ");
68     scanf("%d", &x);
69
70     /* Binarna pretraga */
71     printf("Binarna pretraga: \n");
72     p = bsearch(&x, a, n, sizeof(int), &poredi_int);
73     if (p == NULL)
74         printf("Elementa nema u nizu!\n");
75     else
76         printf("Element je nadjen na poziciji %ld\n", p - a);
77
78     /* Linearna pretraga */
79     printf("Linearna pretraga (lfind): \n");
80     p = lfind(&x, a, &n, sizeof(int), &poredi_int);
81     if (p == NULL)
82         printf("Elementa nema u nizu!\n");
83     else
84         printf("Element je nadjen na poziciji %ld\n", p - a);
85
86     return 0;
87 }

```

Rešenje 3.32

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <search.h>
5
6 #define MAX 100
7
8 /* Funkcija racuna broj delilaca broja x */
9 int broj_delioca(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 korenju 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 poredi_po_broju_delioca(const void *a, const void *b)
38 {
39     int ak = *(int *) a;
40     int bk = *(int *) b;
41     int n_d_a = broj_delioca(ak);
42     int n_d_b = broj_delioca(bk);
43
44     return n_d_a - n_d_b;
45 }
46
47 int main()
48 {
49     size_t n;
50     int i;

```

```

51   int a[MAX];
53   /* Unos dimenzije */
54   printf("Uneti dimenziju niza: ");
55   scanf("%d", &n);
56   if (n > MAX)
57     n = MAX;
58
59   /* Unos elementa niza */
60   printf("Uneti elemente niza:\n");
61   for (i = 0; i < n; i++)
62     scanf("%d", &a[i]);
63
64   /* Sortiranje niza celih brojeva prema broju delilaca */
65   qsort(a, n, sizeof(int), &poredi_po_broju_delilaca);
66
67   /* Prikaz sortiranog niza */
68   printf("Sortirani niz u rastucem poretku prema broju delilaca:\n");
69   for (i = 0; i < n; i++)
70     printf("%d ", a[i]);
71   putchar('\n');
72
73   return 0;
}

```

Rešenje 3.33

```

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 31
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 veca 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
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 {
30     return strcmp((char *) a, (char *) b);
31 }
32
33 /* Funkcija slična prethodnoj, osim što elemente ne uporedjuje
34    leksikografski, već po duzini */
35 int poredi_duzine(const void *a, const void *b)
36 {
37     return strlen((char *) a) - strlen((char *) b);
38 }
39
40 int main()
41 {
42     int i;
43     size_t n;
44     FILE *fp = NULL;
45     char niske[MAX_NISKI][MAX_DUZINA];
46     char *p = NULL;
47     char x[MAX_DUZINA];
48
49     /* Otvaranje datoteke */
50     if ((fp = fopen("niske.txt", "r")) == NULL) {
51         fprintf(stderr, "Neuspelo otvaranje datoteke niske.txt.\n");
52         exit(EXIT_FAILURE);
53     }
54
55     /* Citanje sadrzaja datoteke */
56     for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
57
58     /* Zatvaranje datoteke */
59     fclose(fp);
60     n = i;
61
62     /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort
63        prosledjuje se funkcija kojom se zadaje kriterijum poredjenja 2
64        niske po duzini */
65     qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_leksikografski);
66
67     printf("Leksikografski sortirane niske:\n");
68     for (i = 0; i < n; i++)
69         printf("%s ", niske[i]);
70     printf("\n");
71
72     /* Unos trazene niske */
73     printf("Uneti trazenu nisku: ");
74     scanf("%s", x);
75
76     /* Binarna pretraga */

```

```

77  /* Prosledjuje se pokazivac na funkciju poredi_leksikografski jer
79   je niz vec sortiran leksikografski. */
80   p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
81             &poredi_leksikografski);
82
83   if (p != NULL)
84     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
85           p, (p - (char *) niske) / MAX_DUZINA);
86   else
87     printf("Niska nije pronadjena u nizu\n");
88
89   /* Sortiranje po duzini */
90   qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);
91
92   printf("Niske sortirane po duzini:\n");
93   for (i = 0; i < n; i++)
94     printf("%s ", niske[i]);
95   printf("\n");
96
97   /* Linearna pretraga */
98   p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
99             &poredi_leksikografski);
100
101  if (p != NULL)
102    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
103          p, (p - (char *) niske) / MAX_DUZINA);
104  else
105    printf("Niska nije pronadjena u nizu\n");
106
107  exit(EXIT_SUCCESS);
}

```

Rešenje 3.34

```

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 31
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 -----

```

```

19 | Z | -----> | f | g | h | \0|
-----  

21 Sa leve strane je vertikalno prikazan niz pokazivaca, gde je i-ti  

njegov element pokazivac koji pokazuje na alocirane karaktere i-te  

reci. Njegov tip je char*.  

23  

25 Kako pokazivaci a i b u sledecoj funkciji sadrze adrese elemenata  

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  

moraju i kastovati. Da bi se leksikografski uporedili elementi X i  

29 Z, moraju se uporediti stringovi na koje oni pokazuju, pa zato se  

u sledecoj funkciji poziva strcmp() nad onim na sta pokazuju a i b,  

kastovani na odgovarajuci tip.  

31 ****
int poredi_leksikografski(const void *a, const void *b)
33 {
    return strcmp(*(char **) a, *(char **) b);
35 }

37 /* Funkcija sличna prethodnoj, osim sto elemente ne uporedjuje
   leksikografski, vec po duzini */
39 int poredi_duzine(const void *a, const void *b)
{
    return strlen(*(char **) a) - strlen(*(char **) b);
}
43

45 /* Ovo je funkcija poredjenja za bsearch. Pokazivac b pokazuje na
   element u nizu sa kojim se poredi, pa njega treba kastovati na
   47 char** i dereferencirati, (videti obrazlozenje za prvu funkciju u
   ovom zadatku, a pokazivac a pokazuje na element koji se trazi. U
   49 main funkciji je to x, koji je tipa char*, tako da pokazivac a
   ovde samo treba kastovati i ne dereferencirati. */
int poredi_leksikografski_b(const void *a, const void *b)
51 {
    return strcmp((char *) a, *(char **) b);
53 }

55 int main()
{
57     int i;
size_t n;
59     FILE *fp = NULL;
    char *niske[MAX_NISKI];
61     char **p = NULL;
    char x[MAX_DUZINA];

63     /* Otvaranje datoteke */
65     if ((fp = fopen("niske.txt", "r")) == NULL) {
        fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
        exit(EXIT_FAILURE);
    }
69

```

```

    /* Citanje sadrzaja datoteke */
71   i = 0;
72   while (fscanf(fp, "%s", x) != EOF) {
73     /* Alociranje dovoljne memorije za i-tu nisku */
74     if ((niske[i] = malloc((strlen(x) + 1) * sizeof(char))) == NULL)
75     {
76       fprintf(stderr, "Greska pri alociranju niske\n");
77       exit(EXIT_FAILURE);
78     }
79     /* Kopiranje procitane niske na svoje mesto */
80     strcpy(niske[i], x);
81     i++;
82   }

83   /* Zatvaranje datoteke */
84   fclose(fp);
85   n = i;

86   /* Sortiranje niski leksikografski. Biblioteckoj funkciji qsort se
87   prosledjuje funkcija kojom se zadaje kriterijum poredjenja 2
88   niske po duzini */
89   qsort(niske, n, sizeof(char *), &poredi_leksikografiski);

90   printf("Leksikografski sortirane niske:\n");
91   for (i = 0; i < n; i++)
92     printf("%s ", niske[i]);
93   printf("\n");

94   /* Unos trazene niske */
95   printf("Uneti trazenu nisku: ");
96   scanf("%s", x);

97   /* Binarna pretraga */
98   p = bsearch(x, niske, n, sizeof(char *), &poredi_leksikografiski_b);
99   if (p != NULL)
100     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
101           *p, p - niske);
102   else
103     printf("Niska nije pronadjena u nizu\n");

104   /* Linearna pretraga */
105   p = lfind(x, niske, &n, sizeof(char *), &poredi_leksikografiski_b);
106   if (p != NULL)
107     printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
108           *p, p - niske);
109   else
110     printf("Niska nije pronadjena u nizu\n");

111   /* Sortiramo po duzini */
112   qsort(niske, n, sizeof(char *), &poredi_duzine);

113   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]);
128
129   exit(EXIT_SUCCESS);
}

```

Rešenje 3.35

```

#include <stdio.h>
2 #include <stdlib.h>
# include <string.h>
4 #include <search.h>

6 #define MAX 500

8 /* Struktura sa svim informacijama o pojedinacnom studentu */
typedef struct {
10   char ime[21];
11   char prezime[21];
12   int bodovi;
} Student;

14 /* Funkcija poredjenja za sortiranje po broju bodova. Studenti sa
15 istim brojem bodova se dodatno sortiraju leksikografski po
16 prezimenu */
18 int poredi1(const void *a, const void *b)
{
20   Student *prvi = (Student *) a;
21   Student *drugi = (Student *) b;

22   if (prvi->bodovi > drugi->bodovi)
23     return -1;
24   else if (prvi->bodovi < drugi->bodovi)
25     return 1;
26   else
27     /* Ako su jednaki po broju bodova, treba ih uporediti po
28     prezimenu */
29     return strcmp(prvi->prezime, drugi->prezime);
}

32 /* Funkcija za poredjenje koja se koristi u pretrazi po broju bodova.
33 Prvi parametar je ono sto se trazi u nizu (broj bodova), a drugi
34 parametar je element niza ciji se bodovi porede. */
36 int poredi2(const void *a, const void *b)
{
37   int bodovi = *(int *) a;

```

```

40     Student *s = (Student *) b;
41     return s->bodovi - bodovi;
42 }
43
44 /* Funkcija za poredjenje koja se koristi u pretrazi po prezimenu.
45  Prvi parametar je ono sto se trazi u nizu (prezime), a drugi
46  parametar je element niza cije se prezime poredi. */
47 int poređi3(const void *a, const void *b)
48 {
49     char *prezime = (char *) a;
50     Student *s = (Student *) b;
51     return strcmp(prezime, s->prezime);
52 }
53
54 int main(int argc, char *argv[])
55 {
56     Student kolokvijum[MAX];
57     int i;
58     size_t br_studenata = 0;
59     Student *nadjen = NULL;
60     FILE *fp = NULL;
61     int bodovi;
62     char prezime[21];
63
64     /* Ako je program pozvan sa nedovoljnim brojem argumenata daje se
65      informacija korisniku kako se program koristi i prekida se
66      izvrsavanje. */
67     if (argc < 2) {
68         fprintf(stderr,
69             "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
70             argv[0]);
71         exit(EXIT_FAILURE);
72     }
73
74     /* Otvaranje datoteke */
75     if ((fp = fopen(argv[1], "r")) == NULL) {
76         fprintf(stderr, "Neuspelo otvaranje datoteke %s\n", argv[1]);
77         exit(EXIT_FAILURE);
78     }
79
80     /* Ucitavanje sadrzaja */
81     for (i = 0;
82          fscanf(fp, "%s%s%d", kolokvijum[i].ime,
83                  kolokvijum[i].prezime,
84                  &kolokvijum[i].bodovi) != EOF; i++);
85
86     /* Zatvaranje datoteke */
87     fclose(fp);
88     br_studenata = i;
89
90     /* Sortiranje niza studenata po broju bodova, gde se unutar grupe
91      studenata sa istim brojem bodova sortiranje vrsti po prezimenu */

```

```

92     qsort(kolokvijum, br_studenata, sizeof(Student), &poredi1);
94
94     printf("Studenti sortirani po broju poena opadajuce, ");
95     printf("pa po prezimenu rastuce:\n");
96     for (i = 0; i < br_studenata; i++)
97         printf("%s %s %d\n", kolokvijum[i].ime,
98                kolokvijum[i].prezime, kolokvijum[i].bodovi);
99
100    /* Pretrazivanje studenata po broju bodova se vrsti binarnom
101       pretragom jer je niz sortiran po broju bodova. */
102    printf("Unesite broj bodova: ");
103    scanf("%d", &bodovi);
104
105    nadjen =
106        bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
107                &poredi2);
108
109    if (nadjen != NULL)
110        printf
111            ("Pronadjeni je student sa unetim brojem bodova: %s %s %d\n",
112             nadjen->ime, nadjen->prezime, nadjen->bodovi);
113    else
114        printf("Nema studenta sa unetim brojem bodova\n");
115
116    /* Pretraga po prezimenu se mora vrstiti linearno jer je niz
117       sortiran po bodovima. */
118    printf("Unesite prezime: ");
119    scanf("%s", prezime);
120
121    nadjen =
122        lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
123               &poredi3);
124
125    if (nadjen != NULL)
126        printf
127            ("Pronadjeni je student sa unetim prezimenom: %s %s %d\n",
128             nadjen->ime, nadjen->prezime, nadjen->bodovi);
129    else
130        printf("Nema studenta sa unetim prezimenom\n");
131
132    exit(EXIT_SUCCESS);
133 }

```

Rešenje 3.36

```

2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 #define MAX 128

```

```

1  /* Funkcija poređi dva karaktera */
2  int uporedi_char(const void *pa, const void *pb)
3  {
4      return *(char *) pa - *(char *) pb;
5  }
6
7  /* Funkcija vraca 1 ako su argumenti anagrami, a 0 inace */
8  int anagrami(char s[], char t[])
9  {
10     /* Ako dve niske imaju razlicitu duzinu onda one nisu anagrami */
11     if (strlen(s) != strlen(t))
12         return 0;
13
14     /* Sortiranje niski */
15     qsort(s, strlen(s) / sizeof(char), sizeof(char), &uporedi_char);
16     qsort(t, strlen(t) / sizeof(char), sizeof(char), &uporedi_char);
17
18     /* Ako su niske nakon sortiranja iste onda one jesu anagrami, u
19      suprotnom, nisu */
20     return !strcmp(s, t);
21 }
22
23 int main()
24 {
25     char s[MAX], t[MAX];
26
27     /* Unos niski */
28     printf("Unesite prvu nisku: ");
29     scanf("%s", s);
30     printf("Unesite drugu nisku: ");
31     scanf("%s", t);
32
33     /* Ispituje se da li su niske anagrami */
34     if (anagrami(s, t))
35         printf("jesu\n");
36     else
37         printf("nisu\n");
38
39     return 0;
40 }
41
42 }
```

Rešenje 3.37

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX 10
6 #define MAX_DUZINA 32
7
8 /* Funkcija poređenja */
```

```

9 int uporedi_niske(const void *pa, const void *pb)
11 {
12     return strcmp((char *) pa, (char *) pb);
13 }
14
15 int main()
16 {
17     int i, n;
18     char S[MAX] [MAX_DUZINA];
19
20     /* Unos broja niski */
21     printf("Unesite broj niski:");
22     scanf("%d", &n);
23
24     /* Unos niza niski */
25     printf("Unesite niske:\n");
26     for (i = 0; i < n; i++)
27         scanf("%s", S[i]);
28
29     /* Sortiranje niza niski */
30     qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);
31
32     //*****
33     //Ovaj deo je iskomentarisan jer se u zadatku ne trazi ispis
34     //sortiranih niski. Koriscen je samo u fazi testiranja programa.
35
36     printf("Sortirane niske su:\n");
37     for(i = 0; i < n; i++)
38         printf("%s ", S[i]);
39
40     //*****
41
42     /* Ako postoje dve iste niske u nizu, onda ce one nakon sortiranja
43     niza biti jedna do druge */
44     for (i = 0; i < n - 1; i++)
45         if (strcmp(S[i], S[i + 1]) == 0) {
46             printf("ima\n");
47             return 0;
48         }
49
50     printf("nema\n");
51     return 0;
52 }
```

Rešenje 3.38

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 21
```

```

7  /* Struktura koja predstavlja jednog studenta */
9   typedef struct student {
10     char nalog[8];
11     char ime[MAX];
12     char prezime[MAX];
13     int poeni;
14 } Student;
15
16 /* Funkcija poredi studente prema broju poena, rastuce */
17 int uporedi_poeni(const void *a, const void *b)
18 {
19   Student s = *(Student *) a;
20   Student t = *(Student *) b;
21   return s.poeni - t.poeni;
22 }
23
24 /* Funkcija poredi studente prvo prema godini, zatim prema smeru i na
25 kraju prema indeksu */
26 int uporedi_nalog(const void *a, const void *b)
27 {
28   Student s = *(Student *) a;
29   Student t = *(Student *) b;
30   /* Za svakog studenta iz naloga se izdvaja godina upisa, smer i
31   broj indeksa */
32   int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
33   int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
34   char smer1 = s.nalog[1];
35   char smer2 = t.nalog[1];
36   int indeks1 =
37     (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
38     s.nalog[6] - '0';
39   int indeks2 =
40     (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
41     t.nalog[6] - '0';
42   if (godina1 != godina2)
43     return godina1 - godina2;
44   else if (smer1 != smer2)
45     return smer1 - smer2;
46   else
47     return indeks1 - indeks2;
48 }
49
50 int uporedi_bsearch(const void *a, const void *b)
51 {
52   /* Nalog studenta koji se trazi */
53   char *nalog = (char *) a;
54   /* Kljuc pretrage */
55   Student s = *(Student *) b;
56
57   int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
58   int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
59   char smer1 = nalog[1];

```

```

59     char smer2 = s.nalog[1];
60     int indeks1 =
61         (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] - '0'
62         ;
63     int indeks2 =
64         (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
65         s.nalog[6] - '0';
66     if (godina1 != godina2)
67         return godina1 - godina2;
68     else if (smer1 != smer2)
69         return smer1 - smer2;
70     else
71         return indeks1 - indeks2;
72 }

73 int main(int argc, char **argv)
74 {
75     Student *nadjen = NULL;
76     char nalog_trazeni[8];
77     Student niz_studenata[100];
78     int i = 0, br_studenata = 0;
79     FILE *in = NULL, *out = NULL;

80     /* Ako je broj argumenata komandne linije razlicit i od 2 i od 3,
81      korisnik nije ispravno pozvao program i prijavljuje se greska.
82      */
83     if (argc != 2 && argc != 3) {
84         fprintf(stderr,
85                 "Greska! Program se poziva sa: ./a.out -opcija [nalog]\n"
86             );
87         exit(EXIT_FAILURE);
88     }

89     /* Otvaranje datoteke za citanje */
90     in = fopen("studenti.txt", "r");
91     if (in == NULL) {
92         fprintf(stderr,
93                 "Greska prilikom otvaranja datoteke studenti.txt!\n");
94         exit(EXIT_FAILURE);
95     }

96     /* Otvaranje datoteke za pisanje */
97     out = fopen("izlaz.txt", "w");
98     if (out == NULL) {
99         fprintf(stderr,
100                 "Greska prilikom otvaranja datoteke izlaz.txt!\n");
101         exit(EXIT_FAILURE);
102     }

103     /* Ucitavanje studenta iz ulazne datoteke sve do njenog kraja */
104     while (fscanf
105           (in, "%s %s %s %d", niz_studenata[i].nalog,
106

```

```

109         niz_studenata[i].ime, niz_studenata[i].prezime,
110         &niz_studenata[i].poeni) != EOF)
111     i++;
112
113     br_studenata = i;
114
115     /* Ako je prisutna opcija -p, vrši se sortiranje po poenima */
116     if (strcmp(argv[1], "-p") == 0)
117         qsort(niz_studenata, br_studenata, sizeof(Student),
118               &uporedi_poeni);
119
120     /* Ako je prisutna opcija -n, vrši se sortiranje po nalogu */
121     else if (strcmp(argv[1], "-n") == 0)
122         qsort(niz_studenata, br_studenata, sizeof(Student),
123               &uporedi_nalog);
124
125     /* Sortirani studenti se ispisuju u izlaznu datoteku */
126     for (i = 0; i < br_studenata; i++)
127         fprintf(out, "%s %s %s %d\n",
128                 niz_studenata[i].ime, niz_studenata[i].prezime,
129                 niz_studenata[i].poeni);
130
131     /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
132      studenta... */
133     if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
134         strcpy(nalog_trazen, argv[2]);
135
136         /* ... pronalazi se student sa tim nalogom... */
137         nadjen =
138             (Student *) bsearch(nalog_trazen, niz_studenata,
139                                 br_studenata, sizeof(Student),
140                                 &uporedi_bsearch);
141
142         if (nadjen == NULL)
143             printf("Nije nadjen!\n");
144         else
145             printf("%s %s %s %d\n",
146                   nadjen->nalog, nadjen->ime,
147                   nadjen->prezime, nadjen->poeni);
148     }
149
150     /* Zatvaranje datoteka */
151     fclose(in);
152     fclose(out);
153
154     exit(EXIT_SUCCESS);
155 }
```

4

Dinamičke strukture podataka

4.1 Liste

Zadatak 4.1 Napisati biblioteku za rad sa jednostruko povezanim listom čiji čvorovi sadrže cele brojeve.

- (a) Definisati strukturu `Cvor` kojom se predstavlja čvor liste. Čvor treba da sadrži ceo broj `vrednost` i pokazivač na sledeći čvor liste.
- (b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja kao argument dobija ceo broj, kreira nov čvor liste, inicijalizuje mu polja i vraća njegovu adresu.
- (c) Napisati funkciju `int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na početak liste, čija glava se nalazi na adresi `adresa_glave`.
- (d) Napisati funkciju `Cvor *pronadji_poslednji(Cvor * glava)` koja pronalazi poslednji čvor u listi.
- (e) Napisati funkciju `int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)` koja dodaje novi čvor sa vrednošću `broj` na kraj liste.
- (f) Napisati funkciju `Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)` koja vraća pokazivač na čvor u neopadajuće uređenoj listi iza kojeg bi trebalo dodati nov čvor sa vrednošću `broj`.

- (g) Napisati funkciju `int dodaj_iza(Cvor * tekuci, int broj)` koja iza čvora `tekuci` dodaje novi čvor sa vrednošću `broj`.
- (h) Napisati funkciju `int dodaj_sortirano(Cvor ** adresa_glave, int broj)` koja dodaje novi elemenat u neopadajuće uređenu listu tako da se očuva postojeće uređenje.
- (i) Napisati funkciju `void ispisi_listu(Cvor * glava)` koja ispisuje čvorove liste uokvirene zagradama `[,]` i međusobno razdvojene zapetama.
- (j) Napisati funkciju `Cvor *pretrazi_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor čija je vrednost jednaka argumentu `broj`. Vraća pokazivač na pronađeni čvor ili `NULL` ukoliko ga ne pronađe.
- (k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se prepostavlja 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 prepostavlja da se briše iz neopadajuće uređene liste.
- (n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ukoliko je sve bilo u redu, odnosno 1, ukoliko se dogodila greška prilikom alokacije memorije za nov čvor. **NAPOMENA:** *Sve funkcije za rad sa listom implementirati iterativno.*

Napisati programe koji koriste jednostruko povezanu listu za čuvanje elemenata koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu prekida se učitavanjem kraja ulaza (`EOF`). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste.

- (1) U programu se učitani celi brojevi dodaju na početak liste. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [3, 2]
14
Lista: [14, 3, 2]
5
Lista: [5, 14, 3, 2]
3
Lista: [3, 5, 14, 3, 2]
17
Lista: [17, 3, 5, 14, 3, 2]

Unesite broj koji se trazi: 5
Trazeni broj 5 je u listi!
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [35, 14, 23]

Unesite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!
```

- (2) U programu se učitani celi brojevi dodaju na kraj liste. Unosi se ceo broj čija se sva pojavljivanja u listi brišu. Na ekran se ispisuje sadržaj liste nakon brisanja.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 14, 3]
3
Lista: [2, 3, 14, 3, 3]
17
Lista: [2, 3, 14, 3, 3, 17]
3
Lista: [2, 3, 14, 3, 3, 17, 3]

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 14, 17]
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unosite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [23, 14]
35
Lista: [23, 14, 35]

Unesite broj koji se brise: 3
Lista nakon brisanja: [23, 14, 35]
```

- (3) U programu se učitani celi brojevi dodaju u listu tako da vrednosti budu uređene u neopadajućem poretku. Unosi se ceo broj koji se traži u unetoj listi i na ekran se ispisuje rezultat pretrage. Potom se unosi još jedan ceo broj čija se sva pojavljivanja u listi brišu i prikazuje se aktuelni sadržaj liste nakon brisanja. NAPOMENA: *Prilikom pretraživanja liste i brisanja čvora liste koristiti činjenicu da je lista uređena.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve: (za kraj CTRL+D)
2
Lista: [2]
3
Lista: [2, 3]
14
Lista: [2, 3, 14]
3
Lista: [2, 3, 3, 14]
3
Lista: [2, 3, 3, 3, 14]
5
Lista: [2, 3, 3, 3, 5, 14]

Unesite broj koji se trazi: 14
Trazenii broj 14 je u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [2, 5, 14]
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve: (za kraj CTRL+D)
23
Lista: [23]
14
Lista: [14, 23]
35
Lista: [14, 23, 35]

Unesite broj koji se trazi: 8
Broj 8 se ne nalazi u listi!

Unesite broj koji se brise: 3
Lista nakon brisanja: [14, 23, 35]
```

[Rešenje 4.1]

Zadatak 4.2 Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 4.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: Koristiti iste *main* programe i test primere iz zadatka 4.1.

[Rešenje 4.2]

Zadatak 4.3 Napisati program koji prebrojava pojavlivanja 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_pojavlivanja;
    char etiketa[20];
    struct _Element *sledeci;
} Element;
```

Test 1

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out datoteka.html

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke datoteka.html.
```

[Rešenje 4.3]

Zadatak 4.4 U datoteci se nalaze podaci o studentima. U svakom redu datoteke nalazi se indeks, ime i prezime studenta. Napisati program kome se preko argumenata komandne linije prosleđuje ime datoteke sa studentskim podacima koje program treba da pročita i smesti u listu. Nakon završenog učitavanja svih podataka o studentima, sa standardnog ulaza unose se, jedan po jedan, indeksi studenata koji se traže u učitanoj listi. Posle svakog unetog indeksa, program ispisuje poruku **da** ili **ne**, u zavisnosti od toga da li u listi postoji student sa unetim indeksom ili ne. Prekid unosa indeksa se vrši unošenjem karaktera za kraj ulaza (**EOF**). Poruke o greškama ispisivati na standardni izlaz za greške. **UPUTSTVO:** *Prepostaviti da je 10 karaktera dovoljno za zapis indeksa i da je 20 karaktera maksimalna dužina bilo imena bilo prezimena studenta.*

Primer 1

```
POKRETANJE: ./a.out studenti.txt
STUDENTI.TXT
123/2014 Marko Lukic
3/2014 Ana Sokic
43/2013 Jelena Ilic
41/2009 Marija Zaric
13/2010 Milovan Lazic

INTERAKCIJA SA PROGRAMOM:
3/2014 da: Ana Sokic
235/2008 ne
41/2009 da: Marija Zaric
```

Primer 2

```
POKRETANJE: ./a.out studenti.txt
DATOTEKA STUDENTI.TXT JE PRAZNA

INTERAKCIJA SA PROGRAMOM:
3/2014 ne
235/2008 ne
41/2009 ne
```

[Rešenje 4.4]

Zadatak 4.5 Data je datoteka **brojevi.txt** koja sadrži cele brojeve.

- Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.
- 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

IZLAC:
REZULTAT.TXT
12 15 16
```

Test 2

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAC ZA GREŠKU:
Greska prilikom otvaranja
datoteke brojevi.txt.
```

Test 3

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAC:
REZULTAT.TXT
Rezultat.txt ce biti prazna.
```

Zadatak 4.6 Napisati program koji objedinjuje dve sortirane liste u jednu sortiranu listu. Funkcija ne treba da kreira nove, već da samo preraspodeli postojeće čvorove. Prva lista se učitava iz datoteke čije ime se zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz. **NAPOMENA:** Koristiti biblioteku za rad sa listama celih brojeva iz zadatka 4.1.

Test 1

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out dat1.txt dat2.txt
DAT1.TXT
2 4 6 10 15

DATOTEKA DAT2.TXT NE POSTOJI.

IZLAZ ZA GREŠKU:
Greska prilikom otvaranja datoteke
dat2.txt.
```

Test 3

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out dat1.txt
IZLAZ ZA GREŠKU:
Program se poziva sa:
./a.out dat1.txt dat2.txt!
```

[Rešenje 4.6]

Zadatak 4.7 Date su dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od ovih listi formira novu listu L koja sadrži naizmenično raspoređene čvorove listi L1 i L2: prvi čvor iz L1, prvi čvor iz L2, drugi čvor L1, drugi čvor L2, itd. Ne formirati nove čvorove, već samo postojeće rasporediti u jednu listu. Prva lista se učitava iz datoteke čije se ime zadaje kao prvi argument komandne linije, a druga iz datoteke čije se ime zadaje kao drugi argument komandne linije. Rezultujuću listu ispisati na standardni izlaz.

NAPOMENA: *Iskoristiti testove 2 - 6 za zadatak 4.6.*

Test 1

```
POKRETANJE: ./a.out dat1.txt dat2.txt
DAT1.TXT
2 4 6 10 15

DAT2.TXT
5 6 11 12 14 16

IZLAZ:
2 5 4 6 6 11 10 12 15 14 16
```

Zadatak 4.8 Sadržaj datoteke je aritmetički izraz koji može sadržati zgrade {}, [i (. Napisati program koji učitava sadržaj datoteke **izraz.txt** i korišćenjem

steka utvrđuje da li su zgrade u aritmetičkom izrazu dobro uparene. Program stampa odgovarajuću poruku na standardni izlaz.

Test 1

```
IZRAZ.TXT
{[23 + 5344] * (24 - 234)} - 23
IZLAZ:
Zgrade su ispravno uparene.
```

Test 2

```
IZRAZ.TXT
{[23 + 5] * (9 * 2)} - {23}
IZLAZ:
Zgrade su ispravno uparene.
```

Test 3

```
IZRAZ.TXT
{[2 + 54] / (24 * 87)} + (234 + 23)
IZLAZ:
Zgrade nisu ispravno uparene.
```

Test 4

```
IZRAZ.TXT
{(2 - 14) / (23 + 11)} * (2 + 13)
IZLAZ:
Zgrade nisu ispravno uparene.
```

Test 5

```
DATOTEKA IZRAZ.TXT JE PRAZNA
IZLAZ:
Zgrade su ispravno uparene.
```

Test 6

```
DATOTEKA IZRAZ.TXT NE POSTOJI.
IZLAZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke izraz.txt!
```

[Rešenje 4.8]

Zadatak 4.9 Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije. Poruke o greškama ispisivati na standardni izlaz za greške. UPUTSTVO: Za rešavanje problema koristiti stek implementiran preko liste čiji čvorovi sadrže HTML etikete.

Test 1

```
POKRETANJE: ./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

```
POKRETANJE: ./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

```
POKRETANJE: ./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

```
POKRETANJE: ./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

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKU:
Greska prilikom otvaranja
datoteke datoteka.html.
```

Test 6

```
POKRETANJE: ./a.out datoteka.html
DATOTEKA.HTML JE PRAZNA

IZLAZ:
Etikete su pravilno uparene!
```

[Rešenje 4.9]

Zadatak 4.10 Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve korisničke JMBG brojeve (niske koje sadrže po 13 karaktera) i zahteve (niska koja sadrži najviše 999 karaktera). Prijem zahteva korisnika se prekida unošenjem karaktera za kraj ulaza, (EOF). Službenik redom pregleda zahteve i odlučuje da li zahtev obrađuje odmah ili kasnije. Program mu postavlja pitanje *Da li korisnika vracate na kraj reda?* i ukoliko on da odgovor *Da*, korisnik se stavlja na kraj reda, čime se obrada njegovog zahteva odlaže. Ukoliko odgovor nije *Da*, službenik obrađuje zahtev i podatke o korisniku dopisuje na kraj datoteke *izvestaj.txt*. Ova datoteka, za svaki obrađen zahtev, sadrži JMBG i zahtev usluženog korisnika. Posle svakog *petog* usluženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu. **UPUTSTVO:** Za čuvanje korisničkih zahteva koristiti red implementirani korišćenjem listi.

Primer 1

```
|| INTERAKCIJA SA PROGRAMOM:  
|| Sluzbenik evidentira korisnicke zahteve:  
|| Novi zahtev [CTRL+D za kraj]  
|| JMBG: 1234567890123  
|| Opis problema: Divaranje racuna  
  
Novi zahtev [CTRL+D za kraj]  
JMBG: 2345678901234  
Opis problema: Podizanje novca  
  
Novi zahtev [CTRL+D za kraj]  
JMBG: 3456789012345  
Opis problema: Reklamacija  
  
Novi zahtev [CTRL+D za kraj]  
JMBG:  
  
Sledeci je korisnik sa JMBG: 1234567890123  
i zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Da  
  
Sledeci je korisnik sa JMBG: 2345678901234  
i zahtevom: Podizanje novca  
Da li ga vracate na kraj reda? [Da/Ne] Ne  
  
Sledeci je korisnik sa JMBG: 3456789012345  
i zahtevom: Reklamacija  
Da li ga vracate na kraj reda? [Da/Ne] Da  
  
Sledeci je korisnik sa JMBG: 1234567890123  
i zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Da  
  
Sledeci je korisnik sa JMBG: 3456789012345  
i zahtevom: Reklamacija  
Da li ga vracate na kraj reda? [Da/Ne] Ne  
  
Da li je kraj smene? [Da/Ne] Ne  
  
Sledeci je korisnik sa JMBG: 1234567890123  
i zahtevom: Otvaranje racuna  
Da li ga vracate na kraj reda? [Da/Ne] Ne  
  
IZVESTAJ.TXT  
JMBG: 2345678901234 Zahtev: Podizanje novca  
JMBG: 3456789012345 Zahtev: Reklamacija  
JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

[Rešenje 4.10]

Zadatak 4.11 Napisati biblioteku za rad sa dvostruko povezanim 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 **`

`adresa_kraja, Cvor * tekuci`) koja briše čvor na koji pokazuje pokazivač `tekuci` iz liste čiji se pokazivač na čvor koji je glava liste nalazi na adresi `adresa_glave` i poslednji čvor liste na adresi `adresa_kraja`.

- (b) Napisati funkciju `void ispisi_listu_unazad(Cvor * kraj)` koja ispisuje sadržaj liste od poslednjeg čvora ka glavi liste.

Sve funkcije za rad sa listom implementirati iterativno. Zbog efikasnog izvršavanja operacija dodavanja na kraj liste i ispisivanja liste unazad treba, pored pokazivača na glavu liste, čuvati i pokazivač na poslednji čvor liste. NAPOMENA: *Funkcije testirati koristeći test primere iz zadatka 4.1*

[Rešenje 4.11]

Zadatak 4.12 Grupa od n plesača na kostimima ima brojeve od 1 do n . Plesači najpre formiraju krug tako da broevi sa njihovih kostima rastu u smeru kazaljke na satu. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na svoj kuk i tako redom. Plesač sa brojem n svoju levu ruku spušta na rame plesača sa brojem 1, a desnu na svoj kuk i tako zatvara krug. Svoju plesnu tačku izvode tako što iz formiranog kruga najpre izlazi k -ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug tako što $k - 1$ -vi stavljaju ruku na rame $k + 1$ -og i zatvara 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 poveznu kružnu listu.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 5 3	ULAZ: 8 4	ULAZ: 3 8
IZLAZ: 3 1 5 2 4	IZLAZ: 4 8 5 2 1 3 7 6	IZLAZ ZA GREŠKU: n mora biti uvek veće od k, a 3 < 8!

Zadatak 4.13 Grupa od n plesača na kostimima ima brojeve od 1 do n . Plesači najpre formiraju krug tako da broevi sa njihovih kostima rastu u smeru kazaljke na satu. Svaki plesač levu ruku stavlja na rame plesača sa sledećim većim brojem, a desnu na rame plesača sa prvim manjim brojem. Plesač sa brojem 1 stavlja levu ruku na rame plesača sa brojem 2, a desnu na rame plesača sa brojem n . Plesač sa brojem n svoju desnu ruku spušta na rame plesača sa brojem $n - 1$,

a levu na rame plesača sa brojem 1 i tako zatvara krug. Plesači izvode svoju plesnu tačku tako što iz formiranog kruga 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.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
<pre> ULAZ: 5 3</pre>	<pre> ULAZ: 8 4</pre>	<pre> ULAZ: 5 8</pre>
<pre> IZLAZ: 3 5 4 2 1</pre>	<pre> IZLAZ: 4 8 5 7 6 3 2 1</pre>	<pre> IZLAZ ZA GREŠKU: n mora biti uvek vece od k, a 5 < 8!</pre>

4.2 Stabla

Zadatak 4.14 Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

- Definisati strukturu **Cvor** kojom se opisuje čvor stabla, a koja sadrži ceo broj **broj** i pokazivače **levo** i **desno** redom na levo i desno podstablo.
- Napisati funkciju **Cvor *napravi_cvor(int broj)** koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem **broj**.
- Napisati funkciju **int dodaj_u_stablo(Cvor ** adresa_korena, int broj)** koja u stablu na koje pokazuje argument **adresa_korena** dodaje ceo broj **broj**. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.
- 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.
- Napisati funkciju **Cvor *pronadji_najmanji(Cvor * koren)** koja pronađe čvor koji sadrži najmanju vrednost u stablu sa korenom **koren**.

- (f) Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronađe čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.
- (g) Napisati funkciju `void obrisi_element(Cvor ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.
- (h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor * koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.
- (i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor * koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.
- (j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor * koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.
- (k) Napisati funkciju `void osloboodi_stablo(Cvor ** adresa_korena)` koja oslobođa memoriju zauzetu stablom na koje pokazuje argument `adresa_korena`.

Korišćenjem kreirane biblioteke, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuće stablo: 1 2 9 18 32
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuće stablo: -3 -2 6 8 13 24
```

[Rešenje 4.14]

Zadatak 4.15 Napisati program koji izračunava i na standardni izlaz ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova.

Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku Nedostaje ime ulazne datoteke!. Može se prepostaviti da dužina reči neće biti veća od 50 karaktera.

Test 1

```
POKRETANJE: ./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

```
POKRETANJE: ./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

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

DATOTEKA ULAZ.TXT NE POSTOJI

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

Test 4

```
POKRETANJE: ./a.out

IZLAZ:
Nedostaje ime ulazne datoteke!
```

[Rešenje 4.15]

Zadatak 4.16 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. Pera Peric 064/123-4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči KRAJ, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se prepostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

Primer 1

```
IMENIK.TXT
Pera Peric 011/3240-987
Marko Maric 064/1234-987
Mirko Maric 011/589-333
Sanja Savkovic 063/321-098
Zika Zikic 021/759-858
```

```
INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik.txt
Unesite ime i prezime: Pera Peric
Broj je: 011/3240-987
Unesite ime i prezime: Marko Markovic
Broj nije u imeniku!
Unesite ime i prezime: KRAJ
```

Primer 2

```
DATOTEKA IMENIKI.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
Unesite ime datoteke: imenik1.txt
Greska: Neuspesno otvaranje datoteke
imenik1.txt.
```

[Rešenje 4.16]

Zadatak 4.17 U datoteci *prijemni.txt* nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niska od najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

Test 1

```
PRIJEMNI.TXT
Marko Markovic 45.4 12.3 11
Milan Jevremovic 35.2 1.3 9
Maja Agic 60 19 20
Nadica Zec 54.2 10 15.8
Jovana Milic 23.3 2 5.6

IZLAC:
1. Maja Agic 60.0 19.0 20.0 99.0
2. Nadica Zec 54.2 10.0 15.8 80.0
3. Marko Markovic 45.4 12.3 11.0 68.7
4. Milan Jevremovic 35.2 1.3 9.0 45.5
-----
5. Jovana Milic 23.3 2.0 5.6 30.9
```

Test 2

```
DATOTEKA PRIJEMNI.TXT NE POSTOJI
IZLAC:
Greska: Neuspesno otvaranje datoteke
prijemni.txt.
```

[Rešenje 4.17]

*** Zadatak 4.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata **Ime Prezime DD.MM.** i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronađi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumi - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu **DD.MM.** Takođe, može se pretpostaviti da će ime i prezime osobe biti kraće od 50 karaktera.

Primer 1

```
POKRETANJE: ./a.out rodjendani.txt
RODJENDANI.TXT
Marko Markovic 12.12.
Milan Jevremovic 04.06.
Maja Agic 23.04.
Nadica Zec 01.01.
Jovana Milic 05.05.
```

```
INTERAKCIJA SA PROGRAMOM:
Unesite datum: 23.04.
Slavljenik: Maja Agic
Unesite datum: 01.01.
Slavljenik: Nadica Zec
Unesite datum: 01.05.
Slavljenik: Jovana Milic 05.05.
Unesite datum: 20.12.
Slavljenik: Nadica Zec 01.01.
Unesite datum:
```

Primer 2

```
POKRETANJE: ./a.out rodjendani.txt
DATOTEKA RODJENDANI.TXT NE POSTOJI
INTERAKCIJA SA PROGRAMOM:
Greska: Neuspesno otvaranje datoteke
rodjendani.txt.
```

[Rešenje 4.18]

Zadatak 4.19 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju **int identitet(Cvor * koren1, Cvor * koren2)** koja proverava da li su binarna stabla **koren1** i **koren2** koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo:
10 5 15 3 2 4 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla jesu identicna.
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Prvo stablo:
10 5 15 4 3 2 30 12 14 13 0
Drugo stablo:
10 15 5 3 4 2 12 14 13 30 0
Stabla nisu identicna.
```

[Rešenje 4.19]

*** Zadatak 4.20** Napisati program za rad sa skupovima u kojem se skupovi predstavljaju pomoću binarnih pretraživačkih stabala. Program za dva skupa čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku skupova. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Prvi skup: 1 7 8 9 2 2
Drugi skup: 3 9 6 11 1
Unija: 1 1 2 2 3 6 7 8 9 9 11
Presek: 1 9
Razlika: 2 2 7 8
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Prvi skup: 11 2 7 5
Drugi skup: 4 3 3 7
Unija: 2 3 3 4 5 7 7 11
Presek: 7
Razlika: 2 5 11
```

[Rešenje 4.20]

Zadatak 4.21 Napisati funkciju void sortiraj(int a[], int n) koja sortira niz celih brojeva a dimenzije n korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj n manji od 50 i niz a celih brojeva dužine n, poziva funkciju sortiraj i rezultat ispisuje na standardni izlaz. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
n: 7
a: 1 11 8 6 37 25 30
1 6 8 11 25 30 37
```

Primer 2

```
INTERAKCIJA SA PROGRAMOM:
n: 55
Greska: pogresna dimenzija niza!
```

[Rešenje 4.21]

Zadatak 4.22 Dato je binarno pretraživačko stablo celih brojeva.

- (a) Napisati funkciju koja izračunava broj čvorova stabla.
- (b) Napisati funkciju koja izračunava broj listova stabla.
- (c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.
- (d) Napisati funkciju koja izračunava zbir čvorova stabla.
- (e) Napisati funkciju koja izračunava najveći element stabla.
- (f) Napisati funkciju koja izračunava dubinu stabla.
- (g) Napisati funkciju koja izračunava broj čvorova na i -tom nivou stabla.
- (h) Napisati funkciju koja ispisuje sve elemente na i -tom nivou stabla.
- (i) Napisati funkciju koja izračunava maksimalnu vrednost na i -tom nivou stabla.
- (j) Napisati funkciju koja izračunava zbir čvorova na i -tom nivou stabla.
- (k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x .

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

Test 1

```
POKRETANJE: ./a.out 2 15
ULAZ:
10 5 15 3 2 4 30 12 14 13

IZLAZ:
Broj čvorova: 10
Broj listova: 4
Pozitivni listovi: 2 4 13 30
Zbir čvorova: 108
Najveći element: 30
Dubina stabla: 5
Broj čvorova na 2. nivou: 3
Elementi na 2. nivou: 3 12 30
Maksimalni element na 2. nivou: 30
Zbir elemenata na 2. nivou: 45
Zbir elemenata manjih ili jednakih od 15: 78
```

Test 2

```
POKRETANJE: ./a.out 3 31
ULAZ:
24 53 61 9 7 55 20 16

IZLAZ:
Broj čvorova: 8
Broj listova: 3
Pozitivni listovi: 7 16 55
Zbir čvorova: 245
Najveći element: 61
Dubina stabla: 4
Broj čvorova na 3. nivou: 2
Elementi na 3. nivou: 16 55
Maksimalni element na 3. nivou: 55
Zbir elemenata na 3. nivou: 71
Zbir elemenata manjih ili jednakih od 31: 76
```

[Rešenje 4.22]

Zadatak 4.23 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima. Elementi stabla se učitavaju sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1	Test 2	Test 3
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	ULAZ: 6 11 8 3 -2 IZLAZ: 0.nivo: 6 1.nivo: 3 11 2.nivo: -2 8	ULAZ: 24 53 61 9 7 55 20 16 IZLAZ: 0.nivo: 24 1.nivo: 9 53 2.nivo: 7 20 61 3.nivo: 16 55

[Rešenje 4.23]

* **Zadatak 4.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

Primer 1	Primer 2
INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 14 30 1 0 Stabla su slična kao u ogledalu.	INTERAKCIJA SA PROGRAMOM: Prvo stablo: 11 20 5 3 0 Drugo stablo: 8 20 15 0 Stabla nisu slična kao u ogledalu.

Zadatak 4.25 AVL-stablo je binarno pretraživačko stablo kod kojeg apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor * koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza. NAPOMENA: *Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.*

Test 1

```
|| ULAZ:
  ||| 10 5 15 2 11 16 1 13
  || IZLAZ:
  ||| 7
```

Test 2

```
|| ULAZ:
  ||| 16 30 40 24 10 18 45 22
  || IZLAZ:
  ||| 6
```

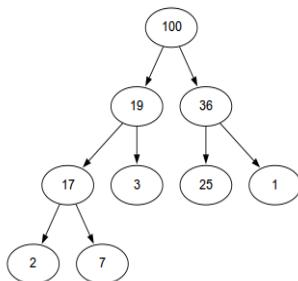
[Rešenje 4.25]

Zadatak 4.26 Binarno stablo celih pozitivnih brojeva se naziva *heap* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor * koren)` koja proverava da li je dato binarno stablo celih brojeva heap. Napisati zatim i glavni program koji kreira stablo zadato slikom 4.1, poziva funkciju `heap` i ispisuje rezultat na standardni izlaz. NAPOMENA: Koristiti biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

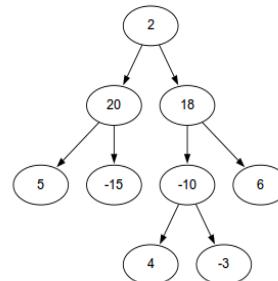
Test 1

```
|| IZLAZ:
  ||| Zadato stablo je heap!
```

[Rešenje 4.26]



Slika 4.1: Zadatak 4.26



Slika 4.2: Zadatak 4.27

Zadatak 4.27 Dato je binarno stablo celih brojeva.

- Napisati funkciju koja pronađe čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

- (b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.
- (c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.
- (d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 4.2 i rezultat ispisuje na standardni izlaz.

Test 1

```
|| IZLAZ:  
|| Vrednost u cvoru sa maksimalnim desnim zbirom: 18  
|| Vrednost u cvoru sa minimalnim levim zbirom: 18  
|| 2 18 -10 4  
|| 2 20 -15
```

4.3 Rešenja

Rešenje 4.1

lista.h

```
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 */  
6 typedef struct cvor {  
    int vrednost;  
    struct cvor *sledeci;  
} Cvor;  
10  
12 /* Funkcija kreira cvor, vrednost novog cvora inicializuje 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  
18 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste  
   ciji se pokazivac glava nalazi na adresi adresa_glave. */  
void oslobodi_listu(Cvor ** adresa_glave);  
20  
22 /* Funkcija dodaje broj 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 dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
38 int dodaj_iza(Cvor * tekuci, int broj);

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 sadran 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
   neopadajuće sortirana. Vraca pokazivac na cvor liste u kome je
   sadran trazeni broj ili NULL u slucaju da takav cvor ne postoji.
   */
50 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

52 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
   obrije stara glava. */
54 void obrisi_cvor(Cvor ** adresa_glave, int broj);

56 /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
   neopadajuće. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrije stara glava liste. */
58 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

60 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
   liste, razdvojene zapetama i uokvirene zagradama. */
62 void ispisi_listu(Cvor * glava);

64 #endif

```

lista.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"

5 Cvor *napravi_cvor(int broj)
{
7     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
       alokacije */
9     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10    if (novi == NULL)
11        return NULL;

13    /* Inicijalizacija polja strukture */
14    novi->vrednost = broj;
15    novi->sledeci = NULL;

17    /* Vraca se adresa novog cvora */
18    return novi;
19}

21 void osloboodi_listu(Cvor ** adresu_glave)
{
23    Cvor *pomocni = NULL;

25    /* Ako lista nije prazna, onda treba oslobooditi memoriju */
26    while (*adresa_glave != NULL) {
27        /* Potrebno je prvo zapamtitи adresu sledeceg cvora i onda
           oslobooditi cvor koji predstavlja glavu liste */
28        pomocni = (*adresa_glave)->sledeci;
29        free(*adresa_glave);

31        /* Sledeci cvor je nova glava liste */
32        *adresa_glave = pomocni;
33    }
35}

37 int dodaj_na_pocetak_liste(Cvor ** adresu_glave, int broj)
{
38    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39    Cvor *novi = napravi_cvor(broj);
40    if (novi == NULL)
41        return 1;

43    /* Novi cvor se uvezuje na pocetak i postaje nova glava liste */
44    novi->sledeci = *adresa_glave;
45    *adresa_glave = novi;

47    /* Vraca se indikator uspesnog dodavanja */
48    return 0;
49}

```

```

    }

51 Cvor *pronadji_poslednji(Cvor * glava)
52 {
53     /* U praznoj listi nema cvorova pa se vraca NULL */
54     if (glava == NULL)
55         return NULL;

56     /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
57     glava se pomera na sledeci cvor. Nakon izlaska iz petlje, glava
58     ce pokazivati na cvor liste koji nema sledbenika, tj. na
59     poslednji cvor liste pa se vraca vrednost pokazivaca glava.

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     return glava;
65 }

66 int dodaj_na_kraj_liste(Cvor ** adresa_glage, int broj)
67 {
68     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
69     Cvor *novi = napravi_cvor(broj);
70     if (novi == NULL)
71         return 1;

72     /* Ako je lista prazna */
73     if (*adresa_glage == NULL) {
74         /* Glava nove liste je upravo novi cvor */
75         *adresa_glage = novi;
76     } else {
77         /* Ako lista nije prazna, pronalazi se poslednji cvor i novi cvor
78         se dodaje na kraj liste kao sledbenik poslednjeg */
79         Cvor *poslednji = pronadji_poslednji(*adresa_glage);
80         poslednji->sledeci = novi;
81     }

82     /* Vraca se indikator uspesnog dodavanja */
83     return 0;
84 }

85 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
86 {
87     /* U praznoj listi nema takvog mesta i vraca se NULL */
88     if (glava == NULL)
89         return NULL;

90     /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
91     pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
92     vecu ili jednaku vrednosti novog cvora.
93 
```

```

103     Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
105     provera da li se doslo do poslednjeg cvora liste pre nego sto se
106     proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
107     sledeci ne postoji, pa ni njegova vrednost. */
108     while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
109         glava = glava->sledeci;
110
111     /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
112      poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci ima
113      vrednost vecu od broj. */
114     return glava;
115 }
116
117 int dodaj_iza(Cvor * tekuci, int broj)
118 {
119     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
120     Cvor *novi = napravi_cvor(broj);
121     if (novi == NULL)
122         return 1;
123
124     /* Novi cvor se dodaje iza tekuceg cvora. */
125     novi->sledeci = tekuci->sledeci;
126     tekuci->sledeci = novi;
127
128     /* Vraca se indikator uspesnog dodavanja */
129     return 0;
130 }
131
132 int dodaj_sortirano(Cvor ** adresa_glove, int broj)
133 {
134     /* Ako je lista prazna */
135     if (*adresa_glove == NULL) {
136         /* Glava nove liste je novi cvor */
137
138         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
139         Cvor *novi = napravi_cvor(broj);
140         if (novi == NULL)
141             return 1;
142
143         *adresa_glove = novi;
144
145         /* Vraca se indikator uspesnog dodavanja */
146         return 0;
147     }
148
149     /* Inace... */
150
151     /* Ako je broj manji ili jednak vrednosti u glavi liste, onda ga
152      treba dodati na pocetak liste */
153     if ((*adresa_glove)->vrednost >= broj) {
154         return dodaj_na_pocetak_liste(adresa_glove, broj);

```

```

155    }
156
157    /* U slučaju da je glava liste cvor sa vrednoscu manjom od broj,
158     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
159    Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glove, broj);
160    return dodaj_iza(pomocni, broj);
161}
162
163Cvor *pretrazi_listu(Cvor * glava, int broj)
164{
165    /* Obilaze se cvorovi liste */
166    for (; glava != NULL; glava = glava->sledeci)
167        /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
168         se obustavlja */
169        if (glava->vrednost == broj)
170            return glava;
171
172    /* Nema trazenog broja u listi i vraca se NULL */
173    return NULL;
174}
175
176Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
177{
178    /* Obilaze se cvorovi liste */
179    /* U uslovu ostanka u petlji, bitan je redosled provera u
180     konjunkciji */
181    while (glava != NULL && glava->vrednost < broj)
182        glava = glava->sledeci;
183
184    /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
185     glava->vrednost veca od trazenog broja i tada treba vratiti
186     NULL, jer tražen broj nije nadjen medju manjim brojevima pri
187     pocetku sortirane liste, pa se moze zakljuciti da ga nema u
188     listi. Drugi nacini, tako sto se doslo do kraja liste i glava je
189     NULL ili tako sto je glava->vrednost == broj. U oba poslednja
190     nacina treba vratiti pokazivac glava bilo da je NULL ili
191     pokazivac na konkretan cvor. */
192    if (glava->vrednost > broj)
193        return NULL;
194    else
195        return glava;
196}
197
198void obrisi_cvor(Cvor ** adres_glove, int broj)
199{
200    Cvor *tekuci = NULL;
201    Cvor *pomocni = NULL;
202
203    /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
204     broju i azurira se pokazivac na glavu liste */
205    while (*adres_glove != NULL && (*adres_glove)->vrednost == broj)
206    {

```

```

205     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
206     adresi adresa_glave */
207     pomocni = (*adresa_glave)->sledeci;
208     free(*adresa_glave);
209     *adresa_glave = pomocni;
210 }
211
212 /* Ako je nakon ovog brisanja lista ostala prazna, izlazi se iz
213 funkcije */
214 if (*adresa_glave == NULL)
215     return;
216
217 /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
218 pokazuje na cvor cija je vrednost razlicita od trazenog broja.
219 Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
220 sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
221 ako je jednak, a ako je razlicit, prelazi se na sledeci cvor.
222 Ovaj postupak se ponavlja dok se ne dodje do poslednjeg cvora.
223 */
224 tekuci = *adresa_glave;
225 while (tekuci->sledeci != NULL)
226     if (tekuci->sledeci->vrednost == broj) {
227         /* tekuci->sledeci treba obrisati, zbog toga se njegova adresa
228         prvo cuva u promenljivoj pomocni. */
229         pomocni = tekuci->sledeci;
230         /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
231             njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
232             sledeci od cvora koji se brise. */
233         tekuci->sledeci = pomocni->sledeci;
234         /* Sada treba osloboditi cvor sa vrednoscu broj. */
235         free(pomocni);
236     } else {
237         /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
238             pomera na sledeci. */
239         tekuci = tekuci->sledeci;
240     }
241 }
242
243 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
244 {
245     Cvor *tekuci = NULL;
246     Cvor *pomocni = NULL;
247
248     /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
249     broju i azurira se pokazivac na glavu liste. */
250     while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
251     {
252         /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
253             adresi adresa_glave. */
254         pomocni = (*adresa_glave)->sledeci;
255         free(*adresa_glave);

```

```

255     *adresa_glave = pomocni;
256 }
257
258 /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
259    prekida. Isto se radi i ukoliko glava liste sadrzi vrednost koja
260    je veca od broja, jer kako je lista sortirana rastuce nema
261    potrebe broj traziti u repu liste. */
262 if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
263     return;
264
265 /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci pokazuje
266    na cvor cija vrednost je manja od trazenog broja, kao i svim
267    cvorovima levo od njega. Cvor se brise ako je jednak, ili, ako
268    je razlicit, prelazi se na sledeci cvor. Ovaj postupak se
269    ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
270    cija vrednost je veca od trazenog broja. */
271 tekuci = *adresa_glave;
272 while (tekuci->sledeci != NULL && tekuci->sledeci->vrednost <= broj
273     )
274     if (tekuci->sledeci->vrednost == broj) {
275         pomocni = tekuci->sledeci;
276         tekuci->sledeci = tekuci->sledeci->sledeci;
277         free(pomocni);
278     } else {
279         /* Ne treba brisati sledeceg od tekuceg jer je manji od
280            trazenog i tekuci se pomera na sledeci cvor. */
281         tekuci = tekuci->sledeci;
282     }
283     return;
284 }
285 void ispisi_listu(Cvor * glava)
286 {
287     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
288        jer se lista nece menjati */
289     putchar('[');
290     /* Unutar zagradica ispisuju se vrednosti u cvorovima liste od
291        pocetka prema kraju liste. */
292     for (; glava != NULL; glava = glava->sledeci) {
293         printf("%d", glava->vrednost);
294         if (glava->sledeci != NULL)
295             printf(", ");
296     }
297     printf("]\n");
298 }
```

main_a.c

```

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

```

4  /* 1) Glavni program */
6  int main()
{
8  /* Lista je prazna na pocetku */
10 Cvor *glava = NULL;
12 Cvor *trazeni = NULL;
14 int broj;
16
18 /* Testiranje dodavanja novog broja na pocetak liste */
20 printf("Unesite brojeve: (za kraj CTRL+D)\n");
22 while (scanf("%d", &broj) > 0) {
24     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
26         memorije za nov cvor. Memoriju alociranu za cvorove liste
28         treba oslobođiti pre napustanja programa. */
30     if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
32         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
34         oslobodi_listu(&glava);
36         exit(EXIT_FAILURE);
38     }
39     printf("\tLista: ");
40     ispisi_listu(glava);
42 }
44
46 /* Testiranje funkcije za pretragu liste */
48 printf("\nUnesite broj koji se trazi: ");
50 scanf("%d", &broj);
52
54 trazeni = pretrazi_listu(glava, broj);
56 if (trazeni == NULL)
58     printf("Broj %d se ne nalazi u listi!\n", broj);
60 else
62     printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
64
66 /* Oslobadja se memorija zauzeta listom */
68 oslobodi_listu(&glava);
70
72 exit(EXIT_SUCCESS);
74 }

```

main_b.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"
4
5 /* 2) Glavni program */
6 int main()
{
8  /* Lista je prazna na pocetku */
10 Cvor *glava = NULL;
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
102
104
106
108
110
112
114
116
118
120
122
124
126
128
130
132
134
136
138
140
142
144
146
148
150
152
154
156
158
160
162
164
166
168
170
172
174
176
178
180
182
184
186
188
190
192
194
196
198
200
202
204
206
208
210
212
214
216
218
220
222
224
226
228
230
232
234
236
238
240
242
244
246
248
250
252
254
256
258
260
262
264
266
268
270
272
274
276
278
280
282
284
286
288
290
292
294
296
298
300
302
304
306
308
310
312
314
316
318
320
322
324
326
328
330
332
334
336
338
340
342
344
346
348
350
352
354
356
358
360
362
364
366
368
370
372
374
376
378
380
382
384
386
388
390
392
394
396
398
400
402
404
406
408
410
412
414
416
418
420
422
424
426
428
430
432
434
436
438
440
442
444
446
448
450
452
454
456
458
460
462
464
466
468
470
472
474
476
478
480
482
484
486
488
490
492
494
496
498
500
502
504
506
508
510
512
514
516
518
520
522
524
526
528
530
532
534
536
538
540
542
544
546
548
550
552
554
556
558
560
562
564
566
568
570
572
574
576
578
580
582
584
586
588
590
592
594
596
598
600
602
604
606
608
610
612
614
616
618
620
622
624
626
628
630
632
634
636
638
640
642
644
646
648
650
652
654
656
658
660
662
664
666
668
670
672
674
676
678
680
682
684
686
688
690
692
694
696
698
700
702
704
706
708
710
712
714
716
718
720
722
724
726
728
730
732
734
736
738
740
742
744
746
748
750
752
754
756
758
760
762
764
766
768
770
772
774
776
778
780
782
784
786
788
790
792
794
796
798
800
802
804
806
808
810
812
814
816
818
820
822
824
826
828
830
832
834
836
838
840
842
844
846
848
850
852
854
856
858
860
862
864
866
868
870
872
874
876
878
880
882
884
886
888
890
892
894
896
898
900
902
904
906
908
910
912
914
916
918
920
922
924
926
928
930
932
934
936
938
940
942
944
946
948
950
952
954
956
958
960
962
964
966
968
970
972
974
976
978
980
982
984
986
988
990
992
994
996
998
999

```

```

10 int broj;

12 /* Testira se funkcija za dodavanja novog broja na kraj liste */
13 printf("Unesite brojeve: (za kraj CTRL+D)\n");
14 while (scanf("%d", &broj) > 0) {
15     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
16        memorije za nov cvor. Memoriju alociranu za cvorove liste
17        treba oslobođiti pre napustanja programa. */
18     if (dodaj_na_kraj_liste(&glava, broj) == 1) {
19         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
20         osloboди_listu(&glava);
21         exit(EXIT_FAILURE);
22     }
23     printf("\tLista: ");
24     ispisi_listu(glava);
25 }

26 /* Testira se funkcije kojom se brise cvor liste */
27 printf("\nUnesite broj koji se brise: ");
28 scanf("%d", &broj);

29 /* Brisu se cvorovi iz liste cije je polje vrednost jednako broju
30    procitanom sa ulaza */
31 obrisi_cvor(&glava, broj);

32 printf("Lista nakon brisanja: ");
33 ispisi_listu(glava);

34 /* Oslobadja se memorija zauzeta listom */
35 osloboди_listu(&glava);

36 exit(EXIT_SUCCESS);
37 }
```

main_c.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"

4 /* 3) Glavni program */
5 int main()
6 {
7     /* Lista je prazna na pocetku */
8     Cvor *glava = NULL;
9     Cvor *trazenii = NULL;
10    int broj;

11    /* Testira se funkcija za dodavanje vrednosti u listu tako da bude
12       uređena neopadajuće */
13    printf("Unosite brojeve (za kraj CTRL+D)\n");
```

```

16  while (scanf("%d", &broj) > 0) {
18      /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
         memorije za nov cvor. Memoriju alociranu za cvorove liste
         treba oslobođiti pre napustanja programa. */
20      if (dodaj_sortirano(&glava, broj) == 1) {
22          fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
23          oslobođi_listu(&glava);
24          exit(EXIT_FAILURE);
25      }
26      printf("\tLista: ");
27      ispisi_listu(glava);
28  }

29  /* Testira se funkcija za pretragu liste */
30  printf("\nUnesite broj koji se trazi: ");
31  scanf("%d", &broj);

32  traženi = pretrazi_listu(glava, broj);
33  if (traženi == NULL)
34      printf("Broj %d se ne nalazi u listi!\n", broj);
35  else
36      printf("Traženi broj %d je u listi!\n", traženi->vrednost);

37  /* Testira se funkcija kojom se brise cvor liste */
38  printf("\nUnesite broj koji se brise: ");
39  scanf("%d", &broj);

40  /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
     procitanom sa ulaza */
41  obrisi_cvor_sortirane_liste(&glava, broj);

42  printf("Lista nakon brisanja: ");
43  ispisi_listu(glava);

44  /* Oslobadja se memorija zauzeta listom */
45  oslobođi_listu(&glava);

46  exit(EXIT_SUCCESS);
47 }
```

Rešenje 4.2

lista.h

```

1 #ifndef _LISTA_H_
2 #define _LISTA_H_

4 /* Struktura kojom je predstavljen cvor liste sadrži celobrojni
   podatak vrednost i pokazivac na sledeći cvor liste. */
6 typedef struct cvor {
```

```

8     int vrednost;
9     struct cvor *sledeci;
10    } Cvor;
11
12   /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na broj,
13      dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
14      na novokreirani cvor ili NULL ako alokacija nije bila uspesna.*/
15 Cvor *napravi_cvor(int broj);
16
17   /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
18      ciji se pokazivac glava nalazi na adresi adresa_glave. */
19 void osloboodi_listu(Cvor ** adresa_glave);
20
21   /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
22      bilo greske pri alokaciji memorije, inace vraca 0. */
23 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);
24
25   /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
26      pri alokaciji memorije, inace vraca 0. */
27 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);
28
29   /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova lista
30      ostane sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
31      memorije, inace vraca 0. */
32 int dodaj_sortirano(Cvor ** adresa_glave, int broj);
33
34   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
35      Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
36      NULL u slucaju da takav cvor ne postoji u listi. */
37 Cvor *pretrazi_listu(Cvor * glava, int broj);
38
39   /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
40      U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
41      neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
42      sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
43      */
44 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);
45
46   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
47      pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
48      obrije stara glava liste. */
49 void obrisi_cvor(Cvor ** adresa_glave, int broj);
50
51   /* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
52      oslanjajući se na cinjenicu da je prosledjena lista sortirana
53      neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
54      promenjen ukoliko se obrije stara glava liste. */
55 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);
56
57   /* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
58      zapetama. */
59 void ispisi_vrednosti(Cvor * glava);

```

```

58  /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
59  liste, razdvojene zapetama i uokvirene zagradama. */
60  void ispisi_listu(Cvor * glava);
61
62 #endif

```

lista.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"

5 Cvor *napravi_cvor(int broj)
{
7  /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8   alokacije */
9  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10 if (novi == NULL)
11    return NULL;

13 /* Inicijalizacija polja strukture */
14 novi->vrednost = broj;
15 novi->sledeci = NULL;

17 /* Vraca se adresa novog cvora */
18 return novi;
19 }

21 void osloboodi_listu(Cvor ** adres_a_glave)
{
23 /* Ako je lista vec prazna */
24 if (*adres_a_glave == NULL)
25    return;

27 /* Ako lista nije prazna, treba oslobooditi memoriju. Pre
28  oslobadjanja memorije za glavu liste, treba oslobooditi rep liste
29  */
30 osloboodi_listu(&(*adres_a_glave)->sledeci);
31 /* Nakon osloboodenog repa, oslobadja se glava liste i azurira se
32  glava u pozivajucoj funkciji tako da odgovara praznoj listi */
33 free(*adres_a_glave);
34 *adres_a_glave = NULL;
35 }

37 int dodaj_na_pocetak_liste(Cvor ** adres_a_glave, int broj)
{
39 /* Kreira se novi cvor i proverava se uspesnost kreiranja */
40 Cvor *novi = napravi_cvor(broj);
41 if (novi == NULL)
42    return 1;

```

```

43     /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
45     novi->sledeci = *adresa_glave;
46     *adresa_glave = novi;
47
48     /* Vraca se indikator uspesnog dodavanja */
49     return 0;
50 }
51
52 int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
53 {
54     /* Ako je lista prazna */
55     if (*adresa_glave == NULL) {
56
57         /* Novi cvor postaje glava liste */
58         Cvor *novi = napravi_cvor(broj);
59         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
60         /*
61         if (novi == NULL)
62             return 1;
63
64         /* Azurira se vrednost na koju pokazuje adresa_glave i ujedno se
65            azurira i pokazivacka promenljiva u pozivajucoj funkciji */
66         *adresa_glave = novi;
67
68         /* Vraca se indikator uspesnog dodavanja */
69         return 0;
70     }
71
72     /* Ako lista nije prazna, broj se dodaje u rep liste. */
73     /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
74        novi broj se dodaje u rep liste u rekursivnom pozivu.
75        Informaciju o uspesnosti alokacije u rekursivnom pozivu funkcija
76        prosledjuje visem rekursivnom pozivu koji tu informaciju vraca u
77        rekursivni poziv iznad, sve dok se ne vrati u main. Tek je iz
78        main funkcije moguce pristupiti pravom pocetku liste i
79        osloboziti je celu, ako ima potrebe. Ako je funkcija vratila 0,
80        onda nije bilo greske. */
81     return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
82 }
83
84 int dodaj_sortirano(Cvor ** adresa_glave, int broj)
85 {
86     /* Ako je lista prazna */
87     if (*adresa_glave == NULL) {
88
89         /* Novi cvor postaje glava liste */
90         Cvor *novi = napravi_cvor(broj);
91
92         /* Ukoliko je bilo greske pri kreiranju novog cvora, vraca se 1 */
93         /*
94         if (novi == NULL)

```

```

93     return 1;
95
96     /* Azurira se glava liste */
97     *adresa_glave = novi;
98
99     /* Vraca se indikator uspesnog dodavanja */
100    return 0;
101}
102
103/* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
104   glavi liste, onda se dodaje na pocetak liste */
105if ((*adresa_glave)->vrednost >= broj)
106    return dodaj_na_pocetak_liste(adresa_glave, broj);
107
108/* Inace, broj treba dodati u rep, tako da rep i sa novim cvorom
109   bude sortirana lista. */
110return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
111}
112
113Cvor *pretrazi_listu(Cvor * glava, int broj)
114{
115    /* U praznoj listi nema vrednosti */
116    if (glava == NULL)
117        return NULL;
118
119    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
120    if (glava->vrednost == broj)
121        return glava;
122
123    /* Inace, pretraga se nastavlja u repu liste */
124    return pretrazi_listu(glava->sledeci, broj);
125}
126
127Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
128{
129    /* Trazenog broja nema ako je lista prazna ili ako je broj manji od
130       vrednosti u glavi liste, jer je lista neopadajuće sortirana */
131    if (glava == NULL || glava->vrednost > broj)
132        return NULL;
133
134    /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
135    if (glava->vrednost == broj)
136        return glava;
137
138    /* Inace, pretraga se nastavlja u repu liste */
139    return pretrazi_listu(glava->sledeci, broj);
140}
141void obrisi_cvor(Cvor ** adresa_glave, int broj)
142{
143    /* U praznoj listi nema cvorova za brisanje. */
144    if (*adresa_glave == NULL)

```

```

145     return;
147 /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
148 obrisi_cvor(&(*adresa_glave)->sledeci, broj);
149
151 /* Preostaje provera da li glavu liste treba obrisati */
152 if ((*adresa_glave)->vrednost == broj) {
153     /* Pomocni pokazuje na cvor koji treba da se obrije */
154     Cvor *pomocni = *adresa_glave;
155     /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
156      brise se cvor koji je bio glava liste. */
157     *adresa_glave = (*adresa_glave)->sledeci;
158     free(pomocni);
159 }
161
163 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
164 {
165     /* Ako je lista prazna ili glava liste sadrzi vrednost koja je veca
166      od broja, kako je lista sortirana rastuce nema potrebe broj
167      traziti u repu liste i zato se funkcija prekida */
168     if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
169         return;
171
172     /* Brisu se cvorovi iz repa koji imaju vrednost broj */
173     obrisi_cvor(&(*adresa_glave)->sledeci, broj);
174
175     /* Preostaje provera da li glavu liste treba obrisati */
176     if ((*adresa_glave)->vrednost == broj) {
177         /* Pomocni pokazuje na cvor koji treba da se obrije */
178         Cvor *pomocni = *adresa_glave;
179         /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi i
180          brise se cvor koji je bio glava liste */
181         *adresa_glave = (*adresa_glave)->sledeci;
182         free(pomocni);
183     }
184 }
186 void ispisi_vrednosti(Cvor * glava)
187 {
188     /* Prazna lista */
189     if (glava == NULL)
190         return;
191
192     /* Ispisuje se vrednost u glavi liste */
193     printf("%d", glava->vrednost);
194
195     /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
196      se poziva ista funkcija za ispis ostalih. */
197     if (glava->sledeci != NULL) {
198         printf(", ");
199     }

```

```

197     ispis_i_vrednosti(glava->sledeci);
198 }
199 }
200
201 void ispis_i_listu(Cvor * glava)
202 {
203     /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
204      jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
205      na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
206      zgrade, a rekurzivno ispisivanje vrednosti u listi prepusta
207      rekurzivnoj pomocnoj funkciji ispis_i_vrednosti, koja ce ispisati
208      elemente razdvojene zapetom i razmakom. */
209     putchar('[');
210     ispis_i_vrednosti(glava);
211     printf("]\n");
212 }
```

Rešenje 4.3

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 #define MAX_DUZINA 20
6
7 /* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
8   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
9 typedef struct _Cvor {
10     char etiketa[20];
11     unsigned broj_pojavljivanja;
12     struct _Cvor *sledeci;
13 } Cvor;
14
15 /* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
16   ili NULL ako alokacija nije uspesno izvrsena */
17 Cvor *napravi_cvor(unsigned br, char *etiketa)
18 {
19     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
20       alokacije */
21     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
22     if (novi == NULL)
23         return NULL;
24
25     /* Inicijalizacija polja strukture */
26     novi->broj_pojavljivanja = br;
27     strcpy(novi->etiketa, etiketa);
28     novi->sledeci = NULL;
29
30     /* Vraca se adresa novog cvora */
31     return novi;
32 }
```

```

33  /* Funkcija oslobođaja dinamicku memoriju zauzetu cvorovima liste */
35  void osloboodi_listu(Cvor ** adresa_glave)
{
37      Cvor *pomocni = NULL;

39      /* Sve dok lista ni bude prazna, brise se tekuća glava liste i
40          azurira se vrednost glave liste */
41      while (*adresa_glave != NULL) {
42          pomocni = (*adresa_glave)->sledeci;
43          free(*adresa_glave);
44          *adresa_glave = pomocni;
45      }
46      /* Nakon izlaska iz petlje pokazivac glava u main funkciji koji se
47          nalazi na adresi adresa_glave bice postavljen na NULL vrednost.
48          */
49 }
50
51 /* Funkcija dodaje novi cvor na početak liste. Povratna vrednost je 1
52     ako je doslo do greske pri alokaciji memorije za nov cvor, odnosno
53     0 */
54 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
55                             char *etiketa)
56 {
57     /* Kreira se novi cvor i proverava se uspesnost alokacije */
58     Cvor *novi = napravi_cvor(br, etiketa);
59     if (novi == NULL)
60         return 1;

61     /* Dodaje se novi cvor na početak liste */
62     novi->sledeci = *adresa_glave;
63     *adresa_glave = novi;

64     /* Vraca se indikator uspesnog dodavanja */
65     return 0;
66 }

67 /* Funkcija vraca cvor koji kao vrednost sadrži traženu etiketu ili
68     NULL ako takav cvor ne postoji u listi */
69 Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
70 {
71     Cvor *tekuci;

72     /* Obilazi se cvor po cvor liste */
73     for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
74         /* Ako tekuci cvor sadrži traženu etiketu, vracamo njegovu
75             vrednost */
76         if (strcmp(tekuci->etiketa, etiketa) == 0)
77             return tekuci;

78     /* Cvor nije pronadjen */
79     return NULL;
80 }

81
82
83

```

```

    }

85 /* Funkcija ispisuje sadrzaj liste */
87 void ispis_listu(Cvor * glava)
{
89     /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete i
90      broj njihovog pojavljivanja u HTML datoteci. */
91     for (; glava != NULL; glava = glava->sledeci)
92         printf("%s - %u\n", glava->etiketa, glava->broj_pojavljenja);
93 }

95 /* Glavni program */
96 int main(int argc, char **argv)
{
97     /* Provera se da li je program pozvan sa ispravnim brojem
98      argumenata. */
99     if (argc != 2) {
100         fprintf(stderr,
101                 "Greska! Program se poziva sa: ./a.out datoteka.html!\n");
102         ;
103         exit(EXIT_FAILURE);
104     }

105     /* Priprema datoteke za citanje */
106     FILE *in = NULL;
107     in = fopen(argv[1], "r");
108     if (in == NULL) {
109         fprintf(stderr,
110                 "Greska prilikom otvaranja datoteke %s!\n", argv[1]);
111         exit(EXIT_FAILURE);
112     }

113     char c;
114     int i = 0;
115     char procitana[MAX_DUZINA];
116     Cvor *glava = NULL;
117     Cvor *trazeni = NULL;

118     /* Cita se karakter po karakter datoteke sve dok se ne procita cela
119      datoteka */
120     while ((c = fgetc(in)) != EOF) {

121         /* Proverava se da li se pocinje sa citanjem nove etikete */
122         if (c == '<') {
123             /* Proverava se da li se cita zatvarajuca etiketa */
124             if ((c = fgetc(in)) == '/') {
125                 i = 0;
126                 while ((c = fgetc(in)) != '>')
127                     procitana[i++] = c;
128             }
129             /* Cita se zatvarajuca etiketa */
130             else {
131
132
133

```

```

135         i = 0;
137         procitana[i++] = c;
138         while ((c = fgetc(in)) != ' ' && c != '>')
139             procitana[i++] = c;
140     }
141     procitana[i] = '\0';
142
143     /* Trazi se procitana etiketa medju postojecim cvorovima liste.
144      Ukoliko ne postoji, dodaje se novi cvor za ucitanu etiketu
145      sa
146      brojem pojavljivanja 1. Inace se uvecava broj pojavljivanja
147      etikete */
148     trazeni = pretrazi_listu(glava, procitana);
149     if (trazeni == NULL) {
150         if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
151             fprintf(stderr, "Neuspela alokacija za nov cvor\n");
152             oslobodi_listu(&glava);
153             exit(EXIT_FAILURE);
154         }
155     } else
156         trazeni->broj_pojavljivanja++;
157 }
158
159 /* Zatvaranje datoteke */
160 fclose(in);
161
162 /* Ispisuje se sadrzaj cvorova liste */
163 ispisi_listu(glava);
164
165 /* Oslobadja se memorija zauzeta listom */
166 oslobodi_listu(&glava);
167
168 exit(EXIT_SUCCESS);
}

```

Rešenje 4.4

```

#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_INDEKS 11
6 #define MAX_IME_PREZIME 21
7
8 /* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
9   studentu */
10 typedef struct _Cvor {
11     char broj_indeksa[MAX_INDEKS];
12     char ime[MAX_IME_PREZIME];
13     char prezime[MAX_IME_PREZIME];
14 }

```

```

14     struct _Cvor *sledeci;
15 } Cvor;
16
17 /* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
18    novi cvor ili NULL ukoliko je doslo do greske */
19 Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
20 {
21     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
22        alokacije */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizacija polja strukture */
28     strcpy(novi->broj_indeksa, broj_indeksa);
29     strcpy(novi->ime, ime);
30     strcpy(novi->prezime, prezime);
31     novi->sledeci = NULL;
32
33     /* Vraca se adresa novog cvora */
34     return novi;
35 }
36
37 /* Funkcija oslobadja memoriju zauzetu cvorovima liste */
38 void osloboodi_listu(Cvor **adresa_glave)
39 {
40     /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
41     if (*adresa_glave == NULL)
42         return;
43
44     /* Rekurzivnim pozivom se oslobadja rep liste */
45     osloboodi_listu(&(*adresa_glave)->sledeci);
46
47     /* Potom se oslobadja i glava liste */
48     free(*adresa_glave);
49
50     /* Proglasava se lista praznom */
51     *adresa_glave = NULL;
52 }
53
54 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
55    do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
56 int dodaj_na_pocetak_liste(Cvor **adresa_glave, char *broj_indeksa,
57                             char *ime, char *prezime)
58 {
59     /* Kreira se novi cvor i proverava se uspesnost alokacije */
60     Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
61     if (novi == NULL)
62         return 1;
63
64     /* Dodaje se novi cvor na pocetak liste */
65     novi->sledeci = *adresa_glave;

```

```

66     *adresa_glave = novi;
68
69     /* Vraca se indikator uspesnog dodavanja */
70     return 0;
71 }
72
73 /* Funkcija ispisuje sadrzaj cvorova liste. */
74 void ispis_listu(Cvor * glava)
75 {
76     /* Pocevsi od glave liste */
77     for (; glava != NULL; glava = glava->sledeci)
78         printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
79                glava->prezime);
80 }
81
82 /* Funkcija vraca cvor koji kao vrednost sadrzi trazen i broj indeksa.
83  U suprotnom funkcija vraca NULL */
84 Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
85 {
86     /* Ako je lista prazna, ne postoji trazen cvor */
87     if (glava == NULL)
88         return NULL;
89
90     /* Poredi se trazen i broj indeksa sa brojem indeksa u glavi liste
91      */
92     if (!strcmp(glava->broj_indeksa, broj_indeksa))
93         return glava;
94
95     /* Ukoliko u glavi liste nije trazen indeks, pretraga se nastavlja
96      u repu liste */
97     return pretrazi_listu(glava->sledeci, broj_indeksa);
98 }
99
100 /* Glavni program */
101 int main(int argc, char **argv)
102 {
103     /* Argumenti komandne linije su neophodni jer se iz komandne linije
104      dobija ime datoteke sa informacijama o studentima */
105     if (argc != 2) {
106         fprintf(stderr,
107                 "Greska! Program se poziva sa: ./a.out ime_datoteke\n");
108         exit(EXIT_FAILURE);
109     }
110
111     /* Priprema datoteke za citanje */
112     FILE *in = NULL;
113     in = fopen(argv[1], "r");
114     if (in == NULL) {
115         fprintf(stderr,
116                 "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
117         exit(EXIT_FAILURE);
118     }

```

```

118  /* Pomocne promenljive za citanje vrednosti koje treba smestiti u
119   listu */
120  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
121  char broj_indeksa[MAX_INDEKS];
122  Cvor *glava = NULL;
123  Cvor *trazeni = NULL;

124  /* Ucitavanje vrednosti u listu */
125  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
126    if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
127      fprintf(stderr, "Neuspela alokacija za nov cvor\n");
128      osloboodi_listu(&glava);
129      exit(EXIT_FAILURE);
130    }

132  /* Datoteka vise nije potrebna i zatvara se. */
133  fclose(in);

136  /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
137  while (scanf("%s", broj_indeksa) != EOF) {
138    trazeni = pretrazi_listu(glava, broj_indeksa);
139    if (trazeni == NULL)
140      printf("ne\n");
141    else
142      printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
143  }

144  /* Oslobadja se memorija zauzeta za cvorove liste. */
145  osloboodi_listu(&glava);

148  exit(EXIT_SUCCESS);
}

```

Rešenje 4.6

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "lista.h"

5  /* Funkcija objedinjuje dve liste ciji se pokazivaci na glave nalaze
6   na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
7   pokazivaca postojećih cvorova listi */
8  Cvor *objedini(Cvor **adresa_glave_1, Cvor **adresa_glave_2)
9  {
10   /* Pokazivaci na pocetne cvorove liste koje se prevezuju */
11   Cvor *lista1 = *adresa_glave_1;
12   Cvor *lista2 = *adresa_glave_2;

13   /* Pokazivac na pocetni cvor rezultujuće liste */
14   Cvor *rezultujuca = NULL;
15 }

```

```

17     Cvor *tekuci = NULL;
19
21     /* Ako su obe liste prazne i rezultat je prazna lista */
23     if (lista1 == NULL && lista2 == NULL)
24         return NULL;
26
28     /* Ako je prva lista prazna, rezultat je druga lista */
30     if (lista1 == NULL)
31         return lista2;
33
35     /* Ako je druga lista prazna, rezultat je prva lista */
37     if (lista2 == NULL)
38         return lista1;
40
42     /* Odredjuje se prvi cvor rezultujuće liste - to je ili prvi cvor
44         liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
46         koji sadrži manju vrednost */
48     if (lista1->vrednost < lista2->vrednost) {
49         rezultujuca = lista1;
50         lista1 = lista1->sledeci;
51     } else {
52         rezultujuca = lista2;
53         lista2 = lista2->sledeci;
54     }
56
58     /* Kako promenljiva rezultujuća pokazuje na početak nove liste, ne
59         sme joj se menjati vrednost. Zato se koristi pokazivac tekuci
60         koji sadrži adresu trenutnog cvora rezultujuće liste */
61     tekuci = rezultujuca;
63
65     /* U svakoj iteraciji petlje rezultujućoj listi se dodaje novi cvor
66         tako da bude uređena neopadajuće. Pokazivac na listu iz koje se
67         uzima cvor se azurira tako da pokazuje na sledeći cvor */
68     while (lista1 != NULL && lista2 != NULL) {
69         if (lista1->vrednost < lista2->vrednost) {
70             tekuci->sledeci = lista1;
71             lista1 = lista1->sledeci;
72         } else {
73             tekuci->sledeci = lista2;
74             lista2 = lista2->sledeci;
75         }
76         tekuci = tekuci->sledeci;
77     }
79
80     /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
81         rezultujuću listu treba nadovezati ostatak druge liste */
83     if (lista1 == NULL)
84         tekuci->sledeci = lista2;
85     else
86         /* U suprotnom treba nadovezati ostatak prve liste */
87         tekuci->sledeci = lista1;
89
90     /* Preko adresa glava polaznih listi vrednosti pokazivaca u

```

```

69      pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
70      prethodnih listi nalaze negde unutar rezultujuće liste. Do njih
71      se može doći prateći pokazivace iz glave rezultujuće liste, tako
72      da stare pokazivace treba postaviti na NULL. */
73      *adresa_glave_1 = NULL;
74      *adresa_glave_2 = NULL;

75      return rezultujuca;
76  }

77 int main(int argc, char **argv)
78 {
79     /* Argumenti komandne linije su neophodni */
80     if (argc != 3) {
81         fprintf(stderr,
82             "Program se poziva sa: ./a.out dat1.txt dat2.txt\n");
83         exit(EXIT_FAILURE);
84     }

85     /* Otvaramo datoteke u kojima se nalaze elementi liste */
86     FILE *in1 = NULL;
87     in1 = fopen(argv[1], "r");
88     if (in1 == NULL) {
89         fprintf(stderr,
90             "Greska prilikom otvaranja datoteke %s.\n", argv[1]);
91         exit(EXIT_FAILURE);
92     }

93     FILE *in2 = NULL;
94     in2 = fopen(argv[2], "r");
95     if (in2 == NULL) {
96         fprintf(stderr,
97             "Greska prilikom otvaranja datoteke %s.\n", argv[2]);
98         exit(EXIT_FAILURE);
99     }

100    /* Liste su na početku prazne */
101    int broj;
102    Cvor *lista1 = NULL;
103    Cvor *lista2 = NULL;

104    /* Ucitavanje liste */
105    while (fscanf(in1, "%d", &broj) != EOF)
106        dodaj_na_kraj_liste(&lista1, broj);

107    while (fscanf(in2, "%d", &broj) != EOF)
108        dodaj_na_kraj_liste(&lista2, broj);

109    /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
110    fclose(in1);
111    fclose(in2);
112
113
114
115
116
117
118
119

```

```

121  /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
122  objedinjavanjem listi */
123  Cvor *rezultat = objedini(&list1, &list2);
124
125  /* Ispis rezultujuce liste. */
126  ispisi_listu(rezultat);
127
128  /* Lista rezultat dobijena je prevezivanjem cvorova polaznih listi.
129  Njenim oslobođanjem bice oslobođena sva zauzeta memorija. */
130  osloboodi_listu(&rezultat);
131
132  exit(EXIT_SUCCESS);
}

```

Rešenje 4.8

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
5   predstavlja zagradu koja se koristi i pokazivac na sledeci cvor
6   liste */
7 typedef struct cvor {
8     char zagrada;
9     struct cvor *sledeci;
10 } Cvor;
11
12 /* Funkcija koja oslobadja memoriju zauzetu stekom */
13 void osloboodi_stek(Cvor ** stek)
14 {
15   Cvor *tekuci;
16   Cvor *pomocni;
17
18   /* Oslobadja se cvor po cvor steka */
19   tekuci = *stek;
20   while (tekuci != NULL) {
21     pomocni = tekuci->sledeci;
22     free(tekuci);
23     tekuci = pomocni;
24   }
25
26   /* Stek se proglašava praznim */
27   *stek = NULL;
28 }
29
30 /* Glavni program */
31 int main()
32 {
33   /* Stek je na pocetku prazen */
34   Cvor *stek = NULL;
35   FILE *ulaz = NULL;

```

```

15     char c;
16     Cvor *pomocni = NULL;
17
18     /* Otvaranje datoteke za citanje izraza */
19     ulaz = fopen("izraz.txt", "r");
20     if (ulaz == NULL) {
21         fprintf(stderr,
22             "Greska prilikom otvaranja datoteke izraz.txt!\n");
23         exit(EXIT_FAILURE);
24     }
25
26     /* Cita se karakter po karakter iz datoteke dok se ne dodje do
27      kraja */
28     while ((c = fgetc(ulaz)) != EOF) {
29         /* Ako je ucitana otvorena zagrada, stavlja se na stek */
30         if (c == '(' || c == '{' || c == '[') {
31             /* Alocira se memorija za novi cvor liste i proverava se
32              uspesnost alokacije */
33             pomocni = (Cvor *) malloc(sizeof(Cvor));
34             if (pomocni == NULL) {
35                 fprintf(stderr, "Greska prilikom alokacije memorije!\n");
36                 /* Oslobadja se memorija zauzeta stekom */
37                 oslobodi_stek(&stek);
38                 /* I prekida se sa izvrsavanjem programa */
39                 exit(EXIT_FAILURE);
40             }
41
42             /* Inicijalizacija polja strukture */
43             pomocni->zagrada = c;
44
45             /* Promena vrha steka */
46             pomocni->sledeci = stek;
47             stek = pomocni;
48         }
49
50         /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
51            prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
52            otvorena zagrada */
53     else {
54         if (c == ')' || c == '}' || c == ']') {
55             if (stek != NULL && ((stek->zagrada == '(' && c == ')')
56                         || (stek->zagrada == '{' && c == '}')
57                         || (stek->zagrada == '[' && c == ']')))
58             {
59                 /* Sa vrha steka se uklanja otvorena zagrada */
60                 pomocni = stek->sledeci;
61                 free(stek);
62                 stek = pomocni;
63             } else {
64                 /* Inace, zaključujemo da zagrade u izrazu nisu ispravno
65                  uparene */
66                 break;
67             }
68         }
69     }
70
71     /* Pisanje rezultata u izraz.txt */
72     if (stek != NULL)
73         fprintf(izraz, "%c", stek->zagrada);
74     else
75         fprintf(izraz, "Nisu ispravno uparene zagrade u izrazu!");
76
77     /* Zatvaranje datoteke */
78     fclose(ulaz);
79
80     /* Islobadjanje memorije */
81     oslobodi_stek(&stek);
82
83     /* Izlazak iz programa */
84     exit(EXIT_SUCCESS);
85

```

```
87         }
88     }
89 }
90
91 /* Procitana je cela datoteka i treba je zatvoriti */
92 fclose(ulaz);
93
94 /* Ako je stek prazan i pročitana je cela datoteka, zagrade su
95    ispravno uparene */
96 if (stek == NULL && c == EOF)
97     printf("Zagrade su ispravno uparene.\n");
98 else {
99     /* U suprotnom se zaključuje da zagrade nisu ispravno uparene */
100    printf("Zagrade nisu ispravno uparene.\n");
101    /* Oslobadja se memorija koja je ostala zauzeta stekom */
102    oslobođi_stek(&stek);
103 }
104
105 exit(EXIT_SUCCESS);
106 }
```

Rešenje 4.9

stek.h

```
1 #ifndef _STEK_H_
2 #define _STEK_H_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <ctype.h>
8
9 #define MAX 100
10
11 #define OTVORENA 1
12 #define ZATVORENA 2
13
14 #define VAN_ETIKETE 0
15 #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2
17
18 /* Struktura kojim se predstavlja cvor liste sadrži ime etikete i
19    pokazivac na sledeći cvor */
20 typedef struct cvor {
21     char etiketa[MAX];
22     struct cvor *sledeći;
23 } Cvor;
24
25 /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca njegovu
```

```

26     adresu ili NULL ako alokacija nije bila uspesna */
27 Cvor *napravi_cvor(char *etiketa);
28
29 /* Funkcija oslobadja memoriju zauzetu stekom */
30 void osloboodi_stek(Cvor ** adresa_vrha);
31
32 /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske pri
   33   alokaciji memorije za novi cvor funkcija vraca 1, inace vraca 0 */
34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);
35
36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
   37   pokazivac razlicit od NULL, tada u niz karaktera na koji on
   38   pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok u
   39   suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan (pa
   40   samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
   41   suprotnom */
42 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);
43
44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
   45   steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);
47
48 /* Funkcija prikazuje stek od vrha prema dnu */
49 void prikazi_stek(Cvor * vrh);
50
51 /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
   52   etiketu, i upisuje je u nisku na koju pokazuje pokazivac etiketa.
   53   Vraca EOF u slucaju da se dodje do kraja datoteke pre nego sto se
   54   procita etiketa. Vraca OTVORENA, ako je procitana otvorena
   55   etiketa, odnosno ZATVORENA, ako je procitana zatvorena etiketa */
56 int uzmi_etiketu(FILE * f, char *etiketa);
57
58 #endif

```

stek.c

```

1 #include "stek.h"
2
3 Cvor *napravi_cvor(char *etiketa)
4 {
5     /* Alocira se memorija za novi cvor liste i proverava se uspesnost
6       alokacije */
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8     if (novi == NULL)
9         return NULL;
10
11     /* Inicijalizacija polja u novom cvoru */
12     if (strlen(etiketa) >= MAX) {
13         fprintf(stderr, "Etiketa je preduga, bice skracena.\n");
14         etiketa[MAX - 1] = '\0';
15     }

```

```

16    strcpy(novi->etiketa, etiketa);
17    novi->sledeci = NULL;
18
19    /* Vraca se adresa novog cvora */
20    return novi;
21}
22
23 void osloboodi_stek(Cvor ** adresa_vrha)
24{
25    Cvor *pomocni;
26
27    /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
28    while (*adresa_vrha != NULL) {
29        /* Potrebno je prvo zapamtitи adresu sledeceg cvora i onda
30         oslobooditi cvor koji predstavlja vrh steka */
31        pomocni = *adresa_vrha;
32        /* Sledeci cvor je novi vrh steka */
33        *adresa_vrha = (*adresa_vrha)->sledeci;
34        free(pomocni);
35    }
36
37    /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
38     adresa_vrha ce pokazivati na NULL. */
39}
40
41 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
42{
43    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
44    Cvor *novi = napravi_cvor(etiketa);
45    if (novi == NULL)
46        return 1;
47
48    /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
49    novi->sledeci = *adresa_vrha;
50    *adresa_vrha = novi;
51    return 0;
52}
53
54 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
55{
56    Cvor *pomocni;
57
58    /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom i
59     vraca se 0 */
60    if (*adresa_vrha == NULL)
61        return 0;
62
63    /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
64     adresu kopira etiketa sa vrha steka */
65    if (etiketa != NULL)
66        strcpy(etiketa, (*adresa_vrha)->etiketa);

```

```
68  /* Element sa vrha steka se uklanja */
69  pomocni = *adresa_vrha;
70  *adresa_vrha = (*adresa_vrha)->sledeci;
71  free(pomocni);
72
73  /* Vraca se indikator uspesno izvrsene radnje */
74  return 1;
75 }
76
77 char *vrh_steka(Cvor * vrh)
78 {
79  /* Prazan stek nema cvor koji je vrh i vraca se NULL */
80  if (vrh == NULL)
81      return NULL;
82
83  /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
84   koji je na vrhu steka. */
85  return vrh->etiketa;
86 }
87
87 void prikazi_stek(Cvor * vrh)
88 {
89  /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
90  for (; vrh != NULL; vrh = vrh->sledeci)
91      printf("<%s>\n", vrh->etiketa);
92 }
93
94 int uzmi_etiketu(FILE * f, char *etiketa)
95 {
96  int c;
97  int i = 0;
98
99  /* Stanje predstavlja informaciju dokle se stalo sa citanjem
100   etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
101   nije zapoceto citanje. */
102
103  /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
104   OTVORENA ili ZATVORENA. */
105  int stanje = VAN_ETIKETE;
106  int tip;
107
108  /* HTML je neosetljiv na razliku izmedju malih i velikih slova, dok
109   to u C-u ne vazi. Zato ce sve etikete biti prevedene u zapis
110   samo malim slovima. */
111  while ((c = fgetc(f)) != EOF) {
112      switch (stanje) {
113      case VAN_ETIKETE:
114          if (c == '<')
115              stanje = PROCITANO_MANJE;
116          break;
117      case PROCITANO_MANJE:
118          if (c == '/') {
119              /* Cita se zatvorena etiketa */
120              tip = ZATVORENA;
```

```

120     } else {
121         if (isalpha(c)) {
122             /* Cita se otvorena etiketa */
123             tip = OTVORENA;
124             etiketa[i++] = tolower(c);
125         }
126     }
127     /* Od sada se cita etiketa i zato se menja stanje */
128     stanje = U_ETIKETI;
129     break;
130 case U_ETIKETI:
131     if (isalpha(c) && i < MAX - 1) {
132         /* Ako je procitani karakter slovo i nije prekoracena
133            dozvoljena duzina etikete, procitani karakter se smanjuje
134            i smesta u etiketu */
135         etiketa[i++] = tolower(c);
136     } else {
137         /* Inace, staje se sa citanjem etikete. Korektno se zavrsava
138            niska koja sadrzi procitanu etiketu i vraca se njen tip */
139         etiketa[i] = '\0';
140         return tip;
141     }
142     break;
143 }
144 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
145    etiketa i vraca se EOF. */
146 return EOF;
147 }
```

main.c

```

#include "stek.h"
2
/* Glavni program */
4 int main(int argc, char **argv)
{
6     /* Na pocetku, stek je prazan i etikete su uparene jer nijedna jos
      nije procitana. */
8     Cvor *vrh = NULL;
10    char etiketa[MAX];
11    int tip;
12    int uparene = 1;
13    FILE *f = NULL;
14
15    /* Ime datoteke se preuzima iz komandne linije. */
16    if (argc < 2) {
17        fprintf(stderr, "Koriscenje: %s ime_html_datoteke\n", argv[0]);
18        exit(EXIT_FAILURE);
19    }
20 }
```

```

20  /* Datoteka se otvara za citanje */
21  if ((f = fopen(argv[1], "r")) == NULL) {
22      fprintf(stderr, "Greska prilikom otvaranja datoteke %s.\n",
23              argv[1]);
24      exit(EXIT_FAILURE);
25  }
26
27  /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
28  while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
29      /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
30         etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
31         potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
32         koje nemaju sadrzaj (npr link). Zbog jednostavnosti
33         pretpostavlja se da njih nema u HTML dokumentu. */
34      if (tip == OTVORENA) {
35          if (strcmp(etiketa, "br") != 0
36              && strcmp(etiketa, "hr") != 0
37              && strcmp(etiketa, "meta") != 0)
38              if (potisni_na_stek(&vrh, etiketa) == 1) {
39                  fprintf(stderr, "Neuspela alokacija za nov cvor\n");
40                  osloboidi_stek(&vrh);
41                  exit(EXIT_FAILURE);
42              }
43      }
44      /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da je
45         u pitanju zatvaranje etikete koja je poslednja otvorena, a jos
46         uvek nije zatvorena. Ona se mora nalaziti na vrhu steka. Ako
47         je taj uslov ispunjen, skida se sa steka, jer je upravo
48         zatvorena. U suprotnom, pronadjena je nepravilnost i etikete
49         nisu pravilno uparene. */
50      else if (tip == ZATVORENA) {
51          if (vrh_steka(vrh) != NULL
52              && strcmp(vrh_steka(vrh), etiketa) == 0)
53              skinji_sa_steka(&vrh, NULL);
54          else {
55              printf("Etikete nisu pravilno uparene\n");
56              printf("(nadjena je etiketa </%s>, etiketa);"
57                  if (vrh_steka(vrh) != NULL)
58                      printf(", a poslednja otvorena je <%s>)\n", vrh_steka(vrh))
59              ;
60              else
61                  printf(" koja nije otvorena)\n");
62              uparene = 0;
63              break;
64          }
65      }
66      /* Završeno je citanje i datoteka se zatvara */
67      fclose(f);
68
69      /* Ako do sada nije pronadjen pogresno uparivanje, stek bi trebalo
70         da bude prazan. Ukoliko nije, tada postoje etikete koje su

```

```

    ostale otvorene */
72 if (uparene) {
    if (vrh_steka(vrh) == NULL)
        printf("Etikete su pravilno uparene!\n");
    else {
        printf("Etikete nisu pravilno uparene\n");
        printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
        /* Oslobođaja se memorija zauzeta stekom */
        osloboodi_stek(&vrh);
    }
}
82 exit(EXIT_SUCCESS);
84 }
```

Rešenje 4.10*red.h*

```

1 #ifndef _RED_H_
2 #define _RED_H_
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define MAX 1000
8 #define JMBG_DUZINA 14
9
10 /* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika i
11   opis njegovog zahteva. */
12 typedef struct {
13     char jmbg[JMBG_DUZINA];
14     char opis[MAX];
15 } Zahtev;
16
17 /* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
18   korisnika i pokazivac na sledeci cvor liste. */
19 typedef struct cvor {
20     Zahtev nalog;
21     struct cvor *sledeci;
22 } Cvor;
23
24 /* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev sa
25   poslate adresom i vraca adresu novog cvora ili NULL ako je doslo do
26   greske pri alokaciji. Prosledjuje joj se pokazivac na zahtev koji
27   treba smestiti u novi cvor zbog smestanja manjeg podatka na
28   sistemski stek. Pokazivac na strukturu Zahtev je manje velicine u
29   bajtovima(B) u odnosu na strukturu Zahtev. */
30 Cvor *napravi_cvor(Zahtev * zahtev);
31 }
```

```

1  /* Funkcija prazni red oslobadjajući memoriju koji je red zauzimao */
33 void osloboodi_red(Cvor ** pocetak, Cvor ** kraj);

35 /* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo do
   greske pri alokaciji memorije za novi cvor, inace vraca 0. */
37 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

39 /* Funkcija skida sa pocetka reda zahtev. Ako je poslednji argument
41   pokazivac razlicit od NULL, tada se u strukturu na koju on
42   pokazuje upisuje zahtev koji je upravo skinut sa reda dok u
43   suprotnom ne upisuje nista. Vraca 0, ako je red bio prazan ili 1 u
44   suprotnom. */
45 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                     Zahtev * zahtev);

47 /* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev korisnika
49   na pocetku reda. Ukoliko je red prazan funkcija vraca NULL. */
50 Zahtev *pocetak_reda(Cvor * pocetak);

51 /* Funkcija prikazuje sadrzaj reda. */
52 void prikazi_red(Cvor * pocetak);

55 #endif

```

red.c

```

1 #include "red.h"

3 Cvor *napravi_cvor(Zahtev * zahtev)
{
5  /* Alocira se memorija za novi cvor liste i proverava uspesnost
   alokacije */
7  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
9  if (novi == NULL)
    return NULL;

11 /* Inicijalizacija polja strukture */
12 novi->nalog = *zahtev;
13 novi->sledeci = NULL;

15 /* Vraca se adresa novog cvora */
16 return novi;
17 }

19 void osloboodi_red(Cvor ** pocetak, Cvor ** kraj)
{
21  Cvor *pomocni = NULL;

23 /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
24 while (*pocetak != NULL) {

```

```

25     /* Potrebno je prvo zapamtitи adresу sledeceg cvora i onda
26         osloboeditи cvor sa pocetka reda */
27     pomocni = *pocetak;
28     *pocetak = (*pocetak)->sledeci;
29     free(pomocni);
30 }
31 /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
32     treba postaviti na NULL. */
33 *kraj = NULL;
34 }

35 int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
36                   Zahtev * zahtev)
37 {
38     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
39     Cvor *novi = napravi_cvor(zahtev);
40     if (novi == NULL)
41         return 1;
42
43     /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
44         kraj, to je podjednako efikasno kao dodavanje na pocetak liste
45         */
46     if (*adresa_kraja != NULL) {
47         (*adresa_kraja)->sledeci = novi;
48         *adresa_kraja = novi;
49     } else {
50         /* Ako je red bio ranije prazan */
51         *adresa_pocetka = novi;
52         *adresa_kraja = novi;
53     }
54
55     /* Vraca se indikator uspesnog dodavanja */
56     return 0;
57 }

58 int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
59                     Zahtev * zahtev)
60 {
61     Cvor *pomocni = NULL;
62
63     /* Ako je red prazan */
64     if (*adresa_pocetka == NULL)
65         return 0;
66
67     /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
68         zahtev koji je na pocetku reda. */
69     if (zahtev != NULL)
70         *zahtev = (*adresa_pocetka)->nalog;
71
72     /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i pokazivac
73         na adresi adresa_pocetka se azurira da pokazuje na sledeci cvor
74         u redu. */
75 }
```

```

77     pomocni = *adresa_pocetka;
78     *adresa_pocetka = (*adresa_pocetka)->sledeci;
79     free(pomocni);
80
81     /* Ukoliko red nakon oslobođanja početnog cvora ostane prazan,
82      potrebno je azurirati i vrednost pokazivaca na adresi
83      adresa_kraja na NULL */
84     if (*adresa_pocetka == NULL)
85         *adresa_kraja = NULL;
86
87     return 1;
88 }
89
90 Zahtev *pocetak_reda(Cvor * pocetak)
91 {
92     /* U praznom redu nema zahteva */
93     if (pocetak == NULL)
94         return NULL;
95
96     /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
97     return &(pocetak->nalog);
98 }
99
100 void prikazi_red(Cvor * pocetak)
101 {
102     /* Prikazuje se sadrzaj reda od pocetka prema kraju */
103     for (; pocetak != NULL; pocetak = pocetak->sledeci)
104         printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);
105
106     printf("\n");
107 }
```

main.c

```

2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include "red.h"
6
7 #define VREME_ZA_PAUZU 5
8
9 /* Glavni program */
10 int main(int argc, char **argv)
11 {
12     /* Red je prazan. */
13     Cvor *pocetak = NULL, *kraj = NULL;
14     Zahtev nov_zahtev;
15     Zahtev *sledeci = NULL;
16     char odgovor[3];
17     int broj_usluzenih = 0;
```

```

18  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog JMBG
   broja i opisa potrebe usluge. */
20 printf("Sluzbenik evidentira korisnicke zahteve:\n");
21 while (1) {
22
23     /* Ucitava se JMBG */
24     printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
25     if (scanf("%s", nov_zahetv.jmbg) == EOF)
26         break;
27
28     /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
       JMBG broja procitao i da bi fgetse nakon toga procitala
       ispravan red sa opisom zahteva */
29     getchar();
30
31     /* Ucitava se opis problema */
32     printf("\tOpis problema: ");
33     fgets(nov_zahetv.opis, MAX - 1, stdin);
34     /* Ako je poslednji karakter nov red, eliminise se */
35     if (nov_zahetv.opis[strlen(nov_zahetv.opis) - 1] == '\n')
36         nov_zahetv.opis[strlen(nov_zahetv.opis) - 1] = '\0';
37
38     /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
39     if (dodaj_u_red(&pocetak, &kraj, &nov_zahetv) == 1) {
40         fprintf(stderr, "Neuspela alokacija za nov cvor\n");
41         osloboodi_red(&pocetak, &kraj);
42         exit(EXIT_FAILURE);
43     }
44
45
46     /* Otvaranje datoteke za dopisivanje izvestaja */
47     FILE *izlaz = fopen("izvestaj.txt", "a");
48     if (izlaz == NULL) {
49         fprintf(stderr, "Neuspesno otvaranje datoteke izvestaj.txt\n");
50         exit(EXIT_FAILURE);
51     }
52
53     /* Dokle god ima korisnika u redu, treba ih usluziti */
54     while (1) {
55         sledeci = pocetak_reda(pocetak);
56         /* Ako nema nikog vise u redu, prekida se petlja */
57         if (sledeci == NULL)
58             break;
59
60         printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
61         printf("i zahtevom: %s\n", sledeci->opis);
62
63         skini_sa_reda(&pocetak, &kraj, &nov_zahetv);
64
65         broj_usluzenih++;
66
67         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
68     }
69
70 }

```

```

70     scanf("%s", odgovor);
72     if (strcmp(odgovor, "Da") == 0)
73         dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
74     else
75         fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
76                 nov_zahtev.opis);
78     if (broj_usluzenih == VREME_ZA_PAUZU) {
79         printf("\nDa li je kraj smene? [Da/Ne] ");
80         scanf("%s", odgovor);
82         if (strcmp(odgovor, "Da") == 0)
83             break;
84         else
85             broj_usluzenih = 0;
86     }
88 /**
90 * Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:
92
93     while (skinisi_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
94         printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
95                nov_zahtev.jmbg);
96         printf("sa zahtevom: %s\n", nov_zahtev.opis);
97         broj_usluzenih++;
98
99         printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
100        scanf("%s", odgovor);
101        if (strcmp(odgovor, "Da") == 0)
102            dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
103        else
104            fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
105                    nov_zahtev.jmbg, nov_zahtev.opis);
106
107        if (broj_usluzenih == VREME_ZA_PAUZU) {
108            printf("\nDa li je kraj smene? [Da/Ne] ");
109            scanf("%s", odgovor);
110            if (strcmp(odgovor, "Da") == 0)
111                break;
112            else
113                broj_usluzenih = 0;
114        }
115    }
116 /**
117 /* Datoteka vise nije potrebna i treba je zatvoriti. */
118 fclose(izlaz);
119
120 /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
121 neusluzenih korisnika, u tom slucaju treba osloboediti memoriju
122 koju zauzima red sa neobradjenim zahtevima korisnika. */

```

```

122     osloboodi_red(&pocetak, &kraj);
124     exit(EXIT_SUCCESS);
}

```

Rešenje 4.11

dvostruko_povezana_lista.h

```

1 #ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
#define _DVOSTRUKO_POVEZANA_LISTA_H_

3 /* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
5    vrednost i pokazivace na sledeci i prethodni cvor liste. */
7 typedef struct cvor {
9     int vrednost;
10    struct cvor *sledeci;
12    struct cvor *prethodni;
13} Cvor;

15 /* Funkcija kreira cvor, vrednost novog cvora inicializuje na broj,
17    dok pokazivac na sledeci cvor postavlja na NULL. Vraca pokazivac
19    na novokreirani cvor ili NULL ako alokacija nije bila uspesna. */
20 Cvor *napravi_cvor(int broj);

22 /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
24    ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji na
26    adresi adresa_kraja. */
27 void osloboodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);

29 /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
31    bilo greske pri alokaciji memorije, inace vraca 0. */
32 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
34                               adresa_kraja, int broj);

36 /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo greske
38    pri alokaciji memorije, inace vraca 0. */
39 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
41                           int broj);

43 /* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
45    novi cvor sa vrednoscu broj. */
46 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

48 /* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
50    dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
51 int dodaj_iza(Cvor * tekuci, int broj);

53 /* Funkcija dodaje broj u sortiranu listu tako da lista ostane
55    sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji memorije,
57    inace vraca 0. */
58

```

```

    inace vraca 0. */
43 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                      broj);

45
/* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
47   Vraca pokazivac na cvor liste u kome je sadrzan trazeni broj ili
48   NULL u slucaju da takav cvor ne postoji u listi. */
49 Cvor *pretrazi_listu(Cvor * glava, int broj);

51 /* Funkcija trazi u listi cvor cija je vrednost jednaka datom broju.
52   U pretrazi oslanja se na cinjenicu da je lista koja se pretrazuje
53   neopadajuce sortirana. Vraca pokazivac na cvor liste u kome je
54   sadrzan trazeni broj ili NULL u slucaju da takav cvor ne postoji.
55 */
55 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

57 /* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi ciji
58   pokazivac glava se nalazi na adresi adresa_glave. */
59 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
                      tekuci);

61
/* Funkcija brise iz liste sve cvorove koji sadrze dati broj. Azurira
63   pokazivac na glavu liste, koji moze biti promenjen u slucaju da se
64   obrije stara glava. */
65 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                      broj);

67
/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
69   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
70   sortirana. Azurira pokazivac na glavu liste, koji moze biti
71   promenjen ukoliko se obrije stara glava liste. */
73 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                         adresa_kraja, int broj);

75 /* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka kraju
76   liste, razdvojene zapetama i uokvirene zagradama. */
77 void ispisi_listu(Cvor * glava);

79 /* Funkcija prikazuje cvorove liste pocevsi od kraja ka glavi liste.
80   */
81 void ispisi_listu_unazad(Cvor * kraj);
81#endif

```

dvostruko_povezana_lista.c

```

2 #include <stdio.h>
2 #include <stdlib.h>
4 #include "dvostruko_povezana_lista.h"

4 Cvor *napravi_cvor(int broj)

```

```

6  {
7      /* Alocira se memorija za novi cvor liste i proverava se uspesnost
8         alokacije */
9      Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10     if (novi == NULL)
11         return NULL;
12
13     /* Inicijalizacija polja strukture */
14     novi->vrednost = broj;
15     novi->sledeci = NULL;
16
17     /* Vraca se adresa novog cvora */
18     return novi;
19 }
20
21 void osloboodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
22 {
23     Cvor *pomocni = NULL;
24
25     /* Ako lista nije prazna, onda treba oslobooditi memoriju */
26     while (*adresa_glave != NULL) {
27         /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
28             oslobooditi memoriju cvora koji predstavlja glavu liste */
29         pomocni = (*adresa_glave)->sledeci;
30         free(*adresa_glave);
31         /* Sledeci cvor je nova glava liste */
32         *adresa_glave = pomocni;
33     }
34     /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj liste
35         treba postaviti na NULL */
36     *adresa_kraja = NULL;
37 }
38
39 int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
40                             adresa_kraja, int broj)
41 {
42     /* Kreira se novi cvor i proverava se uspesnost kreiranja */
43     Cvor *novi = napravi_cvor(broj);
44     if (novi == NULL)
45         return 1;
46
47     /* Sledbenik novog cvora je glava stare liste */
48     novi->sledeci = *adresa_glave;
49
50     /* Ako stara lista nije bila prazna, onda prethodni cvor glave
51         treba da bude novi cvor. Inace, novi cvor je ujedno i pocetni i
52         krajnji */
53     if (*adresa_glave != NULL)
54         (*adresa_glave)->prethodni = novi;
55     else
56         *adresa_kraja = novi;

```

```

58  /* Novi cvor je nova glava liste */
59  *adresa_glave = novi;
60
61  /* Vraca se indikator uspesnog dodavanja */
62  return 0;
63 }
64
65 int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
66                           int broj)
67 {
68  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
69  Cvor *novi = napravi_cvor(broj);
70  if (novi == NULL)
71      return 1;
72
73  /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
74   ujedno i cela lista. Azurira se vrednost na koju pokazuju
75   adresa_glave i adresa_kraja */
76  if (*adresa_glave == NULL) {
77      *adresa_glave = novi;
78      *adresa_kraja = novi;
79  } else {
80      /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
81       sledbenik poslednjeg cvora i azurira se samo pokazivac na kraj
82       liste */
83      (*adresa_kraja)->sledeci = novi;
84      novi->prethodni = (*adresa_kraja);
85      *adresa_kraja = novi;
86  }
87
88  /* Vraca se indikator uspesnog dodavanja */
89  return 0;
90 }

91 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
92 {
93  /* U praznoj listi nema takvog mesta i vraca se NULL */
94  if (glava == NULL)
95      return NULL;
96
97  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
98   pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
99   vrednost vecu ili jednaku od vrednosti novog cvora.
100
101   Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
102   provera da li se doslo do poslednjeg cvora liste pre nego sto se
103   proveri vrednost u sledecem cvoru jer u slucaju poslednjeg,
104   sledeci ne postoji pa ni njegova vrednost. */
105  while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
106      glava = glava->sledeci;
107
108  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do

```

```

110     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeći ima
111     vrednost vecu od broj */
112     return glava;
113 }
114
115 int dodaj_iza(Cvor * tekuci, int broj)
116 {
117     /* Kreira se novi cvor i provera se uspesnost kreiranja */
118     Cvor *novi = napravi_cvor(broj);
119     if (novi == NULL)
120         return 1;
121
122     novi->sledeci = tekuci->sledeci;
123     novi->prethodni = tekuci;
124
125     /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje prethodnik,
126      a potom i tekuci dobija novog sledeceg postavljanjem pokazivaca
127      na ispravne adrese */
128     if (tekuci->sledeci != NULL)
129         tekuci->sledeci->prethodni = novi;
130     tekuci->sledeci = novi;
131
132     /* Vraca se indikator uspesnog dodavanja */
133     return 0;
134 }
135
136 int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
137                      broj)
138 {
139     /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor liste
140      */
141     if (*adresa_glave == NULL) {
142         /* Kreira se novi cvor i proverava se uspesnost kreiranja */
143         Cvor *novi = napravi_cvor(broj);
144         if (novi == NULL)
145             return 1;
146
147         /* Azuriraju se vrednosti pocetka i kraja liste */
148         *adresa_glave = novi;
149         *adresa_kraja = novi;
150
151         /* Vraca se indikator uspesnog dodavanja */
152         return 0;
153     }
154
155     /* Ukoliko je vrednost glave liste veca ili jednaka od nove
156      vrednosti onda novi cvor treba staviti na pocetak liste */
157     if ((*adresa_glave)->vrednost >= broj) {
158         return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
159     }
160
161     /* Pronazi se cvor iza koga treba uvezati novi cvor */

```

```

162     Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
163     /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
164     if (dodaj_iza(pomocni, broj) == 1)
165         return 1;
166     /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
167      novi cvor poslednji u listi. */
168     if (pomocni == *adresa_kraja)
169         *adresa_kraja = pomocni->sledeci;
170
171     return 0;
172 }
173
174 Cvor *pretrazi_listu(Cvor * glava, int broj)
175 {
176     /* Obilaze se cvorovi liste */
177     for (; glava != NULL; glava = glava->sledeci)
178         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
179          se obustavlja */
180         if (glava->vrednost == broj)
181             return glava;
182
183     /* Nema trazenog broja u listi i vraca se NULL */
184     return NULL;
185 }
186
187 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
188 {
189     /* Obilaze se cvorovi liste */
190     /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
191     for (; glava != NULL && glava->vrednost <= broj;
192         glava = glava->sledeci)
193         /* Ako je vrednost tekuceg cvora jednaka zadatom broju, pretraga
194          se obustavlja */
195         if (glava->vrednost == broj)
196             return glava;
197
198     /* Nema trazenog broja u listi i bice vraceno NULL */
199     return NULL;
200 }
201
202 /* Kod dvostruko povezane liste brisanje odredjenog cvora se moze
203 lako realizovati jer on sadrzi pokazivace na svog sledbenika i
204 prethodnika u listi. U funkciji se bise cvor zadat argumentom
205 tekuci */
206 void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja, Cvor *
207                      tekuci)
208 {
209     /* Ako je tekuci NULL pokazivac, nema sta da se brise */
210     if (tekuci == NULL)
211         return;
212
213     /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da

```

```

138     njegov sledbenik bude sledbenik tekuceg cvora */
139     if (tekuci->prethodni != NULL)
140         tekuci->prethodni->sledeci = tekuci->sledeci;
141
142     /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
143      treba da bude prethodnik tekuceg cvora */
144     if (tekuci->sledeci != NULL)
145         tekuci->sledeci->prethodni = tekuci->prethodni;
146
147     /* Ako je glava cvor koji se brise, nova glava liste ce biti
148      sledbenik stare glave */
149     if (tekuci == *adresa_glave)
150         *adresa_glave = tekuci->sledeci;
151
152     /* Ako je cvor koji se brise poslednji u listi, azurira se i
153      pokazivac na kraj liste */
154     if (tekuci == *adresa_kraja)
155         *adresa_kraja = tekuci->prethodni;
156
157     /* Oslobadja se dinamicki alociran prostor za cvor tekuci */
158     free(tekuci);
159 }
160
161 void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int broj
162 )
163 {
164     Cvor *tekuci = *adresa_glave;
165
166     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broj, takvi
167      cvorovi se brisu iz liste. */
168     while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
169         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
170 }
171
172 void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
173                                     adresa_kraja, int broj)
174 {
175     Cvor *tekuci = *adresa_glave;
176
177     /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
178      takvi cvorovi se brisu iz liste. */
179     while ((tekuci =
180             pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
181         obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
182 }
183
184 void ispisi_listu(Cvor * glava)
185 {
186     putchar('[');
187     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
188      pocetka prema kraju liste */
189     for (; glava != NULL; glava = glava->sledeci) {

```

```

264     printf("%d", glava->vrednost);
265     if (glava->sledeci != NULL)
266         printf(", ");
267 }
268
269     printf("]\n");
270 }

271 void ispisi_listu_unazad(Cvor * kraj)
272 {
273     putchar('[');
274     /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od kraja
275      prema pocetku liste */
276     for (; kraj != NULL; kraj = kraj->prethodni) {
277         printf("%d", kraj->vrednost);
278         if (kraj->prethodni != NULL)
279             printf(", ");
280     }
281     printf("]\n");
282 }
```

main_a.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"

5 /* 1) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku */
9     /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
10      da bi operacije poput dodavanja na kraj liste i ispisivanja
11      liste unazad bile efikasne poput dodavanja na pocetak liste i
12      ispisivanja liste od pocetnog do poslednjeg cvora. */
13     Cvor *glava = NULL;
14     Cvor *kraj = NULL;
15     Cvor *trazeni = NULL;
16     int broj;

17     /* Testira se funkcija za dodavanja novog broja na pocetak liste */
18     printf("Unesite brojeve: (za kraj CTRL+D)\n");
19     while (scanf("%d", &broj) > 0) {
20         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
21            memorije za novi cvor. Memoriju alociranu za cvorove liste
22            treba osloboditi pre napustanja programa */
23         if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
24             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
25             osloboodi_listu(&glava, &kraj);
26             exit(EXIT_FAILURE);
27     }
```

```

29     printf("\tLista: ");
30     ispisi_listu(glava);
31 }
32
33 /* Testira se funkcija za pretragu liste */
34 printf("\nUnesite broj koji se trazi u listi: ");
35 scanf("%d", &broj);
36
37 /* Pokazivac traženi dobija vrednost rezultata pretrage */
38 traženi = pretrazi_listu(glava, broj);
39 if (traženi == NULL)
40     printf("Broj %d se ne nalazi u listi!\n", broj);
41 else
42     printf("Traženi broj %d je u listi!\n", traženi->vrednost);
43
44 /* Ispisuje se lista unazad */
45 printf("\nLista ispisana u nazad: ");
46 ispisi_listu_unazad(kraj);
47
48 /* Oslobadja se memorija zauzeta za cvorove liste */
49 oslobodi_listu(&glava, &kraj);
50
51 exit(EXIT_SUCCESS);
52 }
```

main_b.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"
4
5 /* 2) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku. */
9     Cvor *glava = NULL;
10    Cvor *kraj = NULL;
11    int broj;
12
13    /* Testira se funkcija za dodavanja novog broja na kraj liste */
14    printf("Unesite brojeve: (za kraj unesite CTRL+D)\n");
15    while (scanf("%d", &broj) > 0) {
16        /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
17           memorije za novi cvor. Memoriju alociranu za cvorove liste
18           treba osloboditi pre napustanja programa */
19        if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
20            fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
21            oslobodi_listu(&glava, &kraj);
22            exit(EXIT_FAILURE);
23        }
24        printf("\tLista: ");
```

```

    ispisi_listu(glava);
26 }

28 /* Testira se funkcija za brisanje elemenata iz liste */
29 printf("\nUnesite broj koji se brise iz liste: ");
30 scanf("%d", &broj);

32 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
   procitanom sa ulaza */
34 obrisi_cvor(&glava, &kraj, broj);

36 printf("Lista nakon brisanja:  ");
37 ispisi_listu(glava);

38 /* Ispisuje se lista unazad */
39 printf("\nLista ispisana u nazad: ");
40 ispisi_listu_unazad(kraj);

42 /* Oslobadja se memorija zauzeta za cvorove liste */
43 osloboodi_listu(&glava, &kraj);

46 exit(EXIT_SUCCESS);
}

```

main_c.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "dvostruko_povezana_lista.h"

5 /* 3) Glavni program */
6 int main()
7 {
8     /* Lista je prazna na pocetku */
9     Cvor *glava = NULL;
10    Cvor *kraj = NULL;
11    Cvor *trazeni = NULL;
12    int broj;

13   /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
14      bude uredjena neopadajuce */
15   printf("Unosite brojeve (za kraj unesite CTRL+D)\n");
16   while (scanf("%d", &broj) > 0) {
17       /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
18          memorije za novi cvor. Memoriju alociranu za cvorove liste
19          treba oslobooditi pre napustanja programa */
20       if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
21           fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
22           osloboodi_listu(&glava, &kraj);
23           exit(EXIT_FAILURE);
24   }

```

```

27     printf("\tLista: ");
28     ispisi_listu(glava);
29 }
30
31 /* Testira se funkcija za pretragu liste */
32 printf("\nUnesite broj koji se trazi u listi: ");
33 scanf("%d", &broj);
34
35 /* Pokazivac traženi dobija vrednost rezultata pretrage */
36 traženi = pretrazi_listu(glava, broj);
37 if (traženi == NULL)
38     printf("Broj %d se ne nalazi u listi!\n", broj);
39 else
40     printf("Traženi broj %d je u listi!\n", traženi->vrednost);
41
42 /* Testira se funkcija za brisanje elemenata iz liste */
43 printf("\nUnesite broj koji se brise iz liste: ");
44 scanf("%d", &broj);
45
46 /* Brisu se cvorovi iz liste cije polje vrednost je jednako broju
47   procitanom sa ulaza */
48 obrisi_cvor_sortirane_liste(&glava, &kraj, broj);
49
50 printf("Lista nakon brisanja: ");
51 ispisi_listu(glava);
52
53 /* Ispisuje se lista unazad */
54 printf("\nLista ispisana u nazad: ");
55 ispisi_listu_unazad(kraj);
56
57 /* Oslobadja se memorija zauzeta za cvorove liste */
58 oslobođi_listu(&glava, &kraj);
59
60 exit(EXIT_SUCCESS);
}

```

Rešenje 4.14

stabla.h

```

1 #ifndef _STABLA_H_
2 #define _STABLA_H_ 1
3
4 /* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
5    stabla */
6 typedef struct cvor {
7     int broj;
8     struct cvor *levo;
9     struct cvor *desno;
10 } Cvor;

```

```

12 /* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
14 Cvor *napravi_cvor(int broj);

16 /* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
   funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je doslo
   do greske */
18 int dodaj_u_stablo(Cvor ** adresa_korena, int broj);

20 /* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
22 Cvor *pretrazi_stablo(Cvor * koren, int broj);

24 /* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
26 Cvor *pronadji_najmanji(Cvor * koren);

28 /* f) Funkcija koja pronalazi cvor koji sadrzi najvecu vrednost u
   stablu */
30 Cvor *pronadji_najveci(Cvor * koren);

32 /* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
34 void obrisi_element(Cvor ** adresa_korena, int broj);

36 /* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   postablo - Koren - Desno podstablo ) */
38 void ispisi_stablo_infiksno(Cvor * koren);

40 /* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
42 void ispisi_stablo_prefiksno(Cvor * koren);

44 /* j) Funkcija koja ispisuje stablo postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren) */
46 void ispisi_stablo_postfiksno(Cvor * koren);

48 /* k) Funkcija koja oslobadja memoriju zauzetu stablom. */
50 void oslobodi_stablo(Cvor ** adresa_korena);

52 #endif

```

stabla.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 Cvor *napravi_cvor(int broj)
6 {
7     /* Alocira se memorija za novi cvor i proverava se uspesnost
8     alokacije. */

```

```

10    Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
11    if (novi == NULL)
12        return NULL;
13
14    /* Inicijalizuju se polja novog cvora. */
15    novi->broj = broj;
16    novi->levo = NULL;
17    novi->desno = NULL;
18
19    /* Vraca se adresa novog cvora. */
20    return novi;
21}
22
23int dodaj_u_stablo(Cvor **adresa_korena, int broj)
24{
25    /* Ako je stablo prazno */
26    if (*adresa_korena == NULL) {
27
28        /* Kreira se novi cvor */
29        Cvor *novi_cvor = napravi_cvor(broj);
30
31        /* Proverava se uspesnost kreiranja */
32        if (novi_cvor == NULL) {
33
34            /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
35             */
36            return 1;
37        }
38        /* Inace ... */
39
40        /* Novi cvor se proglašava korenom stabla */
41        *adresa_korena = novi_cvor;
42
43        /* I vraca se indikator uspesnosti kreiranja */
44        return 0;
45    }
46
47    /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za zadati
48     broj */
49
50    /* Ako je zadata vrednost manja od vrednosti korena */
51    if (broj < (*adresa_korena)->broj)
52
53        /* Broj se dodaje u levo podstablo */
54        return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
55
56    else
57        /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
58         dodaje u desno podstablo */
59        return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
60}

```

```

62 Cvor *pretrazi_stablo(Cvor * koren, int broj)
63 {
64     /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
65     if (koren == NULL)
66         return NULL;
67
68     /* Ako je trazena vrednost sadrzana u korenu */
69     if (koren->broj == broj) {
70
71         /* Prekidamo pretragu */
72         return koren;
73     }
74
75     /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
76     if (broj < koren->broj)
77
78         /* Pretraga se nastavlja u levom podstablu */
79         return pretrazi_stablo(koren->levo, broj);
80
81     else
82         /* U suprotnom, pretraga se nastavlja u desnom podstablu */
83         return pretrazi_stablo(koren->desno, broj);
84 }
85
86 Cvor *pronadji_najmanji(Cvor * koren)
87 {
88
89     /* Ako je stablo prazno, prekida se pretraga */
90     if (koren == NULL)
91         return NULL;
92
93     /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
94      levo od njega */
95
96     /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
97      najmanju vrednost */
98     if (koren->levo == NULL)
99         return koren;
100
101    /* Inace, pretragu treba nastaviti u levom podstablu */
102    return pronadji_najmanji(koren->levo);
103 }
104
105 Cvor *pronadji_najveci(Cvor * koren)
106 {
107
108     /* Ako je stablo prazno, prekida se pretraga */
109     if (koren == NULL)
110         return NULL;
111
112     /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
113      desno od njega */
114 }
```

```

114  /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
115   najvecu vrednost */
116  if (koren->desno == NULL)
117    return koren;
118
119  /* Inace, pretragu treba nastaviti u desnom podstablu */
120  return pronadji_najveci(koren->desno);
121 }
122
123 void obrisi_element(Cvor ** adresa_korena, int broj)
124 {
125   Cvor *pomocni_cvor = NULL;
126
127   /* Ako je stablo prazno, brisanje nije primenljivo */
128   if (*adresa_korena == NULL)
129     return;
130
131   /* Ako je vrednost koju treba obrisati manja od vrednosti u korenu
132    stabla, ona se eventualno nalazi u levom podstablu, pa treba
133    rekurzivno primeniti postupak na levo podstablu. Koren ovako
134    modifikovanog stabla je nepromenjen. */
135   if (broj < (*adresa_korena)->broj) {
136     obrisi_element(&(*adresa_korena)->levo, broj);
137     return;
138   }
139
140   /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
141    stabla, ona se eventualno nalazi u desnom podstablu pa treba
142    rekurzivno primeniti postupak na desno podstablu. Koren ovako
143    modifikovanog stabla je nepromenjen. */
144   if ((*adresa_korena)->broj < broj) {
145     obrisi_element(&(*adresa_korena)->desno, broj);
146     return;
147   }
148
149   /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
150    jednaka broju koji se brise (tj. slucaj kada treba obrisati
151    koren) */
152
153   /* Ako koren nema sinova, tada se on prosto brise, i rezultat je
154    prazno stablo (vraca se NULL) */
155   if ((*adresa_korena)->levo == NULL
156       && (*adresa_korena)->desno == NULL) {
157     free(*adresa_korena);
158     *adresa_korena = NULL;
159     return;
160   }
161
162   /* Ako koren ima samo levog sina, tada se brisanje vrsti tako sto se
163    brise koren, a novi koren postaje levi sin */
164   if ((*adresa_korena)->levo != NULL

```

```

166     && (*adresa_korena)->desno == NULL) {
167     pomocni_cvor = (*adresa_korena)->levo;
168     free(*adresa_korena);
169     *adresa_korena = pomocni_cvor;
170     return;
171 }
172 /* Ako koren ima samo desnog sina, tada se brisanje vrši tako što
173    se brise koren, a novi koren postaje desni sin */
174 if ((*adresa_korena)->desno != NULL
175     && (*adresa_korena)->levo == NULL) {
176     pomocni_cvor = (*adresa_korena)->desno;
177     free(*adresa_korena);
178     *adresa_korena = pomocni_cvor;
179     return;
180 }
181 /* Slučaj kada koren ima oba sina - najpre se potrazi sledbenik
182    korena (u smislu poretka) u stablu. To je upravo po vrednosti
183    najmanji cvor u desnom podstablju. On se može pronaci npr.
184    funkcijom pronadji_najmanji(). Nakon toga se u koren smesti
185    vrednost tog cvora, a u taj cvor se smesti vrednost korena (tj.
186    broj koji se brise). Zatim se prosto rekurzivno pozove funkcija
187    za brisanje na desno podstablo. S obzirom da u njemu treba
188    obrisati najmanji element, a on zasigurno ima najviše jednog
189    potomka, jasno je da će njegovo brisanje biti obavljen na jedan
190    od jednostavnijih nacija koji su gore opisani. */
191 pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
192 (*adresa_korena)->broj = pomocni_cvor->broj;
193 pomocni_cvor->broj = broj;
194 obrisi_element(&(*adresa_korena)->desno, broj);
195 }
196 }
197 void ispisi_stablo_infiksno(Cvor * koren)
198 {
199 /* Ako stablo nije prazno */
200 if (koren != NULL) {
201
202     /* Prvo se ispisuju svi cvorovi levo od korena */
203     ispisi_stablo_infiksno(koren->levo);
204
205     /* Zatim se ispisuje vrednost u korenu */
206     printf("%d ", koren->broj);
207
208     /* Na kraju se ispisuju cvorovi desno od korena */
209     ispisi_stablo_infiksno(koren->desno);
210 }
211 }
212 }
213 void ispisi_stablo_prefiksno(Cvor * koren)
214 {
215 /* Ako stablo nije prazno */
216

```

```

218 if (koren != NULL) {
219     /* Prvo se ispisuje vrednost u korenu */
220     printf("%d ", koren->broj);
221
222     /* Zatim se ispisuju svi cvorovi levo od korena */
223     ispisi_stablo_prefiksno(koren->levo);
224
225     /* Na kraju se ispisuju svi cvorovi desno od korena */
226     ispisi_stablo_prefiksno(koren->desno);
227 }
228
229 void ispisi_stablo_postfiksno(Cvor * koren)
230 {
231     /* Ako stablo nije prazno */
232     if (koren != NULL) {
233
234         /* Prvo se ispisuju svi cvorovi levo od korena */
235         ispisi_stablo_postfiksno(koren->levo);
236
237         /* Zatim se ispisuju svi cvorovi desno od korena */
238         ispisi_stablo_postfiksno(koren->desno);
239
240         /* Na kraju se ispisuje vrednost u korenu */
241         printf("%d ", koren->broj);
242     }
243 }
244
245 void osloboodi_stablo(Cvor ** adres_a_korena)
246 {
247     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
248     if (*adres_a_korena == NULL)
249         return;
250
251     /* Inace ... */
252     /* Oslobadja se memorija zauzeta levim podstablom */
253     osloboodi_stablo(&(*adres_a_korena)->levo);
254
255     /* Oslobadja se memorija zauzeta desnim podstablom */
256     osloboodi_stablo(&(*adres_a_korena)->desno);
257
258     /* Oslobadja se memorija zauzeta korenom */
259     free(*adres_a_korena);
260
261     /* Proglasava se stablo praznim */
262     *adres_a_korena = NULL;
263 }

```

main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"

5 int main()
{
7     Cvor *koren;
8     int n;
9     Cvor *trazeni_cvor;

11    /* Proglasava se stablo praznim */
12    koren = NULL;
13
15    /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
16       dodavanja */
17    printf("Unesite brojeve (CTRL+D za kraj unosa): ");
18    while (scanf("%d", &n) != EOF) {
19        if (dodaj_u_stablo(&koren, n) == 1) {
20            fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
21            oslobodi_stablo(&koren);
22            return 0;
23        }
24    }
25
26    /* Generisu se trazeni ispisi: */
27    ispisi_stablo_infiksno(koren);
28    printf("\nPrefiksni ispis: ");
29    ispisi_stablo_prefiksno(koren);
30    printf("\nPostfiksni ispis: ");
31    ispisi_stablo_postfiksno(koren);

32    /* Demonstrira se rad funkcije za pretragu */
33    printf("\nTrazi se broj: ");
34    scanf("%d", &n);
35    trazeni_cvor = pretrazi_stablo(koren, n);
36    if (trazeni_cvor == NULL)
37        printf("Broj se ne nalazi u stablu!\n");
38
39    else
40        printf("Broj se nalazi u stablu!\n");
41
42    /* Demonstrira se rad funkcije za brisanje */
43    printf("Brise se broj: ");
44    scanf("%d", &n);
45    obrisi_element(&koren, n);
46    printf("Rezultujuce stablo: ");
47    ispisi_stablo_infiksno(koren);
48    printf("\n");
49
50    /* Oslobadja se memorija zauzeta stablom */

```

```

53     oslobodi_stablo(&koren);
54
55 }

```

Rešenje 4.15

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 #define MAX 50
7
8 /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
   pojavljivanja i redom pokazivace na levo i desno podstablo */
9
10 typedef struct cvor {
11     char *rec;
12     int brojac;
13     struct cvor *levo;
14     struct cvor *desno;
15 } Cvor;
16
17 /* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec)
19 {
20     /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije. */
21     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
22     if (novi_cvor == NULL)
23         return NULL;
24
25     /* Alocira se memorija za zadatu rec: potrebno je rezervisati
       memoriju za svaki karakter reci uključujući i terminirajući nulu
       */
26     novi_cvor->rec = (char *) malloc((strlen(rec) + 1) * sizeof(char));
27     if (novi_cvor->rec == NULL) {
28         free(novi_cvor);
29         return NULL;
30     }
31
32     /* Inicijalizuju se polja u novom cvoru */
33     strcpy(novi_cvor->rec, rec);
34     novi_cvor->brojac = 1;
35     novi_cvor->levo = NULL;
36     novi_cvor->desno = NULL;
37
38     /* Vraca se adresa novog cvora */
39     return novi_cvor;
40 }
41
42
43
44

```

```

46  /* Funkcija koja dodaje novu rec u stablo - ukoliko je dodavanje
47    uspesno povratna vrednost je 0, u suprotnom povratna vrednost je 1
48 */
49 int dodaj_u_stablo(Cvor ** adresu_korena, char *rec)
50 {
51     /* Ako je stablo prazno */
52     if (*adresu_korena == NULL) {
53         /* Kreira se cvor koji sadrzi zadatu rec */
54         Cvor *novi_cvor = napravi_cvor(rec);
55         /* Proverava se uspesnost kreiranja novog cvora */
56         if (novi_cvor == NULL) {
57             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
58             */
59             return 1;
60         }
61         /* Inace... */
62         /* Novi cvor se proglašava korenom stabla */
63         *adresu_korena = novi_cvor;
64
65         /* I vraca se indikator uspesnog dodavanja */
66         return 0;
67     }
68
69     /* Ako stablo nije prazno, trazi odgovarajuca pozicija za novu rec
70     */
71
72     /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
73      levo podstablo */
74     if (strcmp(rec, (*adresu_korena)->rec) < 0)
75         return dodaj_u_stablo(&(*adresu_korena)->levo, rec);
76
77     else
78         /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
79          desno podstablo */
80     if (strcmp(rec, (*adresu_korena)->rec) > 0)
81         return dodaj_u_stablo(&(*adresu_korena)->desno, rec);
82
83     else
84         /* Ako je rec jednaka reci u korenu, uvecava se njen broj
85          pojavljivanja */
86         (*adresu_korena)->brojac++;
87     }
88
89     /* Funkcija koja oslobođa memoriju zauzetu stablom */
90 void osloboodi_stablo(Cvor ** adresu_korena)
91 {
92     /* Ako je stablo prazno, nepotrebno je oslobođati memoriju */
93     if (*adresu_korena == NULL)
94         return;
95
96     /* Inace ... */
97     /* Oslobođaja se memorija zauzeta levim podstablom */

```

```

96    osloboodi_stablo(&(*adresa_korena)->levo);

98    /* Oslobadja se memorija zauzeta desnim podstablom */
100   osloboodi_stablo(&(*adresa_korena)->desno);

102   /* Oslobadja se memorija zauzeta korenom */
103   free((*adresa_korena)->rec);
104   free(*adresa_korena);

106   /* Stablo se proglašava praznim */
107   *adresa_korena = NULL;
108 }

110   /* Funkcija koja pronađe cvor koji sadrži najfrekventniju rec (rec
111    sa najvećim brojem pojavljivanja) */
112 Cvor *nadji_najfrekventniju_rec(Cvor * koren)
113 {
114     Cvor *max, *max_levo, *max_desno;

116     /* Ako je stablo prazno, prekida se sa pretragom */
117     if (koren == NULL)
118         return NULL;

120     /* Pronađe se najfrekventnija rec u levom podstablu */
121     max_levo = nadji_najfrekventniju_rec(koren->levo);

122     /* Pronađe se najfrekventnija rec u desnem podstablu */
123     max_desno = nadji_najfrekventniju_rec(koren->desno);

125     /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
126      podstabla, korena i desnog podstabla */
127     max = koren;
128     if (max_levo != NULL && max_levo->brojac > max->brojac)
129         max = max_levo;
130     if (max_desno != NULL && max_desno->brojac > max->brojac)
131         max = max_desno;

133     /* Vraca se adresa cvora sa najvećim brojem pojavljivanja */
134     return max;
135 }

137   /* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
138    prema brojem pojavljivanja */
139 void prikazi_stablo(Cvor * koren)
140 {
141     /* Ako je stablo prazno, završava se sa ispisom */
142     if (koren == NULL)
143         return;

144     /* Zbog leksikografskog poretkova, prvo se ispisuju sve reci iz levog
145      podstabla */
146     prikazi_stablo(koren->levo);

```

```

148     /* Zatim rec iz korena */
150     printf("%s: %d\n", koren->rec, koren->brojac);
152     /* I nastavlja se sa ispisom reci iz desnog podstabla */
153     prikazi_stablo(koren->desno);
154 }
155
156 /* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
157 niz rec. Maksimalna duzina reci je odredjena argumentom max.
158 Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
159 suprotnom. Rec je niz malih ili velikih slova. */
160 int procitaj_rec(FILE * f, char rec[], int max)
161 {
162     /* Karakter koji se cita */
163     int c;
164
165     /* Indeks pozicije na koju se smesta procitani karakter */
166     int i = 0;
167
168     /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
169      nije stiglo do kraja datoteke... */
170     while (i < max - 1 && (c = fgetc(f)) != EOF) {
171         /* Proverava se da li je procitani karakter slovo */
172         if (isalpha(c))
173             /* Ako jeste, smesta se u niz - pritom se vrsti konverzija u
174              mala slova jer program treba da bude neosetljiv na razliku
175              izmedju malih i velikih slova */
176             rec[i++] = tolower(c);
177
178         else
179             /* Ako nije, proverava se da li je procitano barem jedno slovo
180               nove reci */
181             /* Ako jeste, prekida se sa citanjem */
182             if (i > 0)
183                 break;
184
185         /* U suprotnom se ide na sledecu iteraciju */
186     }
187
188     /* Dodaje se na rec terminirajuca nula */
189     rec[i] = '\0';
190
191     /* Vraca se 0 ako je procitana rec, tj. EOF u suprotnom */
192     return i > 0 ? 0 : EOF;
193 }
194
195 int main(int argc, char **argv)
196 {
197     Cvor *koren = NULL, *max;
198     FILE *f;
199     char rec[MAX];

```

```

200  /* Provera da li je navedeno ime datoteke prilikom pokretanja
202   programa */
204   if (argc < 2) {
205     fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
206     exit(EXIT_FAILURE);
207   }
208
209   /* Priprema datoteke za citanje */
210   if ((f = fopen(argv[1], "r")) == NULL) {
211     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
212             argv[1]);
213     exit(EXIT_FAILURE);
214   }
215
216   /* Ucitavanje reci iz datoteke i smestanje u binarno stablo
217    pretrage uz proveru uspesnosti dodavanja */
218   while (procitaj_rec(f, rec, MAX) != EOF) {
219     if (dodaj_u_stablo(&koren, rec) == 1) {
220       fprintf(stderr, "Neuspelo dodavanje reci %s\n", rec);
221       oslobodi_stablo(&koren);
222       exit(EXIT_FAILURE);
223     }
224   }
225
226   /* Posto je citanjem reci zavrseno, zatvara se datoteka */
227   fclose(f);
228
229   /* Prikazuju se sve reci iz teksta i brojevi njihovih
230    pojavljivanja. */
231   prikazi_stablo(koren);
232
233   /* Pronalazi se najfrekventnija rec */
234   max = nadji_najfrekventniju_rec(koren);
235
236   /* Ako takve reci nema... */
237   if (max == NULL)
238
239     /* Ispisuje se odgovarajuce obavestenje */
240     printf("U tekstu nema reci!\n");
241
242   else
243     /* Inace, ispisuje se broj pojavljivanja reci */
244     printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
245           max->rec, max->brojac);
246
247   /* Oslobadja se dinamicki alociran prostor za stablo */
248   oslobodi_stablo(&koren);
249
250   exit(EXIT_SUCCESS);
251 }
```

Rešenje 4.16

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>

6 #define MAX_IME_DATOTEKE 50
7 #define MAX_CIFARA 13
8 #define MAX_IME_I_PREZIME 100

10 /* Struktura kojom se opisuje cvor stabla: sadrži ime i prezime, broj
    telefona i redom pokazivace na levo i desno podstablo */
12 typedef struct cvor {
13     char ime_i_prezime[MAX_IME_I_PREZIME];
14     char telefon[MAX_CIFARA];
15     struct cvor *levo;
16     struct cvor *desno;
17 } Cvor;

18 /* Funkcija koja kreira novi cvora stabla */
19 Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
20 {
21     /* Alocira se memorija za novi cvor i proverava se uspesnost
        alokacije. */
22     Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
23     if (novi_cvor == NULL)
24         return NULL;

25     /* Inicijalizuju se polja novog cvora */
26     strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
27     strcpy(novi_cvor->telefon, telefon);
28     novi_cvor->levo = NULL;
29     novi_cvor->desno = NULL;

30     /* Vraca se adresa novog cvora */
31     return novi_cvor;
32 }

33 /* Funkcija koja dodaje novu osobu i njen broj telefona u stablo -
    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
    povratna vrednost je 1 */
34 int
35 dodaj_u_stablo(Cvor **adresa_korena, char *ime_i_prezime,
36                  char *telefon)
37 {
38     /* Ako je stablo prazno */
39     if (*adresa_korena == NULL) {
40         /* Kreira se novi cvor */
41         Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
42         /* Proverava se uspesnost kreiranja novog cvora */
43         if (novi_cvor == NULL) {
44             /* Povrati 1 jer je stablo prazno */
45             return 1;
46         }
47         /* Ukoliko je novi cvor uspesno kreiran, postavlja ga na
            poziciju koju je zadat u parametru */
48         *adresa_korena = novi_cvor;
49     }
50 }
```

```

52     /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
53     */
54     return 1;
55 }
56 /* Inace... */
57 /* Novi cvor se proglašava korenom stabla */
58 *adresa_korena = novi_cvor;
59
60 /* I vraca se indikator uspesnog dodavanja */
61 return 0;
62 }
63
64 /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
65   unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
66   treba da bude pretrazivacko po ovom polju */
67
68 /* Ako je zadato ime i prezime leksikografski manje od imena i
69   prezimena sadrzanog u korenju, podaci se dodaju u levo podstablo
70   */
71 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
72     < 0)
73     return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
74                           telefon);
75
76 else
77     /* Ako je zadato ime i prezime leksikografski vece od imena i
78       prezimena sadrzanog u korenju, podaci se dodaju u desno
79       podstablo */
80 if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
81     return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
82                           telefon);
83 }
84
85 /* Funkcija koja oslobadja memoriju zauzetu stablom */
86 void osloboodi_stablo(Cvor ** adresakorena)
87 {
88     /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
89     if (*adresa_korena == NULL)
90         return;
91
92     /* Inace ... */
93     /* Oslobadja se memorija zauzeta levim podstablom */
94     osloboodi_stablo(&(*adresa_korena)->levo);
95
96     /* Oslobadja se memorija zauzeta desnim podstablom */
97     osloboodi_stablo(&(*adresa_korena)->desno);
98
99     /* Oslobadja se memorija zauzeta korenom */
100    free(*adresa_korena);
101
102    /* Stablo se proglašava praznim */
103    *adresa_korena = NULL;

```

```

102 }
104 /* Funkcija koja ispisuje imenik u leksikografskom poretku */
105 /* Napomena: ova funkcija nije trazena u zadatku ali se moze
106   koristiti za proveru da li je stablo lepo kreirano ili ne */
107 void prikazi_stablo(Cvor * koren)
108 {
109   /* Ako je stablo prazno, zavrsava se sa ispisom */
110   if (koren == NULL)
111     return;
112
113   /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
114     podstabla */
115   prikazi_stablo(koren->levo);
116
117   /* Zatim se ispisuju podaci iz korena */
118   printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);
119
120   /* I nastavlja se sa ispisom podataka iz desnog podstabla */
121   prikazi_stablo(koren->desno);
122 }

123 /* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje ime
124   i prezime i broj telefona u odgovarajuce nizove. Maksimalna duzina
125   imena i prezimena odredjena je konstantom MAX_IME_PREZIME, a
126   maksimalna duzina broja telefona konstantom MAX_CIFARA. Funkcija
127   vraca EOF ako nema vise kontakata ili 0 u suprotnom. */
128 int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
129 {
130   /* Karakter koji se cita */
131   int c;
132
133   /* Indeks pozicije na koju se smesta procitani karakter */
134   int i = 0;
135
136   /* Linije datoteke koje se obradjuju su formata Ime Prezime
137     BrojTelefona */
138
139   /* Preskacu se eventualne praznine sa pocetka linije datoteke */
140   while ((c = fgetc(f)) != EOF && isspace(c));
141
142   /* Prvo procitano slovo upisuje se u ime i prezime */
143   if (!feof(f))
144     ime_i_prezime[i++] = c;
145
146   /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
147   cifre tako da se citanje vrsti sve dok se ne naidje na cifru.
148   Pritom treba voditi racuna da li ima dovoljno mesta za smestanje
149   procitanog karaktera i da se slucajno ne dodje do kraja datoteke
150   */
151   while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
152     if (!isdigit(c))

```

```

154     ime_i_prezime[i++] = c;
156
157     else if (i > 0)
158         break;
159
160     /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
161        blanko karaktera */
162     ime_i_prezime[--i] = '\0';
163
164     /* I pocinje se sa citanjem broja telefona */
165     i = 0;
166
167     /* Upisuje se cifra koja je vec procitana */
168     telefon[i++] = c;
169
170     /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
171        karaktera cije prisustvo nije dozvoljeno u broju telefona */
172     while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
173         if (c == '/' || c == '-' || !isdigit(c))
174             telefon[i++] = c;
175         else
176             break;
177
178     /* Upisuje se terminirajuca nula */
179     telefon[i] = '\0';
180
181     /* Vraca se 0 ako je procitan kontakt ili EOF u suprotnom */
182     return !feof(f) ? 0 : EOF;
183 }
184
185 /* Funkcija koja trazi u imeniku osobu sa zadatim imenom i prezimenom
186 */
187 Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
188 {
189     /* Ako je imenik prazan, zavrsava se sa pretragom */
190     if (koren == NULL)
191         return NULL;
192
193     /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
194        zavrsava sa pretragom */
195     if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
196         return koren;
197
198     /* Ako je zadato ime i prezime leksikografski manje od vrednosti u
199        korenu pretraga se nastavlja levo */
200     if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
201         return pretrazi_imenik(koren->levo, ime_i_prezime);
202
203     else
204         /* u suprotnom, pretraga se nastavlja desno */
205         return pretrazi_imenik(koren->desno, ime_i_prezime);

```

```

206 }
208 int main(int argc, char **argv)
{
210     char ime_datoteke[MAX_IME_DATOTEKE];
211     Cvor *koren = NULL;
212     Cvor *trazeni;
213     FILE *f;
214     char ime_i_prezime[MAX_IME_I_PREZIME];
215     char telefon[MAX_CIFARA];
216     char c;
217     int i;
218
219     /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
220     printf("Unesite ime datoteke: ");
221     scanf("%s", ime_datoteke);
222     getchar();
223     if ((f = fopen(ime_datoteke, "r")) == NULL) {
224         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
225                 ime_datoteke);
226         exit(EXIT_FAILURE);
227     }
228
229     /* Citaju se podaci iz datoteke i smestanju u binarno stablo
230      pretrage uz proveru uspesnosti dodavanja */
231     while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
232         if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
233             fprintf(stderr, "Neuspelo dodavanje podataka za osobu %s\n",
234                     ime_i_prezime);
235             oslobodi_stablo(&koren);
236             exit(EXIT_FAILURE);
237         }
238
239     /* Zatvara se datoteka */
240     fclose(f);
241
242     /* Omogucava se pretraga imenika */
243     while (1) {
244         /* Ucitavaja se ime i prezime */
245         printf("Unesite ime i prezime: ");
246         i = 0;
247         while ((c = getchar()) != '\n')
248             ime_i_prezime[i++] = c;
249             ime_i_prezime[i] = '\0';
250
251         /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
252            funkcionalnost */
253         if (strcmp(ime_i_prezime, "KRAJ") == 0)
254             break;
255
256         /* Inace se ispisuje rezultat pretrage */
257         trazeni = pretrazi_imenik(koren, ime_i_prezime);

```

```

258     if (trazeni == NULL)
259         printf("Broj nije u imeniku!\n");
260     else
261         printf("Broj je: %s \n", trazeni->telefon);
262     }
263
264     /* Oslobadja se memorija zauzeta imenikom */
265     osloboodi_stablo(&koren);
266
267     exit(EXIT_SUCCESS);
268 }
```

Rešenje 4.17

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 51
6
7 /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
8    studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9    jezika i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX];
12     char prezime[MAX];
13     double uspeh;
14     double matematika;
15     double jezik;
16     struct cvor_stabla *levo;
17     struct cvor_stabla *desno;
18 } Cvor;
19
20 /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
22                     double matematika, double jezik)
23 {
24     /* Alocira se memorija za novi cvor i proverava se uspesnost
25        alokacije. */
26     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27     if (novi == NULL)
28         return NULL;
29
30     /* Inicijalizuju se polja strukture */
31     strcpy(novi->ime, ime);
32     strcpy(novi->prezime, prezime);
33     novi->uspeh = uspeh;
34     novi->matematika = matematika;
35     novi->jezik = jezik;
36     novi->levo = NULL;
37     novi->desno = NULL;
```

```

39  /* Vraca se adresa kreiranog cvora */
40  return novi;
41 }

43 /* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo -
44    ukoliko je dodavanje uspesno povratna vrednost je 0, u suprotnom
45    povratna vrednost je 1 */
46 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
47                      double uspeh, double matematika, double jezik)
48 {
49  /* Ako je stablo prazno */
50  if (*koren == NULL) {
51    /* Kreira se novi cvor */
52    Cvor *novi_cvor =
53      napravi_cvor(ime, prezime, uspeh, matematika, jezik);
54    /* Proverava se uspesnost kreiranja novog cvora */
55    if (novi_cvor == NULL) {
56      /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
57      */
58      return 1;
59    }
60    /* Inace... */
61    /* Novi cvor se proglašava korenom stabla */
62    *koren = novi_cvor;
63
64    /* I vraca se indikator uspesnog dodavanja */
65    return 0;
66  }
67
68  /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
69    sortirano po ukupnom broju poena */
70  if (uspeh + matematika + jezik >
71      (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
72    return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
73                          matematika, jezik);
74  else
75    return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
76                          matematika, jezik);
77 }

79
80 /* Funkcija kojom se oslobadja memorija zauzeta stablom */
81 void oslobodi_stablo(Cvor ** koren)
82 {
83  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
84  if (*koren == NULL)
85    return;
86
87  /* Inace ... */
88  /* Oslobadja se memorija zauzeta levim podstablom */
89  oslobodi_stablo(&(*koren)->levo);

```

```

91  /* Oslobadja se memorija zauzeta desnim podstabom */
93  oslobadji_stablo(&(*koren)->desno);
95
97  /* Oslobadja se memorija zauzeta korenom */
98  free(*koren);
99
101 /* Stablo se proglašava praznim */
102 *koren = NULL;
103
104 /* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
105  polozili jednaka 0 ispisuju se informacije o ucenicima koji nisu
106  polozili prijemni, a ako je vrednost argumenta razlicita od nule,
107  ispisuju se informacije o ucenicima koji su polozili prijemni */
108 void stampaj(Cvor * koren, int polozili)
109 {
110     /* Stablo je prazno - prekida se sa ispisom */
111     if (koren == NULL)
112         return;
113
114     /* Stampaju se informacije iz levog podstabla */
115     stampaj(koren->levo, polozili);
116
117     /* Stampaju se informacije iz korenog cvora */
118     if (polozili && koren->matematika + koren->jezik >= 10)
119         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
120                koren->prezime, koren->uspeh, koren->matematika,
121                koren->jezik,
122                koren->uspeh + koren->matematika + koren->jezik);
123     else if (!polozili && koren->matematika + koren->jezik < 10)
124         printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
125                koren->prezime, koren->uspeh, koren->matematika,
126                koren->jezik,
127                koren->uspeh + koren->matematika + koren->jezik);
128
129     /* Stampaju se informacije iz desnog podstabla */
130     stampaj(koren->desno, polozili);
131 }
132
133 /* Funkcija koja određuje koliko studenata nije polozilo prijemni
134  ispit */
135 int nisu_polozili(Cvor * koren)
136 {
137     /* Ako je stablo prazno, broj onih koji nisu polozili 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  uvećava za 1 */

```

```

143     if (koren->matematika + koren->jezik < 10)
144         return 1 + nisu_polozili(koren->levo) +
145             nisu_polozili(koren->desno);
146
147     return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
148 }
149
150 int main(int argc, char **argv)
151 {
152     FILE *in;
153     Cvor *koren;
154     char ime[MAX], prezime[MAX];
155     double uspeh, matematika, jezik;
156
157     /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
158     in = fopen("prijemni.txt", "r");
159     if (in == NULL) {
160         fprintf(stderr,
161                 "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
162         exit(EXIT_FAILURE);
163     }
164
165     /* Citanje podataka i dodavanje u stablo uz proveru uspesnosti
166      dodavanja */
167     koren = NULL;
168     while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
169                   &matematika, &jezik) != EOF) {
170         if (dodaj_u_stablo(&koren, ime, prezime, uspeh, matematika, jezik)
171             == 1) {
172             fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
173                     prezime);
174             oslobodi_stablo(&koren);
175             exit(EXIT_FAILURE);
176         }
177     }
178
179     /* Zatvaranje datoteke */
180     fclose(in);
181
182     /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
183     stampaj(koren, 1);
184
185     /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
186      prijemni */
186     if (nisu_polozili(koren) != 0)
187         printf("-----\n");
188
189     /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
190     stampaj(koren, 0);
191
192     /* Oslobadja se memorija zauzeta stablom */

```

```

193     oslobodi_stablo(&koren);
195     exit(EXIT_SUCCESS);
}

```

Rešenje 4.18

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_NISKA 51
6
7 /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
8    osobe, dan i mesec rodjenja i redom pokazivace na levo i desno
9    podstablo */
10 typedef struct cvor_stabla {
11     char ime[MAX_NISKA];
12     char prezime[MAX_NISKA];
13     int dan;
14     int mesec;
15     struct cvor_stabla *levo;
16     struct cvor_stabla *desno;
17 } Cvor;
18
19 /* Funkcija koja kreira novi cvor */
20 Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
22     /* Alocira se memorija */
23     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
24     if (novi == NULL)
25         return NULL;
26
27     /* Inicijalizuju se polja strukture */
28     strcpy(novi->ime, ime);
29     strcpy(novi->prezime, prezime);
30     novi->dan = dan;
31     novi->mesec = mesec;
32     novi->levo = NULL;
33     novi->desno = NULL;
34
35     /* Vraca se adresa novog cvora */
36     return novi;
37 }
38
39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
40    uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
41    dodavanje uspesno povratna vrednost je 0, u suprotnom povratna
42    vrednost je 1 */
43 int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
44                     int dan, int mesec)

```

```

45 {
46     /* Ako je stablo prazno */
47     if (*koren == NULL) {
48
49         /* Kreira se novi cvor */
50         Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
51         /* Proverava se uspesnost kreiranja novog cvora */
52         if (novi_cvor == NULL) {
53             /* I ukoliko je doslo do greske, vraca se odgovarajuca vrednost
54             */
55             return 1;
56         }
57         /* Inace... */
58         /* Novi cvor se proglašava korenom stabla */
59         *koren = novi_cvor;
60
61         /* I vraca se indikator uspesnog dodavanja */
62         return 0;
63     }
64
65     /* Stablo se uređuje po mesecu, a zatim po danu u okviru istog
66      meseca */
67     if (mesec < (*koren)->mesec)
68         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
69     else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
70         return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan, mesec);
71     else
72         return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan, mesec);
73 }
74
75 /* Funkcija vrsti pretragu stabla i vraca cvor sa traženim datumom */
76 Cvor *pretrazi(Cvor * koren, int dan, int mesec)
77 {
78     /* Stablo je prazno, obustavlja se pretraga */
79     if (koren == NULL)
80         return NULL;
81
82     /* Ako je traženi datum u korenu */
83     if (koren->dan == dan && koren->mesec == mesec)
84         return koren;
85
86     /* Ako je mesec traženog datuma manji od meseca sadrzanog u korenu
87      ili ako su meseci isti ali je dan traženog datuma manji od
88      aktuelnog datuma, pretrazuje se levo podstablo - pre toga se
89      svakako proverava da li leva grana postoji - ako ne postoji
90      treba vratiti prvi sledeći, a to je bas vrednost uocenog korena
91      */
92     if (mesec < koren->mesec
93         || (mesec == koren->mesec && dan < koren->dan)) {
94         if (koren->levo == NULL)
95             return koren;

```

```

95     else
96         return pretrazi(koren->levo, dan, mesec);
97     }
98
99     /* Inace se nastavlja pretraga u desnom delu */
100    return pretrazi(koren->desno, dan, mesec);
101}
102
103/* Funkcija koja pronalazi najmanji datum u stablu */
104Cvor *pronadji_najmanji_datum(Cvor * koren)
105{
106    /* Stablo je prazno, obustavlja se pretraga */
107    if (koren == NULL)
108        return NULL;
109
110    /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
111     sadrzi najmanji datum */
112    if (koren->levo == NULL)
113        return koren;
114    else
115        /* Inace, trazimo manji datum u levom podstablu */
116        return pronadji_najmanji_datum(koren->levo);
117}
118
119/* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
120 */
121void datum_u_nisku(int dan, int mesec, char datum[])
122{
123    if (dan < 10) {
124        datum[0] = '0';
125        datum[1] = dan + '0';
126    } else {
127        datum[0] = dan / 10 + '0';
128        datum[1] = dan % 10 + '0';
129    }
130    datum[2] = '.';
131
132    if (mesec < 10) {
133        datum[3] = '0';
134        datum[4] = mesec + '0';
135    } else {
136        datum[3] = mesec / 10 + '0';
137        datum[4] = mesec % 10 + '0';
138    }
139    datum[5] = '.';
140    datum[6] = '\0';
141}
142
143/* Funkcija koja oslobadja memoriju zauzetu stablom */
144void osloboodi_stablo(Cvor ** adresa_korena)
145{
146    /* Stablo je prazno */

```

```

147     if (*adresa_korena == NULL)
148         return;
149
150     /* Oslobadja se memorija zauzeta levim podstablom (ako postoji) */
151     if ((*adresa_korena)->levo)
152         osloboodi_stablo(&(*adresa_korena)->levo);
153
154     /* Oslobadja se memorija zauzeta desnim podstablom (ako postoji) */
155     if ((*adresa_korena)->desno)
156         osloboodi_stablo(&(*adresa_korena)->desno);
157
158     /* Oslobadja se memorija zauzeta korenom */
159     free(*adresa_korena);
160
161     /* Proglasava se stablo praznim */
162     *adresa_korena = NULL;
163 }
164
165 int main(int argc, char **argv)
166 {
167     FILE *in;
168     Cvor *koren;
169     Cvor *slavljenik;
170     char ime[MAX_NISKA], prezime[MAX_NISKA];
171     int dan, mesec;
172     char datum[7];
173
174     /* Provera da li je zadato ime ulazne datoteke */
175     if (argc < 2) {
176         /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
177            programa */
178         fprintf(stderr, "Nedostaje ime ulazne datoteke!\n");
179         exit(EXIT_FAILURE);
180     }
181
182     /* Inace, priprema se datoteka za citanje */
183     in = fopen(argv[1], "r");
184     if (in == NULL) {
185         fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
186                 argv[1]);
187         exit(EXIT_FAILURE);
188     }
189
190     /* I stablo se popunjava podacima uz proveru uspesnosti dodavanja
191        */
192     koren = NULL;
193     while (fscanf(
194             (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
195             if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
196                 fprintf(stderr, "Neuspelo dodavanje podataka za %s %s\n", ime,
197                         prezime);
198                 osloboodi_stablo(&koren);
199             }
200         }
201     }
202 }
```

```

197     exit(EXIT_FAILURE);
198 }
199
200 /* Datoteka se zatvara */
201 fclose(in);
202
203 /* Omogucuje se pretraga podataka */
204 while (1) {
205
206     /* Ucitava se novi datum */
207     printf("Unesite datum: ");
208     if (scanf("%d.%d.", &dan, &mesec) == EOF)
209         break;
210
211     /* Pretrazuje se stablo */
212     slavljenik = pretrazi(koren, dan, mesec);
213
214     /* Ispisuju se pronadjeni podaci */
215
216     /* Ako slavljenik nije pronadjen, to moze znaci da: */
217     /* 1. drvo je prazno */
218     if (slavljenik == NULL && koren == NULL) {
219         printf("Nema podataka o ovom ni o sledecem rođendanu.\n");
220         continue;
221     }
222     /* 2. posle datuma koji je unesen, nema podataka u stablu - u
223      ovom slučaju se pretraga vrši pocevsi od naredne godine i
224      ispisuje se najmanji datum */
225     if (slavljenik == NULL) {
226         slavljenik = pronadji_najmanji_datum(koren);
227         datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
228         printf("Slavljenik: %s %s %s\n", slavljenik->ime,
229               slavljenik->prezime, datum);
230         continue;
231     }
232
233     /* Ako je slavljenik pronadjen, razlikuju se slučajevi: */
234     /* 1. Pronadjeni su tacni podaci */
235     if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
236         printf("Slavljenik: %s %s\n", slavljenik->ime,
237               slavljenik->prezime);
238         continue;
239     }
240
241     /* 2. Pronadjeni su podaci o prvom sledecem rođendanu */
242     datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
243     printf("Slavljenik: %s %s %s\n", slavljenik->ime,
244           slavljenik->prezime, datum);
245 }
246
247 /* Oslobadja se memorija zauzeta stablom */
248 osloboodi_stablo(&koren);

```

```

249     exit(EXIT_SUCCESS);
251 }
```

Rešenje 4.19

```

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

4 /* Uključuje se biblioteka za rad sa stablima */
# include "stabla.h"
6
8 /* Funkcija koja proverava da li su dva stabla koja sadrže cele
brojeve identična. Povratna vrednost funkcije je 1 ako jesu,
odnosno 0 ako nisu */
10 int identitet(Cvor * koren1, Cvor * koren2)
{
12     /* Ako su oba stabla prazna, jednaka su */
13     if (koren1 == NULL && koren2 == NULL)
14         return 1;
16     /* Ako je jedno stablo prazno, a drugo nije, stabla nisu jednakia */
17     if (koren1 == NULL || koren2 == NULL)
18         return 0;
20     /* Ako su oba stabla neprazna i u korenu se nalaze razlike
vrednosti, može se zaključiti da se razlikuju */
21     if (koren1->broj != koren2->broj)
22         return 0;
24     /* Inace, proverava se da li vazi i jednakost levih i desnih
podstabala */
26     return (identitet(koren1->levo, koren2->levo)
27             && identitet(koren1->desno, koren2->desno));
28 }
30
int main()
31 {
32     int broj;
33     Cvor *koren1, *koren2;
35
36     /* Ucitavaju se elementi prvog stabla */
37     koren1 = NULL;
38     printf("Prvo stablo: ");
39     scanf("%d", &broj);
40     while (broj != 0) {
41         if (dodaj_u_stablo(&koren1, broj) == 1) {
42             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
43             osloboodi_stablo(&koren1);
44             return 0;
45     }
```

```

46     scanf("%d", &broj);
47 }
48
49 /* Ucitavaju se elementi drugog stabla */
50 koren2 = NULL;
51 printf("Drugo stablo: ");
52 scanf("%d", &broj);
53 while (broj != 0) {
54     if (dodaj_u_stablo(&koren2, broj) == 1) {
55         fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
56         oslobodi_stablo(&koren2);
57         return 0;
58     }
59     scanf("%d", &broj);
60 }
61
62 /* Poziva se funkcija koja ispituje identitet stabala i ispisuje se
63 njen rezultat. */
64 if (identitet(koren1, koren2))
65     printf("Stabla jesu identicna.\n");
66 else
67     printf("Stabla nisu identicna.\n");
68
69 /* Oslobadja se memorija zauzeta stablima */
70 oslobodi_stablo(&koren1);
71 oslobodi_stablo(&koren2);
72
73 return 0;
74 }
```

Rešenje 4.20

```

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija kreira novo stablo identично stablu koje je dato korenom.
Povratna vrednost funkcije je 0 ukoliko je kopiranje uspesno,
odnosno 1 ukoliko je doslo do greske */
int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
{
    /* Izlaz iz rekurzije */
    if (koren == NULL) {
        *duplikat = NULL;
        return 0;
    }

    /* Duplira se koren stabla i postavlja da bude koren novog stabla
    */
}
```

```

15     *duplikat = napravi_cvor(koren->broj);
16     if (*duplikat == NULL) {
17         return 1;
18     }
19
20
21     /* Rekurzivno se dupliraju levo i desno podstablo i njihove adrese
22      se cuvaju redom u pokazivacima na levo i desno podstablo korena
23      duplikata */
24     int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
25     int kopija_desno =
26         kopiraj_stablo(koren->desno, &(*duplikat)->desno);
27
28     /* Ako je uspesno duplirano i levo i desno podstablo */
29     if (kopija_levo == 0 && kopija_desno == 0)
30         /* Uspesno je duplirano i celo stablo */
31         return 0;
32     /* Inace, prijavljuje se da je doslo do greske */
33     return 1;
34
35 }
36
37 /* Funkcija izracunava uniju dva skupa predstavljena stablima -
38   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.
39   Povratna vrednost funkcije je 0 ukoliko je kreiranje unije
40   uspesno, odnosno 1 ukoliko je doslo do greske */
41 int kreiraj_uniju(Cvor ** adresu_korena1, Cvor * koren2)
42 {
43
44     /* Ako drugo stablo nije prazno */
45     if (koren2 != NULL) {
46         /* 1. Dodaje se njegov koren u prvo stablo */
47         if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
48             return 1;
49         }
50
51         /* 2. Rekurzivno se racuna unija levog i desnog podstabala drugog
52           stabla sa prvim stablom */
53         int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
54         int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);
55
56         /* Ako je unija podstabala uspesno kreirana */
57         if (unija_levo == 0 && unija_desno == 0)
58             /* Uspesno je kreirana i unija stabala */
59             return 0;
60
61         /* U suprotnom se prijavljuje da je doslo do greske */
62         return 1;
63     }
64
65     /* Ako je drugo stablo prazno, nista se ne preduzima */
66     return 0;
67 }
68
69
70

```

```

72  /* Funkcija izracunava presek dva skupa predstavljana stablima -  

73   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.  

74   Povratna vrednost funkcije je 0 ukoliko je kreiranje preseka  

75   uspesno, odnosno 1 ukoliko je doslo do greske */  

76  int kreiraj_presek(Cvor ** adres_a_koren1, Cvor * koren2)  

77  {  

78    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */  

79    if (*adresa_koren1 == NULL)  

80      return 0;  

81  

82    /* Inace... */  

83    /* Kreira se presek levog i desnog podstabla sa drugim stablom, tj.  

84     iz levog i desnog podstabla prvog stabla brisu se svi oni  

85     elementi koji ne postoje u drugom stablu */  

86    int presek_levo = kreiraj_presek(&(*adresa_koren1)->levo, koren2);  

87    int presek_desno =  

88      kreiraj_presek(&(*adresa_koren1)->desno, koren2);  

89    if (presek_levo == 0 && presek_desno == 0) {  

90      /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se on  

91       uklanja iz prvog stabla */  

92      if (pretrazi_stablo(koren2, (*adresa_koren1)->broj) == NULL)  

93        obrisi_element(adresa_koren1, (*adresa_koren1)->broj);  

94  

95      /* Presek stabala je uspesno kreiran */  

96      return 0;  

97    }  

98    /* Inece, prijavljuje se da je doslo do greske */  

99    return 1;  

100  }  

101  

102  /* Funkcija izracunava razliku dva skupa predstavljana stablima -  

103   rezultujuci skup tj. stablo se dobija modifikacijom prvog stabla.  

104   Povratna vrednost funkcije je 0 ukoliko je kreiranje razlike  

105   uspesno, odnosno 1 ukoliko je doslo do greske */  

106  int kreiraj_razliku(Cvor ** adres_a_koren1, Cvor * koren2)  

107  {  

108    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */  

109    if (*adresa_koren1 == NULL)  

110      return 0;  

111  

112    /* Inace... */  

113    /* Kreira se razlika levog i desnog podstabla sa drugim stablom,  

114     tj. iz levog i desnog podstabla prvog stabla se brisu svi oni  

115     elementi koji postoje i u drugom stablu */  

116    int razlika_levo =  

117      kreiraj_razliku(&(*adresa_koren1)->levo, koren2);  

118    int razlika_desno =  

119      kreiraj_razliku(&(*adresa_koren1)->desno, koren2);  

120    if (razlika_levo == 0 && razlika_desno == 0) {  

121      /* Ako se koren prvog stabla nalazi i u drugom stablu tada se on  

122       uklanja se iz prvog stabla */  

123      if (pretrazi_stablo(koren2, (*adresa_koren1)->broj) != NULL)

```

```

124     obrisi_element(adresa_korena1, (*adresa_korena1)->broj);
126     /* Razlika stabala je uspesno kreirana */
127     return 0;
128 }
129
130 /* Ineće, prijavljuje se da je doslo do greske */
131 return 1;
132 }
133
134 int main()
135 {
136     Cvor *skup1;
137     Cvor *skup2;
138     Cvor *pomocni_skup = NULL;
139     int n;
140
141     /* Ucitavaju se elementi prvog skupa */
142     skup1 = NULL;
143     printf("Prvi skup: ");
144     while (scanf("%d", &n) != EOF) {
145         if (dodaj_u_stablo(&skup1, n) == 1) {
146             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
147             osloboodi_stablo(&skup1);
148             return 0;
149         }
150     }
151
152     /* Ucitavaju se elementi drugog skupa */
153     skup2 = NULL;
154     printf("Drugi skup: ");
155     while (scanf("%d", &n) != EOF) {
156         if (dodaj_u_stablo(&skup2, n) == 1) {
157             fprintf(stderr, "Neuspelo dodavanje broja %d\n", n);
158             osloboodi_stablo(&skup2);
159             return 0;
160         }
161     }
162
163     /* Kreira se unija skupova: prvo se napravi kopija prvog skupa kako
164      bi se isti mogao iskoristiti i za preostale operacije */
165     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
166         osloboodi_stablo(&skup1);
167         osloboodi_stablo(&pomocni_skup);
168         return 0;
169     }
170     if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
171         osloboodi_stablo(&pomocni_skup);
172         osloboodi_stablo(&skup2);
173         return 0;
174     }
175     printf("Unija: ");

```

```

176     ispisi_stablo_infiksno(pomocni_skup);
177     putchar('\n');

178     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
179      operacije */
180     oslobodi_stablo(&pomocni_skup);

182     /* Kreira se presek skupova: prvo se napravi kopija prvog skupa
183      kako bi se isti mogao iskoristiti i za preostale operacije */
184     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
185         oslobodi_stablo(&skup1);
186         oslobodi_stablo(&pomocni_skup);
187         return 0;
188     }
189     if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
190         oslobodi_stablo(&pomocni_skup);
191         oslobodi_stablo(&skup2);
192         return 0;
193     }
194     printf("Presek: ");
195     ispisi_stablo_infiksno(pomocni_skup);
196     putchar('\n');

198     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
199      operacije */
200     oslobodi_stablo(&pomocni_skup);

202     /* Kreira se razlika skupova: prvo se napravi kopija prvog skupa
203      kako bi se isti mogao iskoristiti i za preostale operacije */
204     if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
205         oslobodi_stablo(&skup1);
206         oslobodi_stablo(&pomocni_skup);
207         return 0;
208     }
209     if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
210         oslobodi_stablo(&pomocni_skup);
211         oslobodi_stablo(&skup2);
212         return 0;
213     }
214     printf("Razlika: ");
215     ispisi_stablo_infiksno(pomocni_skup);
216     putchar('\n');

218     /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
219      operacije */
220     oslobodi_stablo(&pomocni_skup);

222     /* Oslobadja se memorija zauzeta polaznim skupovima */
223     oslobodi_stablo(&skup1);
224     oslobodi_stablo(&skup2);

226     return 0;

```

```
| }
```

Rešenje 4.21

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 #define MAX 50
8
9 /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
10 cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11 su smestene u niz. */
12 int kreiraj_niz(Cvor * koren, int a[])
13 {
14     int r, s;
15
16     /* Stablo je prazno - u niz je smesteno 0 elemenata */
17     if (koren == NULL)
18         return 0;
19
20     /* Dodaju se u niz elementi iz levog podstabla */
21     r = kreiraj_niz(koren->levo, a);
22
23     /* Tekuća vrednost promenljive r je broj elemenata koji su upisani
24     u niz i na osnovu nje se može odrediti indeks novog elementa */
25
26     /* Smesta se vrednost iz korena */
27     a[r] = koren->broj;
28
29     /* Dodaju se elementi iz desnog podstabla */
30     s = kreiraj_niz(koren->desno, a + r + 1);
31
32     /* Racuna se indeks na koji treba smestiti naredni element */
33     return r + s + 1;
34 }
35
36 /* Funkcija sortira niz tako što najpre elemente niza smesti u
37 stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
38 desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
39 kreiran i sortiran, a 1 ukoliko je doslo do greske.
40
41 Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu" kao
42 sto je to slučaj sa ostalim opisanim algoritmima sortiranja jer se
43 sortiranje vrši u pomocnoj dinamickoj strukturi, a ne razmenom
44 elemenata niza. */
45 int sortiraj(int a[], int n)
46 {
47     int i;

```

```

49     Cvor *koren;
50
51     /* Kreira se stablo smestanjem elemenata iz niza u stablo */
52     koren = NULL;
53     for (i = 0; i < n; i++) {
54         if (dodaj_u_stablo(&koren, a[i]) == 1) {
55             osloboodi_stablo(&koren);
56             return 1;
57         }
58     }
59     /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u niz
60      a */
61     kreiraj_niz(koren, a);
62
63     /* Stablo vise nije potrebno pa se oslobadja memorija koju zauzima
64      */
65     osloboodi_stablo(&koren);
66
67     /* Vraca se indikator uspesnog sortiranja */
68     return 0;
69 }
70
71 int main()
72 {
73     int a[MAX];
74     int n, i;
75
76     /* Ucitavaju se dimenzija i elementi niza */
77     printf("n: ");
78     scanf("%d", &n);
79     if (n < 0 || n > MAX) {
80         printf("Greska: pogresna dimenzija niza!\n");
81         return 0;
82     }
83
84     printf("a: ");
85     for (i = 0; i < n; i++)
86         scanf("%d", &a[i]);
87
88     /* Poziva se funkcija za sortiranje */
89     if (sortiraj(a, n) == 0) {
90         /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
91         for (i = 0; i < n; i++)
92             printf("%d ", a[i]);
93             printf("\n");
94     } else {
95         /* Inace, obavestava se korisnik da je doslo do greske */
96         printf("Greska: problem prilikom sortiranja niza!\n");
97     }
98
99     return 0;
}

```

Rešenje 4.22

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Uključuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* a) Funkcija koja izracunava broj cvorova stabla */
8 int broj_cvorova(Cvor * koren)
9 {
10    /* Ako je stablo prazno, broj cvorova je nula */
11    if (koren == NULL)
12        return 0;
13
14    /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova u
15       levom podstablu i broja cvorova u desnom podstablu - 1 se dodaje
16       zato sto treba racunati i koren */
17    return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
18 }
19
20 /* b) Funkcija koja izracunava broj listova stabla */
21 int broj_listova(Cvor * koren)
22 {
23    /* Ako je stablo prazno, broj listova je nula */
24    if (koren == NULL)
25        return 0;
26
27    /* Proverava se da li je tekuci cvor list */
28    if (koren->levo == NULL && koren->desno == NULL)
29        /* Ako jeste vraca se 1 - to ce kasnije zbog rekurzivnih poziva
30           uvecati broj listova za 1 */
31        return 1;
32
33    /* U suprotnom se prebrojavaju listovi koje se nalaze u podstablima
34     */
35    return broj_listova(koren->levo) + broj_listova(koren->desno);
36 }
37
38 /* 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 */

```

```

52     pozitivni_listovi(koren->levo);
53     pozitivni_listovi(koren->desno);
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 najveci 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 pretrazivackog stabla, vrednosti vece od korena se
76      nalaze u desnom podstablu */
77
78     /* Ako desnog podstabla nema */
79     if (koren->desno == NULL)
80         /* Najveca vrednost je koren */
81         return koren;
82
83     /* Inace, najveca vrednost se trazi 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 >

```

```

104         dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
105     }
106
107     /* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
108     int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
109     {
110         /* Ideja je spustanje kroz stablo sve dok se ne stigne do trazenog
111            nivoa */
112
113         /* Ako nema vise cvorova, nema spustanja niz stablo */
114         if (koren == NULL)
115             return 0;
116
117         /* Ako se stiglo do trazenog nivoa, vraca se 1 - to ce kasnije zbog
118            rekurzivnih poziva uvecati broj cvorova za 1 */
119         if (i == 0)
120             return 1;
121
122         /* Inace, spusta se jedan nivo nize i u levom i u desnom postablu
123            */
124         return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
125             + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
126     }
127
128     /* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
129     void ispis_nivo(Cvor * koren, int i)
130     {
131         /* Ideja je slicna ideji iz prethodne funkcije */
132
133         /* Nema vise cvorova, nema spustanja kroz stablo */
134         if (koren == NULL)
135             return;
136
137         /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
138         if (i == 0) {
139             printf("%d ", koren->broj);
140             return;
141         }
142         /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
143            desnom podstablu */
144         ispis_nivo(koren->levo, i - 1);
145         ispis_nivo(koren->desno, i - 1);
146     }
147
148     /* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
149        stabla */
150     Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
151     {
152         /* Ako je stablo prazno, obustavlja se pretraga */
153         if (koren == NULL)
154             return NULL;

```

```

154 /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
155 if (i == 0)
156     return koren;
157
158 /* Pronalazi se maksimum sa i-tog nivoa levog podstabla */
159 Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);
160
161 /* Pronalazi se maksimum sa i-tog nivoa desnog podstabla */
162 Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);
163
164 /* Trazi se i vraca maksimum izracunatih vrednosti */
165 if (a == NULL && b == NULL)
166     return NULL;
167 if (a == NULL)
168     return b;
169 if (b == NULL)
170     return a;
171 return a->broj > b->broj ? a : b;
172 }
173
174 /* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
175 int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
176 {
177     /* Ako je stablo prazno, zbir je nula */
178     if (koren == NULL)
179         return 0;
180
181     /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
182     if (i == 0)
183         return koren->broj;
184
185     /* Inace, spustanje se nastavlja za jedan nivo nize i traže se sume
186      iz levog i desnog podstabla */
187     return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
188         + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
189 }
190
191
192 /* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
193    manje ili jednake od date vrednosti x */
194 int zbir_manjih_od_x(Cvor * koren, int x)
195 {
196     /* Ako je stablo prazno, zbir je nula */
197     if (koren == NULL)
198         return 0;
199
200     /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
201        prirode pretrazivackog stabla treba obici i levo i desno
202        podstablo */
203     if (koren->broj <= x)
204         return koren->broj + zbir_manjih_od_x(koren->levo, x) +
205             zbir_manjih_od_x(koren->desno, x);

```

```

206     /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
208      medju njima jedino moze biti onih koje zadovoljavaju uslov */
210      return zbir_manjih_od_x(koren->levo, x);
211  }
212
213 int main(int argc, char **argv)
214 {
215     /* Analiza argumenata komandne linije */
216     if (argc != 3) {
217         fprintf(stderr,
218                 "Greska! Program se poziva sa: ./a.out nivo
219                 broj_za_pretragu\n");
220         return 1;
221     }
222     int i = atoi(argv[1]);
223     int x = atoi(argv[2]);
224
225     /* Kreira se stablo uz proveru uspesnosti dodavanja novih vrednosti
226     */
227     Cvor *koren = NULL;
228     int broj;
229     while (scanf("%d", &broj) != EOF) {
230         if (dodaj_u_stablo(&koren, broj) == 1) {
231             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
232             oslobodi_stablo(&koren);
233             return 0;
234         }
235     }
236
237     /* ispisuju se rezultati rada funkcija */
238     printf("Broj cvorova: %d\n", broj_cvorova(koren));
239     printf("Broj listova: %d\n", broj_listova(koren));
240     printf("Pozitivni listovi: ");
241     pozitivni_listovi(koren);
242     printf("\n");
243     printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
244     if (najveci_element(koren) == NULL)
245         printf("Najveci element: ne postoji\n");
246     else
247         printf("Najveci element: %d\n", najveci_element(koren)->broj);
248
249     printf("Dubina stabla: %d\n", dubina_stabla(koren));
250
251     printf("Broj cvorova na %d. nivou: %d\n", i,
252           broj_cvorova_na_itom_nivou(koren, i));
253     printf("Elementi na %d. nivou: ", i);
254     ispis_nivo(koren, i);
255     printf("\n");
256     if (najveci_element_na_itom_nivou(koren, i) == NULL)
257         printf("Nema elemenata na %d. nivou!\n", i);
258     else

```

```

258     printf("Maksimalni element na %d. nivou: %d\n", i,
259             najveci_element_na_itom_nivou(koren, i)->broj);
260
260     printf("Zbir elemenata na %d. nivou: %d\n", i,
261             zbir_cvorova_na_itom_nivou(koren, i));
262     printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
263             zbir_manjih_od_x(koren, x));
264
264     /* Oslobadja se memorija zauzeta stablom */
265     osloboodi_stablo(&koren);
266
268     return 0;
}

```

Rešenje 4.23

```

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

/* Uključuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
    /* Dubina praznog stabla je 0 */
    if (koren == NULL)
        return 0;

    /* Izracunava se dubina levog podstabla */
    int dubina_levo = dubina_stabla(koren->levo);

    /* Izracunava se dubina desnog podstabla */
    int dubina_desno = dubina_stabla(koren->desno);

    /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se dodaje
       jer se racuna i koren */
    return dubina_levo >
        dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispisi_nivo(Cvor * koren, int i)
{
    /* Ideja je slicna ideji iz prethodne funkcije */
    /* Nema vise cvorova, nema spustanja niz stablo */
    if (koren == NULL)
        return;

    /* Ako se stiglo do trazenog nivoa - ispisuje se vrednost */
    if (i == 0) {

```

```

36     printf("%d ", koren->broj);
37     return;
38 }
39 /* Inace, vrsti se spustanje za jedan nivo nize i u levom i u desnom
40    podstablu */
41 ispisi_nivo(koren->levo, i - 1);
42 ispisi_nivo(koren->desno, i - 1);
43 }
44
45 /* Funkcija koja ispisuje stablo po nivoima */
46 void ispisi_stablo_po_nivoima(Cvor * koren)
47 {
48     int i;
49
50     /* Prvo se izracunava dubina stabla */
51     int dubina;
52     dubina = dubina_stabla(koren);
53
54     /* Ispisuje se nivo po nivo stabla */
55     for (i = 0; i < dubina; i++) {
56         printf("%d. nivo: ", i);
57         ispisi_nivo(koren, i);
58         printf("\n");
59     }
60 }
61
62 int main(int argc, char **argv)
63 {
64     Cvor *koren;
65     int broj;
66
67     /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
68        uspesnosti dodavanja */
69     koren = NULL;
70     while (scanf("%d", &broj) != EOF) {
71         if (dodaj_u_stablo(&koren, broj) == 1) {
72             fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
73             osloboodi_stablo(&koren);
74             return 0;
75         }
76     }
77
78     /* Ispisuje se stablo po nivoima */
79     ispisi_stablo_po_nivoima(koren);
80
81     /* Oslobadja se memorija zauzeta stablom */
82     osloboodi_stablo(&koren);
83
84     return 0;
85 }

```

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 izracunava dubinu stabla */
8 int dubina_stabla(Cvor * koren)
9 {
10     /* Dubina praznog stabla je 0 */
11     if (koren == NULL)
12         return 0;
13
14     /* Izracunava se dubina levog podstabla */
15     int dubina_levo = dubina_stabla(koren->levo);
16
17     /* Izracunava 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     /* Izracunava se dubina levog podstabla korena */
38     dubina_levo = dubina_stabla(koren->levo);
39
40     /* Izracunava 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     } else {
49         /* Inace, racuna se samo broj AVL cvorova u podstablima */
50         return avl(koren->levo) + avl(koren->desno);
51     }
52 }
```

```

51    }
52}
53 int main(int argc, char **argv)
54{
55    Cvor *koren;
56    int broj;
57
58    /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
59       uspesnosti dodavanja */
60    koren = NULL;
61    while (scanf("%d", &broj) != EOF) {
62        if (dodaj_u_stablo(&koren, broj) == 1) {
63            fprintf(stderr, "Neuspelo dodavanje broja %d\n", broj);
64            osloboodi_stablo(&koren);
65            return 0;
66        }
67    }
68
69    /* Racuna se i ispisuje broj AVL cvorova */
70    printf("%d\n", avl(koren));
71
72    /* Oslobadja se memorija zauzeta stablom */
73    osloboodi_stablo(&koren);
74
75    return 0;
76}
77

```

Rešenje 4.26

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Ukljucuje se biblioteka za rad sa stablima */
5 #include "stabla.h"
6
7 /* Funkcija proverava da li je zadato binarno stablo celih pozitivnih
8   brojeva hip. Ideja koja ce biti implementirana u osnovi ima
9   pronalazenje maksimalne vrednosti levog i maksimalne vrednosti
10  desnog podstabla - ako je vrednost u korenu veca od izracunatih
11  vrednosti, uoceni fragment stabla zadovoljava uslov za hip. Zato
12  ce funkcija vracati maksimalne vrednosti iz uocenog podstabala ili
13  vrednost -1 ukoliko se zakljuci da stablo nije hip. */
14 int heap(Cvor * koren)
15{
16    int max_levo, max_desno;
17
18    /* Prazno stablo je hip - kao rezultat se vraca 0 kao najmanji
19       pozitivan broj */
20    if (koren == NULL) {
21        return 0;
22    }
23
24    max_levo = heap(koren->levi);
25    max_desno = heap(koren->desni);
26
27    if (max_levo >= koren->vrednost & koren->vrednost >= max_desno)
28        return koren->vrednost;
29    else
30        return -1;
31}
32
33

```

```

22    }
24    /* Ukoliko je stablo list... */
25    if (koren->levo == NULL && koren->desno == NULL) {
26        /* Vraca se njegova vrednost */
27        return koren->broj;
28    }
29    /* Inace... */

30    /* Proverava se svojstvo za levo podstablo */
31    max_levo = heap(koren->levo);

32    /* Proverava se svojstvo za desno podstablo */
33    max_desno = heap(koren->desno);

34    /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
35     ni celo stablo */
36    if (max_levo == -1 || max_desno == -1) {
37        return -1;
38    }

39    /* U suprotnom proverava se da li svojstvo vazi za uoceni cvor */
40    if (koren->broj > max_levo && koren->broj > max_desno) {
41        /* Ako vazi, vraca se vrednost korena */
42        return koren->broj;
43    }

44    /* U suprotnom zaključuje se da stablo nije hip */
45    return -1;
46}

47 int main(int argc, char **argv)
48 {
49    Cvor *koren;
50    int hip_indikator;

51    /* Kreira se stablo prema zadatoj slici */
52    koren = NULL;
53    koren = napravi_cvor(100);
54    koren->levo = napravi_cvor(19);
55    koren->levo->levo = napravi_cvor(17);
56    koren->levo->levo->levo = napravi_cvor(2);
57    koren->levo->levo->desno = napravi_cvor(7);
58    koren->levo->desno = napravi_cvor(3);
59    koren->desno = napravi_cvor(36);
60    koren->desno->levo = napravi_cvor(25);
61    koren->desno->desno = napravi_cvor(1);

62    /* Poziva se funkcija kojom se proverava da li je stablo hip */
63    hip_indikator = heap(koren);

64    /* Ispisuje se rezultat */
65    if (hip_indikator == -1) {
66
67
68
69
70
71
72

```

```
74     printf("Zadato stablo nije hip!\n");
75 } else {
76     printf("Zadato stablo je hip!\n");
77 }
78 /* Oslobadja se memorija zauzeta stablom */
79 oslobodi_stablo(&koren);
80
81     return 0;
82 }
```

5

Ispitni rokovi

5.1 Praktični deo ispita, jun 2015

Zadatak 5.1 Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od 50 karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu za grešku ispisati vrednost **-1** i prekinuti izvršavanje programa.

Test 1

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
5
Programiranje
Matematika
12345
dInAmicNArEc
Ispit

IZLAC:
```

Test 2

```
POKRETANJE: ./a.out ulaz.txt
ULAZ.TXT
2
maksimalano
poena

IZLAC:
```

Test 3

```
|| POKRETANJE: ./a.out ulaz.txt  
|| DATOTEKA ULAZ.TXT NE POSTOJI  
|| IZLAZ ZA GREŠKU:  
|| -1
```

Test 4

```
|| POKRETANJE: ./a.out  
|| IZLAZ ZA GREŠKU:  
|| -1
```

[Rešenje 5.1]

Zadatak 5.2 Data je biblioteka za rad sa binarnim pretraživačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumiraj_n (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 `sumiraj_n` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test 1

```
|| ULAZ:  
|| 2 8 10 3 6 14 13 7 4 0  
|| IZLAZ:  
|| 20
```

Test 2

```
|| ULAZ:  
|| 0 50 14 5 2 4 56 8 52 7 1 0  
|| IZLAZ:  
|| 50
```

[Rešenje 5.2]

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A , a zatim i elementi matrice A . Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od 50. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

Test 1

```
|| ULAZ:
  4 5
  1 2 3 4 5
  -1 2 -3 4 -5
  -5 -4 -3 -2 1
  -1 0 0 0 0
  IZLAZ:
  0
```

Test 2

```
|| ULAZ:
  2 3
  0 0 -5
  1 2 -4
  IZLAZ:
  2
```

Test 3

```
|| ULAZ:
  -2
  IZLAZ ZA GREŠKU:
  -1
```

[Rešenje 5.3]

5.2 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 podnische `needle` u nisci `haystack`, i vraća pokazivač na početak podnische, ili `NULL` ako podniska nije pronađena.

Test 1

```
|| POKRETANJE: ./a.out ulaz.txt test
  ULAZ.TXT
  Ovo je test primer.
  U njemu se rec test javlja
  vise puta. testtesttest
  IZLAZ:
  5
```

Test 2

```
|| POKRETANJE: ./a.out
  IZLAZ ZA GREŠKU:
  -1
```

Test 3

```
|| POKRETANJE: ./a.out ulaz.txt foo
  DATOTEKA ULAZ.TXT NE POSTOJI
  IZLAZ ZA GREŠKU:
  -1
```

Test 4

```
|| POKRETANJE: ./a.out ulaz.txt .
  DATOTEKA ULAZ.TXT JE PRAZNA
  IZLAZ:
  0
```

[Rešenje 5.4]

Zadatak 5.5 Na početku datoteke „trouglovi.txt” nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitava trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac: $P = \sqrt{s * (s - a) * (s - b) * (s - c)}$, gde je s poluobim trougla). U slučaju bilo kakve greške ispisati -1 na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Test 1

```
TROUGLOVI.TXT
4
0 0 0 1.2 1 0
0.3 0.3 0.5 0.5 0.9 1
-2 0 0 0 0 1
-2 0 0 0 0 1

IZLAZ:
2 0 2 2 -1 -1
-2 0 0 0 0 1
0 0 0 1.2 1 0
0.3 0.3 0.5 0.5 0.9 1
```

Test 2

```
TROUGLOVI.TXT
3
1.2 3 2 1.1 4.3

IZLAZ ZA GREŠKU:
-1
```

Test 3

```
DATOTEKA TROUGLOVI.TXT NE POSTOJI

IZLAZ ZA GREŠKU:
-1
```

Test 4

```
TROUGLOVI.TXT
0

IZLAZ:
```

[Rešenje 5.5]

Zadatak 5.6 Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeva. Napisati funkciju

`int prebroj_n(Cvor *koren, int n)`

koja u datom stablu prebrojava čvorove na n -tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou 0. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

Test 1

```
ULAZ:
1 5 3 6 1 4 7 9
IZLAZ:
1
```

Test 2

```
ULAZ:
2 5 3 6 1 0 4 7 9
IZLAZ:
2
```

Test 3

```
ULAZ:
0 4 2 5
IZLAZ:
0
```

Test 4

```
|| ULAZ:  
||   3  
|| IZLAZ:  
||   0
```

Test 5

```
|| ULAZ:  
||   -1 4 5 1 7  
|| IZLAZ:  
||   0
```

[Rešenje [5.6](#)]

5.3 Praktični deo ispita, septembar 2015

Zadatak 5.7 Sa standardnog ulaza se učitavaju neoznačeni celi brojevi x i n . Na standardni izlaz ispisati neoznačen ceo broj koji se dobija od broja x kada se njegov binarni zapis rotira za n mesta udesno (na primer, ako je binarni zapis broja x jednak 00000000000000000000000000001111, i ako je $n=1$ tada na standardni izlaz treba ispisati neoznačen broj čiji je binarni zapis jednak 10000000000000000000000000000000111).

Test 1

```
|| ULAZ:  
||   6 1  
|| IZLAZ:  
||   3
```

Test 2

```
|| ULAZ:  
||   15 3  
|| IZLAZ:  
||   3758096385
```

Test 3

```
|| ULAZ:  
||   31 100  
|| IZLAZ:  
||   4026531841
```

Test 4

```
|| ULAZ:  
||   4 0  
|| IZLAZ:  
||   4
```

Test 5

```
|| ULAZ:  
||   0 5  
|| IZLAZ:  
||   0
```

[Rešenje [5.7](#)]

Zadatak 5.8 Napisati funkciju `int dopuni_listu(Cvor ** adresa_glave)` koja samo čvorovima koji imaju sledbenika u jednostruko povezanoj listi realnih brojeva, dodaje između čvora i njegovog sledbenika nov čvor čija vrednost je aritmetička sredina njihovih vrednosti. Povratna vrednost funkcije treba da bude 1 ukoliko je došlo greške pri alokaciji memorije, inače 0. Ispravnost napisane funkcije testirati koristeći dostupnu biblioteku za rad sa listama i `main` funkciju koja najpre učitava elemente liste, poziva pomenutu funkciju i ispisuje sadržaj liste.

Test 1

	ULAZ:	
	1 2 3 4 5	
	IZLAZ:	
	1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00	

Test 2

	ULAZ:	
	12	
	IZLAZ:	
	12.00	

Test 3

	ULAZ:	
	prazna lista	
	IZLAZ:	

Test 4

	ULAZ:	
	13.3 15.8	
	IZLAZ:	
	13.30 14.55	

[Rešenje 5.8]

Zadatak 5.9 Sa standardnog ulaza se učitava dimenzija n kvadratne celobrojne matrice A ($n > 0$), a zatim i elementi matrice A. Napisati program koji proverava da li je data kvadratna matrica magični kvadrat (magični kvadrat je kvadratna matrica kod koje su sume brojeva u svim redovima i kolonama međusobno jednake). Ukoliko jeste, ispisati na standardnom izlazu sumu brojeva jedne vrste ili kolone te matrice, a ukoliko nije ispisati -. Broj vrsta i broj kolona matrice nije unapred poznat. U slučaju greške ispisati -1 na standardni izlaz za grešku. NAPOMENA: Rešenje koristi biblioteku za rad sa matricama iz zadatka ??.

Test 1

	ULAZ:	
	4	
	1 2 3 4	
	2 1 4 3	
	3 4 2 1	
	4 3 1 2	
	IZLAZ:	
	10	

Test 2

	ULAZ:	
	3	
	1 1 1	
	1 1 1	
	IZLAZ:	
	3	

Test 3

	ULAZ:	
	2	
	1 1	
	2 2	
	IZLAZ:	
	-	

Test 4

	ULAZ:	
	2	
	1 2	
	1 2	
	IZLAZ:	
	-	

Test 5

	ULAZ:	
	1	
	5	
	IZLAZ:	
	5	

Test 6

	ULAZ:	
	0	
	IZLAZ ZA GREŠKU:	
	-1	

[Rešenje 5.9]

5.4 Praktični deo ispita, oktobar 2015

5.5 Praktični deo ispita, januar 2016

5.6 Rešenja

Rešenje 5.1

```
#include <stdio.h>
2 #include <stdlib.h>
# include <ctype.h>
4 #define MAKS 50

6 /* Funkcija vrši dinamicku alokaciju memorije potrebne n linija tj.
n niski od kojih nijedna nije duza od MAKS karaktera. */
8 char **alociranje_memorije(int n)
{
10    char **linije = NULL;
11    int i, j;
12    /* Alocira se prostor za niz vrsti matrice */
13    linije = (char **) malloc(n * sizeof(char *));
14    /* U slučaju neuspesnog otvaranja ispisuje se -1 na stderr i
program završava. */
15    if (linije == NULL)
16        return NULL;
17    /* Alocira se prostor za svaku vrstu matrice. Niska nije duza od
MAKS karaktera, a 1 se dodaje zbog terminirajuce nule. */
18    for (i = 0; i < n; i++) {
19        linije[i] = malloc((MAKS + 1) * sizeof(char));
20        /* Ako alokacija nije prosla uspesno, oslobadaju se svi
prethodno alocirani resursi, i povratna vrednost je NULL */
21        if (linije[i] == NULL) {
22            for (j = 0; j < i; j++)
23                free(linije[j]);
24            free(linije);
25            return NULL;
26        }
27    }
28    return linije;
29 }
30
31 char **oslobadjanje_memorije(char **linije, int n)
32 {
33    int i;
34    /* Oslobadja se prostor rezervisan za svaku vrstu */
35    for (i = 0; i < n; i++)
36        free(linije[i]);
```

```

42     }
43     /* Oslobadja se memorija za niz pokazivaca na vrste */
44     free(linije);
45
46     /* Matrica postaje prazna, tj. nealocirana */
47     return NULL;
48 }
49
50 int main(int argc, char *argv[])
51 {
52     FILE *ulaz;
53     char **linije;
54     int i, j, n;
55
56     /* Proverava argumenata komandne linije. */
57     if (argc != 2) {
58         fprintf(stderr, "-1\n");
59         exit(EXIT_FAILURE);
60     }
61
62     /* Otvaranje datoteke cije ime je navedeno kao argument komandne
63      linije neposredno nakon imena programa koji se poziva. U slučaju
64      neuspesnog otvaranja ispisuje se -1 na stderr i program završava
65      izvršavanje. */
66     ulaz = fopen(argv[1], "r");
67     if (ulaz == NULL) {
68         fprintf(stderr, "-1\n");
69         exit(EXIT_FAILURE);
70     }
71     /* Ucitavanje broja linija. */
72     fscanf(ulaz, "%d", &n);
73
74     /* Alociranje memorije na osnovu ucitanog broja linija. */
75     linije = alociranje_memorije(n);
76
77     /* U slučaju neuspesne alokacije ispisuje se -1 na stderr i program
78      završava. */
79     if (linije == NULL) {
80         fprintf(stderr, "-1\n");
81         exit(EXIT_FAILURE);
82     }
83
84     /* Ucitavanje svih n linija iz datoteke. */
85     for (i = 0; i < n; i++) {
86         fscanf(ulaz, "%s", linije[i]);
87     }
88
89     /* Ispisivanje u odgovarajućem poretku ucitane linije koje
90      zadovoljavaju kriterijum. */
91     for (i = n - 1; i >= 0; i--) {
92         if (isupper(linije[i][0])) {
93             printf("%s\n", linije[i]);
94         }
95     }
96 }
```

```

    }
94  /*
95   * Oslobadjanje memorije koja je dinamicki alocirana. */
96  linije = oslobadjanje_memorije(linije, n);
97
98  /*
99   * Zatvaranje datoteku. */
100 fclose(ulaz);
101 exit(EXIT_SUCCESS);
102 }
```

Rešenje 5.2

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "stabla.h"
4
5 int sumiraj_n(Cvor * koren, int n)
6 {
7     /* Ako je stablo prazno, suma je nula */
8     if (koren == NULL)
9         return 0;
10    /* Inace ... */
11    /* Ako je n jednako nula, vraca se broj iz korena */
12    if (n == 0)
13        return koren->broj;
14    /* Inace, izracunava se suma na (n-1)-om nivou u levom podstablu,
15       kao i suma na (n-1)-om nivou u desnom podstablu i vraca se zbir
16       te dve izracunate vrednosti jer predstavlja zbir svih cvorova na
17       n-tom nivou u pocetnom stablu */
18    return sumiraj_n(koren->levo, n - 1) + sumiraj_n(koren->desno, n -
19               1);
20}
21
22int main()
23{
24    Cvor *koren = NULL;
25    int n;
26    int nivo;
27
28    /* Ucitava se vrednost nivoa */
29    scanf("%d", &nivo);
30    while (1)
31        scanf("%d", &n);
32    /* Ukoliko je korisnik uneo 0, prekida se dalje citanje. */
33    if (n == 0)
```

```

        break;
34     /* Ako nije, dodaje se procitani broj u stablo. */
35     if (dodaj_u_stablo(&koren, n) == 1) {
36         fprintf(stderr, "-1\n", n);
37         oslobodi_stablo(&koren);
38         exit(EXIT_FAILURE);
39     }
40 }

42     /* Ispisuje se rezultat rada trazene funkcije */
43     printf("%d\n", sumiraj_n(koren, nivo));
44
45     /* Oslobadja se memorija */
46     oslobodi_stablo(&koren);

48     exit(EXIT_SUCCESS);
}

```

Rešenje 5.3

```

#include <stdio.h>
2 #include <stdlib.h>
#define MAKS 50

4 /* Funkcija ucitava elemenate matrice sa standardnog ulaza */
6 void ucitaj_matricu(int m[][][MAKS], int v, int k)
{
8     int i, j;
9     for (i = 0; i < v; i++) {
10         for (j = 0; j < k; j++) {
11             scanf("%d", &m[i][j]);
12         }
13     }
14 }

16 /* Funkcija racuna broj negativnih elemenata u k-oj koloni matrice m
   koja ima v vrsta */
18 int broj_negativnih_u_koloni(int m[][][MAKS], int v, int k)
{
20     int broj_negativnih = 0;
21     int i;
22     for (i = 0; i < v; i++) {
23         if (m[i][k] < 0)
24             broj_negativnih++;
25     }
26     return broj_negativnih;
27 }
28
29 int MAKS_indeks(int m[][][MAKS], int v, int k)
{
30     int i, j;

```

```

32     int broj_negativnih;
34     /* Inicijalizujemo na nulu indeks kolone sa maksimalnim brojem
35      negativnih (maks_indeks_kolone), kao i maksimalni broj
36      negativnih elemenata u nekoj koloni (maks_broj_negativnih) */
37     int maks_indeks_kolone = 0;
38     int maks_broj_negativnih = 0;
39
40     for (j = 0; j < k; j++) {
41         /* Racunamo broj negativnih u j-oj koloni */
42         broj_negativnih = broj_negativnih_u_koloni(m, v, j);
43         /* Ukoliko broj negativnih u j-toj koloni veci od trenutnog
44            maksimalnog, azuriramo trenutni maksimalni broj negativnih
45            kao i indeks kolone sa maksimalnim brojem negativnih */
46         if (maks_broj_negativnih < broj_negativnih) {
47             maks_indeks_kolone = j;
48             maks_broj_negativnih = broj_negativnih;
49         }
50     }
51     return maks_indeks_kolone;
52 }
53
54 int main()
55 {
56     int m[MAKS][MAKS];
57     int v, k;
58
59     /* Ucitavanje dimenzije matrice */
60     scanf("%d", &v);
61     if (v < 0 || v > MAKS) {
62         fprintf(stderr, "-1\n");
63         exit(EXIT_FAILURE);
64     }
65     scanf("%d", &k);
66     if (k < 0 || k > MAKS) {
67         fprintf(stderr, "-1\n");
68         exit(EXIT_FAILURE);
69     }
70     /* Ucitavanje elemenata matrice */
71     ucitaj_matricu(m, v, k);
72
73     /* Pronalazenje kolone koja sadrzi najveci broj negativnih
74      elemenata i ispisivanje trazenog rezultata */
75     printf("%d\n", MAKS_indeks(m, v, k));
76     exit(EXIT_SUCCESS);
77 }
```

Rešenje 5.4

```

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

```

4 #define MAKS 128
5
6 int main(int argc, char **argv)
7 {
8     FILE *f;
9     int brojac = 0;
10    char linija[MAKS], *p;
11
12    /* Provera da li je broj argumenata komandne linije 3 */
13    if (argc != 3) {
14        fprintf(stderr, "-1\n");
15        exit(EXIT_FAILURE);
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    /* Ucitavanje iz otvorene datoteke - liniju po liniju */
24    while (fgets(linija, MAKS, f) != NULL) {
25        p = linija;
26        while (*p) {
27            p = strstr(p, argv[2]);
28
29            /* Ukoliko nije podniska tj. p je NULL izlazi se iz petlje */
30            if (p == NULL)
31                break;
32            /* Inace se uvecava brojac */
33            brojac++;
34            /* p se pomera da bi se u sledecoj iteraciji posmatra ostatak
35               linije nakon uocene podniske */
36            p = p + strlen(argv[2]);
37        }
38    }
39    /* Zatvaranje datoteke */
40    fclose(f);
41    /* Ispisivanje vrednosti brojaca */
42    printf("%d\n", brojac);
43
44    exit(EXIT_SUCCESS);
45 }

```

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;
8 } trougao;
9
10 /* Funkcija racuna duzinu duzi */
11 double duzina(double x1, double y1, double x2, double y2)
12 {
13   return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
14 }
15
16 /* Funkcija racuna povrsinu trougla koristeci Heronov obrazac */
17 double povrsina(trougao t)
18 {
19   /* Racunanje duzina stranica trougla */
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   /* Poloubim */
24   double s = (a + b + c) / 2;
25   /* Primena Heronovog obrasca */
26   return sqrt(s * (s - a) * (s - b) * (s - c));
27 }
28
29 /* Funkcija poredi dva trougla: ukoliko je povrsina trougla koji je
30 prvi argument funkcije manja od povrsine trougla koji je drugi
31 element funkcije funkcija vraca 1, ukoliko je veca -1, a ukoliko
32 su povrsine dva trougla jednake vraca nulu. Dakle, funkcija je
33 napisana tako da se moze proslediti funkciji qsort da se niz
34 trouglova sortira po povrsini opadajuce. */
35 int poredi(const void *a, const void *b)
36 {
37   trougao x = *(trougao *) a;
38   trougao y = *(trougao *) b;
39   double xp = povrsina(x);
40   double yp = povrsina(y);
41   if (xp < yp)
42     return 1;
43   if (xp > yp)
44     return -1;
45   return 0;
46 }
47
48 int main()
49 {
50   FILE *f;
51   int n, i;
52   trougao *niz;
53
54   /* Otvaranje datoteke ciji je naziv trouglovi.txt */
55   if ((f = fopen("trouglovi.txt", "r")) == NULL) {
56     fprintf(stderr, "-1\n");
57     exit(EXIT_FAILURE);
58   }

```

```

59  /* Ucitavanje podataka o broju trouglova iz datoteke */
61  if (fscanf(f, "%d", &n) != 1) {
62      fprintf(stderr, "-1\n");
63      exit(EXIT_FAILURE);
64  }
65
66  /* Dinamicka alokacija memorije za niz trouglova duzine n */
67  if ((niz = malloc(n * sizeof(trougao))) == NULL) {
68      fprintf(stderr, "-1\n");
69      exit(EXIT_FAILURE);
70  }
71
72  /* Ucitavanje podataka u niz iz otvorene datoteke */
73  for (i = 0; i < n; i++) {
74      if (fscanf(f, "%lf%lf%lf%lf%lf", &niz[i].xa, &niz[i].ya,
75                  &niz[i].xb, &niz[i].yb, &niz[i].xc, &niz[i].yc) != 6)
76      {
77          fprintf(stderr, "-1\n");
78          exit(EXIT_FAILURE);
79      }
80  }
81
82  /* Pozivanje funkcije qsort da sortira niz na osnovu funkcije
     poredi */
83  qsort(niz, n, sizeof(trougao), &poredi);
84
85  /* Ispisivanje sortiranog niza na standardni izlaz */
86  for (i = 0; i < n; i++)
87      printf("%g %g %g %g %g\n", niz[i].xa, niz[i].ya, niz[i].xb,
88             niz[i].yb, niz[i].xc, niz[i].yc);
89
90  /* Oslobadjanje dinamicki alocirane memorije */
91  free(niz);
92
93  /* Zatvranje datoteke */
94  fclose(f);
95  exit(EXIT_SUCCESS);
}

```

Rešenje 5.6

NAPOMENA: Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 4.14.

main.c

```

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

```

```

4  /*
5   * Funkcija ucitava brojeve sa standardnog ulaza i smesta ih u
6   * stablo. Funkcija vraca 1 u slucaju neuspesnog dodavanja elementa u
7   * stablo, a inace 0. */
8 int ucitaj_stablo(Cvor ** koren)
{
    *koren = NULL;
    int x;
    /* Sve dok ima brojeva na standardnom ulazu, ucitani brojevi se
       dodaju u stablo. Ukoliko funkcija dodaj_u_stablo vrati 1, onda
       je i povratna vrednost iz funkcije ucitaj_stablo 1. */
    while (scanf("%d", &x) == 1)
        if (dodaj_u_stablo(koren, x) == 1)
            return 1;
    return 0;
}

/* Funkcija prebrojava broj cvorova na n-tom nivou u stablu */
22 int prebroj_n(Cvor * koren, int n)
{
    /* Ukoliko je stablo prazno, rezultat je nula. Takodje, ako je n
       negativan broj, na tom nivou nema cvorova (rezultat je nula). */
26    if (koren == NULL || n < 0)
        return 0;
    /* Ukoliko je n = 0, na tom nivou je samo koren. Ukoliko ima jednog
       potomka funkcija vraca 1, inace 0 */
30    if (n == 0) {
        if (koren->levo == NULL && koren->desno != NULL)
            return 1;
        if (koren->levo != NULL && koren->desno == NULL)
            return 1;
        return 0;
    }
    /* Broj cvorova na n-tom nivou je jednak zbiru broja cvorova na
       (n-1)-om nivou levog podstabla i broja cvorova na (n-1)-om nivou
       levog podstabla */
40    return prebroj_n(koren->levo, n - 1) + prebroj_n(koren->desno, n -
        1);
}

int main()
{
    Cvor *koren;
    int n;
    scanf("%d", &n);

    /* Ucitavanje elemenata u stablo. U slucaju neuspesne alokacije
       oslobođuju se alocirana memorija i izlazi se iz programa. */
50    if (ucitaj_stablo(&koren) == 1) {
        fprintf(stderr, "-1\n");
        osloboidi_stablo(&koren);
        exit(EXIT_FAILURE);
    }
}

```

```

56    }
57
58    /* Ispisivanje rezultata */
59    printf("%d\n", prebroj_n(koren, n));
60
61    /* Oslobadjanje dinamicki alociranog stabla */
62    osloboodi_stablo(&koren);
63
64    exit(EXIT_SUCCESS);
}

```

Rešenje 5.7

```

#include <stdio.h>
1
2    /* Funkcija vraca broj ciji binarni zapis se dobija kada se binarni
   3     zapis argumenta x rotira za n mesta udesno */
4    unsigned int rotiraj(unsigned int x, unsigned int n)
5    {
6        int i;
7        unsigned int maska = 1;
8        /* Formiranje maske sa n jedinica na kraju, npr za n=4 maska bi
   9         izgledala: 000...00001111 */
10       /* Maska se moze formirati i naredbom: maska = (1 << n) - 1; U
11        nastavku je drugi nacin. */
12       for (i = 1; i < n; i++)
13           maska = (maska << 1) | 1;
14       /* Kada se x poremeri za n mesta udesno x >> n, poslednjih n bitova
15        binarne reprezentacije broja x ce "ispasti". Za rotaciju je
16        potrebno da se tih n bitova postavi na pocetak broja. Kreirana
17        maska omogucava da se tih n bitova izdvoji sa (maska & x), a
18        zatim se pomeranjem za (sizeof(unsigned) * 8 - n) mesta uлево
19        tih n bitova postavlja na pocetak. Primenom logicke disjunkcije
20        dobija se rotirani broj. */
21       return (x >> n) | ((maska & x) << (sizeof(unsigned) * 8 - n));
22   }
23
24   int main()
25   {
26       unsigned int x, n;
27
28       /* Ucitavanje brojeva sa standardnog ulaza */
29       scanf("%u%u", &x, &n);
30
31       /* Ispisivanje rezultata */
32       printf("%u\n", rotiraj(x, n));
33       return 0;
34   }

```

Rešenje 5.8

liste.h

```

1 #ifndef _LISTE_H_
2 #define _LISTE_H_ 1

4 /* Struktura koja predstavlja cvor liste */
5 typedef struct cvor {
6     double vrednost;
7     struct cvor *sledeci;
8 }
9 Cvor;

10 /* Pomocna funkcija koja kreira cvor. */
11 Cvor * napravi_cvor(double broj);

14 /* Funkcija oslobođaja dinamicku memoriju zauzetu za elemente
   liste ciji se pocetni cvor nalazi na adresi adresa_glove. */
15 void osloboodi_listu(Cvor ** adresa_glove);

18 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
19 Cvor * pronadji_poslednji(Cvor * glava);

22 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije,inace vraca 0. */
23 int dodaj_na_kraj_liste(Cvor ** adresa_glove, double broj);

26 /* Funkcija prikazuje sve elemente liste pocev od glave ka kraju
   liste. */
27 void ispisi_listu(Cvor * glava);

30#endif

```

liste.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "liste.h"

4 /* Pomocna funkcija koja kreira cvor. */
5 Cvor *napravi_cvor(double broj)
6 {
7     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
8     if (novi == NULL)
9         return NULL;
10    /* inicijalizacija polja u novom cvoru */
11    novi->vrednost = broj;
12    novi->sledeci = NULL;

```

```

14     return novi;
16 }
18 /* Funkcija oslobadja dinamicku memoriju zauzetu za elemente liste
   ciji se pocetni cvor nalazi na adresi adresa_glave. */
20 void oslobodi_listu(Cvor ** adresa_glave)
21 {
22     Cvor *pomocni = NULL;
23     while (*adresa_glave != NULL) {
24         pomocni = (*adresa_glave)->sledeci;
25         free(*adresa_glave);
26         *adresa_glave = pomocni;
27     }
28 }
29 }
30 }
31
32 /* Funkcija pronalazi i vraca pokazivac na poslednji element liste,
   ili NULL kao je lista prazna */
33 Cvor *pronadji_poslednji(Cvor * glava)
34 {
35     /* Ako je lista prazna, nema ni poslednjeg cvor i u tom slucaju
       funkcija vraca NULL. */
36     if (glava == NULL)
37         return NULL;
38     while (glava->sledeci != NULL)
39         glava = glava->sledeci;
40     return glava;
41 }
42
43
44 }
45
46 /* Funkcija dodaje novi cvor na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
47 int dodaj_na_kraj_liste(Cvor ** adresa_glave, double broj)
48 {
49     Cvor *novi = napravi_cvor(broj);
50     if (novi == NULL)
51         return 1;
52     if (*adresa_glave == NULL) {
53         *adresa_glave = novi;
54         return 0;
55     }
56     Cvor *poslednji = pronadji_poslednji(*adresa_glave);
57     poslednji->sledeci = novi;
58 }
59
60 }
61
62 /* Funkcija prikazuje elemente liste pocev od glave ka kraju liste.
   */
63 void ispisi_listu(Cvor * glava)
64

```

```

66 {
67     for ( ; glava != NULL; glava = glava->sledeci)
68         printf("%.2lf ", glava->vrednost);
69     putchar('\n');
70 }

```

main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "liste.h"
4
5 /* Funkcija umece novi cvor iza tekuceg u listi */
6 void dodaj_iza(Cvor * tekuci, Cvor * novi)
7 {
8     /* Novi cvor se dodaje iza tekuceg cvora. */
9     novi->sledeci = tekuci->sledeci;
10    tekuci->sledeci = novi;
11 }
12
13
14 /* Funkcija koja dopunjuje listu na nacin opisan u tekstu zadatka.
15    Vraca 1 ukoliko je bilo greske pri alokaciji memorije, inace vraca
16    0. */
17 int dopuni_listu(Cvor ** adresa_glove)
18 {
19     Cvor *tekuci;
20     Cvor *novi;
21     double aritmeticka_sredina;
22     /* U slucaju prazne ili jednoclane liste, funkcija vraca 1 */
23     if (*adresa_glove == NULL || (*adresa_glove)->sledeci == NULL)
24         return 1;
25     /* Promenljiva tekuci se inicializacuje da pokazuje na pocetni
26     cvor */
27     tekuci = *adresa_glove;
28     /* Sve dok ima cvorova u listi racuna se aritmeticka sredina
29     vrednosti u susednim cvorovim i kreira cvor sa tom vrednoscu. U
30     slucaju neupele alokacije novog cvora, funkcija vraca 1. Inace,
31     novi cvor se umece izmedju dva cvora za koje racunata
32     aritmeticka sredina */
33     while (tekuci->sledeci != NULL) {
34         aritmeticka_sredina =
35             ((tekuci)->vrednost + ((tekuci)->sledeci)->vrednost) / 2;
36         novi = napravi_cvor(aritmeticka_sredina);
37         if (novi == NULL)
38             return 1;
39         /* Poziva se funkcija koja umece novi cvor iza tekuceg cvora */
40         dodaj_iza(tekuci, novi);
41         /* Tekuci cvor se pomera na narednog u listi (to je novoumetnuti
42         cvor), a zatim jos jednom da bi pokazivao na naredni cvor iz

```

```

        polazne liste */
44    tekuci = tekuci->sledeci;
45    tekuci = tekuci->sledeci;
46 }
47 return 0;
48 }

50 int main()
{
51     Cvor *glava = NULL;
52     double broj;

53     /* Ucitavanje se vrsti do kraja ulaza. Elementi se dodaju na kraj
54      liste! */
55     while (scanf("%lf", &broj) > 0) {
56         /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
57          memorije za nov cvor. Memoriju alociranu za cvorove liste
58          treba oslobođiti. */
59         if (dodaj_na_kraj_liste(&glava, broj) == 1) {
60             fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
61             osloboди_listu(&glava);
62             exit(EXIT_FAILURE);
63         }
64     }

65     /* Pozivanje funkcije da dopuni listu. Ako je funkcija vratila 1,
66      onda je bilo greske pri alokaciji memorije za nov cvor. Memoriju
67      alociranu za cvorove liste treba oslobođiti. */
68     if (dopuni_listu(&glava) == 1) {
69         fprintf(stderr, "Neuspela alokacija za cvor %d\n", broj);
70         osloboди_listu(&glava);
71         exit(EXIT_FAILURE);
72     }

73     /* Ispisivanje liste */
74     ispisi_listu(glava);

75     /* Oslobođjanje liste */
76     osloboди_listu(&glava);

77     exit(EXIT_SUCCESS);
78 }

```

Rešenje 5.9

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "matrica.h"
4
5 /* Funkcija racuna zbir elemenata v-te vrste */
6 int zbir_vrste(int **M, int n, int v)

```

```

8   {
9     int j, zbir = 0;
10    for (j = 0; j < n; j++)
11      zbir += M[v][j];
12    return zbir;
13  }

14 /* Funkcija racuna zbir elemenata k-te kolone */
15 int zbir_kolone(int **M, int n, int k)
16 {
17   int i, zbir = 0;
18   for (i = 0; i < n; i++)
19     zbir += M[i][k];
20   return zbir;
21 }

22 /* Funkcija proverava da li je kvadrat koji joj se prosledjuje kao
23 argument magican. Ukoliko jeste magican funkcija vraca 1, inace 0.
24 Argument funkcije zbir ce sadrzati zbir elemenata neke vrste ili
25 kolone ukoliko je kvadrat magican. */
26 int magični_kvadrat(int **M, int n, int *zbirmagicnog)
27 {
28   int i, j;
29   int zbir = 0, zbir_pom;
30   /* Promenljivu zbir inicijalizujemo na zbir 0-te vreste */
31   zbir = zbir_vrste(M, n, 0);

32   /* Racunaju se zbirovi u ostalim vrstama i ako neki razlikuje od
33    vrednosti promeljive zbir funkcija vraca 1 */
34   for (i = 1; i < n; i++) {
35     zbir_pom = zbir_vrste(M, n, i);
36     if (zbir_pom != zbir)
37       return 0;
38   }
39   /* Racunaju se zbirovi u svim kolonama i ako neki razlikuje od
40    vrednosti promeljive zbir funkcija vraca 1 */
41   for (j = 0; j < n; j++) {
42     zbir_pom = zbir_kolone(M, n, j);
43     if (zbir_pom != zbir)
44       return 0;
45   }
46   /* Inace su zbirovi svih vrsta i kolona jednaki, postavlja se
47    vresnost u zbirmagicnog i funkcija vraca 1 */
48   *zbirmagicnog = zbir;
49   return 1;
50 }

51 int main()
52 {
53   int n, i, j;
54   int **matrica = NULL;
55   int zbir = 0, zbirmagicnog = 0;

```

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

60   /* Provera da li je n strogo pozitivan */
62   if (n <= 0) {
63     printf("-1\n");
64     exit(EXIT_FAILURE);
65   }

66   /* Dinamicka alokacija matrice dimenzije nxn */
68   matrica = alociraj_matricu(n);
69   if (matrica == NULL) {
70     printf("-1\n");
71     exit(EXIT_FAILURE);
72   }

74   /* Ucitavanje elemenata matrice sa standardnog ulaza */
75   ucitaj_matricu(matrica, n);

76   /* Ispisivanje rezultata na osnovu funkcije magicni_kvadrat */
77   if (magicni_kvadrat(matrica, n, &zbirmagicnog)) {
78     printf("%d\n", zbirmagicnog);
79   } else
80     printf("-\n");

82   /* Oslobadjanje dinamicki alocirane memorije */
83   matrica = dealociraj_matricu(matrica, n);

84   exit(EXIT_SUCCESS);
85 }
```