# PROGRAMIRANJE 2

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

# 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
*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu
*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu
*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu
*Anđelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

PROGRAMIRANJE 2
Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu. Studentski trg 16, 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: Copy Centar, Beograd. Tiraž 200.

CIP Каталогизација у публикацији
Народна библиотека Србије, Београд

004.4(075.8)(076)
004.432.2C(075.8)(076)

PROGRAMIRANJE 2 : zbirka zadataka sa rešenjima / Milena Vujošević Janičić ... [et al.]. - Beograd : Matematički fakultet, 2016 (Beograd : Copy Centar). - VII, 361 str. ; 24 cm

Tiraž 200.

ISBN 978-86-7589-107-9

1. Вујошевић Јаничић, Милена 1980- [аутор]

a) Програмирање - Задаци b) Програмски језик "C"- Задаци
COBISS.SR-ID 221508876



# Sadržaj

# Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanja problema, ispravnom radu sa pokazivačima i dinamički alociranom memorijom, osnovnim algoritmima pretraživanja i sortiranja, kao i radu sa dinamičkim strukturama podataka, poput listi i stabala. Zadaci koji se nalaze u ovoj zbirci predstavljaju objedinjen skup zadataka sa vežbi i praktikuma ovog kursa, kao i primere zadataka sa održanih ispita. Elektronska verzija zbirke i propratna rešenja u elektronskom formatu, dostupna su besplatno u okviru strane kursa `www.programiranje2.matf.bg.ac.rs` u skladu sa navedenom licencom.

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. Dodatak sadrži najvažnije ispitne rokove iz jedne akademske godine. Većina zadataka je rešena, a teži zadaci su obeleženi zvezdicom.

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 potrebni za uspešno savlađivanje koncepata koji se obrađuju u okviru kursa. Takođe, formulacije zadataka smo obogatili primerima koji upotpunjuju razumevanje zahteva zadataka i koji omogućavaju čitaocu zbirke da proveri sopstvena rešenja. Primeri su dati u obliku testova i interakcija sa programom. Testovi su svedene prirode i obuhvataju samo jednostavne ulaze i izlaze iz programa. Interakcija sa programom obuhvata naizmeničnu interakciju čovek-računar u kojoj su ulazi i izlazi isprepletani. U zadacima koji zahtevaju

rad sa argumentima komandne linije, navedeni su i primeri poziva programa, a u zadacima koji demonstriraju rad sa datotekama, i primeri ulaznih ili izlaznih datoteka. Test primeri koji su navedeni uz ispitne zadatke u dodatku su oni koji su korišćni za početno testiranje (koje prethodi ocenjivanju) studentskih radova na ispitima.

Neizmerno 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 uobličavanju ovog materijala.

Svi komentari i sugestije na sadržaj zbirke su dobrodošli i osećajte se slobodnim da ih pošaljete elektronskom poštom bilo kome od autora[1].

*Autori*

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

# 1

# Dinamičke strukture podataka

## 1.1 Liste

**Zadatak 1.1** Napisati biblioteku za rad sa jednostruko povezanom listom čiji čvorovi sadrže cele brojeve.

(a) Definisati strukturu `Cvor` kojom se predstavlja čvor liste, a koja 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 vrednosti u čvorovima 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`. Funkcija vraća pokazivač na pronađeni čvor ili $NULL$ ukoliko ga ne pronađe.

(k) Napisati funkciju `Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)` koja proverava da li se u listi nalazi čvor sa vrednošću `broj`, pri čemu se pretpostavlja da se pretražuje neopadajuće uređena lista.

(l) Napisati funkciju `void obrisi_cvor(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`.

(m) Napisati funkciju `void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)` koja briše sve čvorove u listi koji imaju vrednost jednaku argumentu `broj`, pri čemu se pretpostavlja da se briše iz neopadajuće uređene liste.

(n) Napisati funkciju `void oslobodi_listu(Cvor ** adresa_glave)` koja oslobađa dinamički zauzetu memoriju za čvorove liste.

Funkcije dodavanja novog elementa u postojeću listu poput, `dodaj_na_pocetak_liste`, `dodaj_na_kraj_liste` i `dodaj_sortirano`, treba da vrate 0, ako je sve bilo u redu, odnosno 1, ako 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. Zatim se unosi ceo broj koji se traži u unetoj listi i na standardni izlaz se ispisuje rezultat pretrage.

*Primer 1*

```
Interakcija sa programom:
 Unesite brojeve (CTRL+D za kraj unosa):
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:
 Unesite brojeve (CTRL+D za kraj unosa):
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. Zatim se unosi ceo broj čija se sva pojavljivanja u listi brišu i na standardni izlaz se ispisuje sadržaj liste nakon brisanja.

*Primer 1*

```
Interakcija sa programom:
 Unesite brojeve (CTRL+D za kraj unosa):
2
 Lista: [2]
3
 Lista: [2, 3]
14
 Lista: [2, 3, 14]
3
 Lista: [2, 3, 14, 3]
3
 Lista: [2, 3, 14, 3, 3]
17
 Lista: [2, 3, 14, 3, 3, 17]
3
 Lista: [2, 3, 14, 3, 3, 17, 3]

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 14, 17]
```

*Primer 2*

```
Interakcija sa programom:
 Unesite brojeve (CTRL+D za kraj unosa):
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 standardni izlaz 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 (CTRL+D za kraj unosa):
 2
 Lista: [2]
 3
 Lista: [2, 3]
 14
 Lista: [2, 3, 14]
 3
 Lista: [2, 3, 3, 14]
 3
 Lista: [2, 3, 3, 3, 14]
 5
 Lista: [2, 3, 3, 3, 5, 14]

 Unesite broj koji se trazi:  14
 Trazeni broj 14 je u listi!

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [2, 5, 14]
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
 Unesite brojeve (CTRL+D za kraj unosa):
 23
 Lista: [23]
 14
 Lista: [14, 23]
 35
 Lista: [14, 23, 35]

 Unesite broj koji se trazi:  8
 Broj 8 se ne nalazi u listi!

 Unesite broj koji se brise:  3
 Lista nakon brisanja: [14, 23, 35]
```

**Zadatak 1.2** Napisati biblioteku za rad sa jednostruko povezanim listama koja sadrži sve funkcije iz zadatka 1.1, ali tako da funkcije budu implementirane rekurzivno. NAPOMENA: *Koristiti* ***main*** *funkcije i test primere iz zadatka 1.1.*

**Zadatak 1.3** Napisati program koji prebrojava pojavljivanja etiketa `HTML` datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu `Element` koja sadrži neoznačen broj pojavljivanja etiketi, nisku karaktera koja može da prihvati etiketu veličine do 20 karaktera i pokazivač na sledeći element liste.

*Test 2*

```
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 1*

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

DATOTEKA DATOTEKA.HTML NE POSTOJI.

IZLAZ ZA GREŠKE:
  Greska: Neuspesno otvaranje
  datoteke datoteka.html.
```

**Zadatak 1.4** 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. Pretpostaviti 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
```

*** Zadatak 1.5** Data je datoteka `brojevi.txt` koja sadrži cele brojeve.

(a) Napisati funkciju koja iz zadate datoteke učitava brojeve i smešta ih u listu.

(b) Napisati funkciju koja u jednom prolazu kroz zadatu listu celih brojeva pronalazi maksimalan strogo rastući podniz.

Napisati program koji u datoteku `rezultat.txt` upisuje nađeni strogo rastući podniz.

*Test 1*

```
BROJEVI.TXT
 43 12 15 16 4 2 8

IZLAZ:
REZULTAT.TXT
 12 15 16
```

*Test 2*

```
DATOTEKA BROJEVI.TXT
NE POSTOJI.

IZLAZ ZA GREŠKE:
 Greska: Neuspesno otvaranje
 datoteke brojevi.txt.
```

*Test 3*

```
DATOTEKA BROJEVI.TXT JE PRAZNA

IZLAZ:
REZULTAT.TXT
 Rezultat.txt ce biti prazna.
```

*** Zadatak 1.6** 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.

*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 GREŠKE:
 Greska: Neuspesno otvaranje
 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 GREŠKE:
 Greska: Program se poziva sa:
 ./a.out dat1.txt dat2.txt!
```

*** Zadatak 1.7** 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: *Koristiti testove 2 - 6 za zadatak 1.6.*

*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 1.8** Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [ i (. Napisati program koji učitava sadržaj datoteke `izraz.txt` i korišćenjem steka (engl. *stack*) utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz.

*Test 1*

```
IZRAZ.TXT
 {[23 + 5344] * (24 - 234)} - 23

IZLAZ:
 Zagrade su ispravno uparene.
```

*Test 2*

```
IZRAZ.TXT
 {[23 + 5] * (9 * 2)} - {23}

IZLAZ:
 Zagrade su ispravno uparene.
```

*Test 3*

```
IZRAZ.TXT
 {[2 + 54) / (24 * 87)} + (234 + 23)

IZLAZ:
 Zagrade nisu ispravno uparene.
```

*Test 4*

```
IZRAZ.TXT
 {(2 - 14) / (23 + 11)}} * (2 + 13)

IZLAZ:
 Zagrade nisu ispravno uparene.
```

*Test 5*

```
DATOTEKA IZRAZ.TXT JE PRAZNA

IZLAZ:
 Zagrade su ispravno uparene.
```

*Test 6*

```
DATOTEKA IZRAZ.TXT NE POSTOJI.

IZLAZ ZA GREŠKE:
 Greska: Neuspesno otvaranje
 datoteke izraz.txt!
```

**Zadatak 1.9** 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 GREŠKE:
 Greska: Neuspesno otvaranje
 datoteke datoteka.html.
```

*Test 6*

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

DATOTEKA.HTML JE PRAZNA

IZLAZ:
 Etikete su pravilno uparene!
```

**Zadatak 1.10** Napisati program koji pomaže službeniku u radu na šalteru. Službenik najpre evidentira sve korisničke $JMBG$ brojeve (niske koje sadrže po 13 karaktera) i zahteve (niske koja sadrže 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 (engl.* queue*) implementiran korišćenjem listi.*

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
 Sluzbenik evidentira korisnicke zahteve:
 Novi zahtev [CTRL+D za kraj]
  JMBG:   1234567890123
  Opis problema:   Otvaranje racuna

 Novi zahtev [CTRL+D za kraj]
  JMBG:   2345678901234
  Opis problema:   Podizanje novca

 Novi zahtev [CTRL+D za kraj]
  JMBG:   3456789012345
  Opis problema:   Reklamacija

 Novi zahtev [CTRL+D za kraj]
  JMBG:

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 2345678901234
 i zahtevom: Podizanje novca
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG: 3456789012345
 i zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Da

 Sledeci je korisnik sa JMBG: 3456789012345
 i zahtevom: Reklamacija
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

 Da li je kraj smene? [Da/Ne]   Ne

 Sledeci je korisnik sa JMBG: 1234567890123
 i zahtevom: Otvaranje racuna
  Da li ga vracate na kraj reda? [Da/Ne]   Ne

IZVESTAJ.TXT
  JMBG: 2345678901234 Zahtev: Podizanje novca
  JMBG: 3456789012345 Zahtev: Reklamacija
  JMBG: 1234567890123 Zahtev: Otvaranje racuna
```

**Zadatak 1.11** Napisati biblioteku za rad sa dvostruko povezanom listom celih brojeva koja ima iste funkcionalnosti kao biblioteka iz zadatka 1.1. Dopuniti bibilioteku novom definicijom cvore i funkcijama.

(a) Definisati strukturu `Cvor` kojom se predstavlja čvor liste, a koja sadrži ceo broj `vrednost`, pokazivače na sledeći i prethodni čvor liste.

(b) 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`.

(c) 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: *Koristiti test primere iz zadatka 1.1*

*** Zadatak 1.12** Grupa od $n$ plesača na kostimima ima brojeve od 1 do $n$. Plesači najpre formiraju krug tako da brojevi sa njihovih kostimima 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 stavlja 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 povezanu kružnu listu.*

*Test 1*

```
ULAZ:
 5 3

IZLAZ:
 3 1 5 2 4
```

*Test 2*

```
ULAZ:
 8 4

IZLAZ:
 4 8 5 2 1 3 7 6
```

*Test 3*

```
ULAZ:
 3 8

IZLAZ ZA GREŠKE:
 Greska: n mora biti uvek vece
 od k, a 3 < 8!
```

*** Zadatak 1.13** Grupa od $n$ plesača na kostimima ima brojeve od 1 do $n$. Plesači najpre formiraju krug tako da brojevi 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.*

*Test 1*

```
ULAZ:
 5 3

IZLAZ:
 3 5 4 2 1
```

*Test 2*

```
ULAZ:
 8 4

IZLAZ:
 4 8 5 7 6 3 2 1
```

*Test 3*

```
ULAZ:
 5 8

IZLAZ ZA GREŠKE:
 Greska: n mora biti uvek vece
 od k, a 5 < 8!
```

## 1.2 Stabla

**Zadatak 1.14** Napisati biblioteku za rad sa binarnim pretraživačkim stablima.

(a) Definisati strukturu `Cvor` kojom se opisuje čvor stabla, a koja sadrži ceo broj `broj` i pokazivače `levo` i `desno` redom na levo i desno podstablo.

(b) Napisati funkciju `Cvor *napravi_cvor(int broj)` koja alocira memoriju za novi čvor stabla i vrši njegovu inicijalizaciju zadatim celim brojem `broj`.

(c) Napisati funkciju `int dodaj_u_stablo(Cvor ** adresa_korena, int broj)` koja u stablo na koje pokazuje argument `adresa_korena` dodaje ceo broj `broj`. Povratna vrednost funkcije je 0 ako je dodavanje uspešno, odnosno 1 ukoliko je došlo do greške.

(d) Napisati funkciju `Cvor *pretrazi_stablo(Cvor * koren, int broj)` koja proverava da li se ceo broj `broj` nalazi u stablu sa korenom `koren`. Funkcija vraća pokazivač na čvor stabla koji sadrži traženu vrednost ili *NULL* ukoliko takav čvor ne postoji.

(e) Napisati funkciju `Cvor *pronadji_najmanji(Cvor * koren)` koja pronalazi čvor koji sadrži najmanju vrednost u stablu sa korenom `koren`.

(f) Napisati funkciju `Cvor *pronadji_najveci(Cvor * koren)` koja pronalazi čvor koji sadrži najveću vrednost u stablu sa korenom `koren`.

(g) Napisati funkciju `void obrisi_element(Cvor ** adresa_korena, int broj)` koja briše čvor koji sadrži vrednost `broj` iz stabla na koje pokazuje argument `adresa_korena`.

(h) Napisati funkciju `void ispisi_stablo_infiksno(Cvor * koren)` koja infiksno ispisuje sadržaj stabla sa korenom `koren`. Infiksni ispis podrazumeva ispis levog podstabla, korena, a zatim i desnog podstabla.

(i) Napisati funkciju `void ispisi_stablo_prefiksno(Cvor * koren)` koja prefiksno ispisuje sadržaj stabla sa korenom `koren`. Prefiksni ispis podrazumeva ispis korena, levog podstabla, a zatim i desnog podstabla.

(j) Napisati funkciju `void ispisi_stablo_postfiksno(Cvor * koren)` koja postfiksno ispisuje sadržaj stabla sa korenom `koren`. Postfiksni ispis podrazumeva ispis levog podstabla, desnog podstabla, a zatim i korena.

(k) Napisati funkciju `void oslobodi_stablo(Cvor ** adresa_korena)` koja oslobađa memoriju zauzetu stablom na koje pokazuje argument `adresa_korena`.

Korišćenjem kreirane biblioteke, napisati program koji sa standardnog ulaza učitava cele brojeve sve do kraja ulaza, dodaje ih u binarno pretraživačko stablo i ispisuje stablo u svakoj od navedenih notacija. Zatim omogućiti unos još dva cela broja i demonstrirati rad funkcije za pretragu nad prvim unetim brojem i rad funkcije za brisanje elemenata nad drugim unetim brojem.

*Primer 1*

```
Interakcija sa programom:
Unesite brojeve (CRL+D za kraj unosa):
7 2 1 9 32 18
Infiksni ispis: 1 2 7 9 18 32
Prefiksni ispis: 7 2 1 9 32 18
Postfiksni ispis: 1 2 18 32 9 7
Trazi se broj: 11
Broj se ne nalazi u stablu!
Brise se broj: 7
Rezultujuce stablo: 1 2 9 18 32
```

*Primer 2*

```
Interakcija sa programom:
Unesite brojeve (CRL+D za kraj unosa):
8 -2 6 13 24 -3
Infiksni ispis: -3 -2 6 8 13 24
Prefiksni ispis: 8 -2 -3 6 13 24
Postfiksni ispis: -3 6 -2 24 13 8
Trazi se broj: 6
Broj se nalazi u stablu!
Brise se broj: 14
Rezultujuce stablo: -3 -2 6 8 13 24
```

**Zadatak 1.15** 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 pretraživačkog stabla uređenog leksikografski po rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati na standardni izlaz za grešku poruku *Nedostaje ime ulazne datoteke*!. Može se pretpostaviti 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

IZLAZ ZA GREŠKE:
  Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

*Test 4*

```
POKRETANJE: ./a.out

IZLAZ ZA GREŠKE:
  Greska: Nedostaje ime ulazne datoteke!
```

**Zadatak 1.16** U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona međusobno razdvojeni blanko znakom, na primer *Milos Peric* 064/123 − 4567. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči $KRAJ$, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera, kao i da imenik ne sadrži podatke o osobama sa istim imenom i prezimenom.

*Primer 1*

```
IMENIK.TXT
  Milos 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: Milos Peric
  Broj je: 011/3240-987
  Unesite ime i prezime: Marko Markovic
  Broj nije u imeniku!
  Unesite ime i prezime: KRAJ
```

*Primer 2*

```
DATOTEKA IMENIK1.TXT NE POSTOJI

INTERAKCIJA SA PROGRAMOM:
  Unesite ime datoteke: imenik1.txt
IZLAZ ZA GREŠKE:
  Greska: Neuspesno otvaranje datoteke
  imenik1.txt.
```

**Zadatak 1.17** U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niska od najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na prijemnom ispitu iz maternjeg jezika (realan broj). Za učenika koji u zbiru osvoji manje od 10 poena na oba prijemna ispita smatra se da nije položio prijemni. Napisati program koji na osnovu podataka iz ove datoteke formira i prikazuje rang listu učenika. Rang lista sadrži redni broj učenika, njegovo ime i prezime, broj poena na osnovu uspeha, broj poena na prijemnom ispitu iz matematike, broj poena na prijemnom ispitu iz maternjeg jezika i ukupan broj poena i sortirana je opadajuće po ukupnom broju poena. Na rang listi se prvo navode oni učenici koji su položili prijemni ispit, a potom i učenici koji ga nisu položili. Između ovih dveju grupa učenika postoji i horizontalna linija koja ih vizuelno razdvaja.

*Test 1*

```
PRIJEMNI.TXT
  Marko Markovic 45.4 12.3 11
  Milan Jevremovic 35.2 1.3 9
  Maja Agic 60 19 20
  Nadica Zec 54.2 10 15.8
  Jovana Milic 23.3 2 5.6

IZLAZ:
  1. Maja Agic 60.0 19.0 20.0 99.0
  2. Nadica Zec 54.2 10.0 15.8 80.0
  3. Marko Markovic 45.4 12.3 11.0 68.7
  4. Milan Jevremovic 35.2 1.3 9.0 45.5
  ------------------------------------
  5. Jovana Milic 23.3 2.0 5.6 30.9
```

*Test 2*

```
DATOTEKA PRIJEMNI.TXT NE POSTOJI

IZLAZ ZA GREŠKE:
  Greska: Neuspesno otvaranje datoteke
  prijemni.txt.
```

*** Zadatak 1.18** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije. Svaka linija datoteke je formata *Ime Prezime DD.MM.* i sadži ime osobe, prezime osobe i dan i mesec rođenja. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj unosa. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima - prvo po mesecu, a zatim po danu u okviru istog meseca. Može se pretpostaviti da će svi korišćeni datumi biti validni i *DD.MM.* formata. 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

IZLAZ ZA GREŠKE:
  Greska: Neuspesno otvaranje datoteke
  rodjendani.txt.
```

**Zadatak 1.19** 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 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 1.14.*

*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.
```

*** Zadatak 1.20** 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 1.14.*

*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
```

**Zadatak 1.21** 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 1.14.*

*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!
```

**Zadatak 1.22** 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 1.14.*

*Test 1*

```
POKRETANJE: ./a.out 2 15

ULAZ:
  10 5 15 3 2 4 30 12 14 13

IZLAZ:
  Broj cvorova: 10
  Broj listova: 4
  Pozitivni listovi: 2 4 13 30
  Zbir cvorova: 108
  Najveci element: 30
  Dubina stabla: 5
  Broj cvorova na 2. nivou: 3
  Elementi na 2. nivou: 3 12 30
  Maksimalni element na 2. nivou: 30
  Zbir elemenata na 2. nivou: 45
  Zbir elemenata manjih ili jednakih od 15:
  78
```

*Test 2*

```
POKRETANJE: ./a.out 3 31

ULAZ:
  24 53 61 9 7 55 20 16

IZLAZ:
  Broj cvorova: 8
  Broj listova: 3
  Pozitivni listovi: 7 16 55
  Zbir cvorova: 245
  Najveci element: 61
  Dubina stabla: 4
  Broj cvorova na 3. nivou: 2
  Elementi na 3. nivou: 16 55
  Maksimalni element na 3. nivou: 55
  Zbir elemenata na 3. nivou: 71
  Zbir elemenata manjih ili jednakih od 31:
  76
```

**Zadatak 1.23** 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 1.14.*

*Test 1*

```
Ulaz:
 10 5 15 3 2 4 30 12 14 13

Izlaz:
 0.nivo: 10
 1.nivo: 5 15
 2.nivo: 3 12 30
 3.nivo: 2 4 14
 4.nivo: 13
```

*Test 2*

```
Ulaz:
 6 11 8 3 -2

Izlaz:
 0.nivo: 6
 1.nivo: 3 11
 2.nivo: -2 8
```

*Test 3*

```
Ulaz:
 24 53 61 9 7 55 20 16

Izlaz:
 0.nivo: 24
 1.nivo: 9 53
 2.nivo: 7 20 61
 3.nivo: 16 55
```

*** Zadatak 1.24** Dva binarna stabla su *slična kao u ogledalu* ako su ili oba prazna ili ako oba nisu prazna i levo podstablo svakog stabla je *slično kao u ogledalu* desnom podstablu onog drugog (bitna je struktura stabala, ali ne i njihov sadržaj). Napisati funkciju koja proverava da li su dva binarna pretraživačka stabla *slična kao u ogledalu*, a potom i program koji testira rad funkcije nad stablima čiji se elementi unose sa standardnog ulaza sve do unosa broja 0 i to redom za prvo stablo, pa zatim i za drugo stablo.

*Primer 1*

```
Interakcija sa programom:
 Prvo stablo: 11 20 5 3 0
 Drugo stablo: 8 14 30 1 0
 Stabla su slicna kao u ogledalu.
```

*Primer 2*

```
Interakcija sa programom:
 Prvo stablo: 11 20 5 3 0
 Drugo stablo: 8 20 15 0
 Stabla nisu slicna kao u ogledalu.
```

**Zadatak 1.25** *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 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 1.14.*

*Test 1*

```
Ulaz:
 10 5 15 2 11 16 1 13

Izlaz:
 7
```

*Test 2*

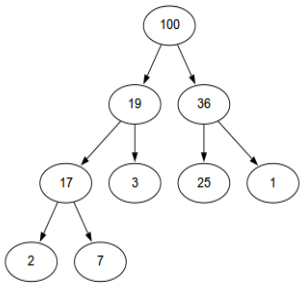
```
Ulaz:
 16 30 40 24 10 18 45 22

Izlaz:
 6
```

**Zadatak 1.26** Binarno stablo celih pozitivnih brojeva se naziva *hip* (engl. *heap*) ako za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int hip(Cvor * koren)`

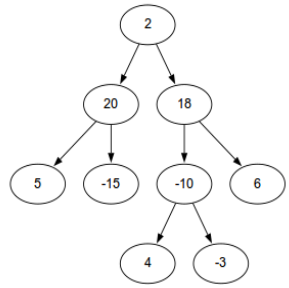
koja proverava da li je dato binarno stablo celih brojeva hip. Napisati zatim i program koji kreira stablo zadato slikom 1.1, poziva funkciju `hip` i ispisuje rezultat na standardni izlaz. NAPOMENA: *Za alokaciju i oslobađanje memorije koristiti funkcije* ***`napravi_cvor`*** *i* ***`oslobodi_stablo`*** *iz zadatka 1.14.*

*Test 1*

```
IZLAZ:
 Zadato stablo je hip!
```




Slika 1.1: *Zadatak 1.26*




Slika 1.2: *Zadatak 1.27*

**Zadatak 1.27** Dato je binarno stablo celih brojeva.

(a) Napisati funkciju koja pronalazi čvor u stablu sa najvećim zbirom vrednosti iz desnog podstabla.

(b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjim zbirom vrednosti iz levog podstabla.

(c) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do najdubljeg čvora.

(d) Napisati funkciju koja štampa sadržaj svih čvorova stabla na putanji od korena do čvora koji ima najmanju vrednost u stablu.

Napisati program koji testira gore navedene funkcije nad stablom zadatim slikom 1.2 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
```

## 1.3 Rešenja

### Rešenje 1.1

*lista.h*

```
#ifndef _LISTA_H_
#define _LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
   podatak vrednost i pokazivac na sledeci cvor liste */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na
   broj, dok pokazivac na sledeci cvor postavlja na NULL. Vraca
   pokazivac na novokreirani cvor ili NULL ako alokacija nije bila
   uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pokazivac glava nalazi na adresi adresa_glave. */
void oslobodi_listu(Cvor ** adresa_glave);

/* Funkcija dodaje broj na pocetak liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

/* Funkcija pronalazi i vraca pokazivac na poslednji cvor liste,
   ili NULL ukoliko je lista prazna. */
Cvor *pronadji_poslednji(Cvor * glava);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

/* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   nov cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

/* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
int dodaj_iza(Cvor * tekuci, int broj);

/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
   memorije, inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, int broj);
```

```c

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom
   broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
   broj ili NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom
   broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
   broj ili NULL ako da takav cvor ne postoji. U pretrazi oslanja
   se na cinjenicu se pretrazuje neopadajuce sortirana lista. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
   Azurira pokazivac na glavu liste, koji moze biti promenjen u
   slucaju da se obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
   neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
   kraju liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

#endif
```

*lista.c*

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

Cvor *napravi_cvor(int broj)
{
  /* Alokacija memorije za novi cvor uz proveru uspesnosti
     alokacije. */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

```

```c
void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  /* Ako lista nije prazna, onda treba osloboditi memoriju */
  while (*adresa_glave != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor koji predstavlja glavu liste */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);

    /* Sledeci cvor je nova glava liste */
    *adresa_glave = pomocni;
  }
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

Cvor *pronadji_poslednji(Cvor * glava)
{
  /* U praznoj listi nema cvorova pa se vraca NULL */
  if (glava == NULL)
    return NULL;

  /* Sve dok glava pokazuje na cvor koji ima sledbenika, pokazivac
     glava se pomera na sledeci cvor. Nakon izlaska iz petlje,
     glava ce pokazivati na cvor liste koji nema sledbenika, tj. na
     poslednji cvor liste, pa se vraca vrednost pokazivaca glava.
     Pokazivac glava je argument funkcije i njegove promene nece se
     odraziti na vrednost pokazivaca glava u pozivajucoj funkciji. */
  while (glava->sledeci != NULL)
    glava = glava->sledeci;

  return glava;
}

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
```

```
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {
    /* Glava nove liste je upravo novi cvor */
    *adresa_glave = novi;
  } else {
    /* Ako lista nije prazna, pronalazi se poslednji cvor i novi
       cvor se dodaje na kraj liste kao sledbenik poslednjeg */
    Cvor *poslednji = pronadji_poslednji(*adresa_glave);
    poslednji->sledeci = novi;
  }

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
  /* U praznoj listi nema takvog mesta i vraca se NULL */
  if (glava == NULL)
    return NULL;

  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
     pokazivao na cvor ciji sledeci ili ne postoji ili ima vrednost
     vecu ili jednaku vrednosti novog cvora. */
  /* Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
     provera da li se doslo do poslednjeg cvora liste pre nego sto
     se proveri vrednost u sledecem cvoru, jer u slucaju poslednjeg,
     sledeci ne postoji, pa ni njegova vrednost. */
  while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
    glava = glava->sledeci;

  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci
     ima vrednost vecu od broj. */
  return glava;
}

int dodaj_iza(Cvor * tekuci, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se dodaje iza tekuceg cvora. */
  novi->sledeci = tekuci->sledeci;
  tekuci->sledeci = novi;

```

```
  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {
    /* Glava nove liste je novi cvor */
    /* Kreiranje novog cvora uz proveru uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
      return 1;

    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Inace, ako je broj manji ili jednak vrednosti u glavi liste,
     onda ga treba dodati na pocetak liste. */
  if ((*adresa_glave)->vrednost >= broj) {
    return dodaj_na_pocetak_liste(adresa_glave, broj);
  }

  /* U slucaju da je glava liste cvor sa vrednoscu manjom od broj,
     tada se pronalazi cvor liste iza koga treba uvezati nov cvor */
  Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  return dodaj_iza(pomocni, broj);
}

Cvor *pretrazi_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  for (; glava != NULL; glava = glava->sledeci)
    /* Ako je vrednost tekuceg cvora jednaka zadatom broju,
       pretraga se obustavlja */
    if (glava->vrednost == broj)
      return glava;

  /* Nema trazenog broja u listi i vraca se NULL */
  return NULL;
}

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
  /* Obilaze se cvorovi liste */
  /* U uslovu ostanka u petlji, bitan je redosled provera u
     konjunkciji. */
  while (glava != NULL && glava->vrednost < broj)
    glava = glava->sledeci;
```

```
178   /* Iz petlje se moglo izaci na vise nacina. Prvi, tako sto je
         glava->vrednost veca od trazenog broja i tada treba vratiti
180      NULL, jer trazen broj nije nadjen medju manjim brojevima pri
         pocetku sortirane liste, pa se moze zakljuciti da ga nema u
182      listi. Drugi nacini, tako sto se doslo do kraja liste i glava
         je NULL ili tako sto je glava->vrednost == broj. U oba
184      poslednja nacina treba vratiti pokazivac glava bilo da je NULL
         ili pokazivac na konkretan cvor. */
186   if (glava->vrednost > broj)
        return NULL;
188   else
        return glava;
190 }

192 void obrisi_cvor(Cvor ** adresa_glave, int broj)
    {
194   Cvor *tekuci = NULL;
      Cvor *pomocni = NULL;
196
      /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
198       broju i azurira se pokazivac na glavu liste */
      while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
        {
200     /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
           adresi adresa_glave */
202     pomocni = (*adresa_glave)->sledeci;
        free(*adresa_glave);
204     *adresa_glave = pomocni;
      }
206
      /* Ako je nakon ovog brisanja lista prazna, vraca se iz funkcije */
208   if (*adresa_glave == NULL)
        return;
210
      /* Od ovog trenutka, u svakoj iteraciji petlje promenljiva tekuci
212       pokazuje na cvor cija je vrednost razlicita od trazenog broja.
         Isto vazi i za sve cvorove levo od tekuceg. Poredi se vrednost
214      sledeceg cvora (ako postoji) sa trazenim brojem. Cvor se brise
         ako je jednak, a ako je razlicit, prelazi se na sledeci. Ovaj
216      postupak se ponavlja dok se ne dodje do poslednjeg cvora. */
      tekuci = *adresa_glave;
218   while (tekuci->sledeci != NULL)
        if (tekuci->sledeci->vrednost == broj) {
220       /* tekuci->sledeci treba obrisati, zbog toga se njegova
             adresa prvo cuva u promenljivoj pomocni. */
222       pomocni = tekuci->sledeci;
          /* Tekucem se preusmerava pokazivac sledeci, preskakanjem
224          njegovog trenutnog sledeceg. Njegov novi sledeci ce biti
             sledeci od cvora koji se brise. */
226       tekuci->sledeci = pomocni->sledeci;
          /* Sada treba osloboditi cvor sa vrednoscu broj. */
```

```
228       free(pomocni);
        } else {
230       /* Inace, ne treba brisati sledeceg od tekuceg i pokazivac se
             pomera na sledeci. */
232       tekuci = tekuci->sledeci;
        }
234   return;
}
236
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
238 {
  Cvor *tekuci = NULL;
240   Cvor *pomocni = NULL;

242   /* Sa pocetka liste se brisu svi cvorovi koji su jednaki datom
        broju i azurira se pokazivac na glavu liste. */
244   while (*adresa_glave != NULL && (*adresa_glave)->vrednost == broj)
        {
      /* Adresu repa liste treba sacuvati pre oslobadjanja cvora na
246        adresi adresa_glave. */
      pomocni = (*adresa_glave)->sledeci;
248     free(*adresa_glave);
      *adresa_glave = pomocni;
250   }

252   /* Ako je nakon ovog brisanja lista ostala prazna, funkcija se
        prekida. Isto se radi i ukoliko glava liste sadrzi vrednost
254     koja je veca od broja, jer kako je lista sortirana rastuce
        nema potrebe broj traziti u repu liste. */
256   if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
      return;
258
  /* Od ovog trenutka se u svakoj iteraciji pokazivac tekuci
260     pokazuje na cvor cija vrednost je manja od trazenog broja, kao
      i svim cvorovima levo od njega. Cvor se brise ako je jednak,
262     ili, ako je razlicit, prelazi se na sledeci cvor. Ovaj postupak
      se ponavlja dok se ne dodje do poslednjeg cvora ili prvog cvora
264     cija vrednost je veca od trazenog broja. */
  tekuci = *adresa_glave;
266   while (tekuci->sledeci != NULL
          && tekuci->sledeci->vrednost <= broj)
268     if (tekuci->sledeci->vrednost == broj) {
        pomocni = tekuci->sledeci;
270     tekuci->sledeci = tekuci->sledeci->sledeci;
        free(pomocni);
272   } else {
        /* Ne treba brisati sledeceg od tekuceg jer je manji od
274        trazenog i tekuci se pomera na sledeci cvor. */
        tekuci = tekuci->sledeci;
276   }
  return;
278 }
```

```
280 void ispisi_listu(Cvor * glava)
{
282   /* Funkciji se ne salje adresa promenljive koja cuva glavu liste
         jer se lista nece menjati */
284   putchar('[');
      /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
286      pocetka prema kraju liste. */
      for (; glava != NULL; glava = glava->sledeci) {
288     printf("%d", glava->vrednost);
        if (glava->sledeci != NULL)
290       printf(", ");
      }
292   printf("]\n");
}
```

*main_a.c*

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

5  int main()
   {
7    /* Lista je prazna na pocetku */
     Cvor *glava = NULL;
9    Cvor *trazeni = NULL;
     int broj;
11
     /* Testiranje funkcije za dodavanja novog broja na pocetak liste */
13   printf("Unesite brojeve (CTRL+D za kraj unosa): ");
     while (scanf("%d", &broj) > 0) {
15     /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
          memorije za nov cvor. Memoriju alociranu za cvorove liste
17        treba osloboditi pre napustanja programa. */
       if (dodaj_na_pocetak_liste(&glava, broj) == 1) {
19       fprintf(stderr,
                 "Greska: Neuspesna alokacija memorije za cvor.\n");
21       oslobodi_listu(&glava);
         exit(EXIT_FAILURE);
23     }
       printf("\tLista: ");
25     ispisi_listu(glava);
     }
27
     /* Testiranje funkcije za pretragu liste */
29   printf("\nUnesite broj koji se trazi: ");
     scanf("%d", &broj);
31
     trazeni = pretrazi_listu(glava, broj);
33   if (trazeni == NULL)
```

```
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

*main_b.c*

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

int main()
{
  /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
  int broj;

  /* Testira se funkcija za dodavanja novog broja na kraj liste */
  printf("Unesite brojeve: (za kraj CTRL+D)\n");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_na_kraj_liste(&glava, broj) == 1) {
      fprintf(stderr,
              "Greska: Neuspesna alokacija memorije za cvor.\n");
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcije kojom se brise cvor liste */
  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
  obrisi_cvor(&glava, broj);

  printf("Lista nakon brisanja:  ");
  ispisi_listu(glava);

  /* Oslobadja se memorija zauzeta listom */
  oslobodi_listu(&glava);
```

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

*main_c.c*

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

int main()
{
  /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se funkcija za dodavanje vrednosti u listu tako da
     bude uredjena neopadajuce */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za nov cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa. */
    if (dodaj_sortirano(&glava, broj) == 1) {
      fprintf(stderr,
              "Greska: Neuspesna alokacija memorije za cvor.\n");
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcija za pretragu liste */
  printf("\nUnesite broj koji se trazi: ");
  scanf("%d", &broj);

  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  /* Testira se funkcija kojom se brise cvor liste */
  printf("\nUnesite broj koji se brise: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
  obrisi_cvor_sortirane_liste(&glava, broj);

  printf("Lista nakon brisanja:  ");
```

```
47    ispisi_listu(glava);

49    /* Oslobadja se memorija zauzeta listom */
      oslobodi_listu(&glava);
51
      exit(EXIT_SUCCESS);
53  }
```

### Rešenje 1.2

*lista.h*

```
1   #ifndef _LISTA_H_
    #define _LISTA_H_
3
    /* Struktura kojom je predstavljen cvor liste sadrzi celobrojni
5      podatak vrednost i pokazivac na sledeci cvor liste. */
    typedef struct cvor {
7     int vrednost;
      struct cvor *sledeci;
9   } Cvor;

11  /* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na
       broj, dok pokazivac na sledeci cvor postavlja na NULL. Vraca
13     pokazivac na novokreirani cvor ili NULL ako alokacija nije bila
       uspesna. */
15  Cvor *napravi_cvor(int broj);

17  /* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
       ciji se pokazivac glava nalazi na adresi adresa_glave. */
19  void oslobodi_listu(Cvor ** adresa_glave);

21  /* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
       bilo greske pri alokaciji memorije, inace vraca 0. */
23  int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj);

25  /* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo
       greske pri alokaciji memorije, inace vraca 0. */
27  int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj);

29  /* Funkcija dodaje broj u rastuce sortiranu listu tako da nova
       lista ostane sortirana. Vraca 1 ukoliko je bilo greske pri
31     alokaciji memorije, inace vraca 0. */
    int dodaj_sortirano(Cvor ** adresa_glave, int broj);
33
    /* Funkcija trazi u listi cvor cija je vrednost jednaka datom
35     broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
       broj ili NULL u slucaju da takav cvor ne postoji u listi. */
37  Cvor *pretrazi_listu(Cvor * glava, int broj);
```

```c
/* Funkcija trazi u listi cvor cija je vrednost jednaka datom
   broju. U pretrazi oslanja se na cinjenicu da je lista koja se
   pretrazuje neopadajuce sortirana. Vraca pokazivac na cvor liste
   u kome je sadrzan trazeni broj ili NULL ako takav cvor ne
   postoji. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
   Azurira pokazivac na glavu liste, koji moze biti promenjen u
   slucaju da se obrise stara glava liste. */
void obrisi_cvor(Cvor ** adresa_glave, int broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista sortirana
   neopadajuce. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj);

/* Funkcija ispisuje samo vrednosti cvorova liste razdvojene
   zapetama. */
void ispisi_vrednosti(Cvor * glava);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
   kraju liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

#endif
```

*lista.c*

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

Cvor *napravi_cvor(int broj)
{
  /* Alokacija memorije za novi cvor uz proveru uspesnosti */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave)
{
```

```
  /* Ako je lista vec prazna */
  if (*adresa_glave == NULL)
    return;

  /* Ako lista nije prazna, treba osloboditi memoriju. Treba
     osloboditi rep, pre oslobadjanja memorije za glavu liste. */
  oslobodi_listu(&(*adresa_glave)->sledeci);
  /* Nakon oslobodjenog repa, oslobadja se glava liste i azurira se
     glava u pozivajucoj funkciji tako da odgovara praznoj listi */
  free(*adresa_glave);
  *adresa_glave = NULL;
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na pocetak i postaje nova glave liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

int dodaj_na_kraj_liste(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {

    /* Novi cvor postaje glava liste */
    Cvor *novi = napravi_cvor(broj);
    /* Ako je bilo greske pri kreiranju novog cvora, vraca se 1 */
    if (novi == NULL)
      return 1;

    /* Azuriranjem vrednosti na koju pokazuje adresa_glave, ujedno
       se azurira i pokazivacka promenljiva u pozivajucoj funkciji */
    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Ako lista nije prazna, broj se dodaje u rep liste. */
  /* Prilikom dodavanja u listu na kraj u velikoj vecini slucajeva
     novi broj se dodaje u rep liste u rekurzivnom pozivu.
     Informaciju o uspesnosti alokacije u rekurzivnom pozivu
     funkcija prosledjuje visem rekurzivnom pozivu koji tu
```

```
     informaciju vraca u rekurzivni poziv iznad, sve dok se ne
     vrati u main. Tek je iz main funkcije moguce pristupiti pravom
     pocetku liste i osloboditi je celu, ako ima potrebe. Ako je
     funkcija vratila 0, onda nije bilo greske. */
  return dodaj_na_kraj_liste(&(*adresa_glave)->sledeci, broj);
}

int dodaj_sortirano(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna */
  if (*adresa_glave == NULL) {

    /* Novi cvor postaje glava liste */
    Cvor *novi = napravi_cvor(broj);

    /* Ako je bilo greske pri kreiranju novog cvora, vraca se 1 */
    if (novi == NULL)
      return 1;

    /* Azurira se glava liste */
    *adresa_glave = novi;

    /* Vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Lista nije prazna. Ako je broj manji ili jednak od vrednosti u
     glavi liste, onda se dodaje na pocetak liste */
  if ((*adresa_glave)->vrednost >= broj)
    return dodaj_na_pocetak_liste(adresa_glave, broj);

  /* Inace, broj treba dodati u rep liste, tako da rep i sa novim
     cvorom bude sortirana lista. */
  return dodaj_sortirano(&(*adresa_glave)->sledeci, broj);
}

Cvor *pretrazi_listu(Cvor * glava, int broj)
{
  /* U praznoj listi nema vrednosti */
  if (glava == NULL)
    return NULL;

  /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
  if (glava->vrednost == broj)
    return glava;

  /* Inace, pretraga se nastavlja u repu liste */
  return pretrazi_listu(glava->sledeci, broj);
}

Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
```

```
  /* Trazenog broja nema ako je lista prazna ili broj manji od
     vrednosti u glavi liste, jer je lista neopadajuce sortirana */
  if (glava == NULL || glava->vrednost > broj)
    return NULL;

  /* Ako glava liste sadrzi trazeni broj, vraca se pokazivac glava */
  if (glava->vrednost == broj)
    return glava;

  /* Inace, pretraga se nastavlja u repu liste */
  return pretrazi_listu(glava->sledeci, broj);
}

void obrisi_cvor(Cvor ** adresa_glave, int broj)
{
  /* U praznoj listi nema cvorova za brisanje. */
  if (*adresa_glave == NULL)
    return;

  /* Prvo se brisu cvorovi iz repa koji imaju vrednost broj */
  obrisi_cvor(&(*adresa_glave)->sledeci, broj);

  /* Preostaje provera da li glavu liste treba obrisati */
  if ((*adresa_glave)->vrednost == broj) {
    /* Pomocni pokazuje na cvor koji treba da se obrise */
    Cvor *pomocni = *adresa_glave;
    /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi
       i brise se cvor koji je bio glava liste. */
    *adresa_glave = (*adresa_glave)->sledeci;
    free(pomocni);
  }
}

void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, int broj)
{
  /* Ako je lista prazna ili glava liste sadrzi vrednost koja je
     veca od broja, kako je lista sortirana rastuce nema potrebe
     broj traziti u repu liste i zato se funkcija prekida */
  if (*adresa_glave == NULL || (*adresa_glave)->vrednost > broj)
    return;

  /* Brisu se cvorovi iz repa koji imaju vrednost broj */
  obrisi_cvor(&(*adresa_glave)->sledeci, broj);

  /* Preostaje provera da li glavu liste treba obrisati */
  if ((*adresa_glave)->vrednost == broj) {
    /* Pomocni pokazuje na cvor koji treba da se obrise */
    Cvor *pomocni = *adresa_glave;
    /* Azurira se pokazivac na glavu da pokazuje na sledeci u listi
       i brise se cvor koji je bio glava liste */
    *adresa_glave = (*adresa_glave)->sledeci;
    free(pomocni);
```

```
  }
}

void ispisi_vrednosti(Cvor * glava)
{
  /* Prazna lista */
  if (glava == NULL)
    return;

  /* Ispisuje se vrednost u glavi liste */
  printf("%d", glava->vrednost);

  /* Ako rep nije prazan, ispisuje se znak ',' i razmak. Rekurzivno
     se poziva ista funkcija za ispis ostalih. */
  if (glava->sledeci != NULL) {
    printf(", ");
    ispisi_vrednosti(glava->sledeci);
  }
}

void ispisi_listu(Cvor * glava)
{
  /* Funkciji se ne salje adresa promenljive koja cuva glavu liste,
     jer nece menjati listu, pa nema ni potrebe da azuriza pokazivac
     na glavu liste iz pozivajuce funkcije. Ona ispisuje samo
     zagrade, a rekurzivno ispisivanje vrednosti u listi prepusta
     rekurzivnoj pomocnoj funkciji ispisi_vrednosti, koja ce ispisati
     elemente razdvojene zapetom i razmakom. */
  putchar('[');
  ispisi_vrednosti(glava);
  printf("]\n");
}
```

### Rešenje 1.3

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

#define MAX_DUZINA 20

/* Struktura kojom je predstavljen cvor liste sadrzi naziv etikete,
   broj pojavljivanja etikete i pokazivac na sledeci cvor liste */
typedef struct _Cvor {
  char etiketa[20];
  unsigned broj_pojavljivanja;
  struct _Cvor *sledeci;
} Cvor;

/* Pomocna funkcija koja kreira cvor. Vraca pokazivac na novi cvor
   ili NULL ako alokacija nije uspesno izvrsena */
```

```c
Cvor *napravi_cvor(unsigned br, char *etiketa)
{
  /* Alokacija memorije za cvor uz proveru uspesnosti alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->broj_pojavljivanja = br;
  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

/* Funkcija oslobadja dinamicku memoriju zauzetu cvorovima liste */
void oslobodi_listu(Cvor ** adresa_glave)
{
  Cvor *pomocni = NULL;

  /* Sve dok lista ni bude prazna, brise se tekuca glava liste i
     azurira se vrednost glave liste */
  while (*adresa_glave != NULL) {
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    *adresa_glave = pomocni;
  }
  /* Pokazivac glava u main funkciji, na adresi adresa_glave, bice
     postavljen na NULL vrednost po izlasku iz petlje. */
}

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
   do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, unsigned br,
                           char *etiketa)
{
  /* Kreira se novi cvor i proverava se uspesnost alokacije */
  Cvor *novi = napravi_cvor(br, etiketa);
  if (novi == NULL)
    return 1;

  /* Dodaje se novi cvor na pocetak liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

/* Funkcija vraca cvor koji kao vrednost sadrzi trazenu etiketu ili
   NULL ako takav cvor ne postoji u listi. */
```

```c
Cvor *pretrazi_listu(Cvor * glava, char etiketa[])
{
  Cvor *tekuci;

  /* Obilazi se cvor po cvor liste */
  for (tekuci = glava; tekuci != NULL; tekuci = tekuci->sledeci)
    /* Ako tekuci cvor sadrzi trazenu etiketu, vraca se njegova
       vrednost */
    if (strcmp(tekuci->etiketa, etiketa) == 0)
      return tekuci;

  /* Cvor nije pronadjen */
  return NULL;
}

/* Funkcija ispisuje sadrzaj liste */
void ispisi_listu(Cvor * glava)
{
  /* Pocevsi od cvora koji je glava lista, ispisuju se sve etikete
     i broj njihovog pojavljivanja u HTML datoteci. */
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s - %u\n", glava->etiketa, glava->broj_pojavljivanja);
}

int main(int argc, char **argv)
{
  /* Proverava se da li je program pozvan sa ispravnim brojem
     argumenata komandne linije. */
  if (argc != 2) {
    fprintf(stderr,
            "Greska: Program se poziva sa: ./a.out datoteka.html\n");
    exit(EXIT_FAILURE);
  }

  /* Priprema datoteke za citanje */
  FILE *in = NULL;
  in = fopen(argv[1], "r");
  if (in == NULL) {
    fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke %s!\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  char c;
  int i = 0;
  char procitana[MAX_DUZINA];
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;

  /* Cita se datoteka, karakter po karakter, dok se ne procita
     karakter za kraj sadrzaja datoteke. */
  while ((c = fgetc(in)) != EOF) {
```

```
121     /* Proverava se da li se pocinje sa citanjem nove etikete */
        if (c == '<') {
123       /* Proverava se da li se cita zatvarajuca etiketa */
          if ((c = fgetc(in)) == '/') {
125         i = 0;
            while ((c = fgetc(in)) != '>')
127           procitana[i++] = c;
          }
129       /* Cita se otvarajuca etiketa */
          else {
131         i = 0;
            procitana[i++] = c;
133         while ((c = fgetc(in)) != ' ' && c != '>')
              procitana[i++] = c;
135       }
          procitana[i] = '\0';
137
          /* Trazi se procitana etiketa medju postojecim cvorovima
139          liste. Ako ne postoji, dodaje se novi cvor za ucitanu
             etiketu sa brojem pojavljivanja 1. Inace se uvecava broj
141          pojavljivanja etikete. */
          trazeni = pretrazi_listu(glava, procitana);
143       if (trazeni == NULL) {
            if (dodaj_na_pocetak_liste(&glava, 1, procitana) == 1) {
145           fprintf(stderr,
                    "Greska: Neuspesna alokacija memorije za nov cvor\n
    ");
147           oslobodi_listu(&glava);
              exit(EXIT_FAILURE);
149         }
          } else
151         trazeni->broj_pojavljivanja++;
        }
153   }

155   /* Zatvara se datoteka */
      fclose(in);
157
      /* Ispisuje se sadrzaj cvorova liste */
159   ispisi_listu(glava);

161   /* Oslobadja se memorija zauzeta listom */
      oslobodi_listu(&glava);
163
      exit(EXIT_SUCCESS);
165 }
```

### Rešenje 1.4

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

```c
#include <string.h>

#define MAX_INDEKS 11
#define MAX_IME_PREZIME 21

/* Struktura kojom se predstavlja cvor liste koji sadrzi podatke o
   studentu */
typedef struct _Cvor {
  char broj_indeksa[MAX_INDEKS];
  char ime[MAX_IME_PREZIME];
  char prezime[MAX_IME_PREZIME];
  struct _Cvor *sledeci;
} Cvor;

/* Funkcija kreira i inicijalizuje cvor liste i vraca pokazivac na
   novi cvor ili NULL ukoliko je doslo do greske */
Cvor *napravi_cvor(char *broj_indeksa, char *ime, char *prezime)
{
  /* Alokacija memorije za cvor uz proveru uspesnosti alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  strcpy(novi->broj_indeksa, broj_indeksa);
  strcpy(novi->ime, ime);
  strcpy(novi->prezime, prezime);
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

/* Funkcija oslobadja memoriju zauzetu cvorovima liste */
void oslobodi_listu(Cvor ** adresa_glave)
{
  /* Ako je lista prazna, nema potrebe oslobadjati memoriju */
  if (*adresa_glave == NULL)
    return;

  /* Rekurzivnim pozivom se oslobadja rep liste */
  oslobodi_listu(&(*adresa_glave)->sledeci);

  /* Potom se oslobadja i glava liste */
  free(*adresa_glave);

  /* Lista se proglasava praznom */
  *adresa_glave = NULL;
}

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ako je doslo
   do greske pri alokaciji memorije za nov cvor, inace vraca 0. */
```

```
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, char *broj_indeksa,
                           char *ime, char *prezime)
{
  /* Kreira se novi cvor i proverava se uspesnost alokacije */
  Cvor *novi = napravi_cvor(broj_indeksa, ime, prezime);
  if (novi == NULL)
    return 1;

  /* Dodaje se novi cvor na pocetak liste */
  novi->sledeci = *adresa_glave;
  *adresa_glave = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

/* Funkcija ispisuje sadrzaj cvorova liste. */
void ispisi_listu(Cvor * glava)
{
  /* Pocevsi od glave liste */
  for (; glava != NULL; glava = glava->sledeci)
    printf("%s %s %s\n", glava->broj_indeksa, glava->ime,
           glava->prezime);
}

/* Funkcija vraca cvor koji kao vrednost sadrzi trazeni broj
   indeksa. U suprotnom funkcija vraca NULL */
Cvor *pretrazi_listu(Cvor * glava, char *broj_indeksa)
{
  /* Ako je lista prazna, ne postoji trazeni cvor */
  if (glava == NULL)
    return NULL;

  /* Poredi se trazeni broj indeksa sa indeksom u glavi liste */
  if (!strcmp(glava->broj_indeksa, broj_indeksa))
    return glava;

  /* Ukoliko u glavi liste nije trazeni indeks, pretraga se
     nastavlja u repu liste */
  return pretrazi_listu(glava->sledeci, broj_indeksa);
}

int main(int argc, char **argv)
{
  /* Argumenti komandne linije su neophodni jer se iz komandne
     linije dobija ime datoteke sa informacijama o studentima */
  if (argc != 2) {
    fprintf(stderr,
            "Greska: Program se poziva sa: ./a.out ime_datoteke\n");
    exit(EXIT_FAILURE);
  }

```

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

  /* Deklaracije pomocnih promenljiva za citanje vrednosti koje
     treba smestiti u listu */
  char ime[MAX_IME_PREZIME], prezime[MAX_IME_PREZIME];
  char broj_indeksa[MAX_INDEKS];
  Cvor *glava = NULL;
  Cvor *trazeni = NULL;

  /* Ucitavanje vrednosti u listu */
  while (fscanf(in, "%s %s %s", broj_indeksa, ime, prezime) != EOF)
    if (dodaj_na_pocetak_liste(&glava, broj_indeksa, ime, prezime)) {
      fprintf(stderr,
              "Greska: Neuspesna alokacija memorije za nov cvor\n");
      oslobodi_listu(&glava);
      exit(EXIT_FAILURE);
    }

  /* Datoteka vise nije potrebna i zatvara se. */
  fclose(in);

  /* Ucitava se indeks po indeks studenta koji se trazi u listi. */
  while (scanf("%s", broj_indeksa) != EOF) {
    trazeni = pretrazi_listu(glava, broj_indeksa);
    if (trazeni == NULL)
      printf("ne\n");
    else
      printf("da: %s %s\n", trazeni->ime, trazeni->prezime);
  }

  /* Oslobadja se memorija zauzeta za cvorove liste. */
  oslobodi_listu(&glava);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.6

NAPOMENA: *Rešenje koristi biblioteku za rad sa listama iz zadatka 1.1.*

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

```
/* Funkcija objedinjuje dve liste ciji se pokazivaci na glave
   nalaze na adresama adresa_glave_1 i adresa_glave_2 prevezivanjem
   pokazivaca postojecih cvorova listi */
Cvor *objedini(Cvor ** adresa_glave_1, Cvor ** adresa_glave_2)
{
  /* Pokazivaci na pocetne cvorove listi koje se prevezuju */
  Cvor *lista1 = *adresa_glave_1;
  Cvor *lista2 = *adresa_glave_2;

  /* Pokazivac na pocetni cvor rezultujuce liste */
  Cvor *rezultujuca = NULL;
  Cvor *tekuci = NULL;

  /* Ako su obe liste prazne i rezultat je prazna lista */
  if (lista1 == NULL && lista2 == NULL)
    return NULL;

  /* Ako je prva lista prazna, rezultat je druga lista */
  if (lista1 == NULL)
    return lista2;

  /* Ako je druga lista prazna, rezultat je prva lista */
  if (lista2 == NULL)
    return lista1;

  /* Odredjuje se prvi cvor rezultujuce liste - to je ili prvi cvor
     liste lista1 ili prvi cvor liste lista2 u zavisnosti od toga
     koji sadrzi manju vrednost */
  if (lista1->vrednost < lista2->vrednost) {
    rezultujuca = lista1;
    lista1 = lista1->sledeci;
  } else {
    rezultujuca = lista2;
    lista2 = lista2->sledeci;
  }
  /* Kako promenljiva rezultujuca pokazuje na pocetak nove liste,
     ne sme joj se menjati vrednost. Zato se koristi pokazivac
     tekuci koji sadrzi adresu trenutnog cvora rezultujuce liste */
  tekuci = rezultujuca;

  /* U svakoj iteraciji petlje rezultujucoj listi se dodaje novi
     cvor tako da bude uredjena neopadajuce. Pokazivac na listu iz
     koje se uzima cvor se azurira tako da pokazuje na sledeci. */
  while (lista1 != NULL && lista2 != NULL) {
    if (lista1->vrednost < lista2->vrednost) {
      tekuci->sledeci = lista1;
      lista1 = lista1->sledeci;
    } else {
      tekuci->sledeci = lista2;
      lista2 = lista2->sledeci;
    }
    tekuci = tekuci->sledeci;
```

```
57    }
59    /* Ako se iz petlje izaslo jer se stiglo do kraja prve liste, na
         rezultujucu listu treba nadovezati ostatak druge liste */
61    if (lista1 == NULL)
        tekuci->sledeci = lista2;
63    else
        /* U suprotnom treba nadovezati ostatak prve liste */
65      tekuci->sledeci = lista1;
67    /* Preko adresa glava polaznih listi vrednosti pokazivaca u
         pozivajucoj funkciji se postavljaju na NULL jer se svi cvorovi
69       prethodnih listi nalaze negde unutar rezultujuce liste. Do njih
         se moze doci prateci pokazivace iz glave rezultujuce liste, tako
71       da stare pokazivace treba postaviti na NULL. */
    *adresa_glave_1 = NULL;
73  *adresa_glave_2 = NULL;
75    return rezultujuca;
  }
77
  int main(int argc, char **argv)
79{
    /* Argumenti komandne linije su neophodni */
81  if (argc != 3) {
      fprintf(stderr,
83            "Greska: Program se poziva sa: ./a.out dat1.txt dat2.txt\
      n");
      exit(EXIT_FAILURE);
85  }
87  /* Otvaraju se datoteke u kojima se nalaze elementi listi */
    FILE *in1 = NULL;
89  in1 = fopen(argv[1], "r");
    if (in1 == NULL) {
91    fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke %s.\n", argv[1]);
93    exit(EXIT_FAILURE);
    }
95
    FILE *in2 = NULL;
97  in2 = fopen(argv[2], "r");
    if (in2 == NULL) {
99    fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke %s.\n", argv[2]);
101   exit(EXIT_FAILURE);
    }
103
    /* Liste su na pocetku prazne */
105 int broj;
    Cvor *lista1 = NULL;
107 Cvor *lista2 = NULL;
```

```
  /* Ucitavanje listi */
  while (fscanf(in1, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&lista1, broj);

  while (fscanf(in2, "%d", &broj) != EOF)
    dodaj_na_kraj_liste(&lista2, broj);

  /* Datoteke vise nisu potrebne i treba ih zatvoriti. */
  fclose(in1);
  fclose(in2);

  /* Pokazivac rezultat ce pokazivati na glavu liste dobijene
     objedinjavanjem listi */
  Cvor *rezultat = objedini(&lista1, &lista2);

  /* Ispis rezultujuce liste. */
  ispisi_listu(rezultat);

  /* Lista rezultat dobijena je prevezivanjem cvorova polaznih
     listi. Njenim oslobadjanjem oslobadja se sva zauzeta memorija.
   */
  oslobodi_listu(&rezultat);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.8

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

/* Struktura kojom je predstavljen cvor liste sadrzi karakter koji
   predstavlja vidjenu zagradu i pokazivac na sledeci cvor liste */
typedef struct cvor {
  char zagrada;
  struct cvor *sledeci;
} Cvor;

/* Funkcija koja oslobadja memoriju zauzetu stekom */
void oslobodi_stek(Cvor ** stek)
{
  Cvor *tekuci;
  Cvor *pomocni;

  /* Oslobadja se cvor po cvor steka */
  tekuci = *stek;
  while (tekuci != NULL) {
    pomocni = tekuci->sledeci;
    free(tekuci);
    tekuci = pomocni;
```

```
  }

  /* Stek se proglasava praznim */
  *stek = NULL;
}

int main()
{
  /* Stek je na pocetku prazan */
  Cvor *stek = NULL;
  FILE *ulaz = NULL;
  char c;
  Cvor *pomocni = NULL;

  /* Otvaranje datotoke za citanje izraza */
  ulaz = fopen("izraz.txt", "r");
  if (ulaz == NULL) {
    fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke izraz.txt!\n");
    exit(EXIT_FAILURE);
  }

  /* Cita karakter po karakter iz datoteke */
  while ((c = fgetc(ulaz)) != EOF) {
    /* Ako je ucitana otvorena zagrada, stavlja se na stek */
    if (c == '(' || c == '{' || c == '[') {
      /* Alocira se memorija za novi cvor liste i proverava se
         uspesnost alokacije */
      pomocni = (Cvor *) malloc(sizeof(Cvor));
      if (pomocni == NULL) {
        fprintf(stderr, "Greska: Neuspesna alokacija memorije!\n");
        /* Oslobadjanje memorije zauzete stekom */
        oslobodi_stek(&stek);
        /* Prekid izvrsavanja programa */
        exit(EXIT_FAILURE);
      }

      /* Inicijalizacija polja strukture */
      pomocni->zagrada = c;

      /* Promena vrha steka */
      pomocni->sledeci = stek;
      stek = pomocni;
    }
    /* Ako je ucitana zatvorena zagrada, proverava se da li je stek
       prazan i ako nije, da li se na vrhu steka nalazi odgovarajuca
       otvorena zagrada */
    else {
      if (c == ')' || c == '}' || c == ']') {
        if (stek != NULL && ((stek->zagrada == '(' && c == ')')
                             || (stek->zagrada == '{' && c == '}')
                             || (stek->zagrada == '[' && c == ']')))
```

```
    {
75          /* Sa vrha steka se uklanja otvorena zagrada */
            pomocni = stek->sledeci;
77          free(stek);
            stek = pomocni;
79        } else {
            /* Dakle, zagrade u izrazu nisu ispravno uparene */
81          break;
          }
83      }
    }
85  }

87  /* Procitana je cela datoteka i treba je zatvoriti. */
  fclose(ulaz);
89
  /* Ako je stek prazan i procitana je cela datoteka, zagrade su
91     ispravno uparene. */
  if (stek == NULL && c == EOF)
93    printf("Zagrade su ispravno uparene.\n");
  else {
95    /* U suprotnom se zakljucuje da zagrade nisu ispravno uparene. */
    printf("Zagrade nisu ispravno uparene.\n");
97    /* Oslobadja se memorija koja je ostala zauzeta stekom. */
    oslobodi_stek(&stek);
99  }

101 exit(EXIT_SUCCESS);
}
```

### Rešenje 1.9

*stek.h*

```
#ifndef _STEK_H_
2 #define _STEK_H_

4 #include <stdio.h>
  #include <stdlib.h>
6 #include <string.h>
  #include <ctype.h>
8
  #define MAX 100
10
  #define OTVORENA 1
12 #define ZATVORENA 2

14 #define VAN_ETIKETE 0
  #define PROCITANO_MANJE 1
16 #define U_ETIKETI 2
```

```
18 /* Struktura kojim se predstavlja cvor liste sadrzi ime etikete i
      pokazivac na sledeci cvor */
20 typedef struct cvor {
     char etiketa[MAX];
22   struct cvor *sledeci;
   } Cvor;
24
   /* Funkcija kreira novi cvor, upisuje u njega etiketu i vraca
26    njegovu adresu ili NULL ako alokacija nije bila uspesna */
   Cvor *napravi_cvor(char *etiketa);
28
   /* Funkcija oslobadja memoriju zauzetu stekom */
30 void oslobodi_stek(Cvor ** adresa_vrha);

32 /* Funkcija postavlja na vrh steka novu etiketu. U slucaju greske
      pri alokaciji za novi cvor funkcija vraca 1, inace vraca 0 */
34 int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa);

36 /* Funkcija skida sa vrha steka etiketu. Ako je drugi argument
      pokazivac razlicit od NULL, tada u niz karaktera na koji on
38    pokazuje upisuje ime etikete koja je upravo skinuta sa steka dok
      u suprotnom ne radi nista. Funkcija vraca 0 ako je stek prazan
40    (pa samim tim nije bilo moguce skinuti vrednost sa steka) ili 1 u
      suprotnom */
42 int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa);

44 /* Funkcija vraca pokazivac na string koji sadrzi etiketu na vrhu
      steka. Ukoliko je stek prazan, vraca NULL */
46 char *vrh_steka(Cvor * vrh);

48 /* Funkcija prikazuje stek od vrha prema dnu */
   void prikazi_stek(Cvor * vrh);
50
   /* Funkcija iz datoteke kojoj odgovara pokazivac f cita sledecu
52    etiketu, i upisuje je u nisku na koju pokazuje pokazivac
      etiketa. Vraca EOF u slucaju da se dodje do kraja datoteke pre
54    nego sto se procita etiketa. Vraca OTVORENA, ako je procitana
      otvorena etiketa, odnosno ZATVORENA, ako je procitana zatvorena
56    etiketa */
   int uzmi_etiketu(FILE * f, char *etiketa);
58
   #endif
```

*stek.c*

```
1  #include "stek.h"

3  Cvor *napravi_cvor(char *etiketa)
   {
5    /* Alokacija memorije za novi cvor uz proveru uspesnosti */
```

```
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja u novom cvoru */
  if (strlen(etiketa) >= MAX) {
    fprintf(stderr, "Greska: Etiketa je preduga, bice skracena.\n");
    etiketa[MAX - 1] = '\0';
  }
  strcpy(novi->etiketa, etiketa);
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_stek(Cvor ** adresa_vrha)
{
  Cvor *pomocni;

  /* Sve dok stek nije prazan, brise se cvor koji je vrh steka */
  while (*adresa_vrha != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi cvor koji predstavlja vrh steka */
    pomocni = *adresa_vrha;
    /* Sledeci cvor je novi vrh steka */
    *adresa_vrha = (*adresa_vrha)->sledeci;
    free(pomocni);
  }

  /* Nakon izlaska iz petlje stek je prazan i pokazivac na adresi
     adresa_vrha ce pokazivati na NULL. */
}

int potisni_na_stek(Cvor ** adresa_vrha, char *etiketa)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(etiketa);
  if (novi == NULL)
    return 1;

  /* Novi cvor se uvezuje na vrh i postaje nov vrh steka */
  novi->sledeci = *adresa_vrha;
  *adresa_vrha = novi;
  return 0;
}

int skini_sa_steka(Cvor ** adresa_vrha, char *etiketa)
{
  Cvor *pomocni;

  /* Pokusaj skidanja vrednosti sa praznog steka rezultuje greskom
```

```
     i vraca se 0 */
  if (*adresa_vrha == NULL)
    return 0;

  /* Ako adresa na koju se smesta etiketa nije NULL, onda se na tu
     adresu kopira etiketa sa vrha steka */
  if (etiketa != NULL)
    strcpy(etiketa, (*adresa_vrha)->etiketa);

  /* Element sa vrha steka se uklanja */
  pomocni = *adresa_vrha;
  *adresa_vrha = (*adresa_vrha)->sledeci;
  free(pomocni);

  /* Vraca se indikator uspesno izvrsene radnje */
  return 1;
}

char *vrh_steka(Cvor * vrh)
{
  /* Prazan stek nema cvor koji je vrh i vraca se NULL */
  if (vrh == NULL)
    return NULL;

  /* Inace, vraca se pokazivac na nisku etiketa koja je polje cvora
     koji je na vrhu steka. */
  return vrh->etiketa;
}

void prikazi_stek(Cvor * vrh)
{
  /* Ispisuje se spisak etiketa na steku od vrha ka dnu. */
  for (; vrh != NULL; vrh = vrh->sledeci)
    printf("<%s>\n", vrh->etiketa);
}

int uzmi_etiketu(FILE * f, char *etiketa)
{
  int c;
  int i = 0;
  /* Stanje predstavlja informaciju dokle se stalo sa citanjem
     etikete. Inicijalizuje se vrednoscu VAN_ETIKETE jer jos uvek
     nije zapoceto citanje. */
  /* Tip predstavlja informaciju o tipu etikete. Uzima vrednosti
     OTVORENA ili ZATVORENA. */
  int stanje = VAN_ETIKETE;
  int tip;

  /* HTML je neosetljiv na razliku izmedju malih i velikih slova,
     dok to u C-u ne vazi. Zato ce sve etikete biti prevedene u
     zapis samo malim slovima. */
  while ((c = fgetc(f)) != EOF) {
```

```
    switch (stanje) {
111 case VAN_ETIKETE:
      if (c == '<')
113     stanje = PROCITANO_MANJE;
      break;
115 case PROCITANO_MANJE:
      if (c == '/') {
117     /* Cita se zatvorena etiketa */
        tip = ZATVORENA;
119   } else {
        if (isalpha(c)) {
121       /* Cita se otvorena etiketa */
          tip = OTVORENA;
123       etiketa[i++] = tolower(c);
        }
125   }
      /* Od sada se cita etiketa i zato se menja stanje */
127   stanje = U_ETIKETI;
      break;
129 case U_ETIKETI:
      if (isalpha(c) && i < MAX - 1) {
131     /* Ako je procitani karakter slovo i nije prekoracena
           dozvoljena duzina etikete, procitani karakter se
133        smanjuje i smesta u etiketu */
        etiketa[i++] = tolower(c);
135   } else {
        /* Inace, staje se sa citanjem etikete. Korektno se
137        zavrsava niska koja sadrzi procitanu etiketu i vraca se
           njen tip */
139     etiketa[i] = '\0';
        return tip;
141   }
      break;
143 }
  }
145 /* Doslo se do kraja datoteke pre nego sto je procitana naredna
       etiketa i vraca se EOF. */
147 return EOF;
}
```

*main.c*

```
#include "stek.h"
2
int main(int argc, char **argv)
4 {
  /* Na pocetku, stek je prazan i etikete su uparene jer nijedna
6    jos nije procitana. */
  Cvor *vrh = NULL;
8 char etiketa[MAX];
  int tip;
```

```
10  int uparene = 1;
    FILE *f = NULL;
12
    /* Ime datoteke se preuzima iz komandne linije. */
14  if (argc < 2) {
      fprintf(stderr, "Greska:");
16    fprintf(stderr,
              "Program se poziva sa:\n %s ime_html_datoteke\n",
18            argv[0]);
      exit(EXIT_FAILURE);
20  }

22  /* Datoteka se otvara za citanje */
    if ((f = fopen(argv[1], "r")) == NULL) {
24    fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
              argv[1]);
26    exit(EXIT_FAILURE);
    }
28
    /* Cita se etiketa po etiketa, sve dok ih ima u datoteci. */
30  while ((tip = uzmi_etiketu(f, etiketa)) != EOF) {
      /* Ako je otvorena etiketa, stavlja se na stek. Izuzetak su
32       etikete <br>, <hr> i <meta> koje nemaju sadrzaj, pa ih nije
         potrebno zatvoriti. U HTML-u postoje jos neke etikete koje
34       koje nemaju sadrzaj (npr link). Zbog jednostavnosti
         pretpostavlja se da njih nema u HTML dokumentu. */
36    if (tip == OTVORENA) {
        if (strcmp(etiketa, "br") != 0
38          && strcmp(etiketa, "hr") != 0
            && strcmp(etiketa, "meta") != 0)
40        if (potisni_na_stek(&vrh, etiketa) == 1) {
            fprintf(stderr,
42                  "Greska: Neuspesna alokacija memorije za nov cvor\n
    ");
            oslobodi_stek(&vrh);
44          exit(EXIT_FAILURE);
          }
46    }
      /* Ako je zatvorena etiketa, tada je uslov dobre uparenosti da
48       je u pitanju zatvaranje etikete koja je poslednja otvorena,
         a jos uvek nije zatvorena. Ona se mora nalaziti na vrhu
50       steka. Ako je taj uslov ispunjen, skida se sa steka, jer je
         upravo zatvorena. U suprotnom, pronadjena je nepravilnost i
52       etikete nisu pravilno uparene. */
      else if (tip == ZATVORENA) {
54      if (vrh_steka(vrh) != NULL
            && strcmp(vrh_steka(vrh), etiketa) == 0)
56        skini_sa_steka(&vrh, NULL);
        else {
58        printf("Etikete nisu pravilno uparene\n");
          printf("(nadjena je etiketa </%s>", etiketa);
60        if (vrh_steka(vrh) != NULL)
```

```
          printf(", a poslednja otvorena je <%s>)\n",
                 vrh_steka(vrh));
        else
          printf(" koja nije otvorena)\n");
        uparene = 0;
        break;
      }
    }
  }
  /* Zavrseno je citanje i datoteka se zatvara */
  fclose(f);

  /* Ako do sada nije pronadjeno pogresno uparivanje, stek bi
     trebalo da bude prazan. Ukoliko nije, tada postoje etikete
     koje su ostale otvorene */
  if (uparene) {
    if (vrh_steka(vrh) == NULL)
      printf("Etikete su pravilno uparene!\n");
    else {
      printf("Etikete nisu pravilno uparene\n");
      printf("(etiketa <%s> nije zatvorena)\n", vrh_steka(vrh));
      /* Oslobadja se memorija zauzeta stekom */
      oslobodi_stek(&vrh);
    }
  }

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.10

*red.h*

```
#ifndef _RED_H_
#define _RED_H_

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

#define MAX 1000
#define JMBG_DUZINA 14

/* Struktura predstavlja zahtev korisnika. Obuhvata JMBG korisnika
   i opis njegovog zahteva. */
typedef struct {
  char jmbg[JMBG_DUZINA];
  char opis[MAX];
} Zahtev;

/* Struktura kojom je predstavljen cvor liste, obuhvata zahtev
```

```
   korisnika i pokazivac na sledeci cvor liste. */
typedef struct cvor {
  Zahtev nalog;
  struct cvor *sledeci;
} Cvor;

/* Funkcija kreira novi cvor, inicijalizuje polje nalog na zahtev
   sa poslate adrese i vraca adresu novog cvora ili NULL ako je
   doslo do greske pri alokaciji. Prosledjuje joj se pokazivac na
   zahtev koji treba smestiti u novi cvor zbog smestanja manjeg
   podatka na sistemski stek. Pokazivac na strukturu Zahtev je
   manje velicine u bajtovima(B) u odnosu na strukturu Zahtev. */
Cvor *napravi_cvor(Zahtev * zahtev);

/* Funkcija prazni red oslobadjajuci memoriju koji je red zauzimao */
void oslobodi_red(Cvor ** pocetak, Cvor ** kraj);

/* Funkcija dodaje na kraj reda novi zahtev. Vraca 1 ako je doslo
   do greske pri alokaciji memorije za novi cvor, inace vraca 0. */
int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                Zahtev * zahtev);

/* Funkcija skida sa pocetka reda zahtev. Ako je argument zahtev
   pokazivac razlicit od NULL, u strukturu na koju on pokazuje
   upisuje se zahtev upravo skinut sa reda, inace se ne upisuje
   nista. Funkcija vraca 0, ako je red bio prazan, inace vraca 1. */
int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                  Zahtev * zahtev);

/* Funkcija vraca pokazivac na strukturu koja sadrzi zahtev
   korisnika na pocetku reda. Ako je red prazan, vraca NULL. */
Zahtev *pocetak_reda(Cvor * pocetak);

/* Funkcija prikazuje sadrzaj reda. */
void prikazi_red(Cvor * pocetak);

#endif
```

*red.c*

```
#include "red.h"

Cvor *napravi_cvor(Zahtev * zahtev)
{
  /* Alokacija memorije za novi cvor uz proveru uspesnosti */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->nalog = *zahtev;
```

```
12    novi->sledeci = NULL;

14    /* Vraca se adresa novog cvora */
      return novi;
16  }

18  void oslobodi_red(Cvor ** pocetak, Cvor ** kraj)
    {
20    Cvor *pomocni = NULL;

22    /* Sve dok red nije prazan brise se cvor koji je pocetka reda */
      while (*pocetak != NULL) {
24      /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
           osloboditi cvor sa pocetka reda */
26      pomocni = *pocetak;
        *pocetak = (*pocetak)->sledeci;
28      free(pomocni);
      }
30    /* Nakon izlaska iz petlje red je prazan. Pokazivac na kraj reda
         treba postaviti na NULL. */
32    *kraj = NULL;
    }
34
    int dodaj_u_red(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
36                  Zahtev * zahtev)
    {
38    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
      Cvor *novi = napravi_cvor(zahtev);
40    if (novi == NULL)
        return 1;
42
      /* U red se uvek dodaje na kraj. Zbog postojanja pokazivaca na
44       kraj, to je efikasno koliko i dodavanje na pocetak liste */
      if (*adresa_kraja != NULL) {
46      (*adresa_kraja)->sledeci = novi;
        *adresa_kraja = novi;
48    } else {
        /* Ako je red bio ranije prazan */
50      *adresa_pocetka = novi;
        *adresa_kraja = novi;
52    }

54    /* Vraca se indikator uspesnog dodavanja */
      return 0;
56  }

58  int skini_sa_reda(Cvor ** adresa_pocetka, Cvor ** adresa_kraja,
                      Zahtev * zahtev)
60  {
      Cvor *pomocni = NULL;
62
      /* Ako je red prazan */
```

```
  if (*adresa_pocetka == NULL)
    return 0;

  /* Ako je prosledjen pokazivac zahtev, na tu adresu se prepisuje
     zahtev koji je na pocetku reda.  */
  if (zahtev != NULL)
    *zahtev = (*adresa_pocetka)->nalog;

  /* Oslobadja se memorija zauzeta cvorom sa pocetka reda i azurira
     se pokazivac na adresi adresa_pocetka da pokazuje na sledeci
     cvor u redu. */
  pomocni = *adresa_pocetka;
  *adresa_pocetka = (*adresa_pocetka)->sledeci;
  free(pomocni);

  /* Ukoliko red nakon oslobadjanja pocetnog cvora ostane prazan,
     potrebno je azurirati i vrednost pokazivaca na adresi
     adresa_kraja na NULL */
  if (*adresa_pocetka == NULL)
    *adresa_kraja = NULL;

  return 1;
}

Zahtev *pocetak_reda(Cvor * pocetak)
{
  /* U praznom redu nema zahteva */
  if (pocetak == NULL)
    return NULL;

  /* Inace, vraca se pokazivac na zahtev sa pocetka reda */
  return &(pocetak->nalog);
}

void prikazi_red(Cvor * pocetak)
{
  /* Prikazuje se sadrzaj reda od pocetka prema kraju */
  for (; pocetak != NULL; pocetak = pocetak->sledeci)
    printf("%s %s\n", (pocetak->nalog).jmbg, (pocetak->nalog).opis);

  printf("\n");
}
```

*main.c*

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

#define VREME_ZA_PAUZU 5
```

```c

int main(int argc, char **argv)
{
  /* Red je prazan. */
  Cvor *pocetak = NULL, *kraj = NULL;
  Zahtev nov_zahtev;
  Zahtev *sledeci = NULL;
  char odgovor[3];
  int broj_usluzenih = 0;

  /* Sluzbenik evidentira korisnicke zahteve unosenjem njihovog
     JMBG broja i opisa potrebne usluge. */
  printf("Sluzbenik evidentira korisnicke zahteve:\n");
  while (1) {
    /* Ucitava se JMBG broj */
    printf("\nNovi zahtev [CTRL+D za kraj]\n\tJMBG: ");
    if (scanf("%s", nov_zahtev.jmbg) == EOF)
      break;

    /* Neophodan je poziv funkcije getchar da bi se i nov red nakon
       JMBG broja procitao i da bi fgets nakon toga procitala
       ispravan red sa opisom zahteva */
    getchar();

    /* Ucitava se opis problema */
    printf("\tOpis problema: ");
    fgets(nov_zahtev.opis, MAX - 1, stdin);
    /* Ako je poslednji karakter nov red, eliminise se */
    if (nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] == '\n')
      nov_zahtev.opis[strlen(nov_zahtev.opis) - 1] = '\0';

    /* Dodaje se zahtev u red i proverava se uspesnost dodavanja */
    if (dodaj_u_red(&pocetak, &kraj, &nov_zahtev) == 1) {
      fprintf(stderr,
              "Greska: Neuspesna alokacija memorije za nov cvor\n");
      oslobodi_red(&pocetak, &kraj);
      exit(EXIT_FAILURE);
    }
  }

  /* Otvaranje datoteke za dopisivanje izvestaja */
  FILE *izlaz = fopen("izvestaj.txt", "a");
  if (izlaz == NULL) {
    fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke izvestaj.txt\n");
    exit(EXIT_FAILURE);
  }

  /* Dokle god ima korisnika u redu, treba ih usluziti */
  while (1) {
    sledeci = pocetak_reda(pocetak);
    /* Ako nema nikog vise u redu, prekida se petlja */
```

```
    if (sledeci == NULL)
      break;

    printf("\nSledeci je korisnik sa JMBG: %s\n", sledeci->jmbg);
    printf("i zahtevom: %s\n", sledeci->opis);

    skini_sa_reda(&pocetak, &kraj, &nov_zahtev);

    broj_usluzenih++;

    printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
    scanf("%s", odgovor);

    if (strcmp(odgovor, "Da") == 0)
      dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
    else
      fprintf(izlaz, "JMBG: %s\tZahtev: %s\n", nov_zahtev.jmbg,
              nov_zahtev.opis);

    if (broj_usluzenih == VREME_ZA_PAUZU) {
      printf("\nDa li je kraj smene? [Da/Ne] ");
      scanf("%s", odgovor);

      if (strcmp(odgovor, "Da") == 0)
        break;
      else
        broj_usluzenih = 0;
    }
  }

  /*****************************************************************
    Usluzivanje korisnika moze da se izvrsi i na sledeci nacin:    */
  /*****************************************************************
    while (skini_sa_reda(&pocetak, &kraj, &nov_zahtev)) {
      printf("\nSledeci je korisnik sa JMBG brojem: %s\n",
            nov_zahtev.jmbg);
      printf("sa zahtevom: %s\n", nov_zahtev.opis);
      broj_usluzenih++;

      printf("\tDa li ga vracate na kraj reda? [Da/Ne] ");
      scanf("%s", odgovor);
      if (strcmp(odgovor, "Da") == 0)
        dodaj_u_red(&pocetak, &kraj, &nov_zahtev);
      else
        fprintf(izlaz, "JMBG: %s\tZahtev: %s\n",
                nov_zahtev.jmbg, nov_zahtev.opis);

      if (broj_usluzenih == VREME_ZA_PAUZU) {
        printf("\nDa li je kraj smene? [Da/Ne] ");
        scanf("%s", odgovor);
        if (strcmp(odgovor, "Da") == 0)
          break;
```

```
        else
          broj_usluzenih = 0;
      }
    }
  ****************************************************************/

  /* Datoteka vise nije potrebna i treba je zatvoriti. */
  fclose(izlaz);

  /* Ukoliko je sluzbenik prekinuo sa radom, mozda je bilo jos
     neusluzenih korisnika, u tom slucaju treba osloboditi memoriju
     koju zauzima red sa neobradjenim zahtevima korisnika. */
  oslobodi_red(&pocetak, &kraj);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.11

*dvostruko_povezana_lista.h*

```
#ifndef _DVOSTRUKO_POVEZANA_LISTA_H_
#define _DVOSTRUKO_POVEZANA_LISTA_H_

/* Struktura kojom je predstavljen cvor liste sadrzi celobrojnu
   vrednost i pokazivace na sledeci i prethodni cvor liste. */
typedef struct cvor {
  int vrednost;
  struct cvor *sledeci;
  struct cvor *prethodni;
} Cvor;

/* Funkcija kreira cvor, vrednost novog cvora inicijalizuje na
   broj, dok pokazivac na sledeci cvor postavlja na NULL. Vraca
   pokazivac na novokreirani cvor ili NULL ako alokacija nije bila
   uspesna. */
Cvor *napravi_cvor(int broj);

/* Funkcija oslobadja dinamicku memoriju zauzetu za cvorove liste
   ciji se pocetni cvor nalazi na adresi adresa_glave, a poslednji
   na adresi adresa_kraja. */
void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja);

/* Funkcija dodaje novi cvor na pocetak liste. Vraca 1 ukoliko je
   bilo greske pri alokaciji memorije, inace vraca 0. */
int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
                           adresa_kraja, int broj);

/* Funkcija dodaje broj na kraj liste. Vraca 1 ukoliko je bilo
   greske pri alokaciji memorije, inace vraca 0. */
```

```c
int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                        int broj);

/* Pomocna funkcija pronalazi cvor u listi iza koga treba umetnuti
   novi cvor sa vrednoscu broj. */
Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj);

/* Funkcija dodaje broj iza zadatog cvora. Vraca 1 ukoliko je
   dodavanje uspesno, odnosno 0 ukoliko je doslo do greske. */
int dodaj_iza(Cvor * tekuci, int broj);

/* Funkcija dodaje broj u sortiranu listu tako da lista ostane
   sortirana. Vraca 1 ukoliko je bilo greske pri alokaciji
   memorije, inace vraca 0. */
int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                    broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom
   broju. Vraca pokazivac na cvor liste u kome je sadrzan trazeni
   broj ili NULL u slucaju da takav cvor ne postoji u listi. */
Cvor *pretrazi_listu(Cvor * glava, int broj);

/* Funkcija trazi u listi cvor cija je vrednost jednaka datom
   broju. U pretrazi oslanja se na cinjenicu da je lista koja se
   pretrazuje neopadajuce sortirana. Vraca pokazivac na cvor liste
   koji sadrzi trazeni broj ili NULL u slucaju da takav cvor ne
   postoji. */
Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj);

/* Funkcija brise cvor na koji pokazuje pokazivac tekuci u listi
   ciji pokazivac glava se nalazi na adresi adresa_glave. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                   Cvor * tekuci);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj.
   Azurira pokazivac na glavu liste, koji moze biti promenjen u
   slucaju da se obrise stara glava. */
void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                 broj);

/* Funkcija brise iz liste sve cvorove koji sadrze dati broj,
   oslanjajuci se na cinjenicu da je prosledjena lista neopadajuce
   sortirana. Azurira pokazivac na glavu liste, koji moze biti
   promenjen ukoliko se obrise stara glava liste. */
void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                 adresa_kraja, int broj);

/* Funkcija prikazuje vrednosti cvorova liste pocev od glave ka
   kraju liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu(Cvor * glava);

/* Funkcija prikazuje vrednosti cvorova liste pocevsi od kraja ka
```

```
   glavi liste, razdvojene zapetama i uokvirene zagradama. */
void ispisi_listu_unazad(Cvor * kraj);

#endif
```

*dvostruko_povezana_lista.c*

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

Cvor *napravi_cvor(int broj)
{
  /* Alokacija memorije za novi cvor uz proveru uspesnosti */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
  if (novi == NULL)
    return NULL;

  /* Inicijalizacija polja strukture */
  novi->vrednost = broj;
  novi->sledeci = NULL;

  /* Vraca se adresa novog cvora */
  return novi;
}

void oslobodi_listu(Cvor ** adresa_glave, Cvor ** adresa_kraja)
{
  Cvor *pomocni = NULL;

  /* Ako lista nije prazna, onda treba osloboditi memoriju */
  while (*adresa_glave != NULL) {
    /* Potrebno je prvo zapamtiti adresu sledeceg cvora i onda
       osloboditi memoriju cvora koji predstavlja glavu liste */
    pomocni = (*adresa_glave)->sledeci;
    free(*adresa_glave);
    /* Sledeci cvor je nova glava liste */
    *adresa_glave = pomocni;
  }
  /* Nakon izlaska iz petlje lista je prazna. Pokazivac na kraj
     liste treba postaviti na NULL */
  *adresa_kraja = NULL;
}

int dodaj_na_pocetak_liste(Cvor ** adresa_glave, Cvor **
                           adresa_kraja, int broj)
{
  /* Kreira se novi cvor i proverava se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;
```

```
45
   /* Sledbenik novog cvora je glava stare liste */
47   novi->sledeci = *adresa_glave;

49   /* Ako stara lista nije bila prazna, onda prethodni cvor glave
        treba da bude novi cvor. Inace, novi cvor je u isto vreme i
51      pocetni i krajnji. */
   if (*adresa_glave != NULL)
53     (*adresa_glave)->prethodni = novi;
   else
55     *adresa_kraja = novi;

57   /* Novi cvor je nova glava liste */
   *adresa_glave = novi;
59
   /* Vraca se indikator uspesnog dodavanja */
61   return 0;
}
63
int dodaj_na_kraj_liste(Cvor ** adresa_glave, Cvor ** adresa_kraja,
65                         int broj)
{
67   /* Kreira se novi cvor i proverava se uspesnost kreiranja */
   Cvor *novi = napravi_cvor(broj);
69   if (novi == NULL)
     return 1;
71
   /* U slucaju prazne liste, glava nove liste je upravo novi cvor i
73      ujedno i cela lista. Azuriraju se vrednosti na koje pokazuju
        adresa_glave i adresa_kraja */
75   if (*adresa_glave == NULL) {
     *adresa_glave = novi;
77     *adresa_kraja = novi;
   } else {
79     /* Ako lista nije prazna, novi cvor se dodaje na kraj liste kao
          sledbenik poslednjeg cvora i azurira se samo pokazivac na
81        kraj liste */
     (*adresa_kraja)->sledeci = novi;
83     novi->prethodni = (*adresa_kraja);
     *adresa_kraja = novi;
85   }

87   /* Vraca se indikator uspesnog dodavanja */
   return 0;
89 }

91 Cvor *pronadji_mesto_umetanja(Cvor * glava, int broj)
{
93   /* U praznoj listi nema takvog mesta i vraca se NULL */
   if (glava == NULL)
95     return NULL;
```

```c
  /* Pokazivac glava se pomera na sledeci cvor sve dok ne bude
     pokazivala na cvor ciji sledeci cvor ili ne postoji ili ima
     vrednost vecu ili jednaku od vrednosti novog cvora. */
  /* Zbog izracunavanja izraza u C-u prvi deo konjunkcije mora biti
     provera da li se doslo do poslednjeg cvora liste pre nego sto
     se proveri vrednost u sledecem cvoru jer u slucaju poslednjeg,
     sledeci ne postoji pa ni njegova vrednost. */
  while (glava->sledeci != NULL && glava->sledeci->vrednost < broj)
    glava = glava->sledeci;

  /* Iz petlje se moglo izaci pomeranjem pokazivaca glava do
     poslednjeg cvora ili, ranije, nailaskom na cvor ciji sledeci
     ima vrednost vecu od broj */
  return glava;
}

int dodaj_iza(Cvor * tekuci, int broj)
{
  /* Kreira se novi cvor i provera se uspesnost kreiranja */
  Cvor *novi = napravi_cvor(broj);
  if (novi == NULL)
    return 1;

  novi->sledeci = tekuci->sledeci;
  novi->prethodni = tekuci;

  /* Ako tekuci ima sledeceg, onda se sledecem dodeljuje
     prethodnik, a potom i tekuci dobija novog sledeceg
     postavljanjem pokazivaca na ispravne adrese */
  if (tekuci->sledeci != NULL)
    tekuci->sledeci->prethodni = novi;
  tekuci->sledeci = novi;

  /* Vraca se indikator uspesnog dodavanja */
  return 0;
}

int dodaj_sortirano(Cvor ** adresa_glave, Cvor ** adresa_kraja, int
                    broj)
{
  /* Ako je lista prazna, novi cvor je i prvi i poslednji cvor */
  if (*adresa_glave == NULL) {
    /* Kreira se novi cvor i proverava se uspesnost kreiranja */
    Cvor *novi = napravi_cvor(broj);
    if (novi == NULL)
      return 1;

    /* Azuriraju se vrednosti pocetka i kraja liste */
    *adresa_glave = novi;
    *adresa_kraja = novi;

    /* Vraca se indikator uspesnog dodavanja */
```

```
149     return 0;
  }
151
  /* Ukoliko je vrednost glave liste veca ili jednaka od nove
153     vrednosti onda novi cvor treba staviti na pocetak liste */
  if ((*adresa_glave)->vrednost >= broj) {
155   return dodaj_na_pocetak_liste(adresa_glave, adresa_kraja, broj);
  }
157
  /* Pronazi se cvor iza koga treba uvezati novi cvor */
159 Cvor *pomocni = pronadji_mesto_umetanja(*adresa_glave, broj);
  /* Dodaje se novi cvor uz proveru uspesnosti dodavanja */
161 if (dodaj_iza(pomocni, broj) == 1)
    return 1;
163 /* Ako pomocni cvor pokazuje na poslednji element liste, onda je
     novi cvor poslednji u listi. */
165 if (pomocni == *adresa_kraja)
    *adresa_kraja = pomocni->sledeci;
167
  return 0;
169 }

171 Cvor *pretrazi_listu(Cvor * glava, int broj)
{
173   /* Obilaze se cvorovi liste */
  for (; glava != NULL; glava = glava->sledeci)
175     /* Ako je vrednost tekuceg cvora jednaka zadatom broju,
        pretraga se obustavlja */
177     if (glava->vrednost == broj)
        return glava;
179
  /* Nema trazenog broja u listi i vraca se NULL */
181 return NULL;
}
183
185 Cvor *pretrazi_sortiranu_listu(Cvor * glava, int broj)
{
187   /* Obilaze se cvorovi liste */
  /* U uslovu ostanka u petlji, bitan je redosled u konjunkciji */
  for (; glava != NULL && glava->vrednost <= broj;
189        glava = glava->sledeci)
     /* Ako je vrednost tekuceg cvora jednaka zadatom broju,
191        pretraga se obustavlja */
     if (glava->vrednost == broj)
193       return glava;

195   /* Nema trazenog broja u listi i bice vraceno NULL */
  return NULL;
197 }

199 /* Funkcija brise cvor zadat argumentom tekuci. Brisanje odredjenog
    cvora dvostruko povezane liste moze se lako realizovati jer cvor
```

```
   sadrzi pokazivace na svog sledbenika i prethodnika u listi. */
void obrisi_tekuci(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                   Cvor * tekuci)
{
  /* Ako je tekuci NULL pokazivac, nema potrebe za brisanjem */
  if (tekuci == NULL)
    return;

  /* Ako postoji prethodnik tekuceg cvora, onda se postavlja da
     njegov sledbenik bude sledbenik tekuceg cvora */
  if (tekuci->prethodni != NULL)
    tekuci->prethodni->sledeci = tekuci->sledeci;

  /* Ako postoji sledbenik tekuceg cvora, onda njegov prethodnik
     treba da bude prethodnik tekuceg cvora */
  if (tekuci->sledeci != NULL)
    tekuci->sledeci->prethodni = tekuci->prethodni;

  /* Ako je glava cvor koji se brise, nova glava liste ce biti
     sledbenik stare glave. */
  if (tekuci == *adresa_glave)
    *adresa_glave = tekuci->sledeci;

  /* Ako je cvor koji se brise poslednji u listi, azurira se i
     pokazivac na kraj liste. */
  if (tekuci == *adresa_kraja)
    *adresa_kraja = tekuci->prethodni;

  /* Oslobadja se dinamicki alociran prostor za cvor tekuci. */
  free(tekuci);
}

void obrisi_cvor(Cvor ** adresa_glave, Cvor ** adresa_kraja,
                 int broj)
{
  Cvor *tekuci = *adresa_glave;

  /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
     takvi cvorovi se brisu iz liste. */
  while ((tekuci = pretrazi_listu(*adresa_glave, broj)) != NULL)
    obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}

void obrisi_cvor_sortirane_liste(Cvor ** adresa_glave, Cvor **
                                 adresa_kraja, int broj)
{
  Cvor *tekuci = *adresa_glave;

  /* Sve dok ima cvorova cija je vrednost jednaka zadatom broju,
     takvi cvorovi se brisu iz liste. */
  while ((tekuci =
          pretrazi_sortiranu_listu(*adresa_glave, broj)) != NULL)
```

```
    obrisi_tekuci(adresa_glave, adresa_kraja, tekuci);
}

void ispisi_listu(Cvor * glava)
{
  putchar('[');
  /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
     pocetka prema kraju liste. */
  for (; glava != NULL; glava = glava->sledeci) {
    printf("%d", glava->vrednost);
    if (glava->sledeci != NULL)
      printf(", ");
  }

  printf("]\n");
}

void ispisi_listu_unazad(Cvor * kraj)
{
  putchar('[');
  /* Unutar zagrada ispisuju se vrednosti u cvorovima liste od
     kraja prema pocetku liste. */
  for (; kraj != NULL; kraj = kraj->prethodni) {
    printf("%d", kraj->vrednost);
    if (kraj->prethodni != NULL)
      printf(", ");
  }
  printf("]\n");
}
```

*main_a.c*

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

int main()
{
  /* Lista je prazna na pocetku */
  /* Cuvaju se pokazivaci na glavu liste i na poslednji cvor liste,
     da bi operacije poput dodavanja na kraj liste i ispisivanja
     liste unazad bile efikasne poput dodavanja na pocetak liste i
     ispisivanja liste od pocetnog do poslednjeg cvora. */
  Cvor *glava = NULL;
  Cvor *kraj = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se funkcija za dodavanje novog broja na pocetak liste */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
  while (scanf("%d", &broj) > 0) {
```

```
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
       memorije za novi cvor. Memoriju alociranu za cvorove liste
       treba osloboditi pre napustanja programa */
    if (dodaj_na_pocetak_liste(&glava, &kraj, broj) == 1) {
      fprintf(stderr,
              "Greska: Neuspesna alokacija memorije za cvor.\n");
      oslobodi_listu(&glava, &kraj);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcija za pretragu liste */
  printf("\nUnesite broj koji se trazi u listi: ");
  scanf("%d", &broj);

  /* Pokazivac trazeni dobija vrednost rezultata pretrage */
  trazeni = pretrazi_listu(glava, broj);
  if (trazeni == NULL)
    printf("Broj %d se ne nalazi u listi!\n", broj);
  else
    printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);

  /* Ispisuje se lista unazad */
  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(kraj);

  /* Oslobadja se memorija zauzeta za cvorove liste */
  oslobodi_listu(&glava, &kraj);

  exit(EXIT_SUCCESS);
}
```

*main_b.c*

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

int main()
{
  /* Lista je prazna na pocetku. */
  Cvor *glava = NULL;
  Cvor *kraj = NULL;
  int broj;

  /* Testira se funkcija za dodavanje novog broja na kraj liste */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
```

```
        memorije za novi cvor. Memoriju alociranu za cvorove liste
        treba osloboditi pre napustanja programa */
    if (dodaj_na_kraj_liste(&glava, &kraj, broj) == 1) {
      fprintf(stderr,
              "Greska: Neuspesna alokacija memorije za cvor.\n");
      oslobodi_listu(&glava, &kraj);
      exit(EXIT_FAILURE);
    }
    printf("\tLista: ");
    ispisi_listu(glava);
  }

  /* Testira se funkcija za brisanje elemenata iz liste */
  printf("\nUnesite broj koji se brise iz liste: ");
  scanf("%d", &broj);

  /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
  obrisi_cvor(&glava, &kraj, broj);

  printf("Lista nakon brisanja: ");
  ispisi_listu(glava);

  /* Ispisuje se lista unazad */
  printf("\nLista ispisana u nazad: ");
  ispisi_listu_unazad(kraj);

  /* Oslobadja se memorija zauzeta za cvorove liste */
  oslobodi_listu(&glava, &kraj);

  exit(EXIT_SUCCESS);
}
```

*main_c.c*

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

int main()
{
  /* Lista je prazna na pocetku */
  Cvor *glava = NULL;
  Cvor *kraj = NULL;
  Cvor *trazeni = NULL;
  int broj;

  /* Testira se funkcija za dodavanje vrednosti u listu tako da ona
     bude uredjena neopadajuce */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
  while (scanf("%d", &broj) > 0) {
    /* Ako je funkcija vratila 1, onda je bilo greske pri alokaciji
```

```
18          memorije za novi cvor. Memoriju alociranu za cvorove liste
            treba osloboditi pre napustanja programa */
20        if (dodaj_sortirano(&glava, &kraj, broj) == 1) {
            fprintf(stderr,
22                  "Greska: Neuspesna alokacija memorije za cvor.\n");
            oslobodi_listu(&glava, &kraj);
24          exit(EXIT_FAILURE);
          }
26        printf("\tLista: ");
          ispisi_listu(glava);
28      }

30      /* Testira se funkcija za pretragu liste */
        printf("\nUnesite broj koji se trazi u listi: ");
32      scanf("%d", &broj);

34      /* Pokazivac trazeni dobija vrednost rezultata pretrage */
        trazeni = pretrazi_listu(glava, broj);
36      if (trazeni == NULL)
          printf("Broj %d se ne nalazi u listi!\n", broj);
38      else
          printf("Trazeni broj %d je u listi!\n", trazeni->vrednost);
40
        /* Testira se funkcija za brisanje elemenata iz liste */
42      printf("\nUnesite broj koji se brise iz liste: ");
        scanf("%d", &broj);
44
        /* Brisu se cvorovi liste cija vrednost je jednaka unetom broju */
46      obrisi_cvor_sortirane_liste(&glava, &kraj, broj);

48      printf("Lista nakon brisanja: ");
        ispisi_listu(glava);
50
        /* Ispisuje se lista unazad */
52      printf("\nLista ispisana u nazad: ");
        ispisi_listu_unazad(kraj);
54
        /* Oslobadja se memorija zauzeta za cvorove liste */
56      oslobodi_listu(&glava, &kraj);

58      exit(EXIT_SUCCESS);
    }
```

### Rešenje 1.14

*stabla.h*

```
#ifndef _STABLA_H_
#define _STABLA_H_ 1

/* a) Struktura kojom se predstavlja cvor binarnog pretrazivackog
   stabla */
typedef struct cvor {
  int broj;
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* b) Funkcija koja alocira memoriju za novi cvor stabla,
   inicijalizuje polja strukture i vraca pokazivac na novi cvor */
Cvor *napravi_cvor(int broj);

/* c) Funkcija koja dodaje zadati broj u stablo. Povratna vrednost
   funkcije je 0 ako je dodavanje uspesno, odnosno 1 ukoliko je
   doslo do greske */
int dodaj_u_stablo(Cvor ** adresa_korena, int broj);

/* d) Funkcija koja proverava da li se zadati broj nalazi stablu */
Cvor *pretrazi_stablo(Cvor * koren, int broj);

/* e) Funkcija koja pronalazi cvor koji sadrzi najmanju vrednost u
   stablu */
Cvor *pronadji_najmanji(Cvor * koren);

/* f) Funkcija koja pronalazi cvor koji sadrzi najvecu vrednost u
   stablu */
Cvor *pronadji_najveci(Cvor * koren);

/* g) Funkcija koja brise cvor stabla koji sadrzi zadati broj */
void obrisi_element(Cvor ** adresa_korena, int broj);

/* h) Funkcija koja ispisuje stablo u infiksnoj notaciji (Levo
   postablo - Koren - Desno podstablo ) */
void ispisi_stablo_infiksno(Cvor * koren);

/* i) Funkcija koja ispisuje stablo u prefiksnoj notaciji ( Koren -
   Levo podstablo - Desno podstablo ) */
void ispisi_stablo_prefiksno(Cvor * koren);

/* j) Funkcija koja ispisuje stablo u postfiksnoj notaciji ( Levo
   podstablo - Desno postablo - Koren) */
void ispisi_stablo_postfiksno(Cvor * koren);

/* k) Funkcija koja oslobadja memoriju zauzetu stablom */
```

```
48 void oslobodi_stablo(Cvor ** adresa_korena);

50 #endif
```

*stabla.c*

```
#include <stdio.h>
2 #include <stdlib.h>
#include "stabla.h"
4
Cvor *napravi_cvor(int broj)
6 {
  /* Alocira se memorija za novi cvor i proverava se uspesnost
8     alokacije */
  Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
10  if (novi == NULL)
    return NULL;
12
  /* Inicijalizuju se polja novog cvora */
14  novi->broj = broj;
  novi->levo = NULL;
16  novi->desno = NULL;

18  /* Vraca se adresa novog cvora */
  return novi;
20 }

22 int dodaj_u_stablo(Cvor ** adresa_korena, int broj)
{
24  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
26
    /* Kreira se novi cvor */
28    Cvor *novi_cvor = napravi_cvor(broj);

30    /* Proverava se uspesnost kreiranja */
    if (novi_cvor == NULL) {
32      /* Ako je doslo do greske, vraca se odgovarajuca vrednost */
      return 1;
34    }

36    /* Inace, novi cvor se proglasava korenom stabla */
    *adresa_korena = novi_cvor;
38
    /* I vraca se indikator uspesnosti kreiranja */
40    return 0;
  }
42
  /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za
44     zadati broj */
```

```
  /* Ako je zadata vrednost manja od vrednosti korena */
  if (broj < (*adresa_korena)->broj)
    /* Broj se dodaje u levo podstablo */
    return dodaj_u_stablo(&(*adresa_korena)->levo, broj);
  else
    /* Inace, broj je veci (ili jednak) od vrednosti u korenu pa se
       dodaje u desno podstablo */
    return dodaj_u_stablo(&(*adresa_korena)->desno, broj);
}

Cvor *pretrazi_stablo(Cvor * koren, int broj)
{
  /* Ako je stablo prazno, vrednost se sigurno ne nalazi u njemu */
  if (koren == NULL)
    return NULL;

  /* Ako je trazena vrednost sadrzana u korenu */
  if (koren->broj == broj) {
    /* Prekida se pretraga */
    return koren;
  }

  /* Inace, ako je broj manji od vrednosti sadrzane u korenu */
  if (broj < koren->broj)
    /* Pretraga se nastavlja u levom podstablu */
    return pretrazi_stablo(koren->levo, broj);
  else
    /* U suprotnom, pretraga se nastavlja u desnom podstablu */
    return pretrazi_stablo(koren->desno, broj);
}

Cvor *pronadji_najmanji(Cvor * koren)
{
  /* Ako je stablo prazno, prekida se pretraga */
  if (koren == NULL)
    return NULL;

  /* Vrednosti koje su manje od vrednosti u korenu stabla nalaze se
     levo od njega */

  /* Ako je koren cvor koji nema levo podstablo, onda on sadrzi
     najmanju vrednost */
  if (koren->levo == NULL)
    return koren;

  /* Inace, pretragu treba nastaviti u levom podstablu */
  return pronadji_najmanji(koren->levo);
}

Cvor *pronadji_najveci(Cvor * koren)
{
  /* Ako je stablo prazno, prekida se pretraga */
```

```
98    if (koren == NULL)
        return NULL;
100
      /* Vrednosti koje su vece od vrednosti u korenu stabla nalaze se
102       desno od njega */

104   /* Ako je koren cvor koji nema desno podstablo, onda on sadrzi
         najvecu vrednost */
106   if (koren->desno == NULL)
        return koren;
108
      /* Inace, pretragu treba nastaviti u desnom podstablu */
110   return pronadji_najveci(koren->desno);
  }
112
  void obrisi_element(Cvor ** adresa_korena, int broj)
114 {
      Cvor *pomocni_cvor = NULL;
116
      /* Ako je stablo prazno, brisanje nije primenljivo */
118   if (*adresa_korena == NULL)
        return;
120
      /* Ako je vrednost koju treba obrisati manja od vrednosti u
122       korenu stabla, ona se eventualno nalazi u levom podstablu, pa
         treba rekurzivno primeniti postupak na levo podstablo. Koren
124       ovako modifikovanog stabla je nepromenjen. */
      if (broj < (*adresa_korena)->broj) {
126     obrisi_element(&(*adresa_korena)->levo, broj);
        return;
128   }

130   /* Ako je vrednost koju treba obrisati veca od vrednosti u korenu
         stabla, ona se eventualno nalazi u desnom podstablu pa treba
132       rekurzivno primeniti postupak na desno podstablo. Koren ovako
         modifikovanog stabla je nepromenjen. */
134   if ((*adresa_korena)->broj < broj) {
        obrisi_element(&(*adresa_korena)->desno, broj);
136     return;
      }
138
      /* Slede podslucajevi vezani za slucaj kada je vrednost u korenu
140       jednaka broju koji se brise tj. slucaj kada treba obrisati
         koren */
142
      /* 1. Ako koren nema sinova, tada se on prosto brise, i rezultat
144       je prazno stablo (vraca se NULL) */
      if ((*adresa_korena)->levo == NULL
146        && (*adresa_korena)->desno == NULL) {
        free(*adresa_korena);
148     *adresa_korena = NULL;
        return;
```

```
150   }
152   /* 2. Ako koren ima samo levog sina, tada se brisanje vrsi tako
         sto se brise koren, a novi koren postaje levi sin */
154   if ((*adresa_korena)->levo != NULL
          && (*adresa_korena)->desno == NULL) {
156     pomocni_cvor = (*adresa_korena)->levo;
        free(*adresa_korena);
158     *adresa_korena = pomocni_cvor;
        return;
160   }
162   /* 3. Ako koren ima samo desnog sina, tada se brisanje vrsi tako
         sto se brise koren, a novi koren postaje desni sin */
164   if ((*adresa_korena)->desno != NULL
          && (*adresa_korena)->levo == NULL) {
166     pomocni_cvor = (*adresa_korena)->desno;
        free(*adresa_korena);
168     *adresa_korena = pomocni_cvor;
        return;
170   }
172   /* 4. Ako koren ima oba sina, najpre se potrazi sledbenik korena
         (u smislu poretka) u stablu. To je upravo po vrednosti
174      najmanji cvor u desnom podstablu koji se moze pronaci npr.
         funkcijom pronadji_najmanji(). Potom se u koren smesti
176      vrednost pronadjenog cvora, a u taj cvor se smesti vrednost
         korena (tj. broj koji se brise). Zatim se rekurzivno pozove
178      funkcija za brisanje nad desnim podstablom. S obzirom da u
         njemu treba obrisati najmanji element, a on zasigurno ima
180      najvise jednog potomka, jasno je da ce njegovo brisanje biti
         obavljeno na jedan od jednostavnijih nacina koji su gore
182      opisani. */
  pomocni_cvor = pronadji_najmanji((*adresa_korena)->desno);
184 (*adresa_korena)->broj = pomocni_cvor->broj;
  pomocni_cvor->broj = broj;
186 obrisi_element(&(*adresa_korena)->desno, broj);
}
188
void ispisi_stablo_infiksno(Cvor * koren)
190 {
  /* Ako stablo nije prazno */
192 if (koren != NULL) {
194   /* Prvo se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_infiksno(koren->levo);
196
    /* Zatim se ispisuje vrednost u korenu */
198 printf("%d ", koren->broj);
200   /* Na kraju se ispisuju cvorovi desno od korena */
    ispisi_stablo_infiksno(koren->desno);
```

```
202   }
}

void ispisi_stablo_prefiksno(Cvor * koren)
{
  /* Ako stablo nije prazno */
  if (koren != NULL) {

    /* Prvo se ispisuje vrednost u korenu */
    printf("%d ", koren->broj);

    /* Zatim se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_prefiksno(koren->levo);

    /* Na kraju se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_prefiksno(koren->desno);
  }
}

void ispisi_stablo_postfiksno(Cvor * koren)
{

  /* Ako stablo nije prazno */
  if (koren != NULL) {

    /* Prvo se ispisuju svi cvorovi levo od korena */
    ispisi_stablo_postfiksno(koren->levo);

    /* Zatim se ispisuju svi cvorovi desno od korena */
    ispisi_stablo_postfiksno(koren->desno);

    /* Na kraju se ispisuje vrednost u korenu */
    printf("%d ", koren->broj);
  }
}

void oslobodi_stablo(Cvor ** adresa_korena)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
  if (*adresa_korena == NULL)
    return;

  /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);

  /* Oslobadja se memorija zauzetu desnim podstablom */
  oslobodi_stablo(&(*adresa_korena)->desno);

  /* Oslobadja se memorija zauzeta korenom */
  free(*adresa_korena);

  /* Proglasava se stablo praznim */
```

```
254   *adresa_korena = NULL;
}
```

*main.c*

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

int main()
{
  Cvor *koren;
  int n;
  Cvor *trazeni_cvor;

  /* Proglasava se stablo praznim */
  koren = NULL;

  /* Citaju se vrednosti i dodaju u stablo uz proveru uspesnosti
     dodavanja */
  printf("Unesite brojeve (CTRL+D za kraj unosa): ");
  while (scanf("%d", &n) != EOF) {
    if (dodaj_u_stablo(&koren, n) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Generisu se trazeni ispisi: */
  printf("\nInfiksni ispis: ");
  ispisi_stablo_infiksno(koren);
  printf("\nPrefiksni ispis: ");
  ispisi_stablo_prefiksno(koren);
  printf("\nPostfiksni ispis: ");
  ispisi_stablo_postfiksno(koren);

  /* Demonstrira se rad funkcije za pretragu */
  printf("\nTrazi se broj: ");
  scanf("%d", &n);
  trazeni_cvor = pretrazi_stablo(koren, n);
  if (trazeni_cvor == NULL)
    printf("Broj se ne nalazi u stablu!\n");
  else
    printf("Broj se nalazi u stablu!\n");

  /* Demonstrira se rad funkcije za brisanje */
  printf("Brise se broj: ");
  scanf("%d", &n);
  obrisi_element(&koren, n);
  printf("Rezultujuce stablo: ");
```

```
47    ispisi_stablo_infiksno(koren);
      printf("\n");
49
      /* Oslobadja se memorija zauzeta stablom */
51    oslobodi_stablo(&koren);

53    exit(EXIT_SUCCESS);
  }
```

### Rešenje 1.15

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include <string.h>
4 #include <ctype.h>

6 #define MAX 50

8 /* Struktura kojom se opisuje cvor stabla: sadrzi rec, njen broj
     pojavljivanja i redom pokazivace na levo i desno podstablo */
10 typedef struct cvor {
    char *rec;
12  int brojac;
    struct cvor *levo;
14  struct cvor *desno;
  } Cvor;
16
  /* Funkcija koja kreira novi cvora stabla */
18 Cvor *napravi_cvor(char *rec)
  {
20  /* Alocira se memorija za novi cvor i proverava se uspesnost
       alokacije */
22  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
    if (novi_cvor == NULL)
24    return NULL;

26  /* Alocira se memorija za zadatu rec: potrebno je rezervisati
       memoriju za svaki karakter reci ukljucujuci i terminirajucu
28     nulu */
    novi_cvor->rec =
30      (char *) malloc((strlen(rec) + 1) * sizeof(char));
    if (novi_cvor->rec == NULL) {
32    free(novi_cvor);
      return NULL;
34  }

36  /* Inicijalizuju se polja u novom cvoru */
    strcpy(novi_cvor->rec, rec);
38  novi_cvor->brojac = 1;
    novi_cvor->levo = NULL;
40  novi_cvor->desno = NULL;
```

```c
  /* Vraca se adresa novog cvora */
  return novi_cvor;
}

/* Funkcija koja dodaje novu rec u stablo. Ako je dodavanje uspesno
   povratna vrednost je 0, u suprotnom povratna vrednost je 1. */
int dodaj_u_stablo(Cvor ** adresa_korena, char *rec)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
    /* Kreira se cvor koji sadrzi zadatu rec */
    Cvor *novi_cvor = napravi_cvor(rec);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
      /* Ako je doslo do greske, vraca se odgovarajuca vrednost */
      return 1;
    }

    /* Inace, novi cvor se proglasava korenom stabla */
    *adresa_korena = novi_cvor;

    /* I vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novu
     rec */

  /* Ako je rec leksikografski manja od reci u korenu ubacuje se u
     levo podstablo */
  if (strcmp(rec, (*adresa_korena)->rec) < 0)
    return dodaj_u_stablo(&(*adresa_korena)->levo, rec);
  else
    /* Ako je rec leksikografski veca od reci u korenu ubacuje se u
       desno podstablo */
  if (strcmp(rec, (*adresa_korena)->rec) > 0)
    return dodaj_u_stablo(&(*adresa_korena)->desno, rec);
  else {
    /* Ako je rec jednaka reci u korenu, uvecava se njen broj
       pojavljivanja */
    (*adresa_korena)->brojac++;

    /* I vraca se indikator uspesnog dodavanja */
    return 0;
  }
}

/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
```

```
  if (*adresa_korena == NULL)
    return;

  /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*adresa_korena)->levo);

  /* Oslobadja se memorija zauzeta desnim podstablom */
  oslobodi_stablo(&(*adresa_korena)->desno);

  /* Oslobadja se memorija zauzeta korenom */
  free((*adresa_korena)->rec);
  free(*adresa_korena);

  /* Stablo se proglasava praznim */
  *adresa_korena = NULL;
}

/* Funkcija koja pronalazi cvor koji sadrzi najfrekventniju rec
   (rec sa najvecim brojem pojavljivanja) */
Cvor *nadji_najfrekventniju_rec(Cvor * koren)
{
  Cvor *max, *max_levo, *max_desno;

  /* Ako je stablo prazno, prekida se sa pretragom */
  if (koren == NULL)
    return NULL;

  /* Pronalazi se najfrekventnija rec u levom podstablu */
  max_levo = nadji_najfrekventniju_rec(koren->levo);

  /* Pronalazi se najfrekventnija rec u desnom podstablu */
  max_desno = nadji_najfrekventniju_rec(koren->desno);

  /* Trazi se maksimum vrednosti pojavljivanja reci iz levog
     podstabla, korena i desnog podstabla */
  max = koren;
  if (max_levo != NULL && max_levo->brojac > max->brojac)
    max = max_levo;
  if (max_desno != NULL && max_desno->brojac > max->brojac)
    max = max_desno;

  /* Vraca se adresa cvora sa najvecim brojem pojavljivanja */
  return max;
}

/* Funkcija koja ispisuje reci iz stabla u leksikografskom poretku
   pracene brojem pojavljivanja */
void prikazi_stablo(Cvor * koren)
{
  /* Ako je stablo prazno, zavrsava se sa ispisom */
  if (koren == NULL)
    return;
```

```c
  /* Zbog leksikografskog poretka, prvo se ispisuju sve reci iz
     levog podstabla */
  prikazi_stablo(koren->levo);

  /* Zatim rec iz korena */
  printf("%s: %d\n", koren->rec, koren->brojac);

  /* I nastavlja se sa ispisom reci iz desnog podstabla */
  prikazi_stablo(koren->desno);
}

/* Funkcija ucitava sledecu rec iz zadate datoteke f i upisuje je u
   niz rec. Maksimalna duzina reci je odredjena argumentom max.
   Funkcija vraca EOF ako u datoteci nema vise reci ili 0 u
   suprotnom. Rec je niz malih ili velikih slova. */
int procitaj_rec(FILE * f, char rec[], int max)
{
  /* Karakter koji se cita */
  int c;

  /* Indeks pozicije na koju se smesta procitani karakter */
  int i = 0;

  /* Sve dok ima mesta za jos jedan karakter u nizu i dokle se god
     nije stiglo do kraja datoteke */
  while (i < max - 1 && (c = fgetc(f)) != EOF) {
    /* Proverava se da li je procitani karakter slovo */
    if (isalpha(c))
      /* Ako jeste, smesta se u niz - pritom se vrsi konverzija u
         mala slova jer program treba da bude neosetljiv na razliku
         izmedju malih i velikih slova */
      rec[i++] = tolower(c);
    else
      /* Ako nije, proverava se da li je procitano barem jedno
         slovo nove reci */
      /* Ako jeste, prekida se sa citanjem */
    if (i > 0)
      break;

    /* U suprotnom se ide na sledecu iteraciju */
  }

  /* Dodaje se na rec terminirajuca nula */
  rec[i] = '\0';

  /* Vraca se 0 ako je uspesno procitana rec, tj. EOF u suprotnom */
  return i > 0 ? 0 : EOF;
}

int main(int argc, char **argv)
{
```

```
  Cvor *koren = NULL, *max;
  FILE *f;
  char rec[MAX];

  /* Proverava se da li je navedeno ime datoteke prilikom
     pokretanja programa */
  if (argc < 2) {
    fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
    exit(EXIT_FAILURE);
  }

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

  /* Ucitavaju se reci iz datoteke i smestaju u binarno stablo
     pretrage uz proveru uspesnosti dodavanja */
  while (procitaj_rec(f, rec, MAX) != EOF) {
    if (dodaj_u_stablo(&koren, rec) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje reci %s.\n", rec);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Posto je citanje reci zavrseno, zatvara se datoteka */
  fclose(f);

  /* Prikazuju se sve reci iz teksta i brojevi njihovih
     pojavljivanja */
  prikazi_stablo(koren);

  /* Pronalazi se najfrekventnija rec */
  max = nadji_najfrekventniju_rec(koren);

  /* Ako takve reci nema */
  if (max == NULL)
    /* Ispisuje se odgovarajuce obavestenje */
    printf("U tekstu nema reci!\n");
  else
    /* Inace, ispisuje se broj pojavljivanja reci */
    printf("Najcesca rec: %s (pojavljuje se %d puta)\n",
           max->rec, max->brojac);

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

**Rešenje 1.16**

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

#define MAX_IME_DATOTEKE 50
#define MAX_CIFARA 13
#define MAX_IME_I_PREZIME 100

/* Struktura kojom se opisuje cvor stabla: sadrzi ime i prezime,
   broj telefona i redom pokazivace na levo i desno podstablo */
typedef struct cvor {
  char ime_i_prezime[MAX_IME_I_PREZIME];
  char telefon[MAX_CIFARA];
  struct cvor *levo;
  struct cvor *desno;
} Cvor;

/* Funkcija koja kreira novi cvora stabla */
Cvor *napravi_cvor(char *ime_i_prezime, char *telefon)
{
  /* Alocira se memorija za novi cvor i proverava se uspesnost
     alokacije */
  Cvor *novi_cvor = (Cvor *) malloc(sizeof(Cvor));
  if (novi_cvor == NULL)
    return NULL;

  /* Inicijalizuju se polja novog cvora */
  strcpy(novi_cvor->ime_i_prezime, ime_i_prezime);
  strcpy(novi_cvor->telefon, telefon);
  novi_cvor->levo = NULL;
  novi_cvor->desno = NULL;

  /* Vraca se adresa novog cvora */
  return novi_cvor;
}

/* Funkcija koja dodaje novu osobu i njen broj telefona u stablo.
   Ukoliko je dodavanje uspesno povratna vrednost funkcije je 0,
   dok je u suprotnom povratna vrednost 1 */
int
dodaj_u_stablo(Cvor ** adresa_korena, char *ime_i_prezime,
               char *telefon)
{
  /* Ako je stablo prazno */
  if (*adresa_korena == NULL) {
    /* Kreira se novi cvor */
    Cvor *novi_cvor = napravi_cvor(ime_i_prezime, telefon);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
```

```
51        /* Ako je doslo do greske, vraca se odgovarajuca vrednost */
          return 1;
53      }

55      /* Inace, novi cvor se proglasava korenom stabla */
        *adresa_korena = novi_cvor;
57
        /* I vraca se indikator uspesnog dodavanja */
59      return 0;
    }
61
    /* Ako stablo nije prazno, trazi se odgovarajuca pozicija za novi
63     unos. Kako pretragu treba vrsiti po imenu i prezimenu, stablo
       treba da bude pretrazivacko po ovom polju.
65
       Ako je zadato ime i prezime leksikografski manje od imena i
67     prezimena koje se nalazi u korenu, podaci se dodaju u levo
       podstablo */
69  if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime)
        < 0)
71    return dodaj_u_stablo(&(*adresa_korena)->levo, ime_i_prezime,
                            telefon);
73  else
      /* Ako je zadato ime i prezime leksikografski vece od imena i
75       prezimena sadrzanog u korenu, podaci se dodaju u desno
         podstablo */
77  if (strcmp(ime_i_prezime, (*adresa_korena)->ime_i_prezime) > 0)
      return dodaj_u_stablo(&(*adresa_korena)->desno, ime_i_prezime,
79                          telefon);

81  /* Pretostavka zadatka je da nema istih imena i prezimena u
       datoteci, pa se sledeca naredba nikada neci ni izvrsiti */
83  return 0;
}
85
/* Funkcija koja oslobadja memoriju zauzetu stablom */
87 void oslobodi_stablo(Cvor ** adresa_korena)
{
89  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
    if (*adresa_korena == NULL)
91    return;

93  /* Oslobadja se memorija zauzeta levim podstablom */
    oslobodi_stablo(&(*adresa_korena)->levo);
95
    /* Oslobadja se memorija zauzeta desnim podstablom */
97  oslobodi_stablo(&(*adresa_korena)->desno);

99  /* Oslobadja se memorija zauzeta korenom */
    free(*adresa_korena);
101
    /* Stablo se proglasava praznim */
```

```c
  *adresa_korena = NULL;
}

/* Funkcija koja ispisuje imenik u leksikografskom poretku */
/* Napomena: ova funkcija nije trazena u zadatku ali se moze
   koristiti za proveru da li je stablo uspesno kreirano. */
void prikazi_stablo(Cvor * koren)
{
  /* Ako je stablo prazno, zavrsava se sa ispisom */
  if (koren == NULL)
    return;

  /* Zbog leksikografskog poretka, prvo se ispisuju podaci iz levog
     podstabla */
  prikazi_stablo(koren->levo);

  /* Zatim se ispisuju podaci iz korena */
  printf("%s: %s\n", koren->ime_i_prezime, koren->telefon);

  /* I nastavlja se sa ispisom podataka iz desnog podstabla */
  prikazi_stablo(koren->desno);
}

/* Funkcija ucitava sledeci kontakt iz zadate datoteke i upisuje
   ime i prezime i broj telefona u odgovarajuce nizove. Vraca EOF
   ako nema vise kontakata ili 0 u suprotnom. Maksimalna duzina
   imena i prezimena odredjena je konstantom MAX_IME_PREZIME, a
   maksimalna duzina broja telefona konstantom MAX_CIFARA.  */
int procitaj_kontakt(FILE * f, char *ime_i_prezime, char *telefon)
{
  /* Karakter koji se cita */
  int c;

  /* Indeks pozicije na koju se smesta procitani karakter */
  int i = 0;

  /* Linije datoteke koje se obradjuju su formata Ime Prezime
     BrojTelefona */

  /* Preskacu se eventualne praznine sa pocetka linije datoteke */
  while ((c = fgetc(f)) != EOF && isspace(c));

  /* Prvo procitano slovo se upisuje u ime i prezime */
  if (!feof(f))
    ime_i_prezime[i++] = c;

  /* Naznaka kraja citanja imena i prezimena ce biti pojava prve
     cifre tako da se citanje vrsi sve dok se ne naidje na cifru.
     Pritom treba voditi racuna da li ima dovoljno mesta za
     smestanje procitanog karaktera i da se slucajno ne dodje do
     kraja datoteke */
  while (i < MAX_IME_I_PREZIME - 1 && (c = fgetc(f)) != EOF) {
```

```c
    if (!isdigit(c))
      ime_i_prezime[i++] = c;
    else if (i > 0)
      break;
  }

  /* Upisuje se terminirajuca nula na mesto poslednjeg procitanog
     blanko karaktera */
  ime_i_prezime[--i] = '\0';

  /* I pocinje se sa citanjem broja telefona */
  i = 0;

  /* Upisuje se cifra koja je vec procitana */
  telefon[i++] = c;

  /* I citaju se preostale cifre. Naznaka kraja ce biti pojava
     karaktera cije prisustvo nije dozvoljeno u broju telefona */
  while (i < MAX_CIFARA - 1 && (c = fgetc(f)) != EOF)
    if (c == '/' || c == '-' || isdigit(c))
      telefon[i++] = c;
    else
      break;

  /* Upisuje se terminirajuca nula */
  telefon[i] = '\0';

  /* Vraca se 0 ako je uspesno procitan kontakt ili EOF u suprotnom
   */
  return !feof(f) ? 0 : EOF;
}

/* Funkcija koja trazi u imeniku osobu sa zadatim imenom i
   prezimenom */
Cvor *pretrazi_imenik(Cvor * koren, char *ime_i_prezime)
{
  /* Ako je imenik prazan, zavrsava se sa pretragom */
  if (koren == NULL)
    return NULL;

  /* Ako je trazeno ime i prezime sadrzano u korenu, takodje se
     zavrsava sa pretragom */
  if (strcmp(koren->ime_i_prezime, ime_i_prezime) == 0)
    return koren;

  /* Ako je zadato ime i prezime leksikografski manje od vrednosti
     u korenu pretraga se nastavlja levo */
  if (strcmp(ime_i_prezime, koren->ime_i_prezime) < 0)
    return pretrazi_imenik(koren->levo, ime_i_prezime);
  else
    /* U suprotnom, pretraga se nastavlja desno */
    return pretrazi_imenik(koren->desno, ime_i_prezime);
```

```
207 }
209 int main(int argc, char **argv)
    {
211   char ime_datoteke[MAX_IME_DATOTEKE];
      Cvor *koren = NULL;
213   Cvor *trazeni;
      FILE *f;
215   char ime_i_prezime[MAX_IME_I_PREZIME];
      char telefon[MAX_CIFARA];
217   char c;
      int i;
219
      /* Ucitava se ime datoteke i vrsi se njena priprema za citanje */
221   printf("Unesite ime datoteke: ");
      scanf("%s", ime_datoteke);
223   getchar();
      if ((f = fopen(ime_datoteke, "r")) == NULL) {
225     fprintf(stderr, "Greska: Neuspesno otvaranje datoteke %s.\n",
              ime_datoteke);
227   exit(EXIT_FAILURE);
      }
229
      /* Citaju se podaci iz datoteke i smestanju u binarno stablo
231      pretrage uz proveru uspesnosti dodavanja */
      while (procitaj_kontakt(f, ime_i_prezime, telefon) != EOF)
233     if (dodaj_u_stablo(&koren, ime_i_prezime, telefon) == 1) {
        fprintf(stderr,
235               "Greska: Neuspelo dodavanje podataka za osobu %s.\n",
                ime_i_prezime);
237     oslobodi_stablo(&koren);
        exit(EXIT_FAILURE);
239     }

241   /* Datoteka se zatvara */
      fclose(f);
243
      /* Omogucava se pretraga imenika */
245   while (1) {
        /* Ucitava se ime i prezime */
247     printf("Unesite ime i prezime: ");
        i = 0;
249     while ((c = getchar()) != '\n')
          ime_i_prezime[i++] = c;
251     ime_i_prezime[i] = '\0';

253     /* Ako je korisnik uneo naznaku za kraj pretrage, obustavlja se
           pretraga */
255     if (strcmp(ime_i_prezime, "KRAJ") == 0)
          break;
257
        /* Inace se ispisuje rezultat pretrage */
```

```
259     trazeni = pretrazi_imenik(koren, ime_i_prezime);
        if (trazeni == NULL)
261       printf("Broj nije u imeniku!\n");
        else
263       printf("Broj je: %s \n", trazeni->telefon);
    }
265
    /* Oslobadja se memorija zauzeta imenikom */
267 oslobodi_stablo(&koren);

269 exit(EXIT_SUCCESS);
}
```

### Rešenje 1.17

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

5  #define MAX 51

7  /* Struktura koja definise cvorove stabla: sadrzi ime i prezime
      studenta, ukupan uspeh, uspeh iz matematike, uspeh iz maternjeg
9     jezika i redom pokazivace na levo i desno podstablo */
   typedef struct cvor_stabla {
11   char ime[MAX];
     char prezime[MAX];
13   double uspeh;
     double matematika;
15   double jezik;
     struct cvor_stabla *levo;
17   struct cvor_stabla *desno;
   } Cvor;
19
   /* Funkcija kojom se kreira cvor stabla */
21 Cvor *napravi_cvor(char ime[], char prezime[], double uspeh,
                      double matematika, double jezik)
23 {
     /* Alocira se memorija za novi cvor i proverava se uspesnost
25       alokacije */
     Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
27   if (novi == NULL)
       return NULL;
29
     /* Inicijalizuju se polja strukture */
31   strcpy(novi->ime, ime);
     strcpy(novi->prezime, prezime);
33   novi->uspeh = uspeh;
     novi->matematika = matematika;
35   novi->jezik = jezik;
     novi->levo = NULL;
```

```
  novi->desno = NULL;

  /* Vraca se adresa kreiranog cvora */
  return novi;
}

/* Funkcija koja dodaje cvor sa zadatim vrednostima u stablo.
   Ukoliko je dodavanje uspesno, povratna vrednost funkcije je 0,
   dok je u suprotnom povratna vrednost 1 */
int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                   double uspeh, double matematika, double jezik)
{
  /* Ako je stablo prazno */
  if (*koren == NULL) {
    /* Kreira se novi cvor */
    Cvor *novi_cvor =
        napravi_cvor(ime, prezime, uspeh, matematika, jezik);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
      /* I ukoliko je doslo do greske, vraca se odgovarajuca
         vrednost */
      return 1;
    }

    /* Inace, novi cvor se proglasava korenom stabla */
    *koren = novi_cvor;

    /* I vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Ako stablo nije prazno, dodaje se cvor u stablo tako da bude
     sortirano po ukupnom broju poena */
  if (uspeh + matematika + jezik >
      (*koren)->uspeh + (*koren)->matematika + (*koren)->jezik)
    return dodaj_u_stablo(&(*koren)->levo, ime, prezime, uspeh,
                          matematika, jezik);
  else
    return dodaj_u_stablo(&(*koren)->desno, ime, prezime, uspeh,
                          matematika, jezik);
}

/* Funkcija kojom se oslobadja memorija zauzeta stablom */
void oslobodi_stablo(Cvor ** koren)
{
  /* Ako je stablo prazno, nepotrebno je oslobadjati memoriju */
  if (*koren == NULL)
    return;

  /* Oslobadja se memorija zauzeta levim podstablom */
  oslobodi_stablo(&(*koren)->levo);
```

```c
  /* Oslobadja se memorija zauzeta desnim podstablom */
  oslobodi_stablo(&(*koren)->desno);

  /* Oslobadja se memorija zauzeta korenom */
  free(*koren);

  /* Stablo se proglasava praznim */
  *koren = NULL;
}


/* Funkcija ispisuje sadrzaj stabla. Ukoliko je vrednost argumenta
   polozili jednaka 0, ispisuju se informacije o ucenicima koji
   nisu polozili prijemni, a ako je vrednost argumenta razlicita od
   nule, ispisuju se informacije o ucenicima koji su polozili
   prijemni. */
void stampaj(Cvor * koren, int polozili)
{
  /* Stablo je prazno - prekida se sa ispisom */
  if (koren == NULL)
    return;

  /* Stampaju se informacije iz levog podstabla */
  stampaj(koren->levo, polozili);

  /* Stampaju se informacije iz korenog cvora */
  if (polozili && koren->matematika + koren->jezik >= 10)
    printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
           koren->prezime, koren->uspeh, koren->matematika,
           koren->jezik,
           koren->uspeh + koren->matematika + koren->jezik);
  else if (!polozili && koren->matematika + koren->jezik < 10)
    printf("%s %s %.1lf %.1lf %.1lf %.1lf\n", koren->ime,
           koren->prezime, koren->uspeh, koren->matematika,
           koren->jezik,
           koren->uspeh + koren->matematika + koren->jezik);

  /* Stampaju se informacije iz desnog podstabla */
  stampaj(koren->desno, polozili);
}

/* Funkcija koja odredjuje koliko studenata nije polozilo prijemni
   ispit */
int nisu_polozili(Cvor * koren)
{
  /* Ako je stablo prazno, broj onih koji nisu polozili je 0 */
  if (koren == NULL)
    return 0;

  /* Pretraga se vrsi i u levom i u desnom podstablu. Ako uslov za
     polaganje nije ispunjen za koreni cvor, broj studenata se
     uvecava za 1 */
```

```c
  if (koren->matematika + koren->jezik < 10)
    return 1 + nisu_polozili(koren->levo) +
        nisu_polozili(koren->desno);

  /* Inace, nastavlja se prebrojavanje u podstablima */
  return nisu_polozili(koren->levo) + nisu_polozili(koren->desno);
}

int main(int argc, char **argv)
{
  FILE *in;
  Cvor *koren;
  char ime[MAX], prezime[MAX];
  double uspeh, matematika, jezik;

  /* Otvara se datoteke sa rezultatima sa prijemnog za citanje */
  in = fopen("prijemni.txt", "r");
  if (in == NULL) {
    fprintf(stderr,
            "Greska: Neuspesno otvaranje datoteke prijemni.txt.\n");
    exit(EXIT_FAILURE);
  }

  /* Citaju se podaci i dodaju u stablo uz proveru uspesnosti
     dodavanja */
  koren = NULL;
  while (fscanf(in, "%s %s %lf %lf %lf", ime, prezime, &uspeh,
                &matematika, &jezik) != EOF) {
    if (dodaj_u_stablo
        (&koren, ime, prezime, uspeh, matematika, jezik)
        == 1) {
      fprintf(stderr,
              "Greska: Neuspelo dodavanje podataka za %s %s.\n",
              ime, prezime);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

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

  /* Stampaju se prvo podaci o ucenicima koji su polozili prijemni */
  stampaj(koren, 1);

  /* Linija se iscrtava samo ako postoje ucenici koji nisu polozili
     prijemni */
  if (nisu_polozili(koren) != 0)
    printf("-------------------------------------\n");

  /* Stampaju se podaci o ucenicima koji nisu polozili prijemni */
  stampaj(koren, 0);
```

```
193
   /* Oslobadja se memorija zauzeta stablom */
195   oslobodi_stablo(&koren);

197   exit(EXIT_SUCCESS);
}
```

### Rešenje 1.18

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

5  #define MAX_NISKA 51

7  /* Struktura koja opisuje jedan cvor stabla: sadrzi ime i prezime
      osobe, dan i mesec rodjenja i redom pokazivace na levo i desno
9     podstablo */
   typedef struct cvor_stabla {
11   char ime[MAX_NISKA];
     char prezime[MAX_NISKA];
13   int dan;
     int mesec;
15   struct cvor_stabla *levo;
     struct cvor_stabla *desno;
17 } Cvor;

19 /* Funkcija koja kreira novi cvor */
   Cvor *napravi_cvor(char ime[], char prezime[], int dan, int mesec)
21 {
     /* Alocira se memorija */
23   Cvor *novi = (Cvor *) malloc(sizeof(Cvor));
     if (novi == NULL)
25     return NULL;

27   /* Inicijalizuju se polja strukture */
     strcpy(novi->ime, ime);
29   strcpy(novi->prezime, prezime);
     novi->dan = dan;
31   novi->mesec = mesec;
     novi->levo = NULL;
33   novi->desno = NULL;

35   /* Vraca se adresa novog cvora */
     return novi;
37 }

39 /* Funkcija koja dodaje novi cvor u stablo. Stablo treba da bude
      uredjeno po datumu - prvo po mesecu, a zatim po danu. Ukoliko je
41    dodavanje uspesno povratna vrednost funkcije je 0, u suprotnom
      povratna vrednost je 1 */
```

```c
int dodaj_u_stablo(Cvor ** koren, char ime[], char prezime[],
                   int dan, int mesec)
{
  /* Ako je stablo prazno */
  if (*koren == NULL) {

    /* Kreira se novi cvor */
    Cvor *novi_cvor = napravi_cvor(ime, prezime, dan, mesec);
    /* Proverava se uspesnost kreiranja novog cvora */
    if (novi_cvor == NULL) {
      /* I ukoliko je doslo do greske, vraca se odgovarajuca
         vrednost */
      return 1;
    }
    /* Inace, novi cvor se proglasava korenom stabla */
    *koren = novi_cvor;

    /* I vraca se indikator uspesnog dodavanja */
    return 0;
  }

  /* Stablo se uredjuje po mesecu, a zatim po danu u okviru istog
     meseca */
  if (mesec < (*koren)->mesec)
    return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan,
                          mesec);
  else if (mesec == (*koren)->mesec && dan < (*koren)->dan)
    return dodaj_u_stablo(&(*koren)->levo, ime, prezime, dan,
                          mesec);
  else
    return dodaj_u_stablo(&(*koren)->desno, ime, prezime, dan,
                          mesec);
}

/* Funkcija vrsi pretragu stabla i vraca cvor sa trazenim datumom */
Cvor *pretrazi(Cvor * koren, int dan, int mesec)
{
  /* Ako je stablo prazno, obustavlja se pretraga */
  if (koren == NULL)
    return NULL;

  /* Ako je trazeni datum u korenu */
  if (koren->dan == dan && koren->mesec == mesec)
    /* Vraca se njegova vrednost */
    return koren;

  /* Ako je mesec trazenog datuma manji od meseca sadrzanog u
     korenu ili ako su meseci isti, ali je dan trazenog datuma
     manji od aktuelnog datuma, pretrazuje se levo podstablo. Pre
     toga se svakako proverava da li leva grana postoji. Ako ne
     postoji treba vratiti prvi sledeci, a to je bas vrednost
     uocenog korena */
```

```c
  if (mesec < koren->mesec
      || (mesec == koren->mesec && dan < koren->dan)) {
    if (koren->levo == NULL)
      return koren;
    else
      return pretrazi(koren->levo, dan, mesec);
  }

  /* Inace se nastavlja pretraga u desnom delu */
  return pretrazi(koren->desno, dan, mesec);
}

/* Funkcija koja pronalazi najmanji datum u stablu */
Cvor *pronadji_najmanji_datum(Cvor * koren)
{
  /* Ako je stablo prazno, obustavlja se pretraga */
  if (koren == NULL)
    return NULL;

  /* Ako ne postoji leva grana korena, zbog uredjenja stabla koren
     sadrzi najmanji datum */
  if (koren->levo == NULL)
    return koren;
  else
    /* Inace, trazi se manji datum u levom podstablu */
    return pronadji_najmanji_datum(koren->levo);
}

/* Funkcija koja za dati dan i mesec odredjuje nisku formata DD.MM.
 */
void datum_u_nisku(int dan, int mesec, char datum[])
{
  if (dan < 10) {
    datum[0] = '0';
    datum[1] = dan + '0';
  } else {
    datum[0] = dan / 10 + '0';
    datum[1] = dan % 10 + '0';
  }
  datum[2] = '.';

  if (mesec < 10) {
    datum[3] = '0';
    datum[4] = mesec + '0';
  } else {
    datum[3] = mesec / 10 + '0';
    datum[4] = mesec % 10 + '0';
  }
  datum[5] = '.';
  datum[6] = '\0';
}
```

```c
/* Funkcija koja oslobadja memoriju zauzetu stablom */
void oslobodi_stablo(Cvor ** adresa_korena)
{
  /* Stablo je prazno */
  if (*adresa_korena == NULL)
    return;

  /* Ako postoji levo podstablo, oslobadja se memorija koju zauzima
   */
  if ((*adresa_korena)->levo)
    oslobodi_stablo(&(*adresa_korena)->levo);

  /* Ako postoji desno podstablo, oslobadja se memorija koju
     zauzima */
  if ((*adresa_korena)->desno)
    oslobodi_stablo(&(*adresa_korena)->desno);

  /* Oslobadja se memorija zauzeta korenom */
  free(*adresa_korena);

  /* Stablo se proglasava praznim */
  *adresa_korena = NULL;
}

int main(int argc, char **argv)
{
  FILE *in;
  Cvor *koren;
  Cvor *slavljenik;
  char ime[MAX_NISKA], prezime[MAX_NISKA];
  int dan, mesec;
  char datum[7];

  /* Provera da li je zadato ime ulazne datoteke */
  if (argc < 2) {
    /* Ako nije, ispisuje se poruka i prekida se sa izvrsavanjem
       programa */
    fprintf(stderr, "Greska: Nedostaje ime ulazne datoteke!\n");
    exit(EXIT_FAILURE);
  }

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

  /* Stablo se popunjava podacima uz proveru uspesnosti dodavanja */
  koren = NULL;
  while (fscanf
```

```
199           (in, "%s %s %d.%d.", ime, prezime, &dan, &mesec) != EOF)
      if (dodaj_u_stablo(&koren, ime, prezime, dan, mesec) == 1) {
201     fprintf(stderr,
                "Greska: Neuspelo dodavanje podataka za %s %s.\n",
203             ime, prezime);
        oslobodi_stablo(&koren);
205     exit(EXIT_FAILURE);
      }
207
    /* Zatvara se datoteka */
209 fclose(in);

211 /* Omogucuje se pretraga podataka */
    while (1) {
213   /* Ucitava se novi datum */
      printf("Unesite datum: ");
215   if (scanf("%d.%d.", &dan, &mesec) == EOF)
        break;
217
      /* Pretrazuje se stablo */
219   slavljenik = pretrazi(koren, dan, mesec);

221   /* Ispisuju se pronadjeni podaci */

223   /* Ako slavljenik nije pronadjen, to moze znaci da: */
      /* 1. Drvo je prazno */
225   if (slavljenik == NULL && koren == NULL) {
        printf("Nema podataka o ovom ni o sledecem rodjendanu.\n");
227     continue;
      }
229   /* 2. Posle datuma koji je unesen, nema podataka u stablu - u
         ovom slucaju se pretraga vrsi pocevsi od naredne godine i
231      ispisuje se najmanji datum */
      if (slavljenik == NULL) {
233     slavljenik = pronadji_najmanji_datum(koren);
        datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
235     printf("Slavljenik: %s %s %s\n", slavljenik->ime,
                slavljenik->prezime, datum);
237     continue;
      }
239
      /* Ako je slavljenik pronadjen, razlikuju se slucajevi: */
241   /* 1. Pronadjeni su tacni podaci */
      if (slavljenik->dan == dan && slavljenik->mesec == mesec) {
243     printf("Slavljenik: %s %s\n", slavljenik->ime,
                slavljenik->prezime);
245     continue;
      }
247
      /* 2. Pronadjeni su podaci o prvom sledecem rodjendanu */
249   datum_u_nisku(slavljenik->dan, slavljenik->mesec, datum);
      printf("Slavljenik: %s %s %s\n", slavljenik->ime,
```

```
            slavljenik->prezime, datum);
  }

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.19

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja proverava da li su dva stabla koja sadrze cele
   brojeve identicna. Povratna vrednost funkcije je 1 ako jesu,
   odnosno 0 ako nisu */
int identitet(Cvor * koren1, Cvor * koren2)
{
  /* Ako su oba stabla prazna, identicna su */
  if (koren1 == NULL && koren2 == NULL)
    return 1;

  /* Ako je jedno stablo prazno, a drugo nije, stabla nisu
     identicna */
  if (koren1 == NULL || koren2 == NULL)
    return 0;

  /* Ako su oba stabla neprazna i u korenima se nalaze razlicite
     vrednosti, moze se zakljuciti da se razlikuju */
  if (koren1->broj != koren2->broj)
    return 0;

  /* Inace, proverava se da li vazi identitet i levih i desnih
     podstabala */
  return (identitet(koren1->levo, koren2->levo)
          && identitet(koren1->desno, koren2->desno));
}

int main()
{
  int broj;
  Cvor *koren1, *koren2;

  /* Ucitavaju se elementi prvog stabla */
```

```
  koren1 = NULL;
  printf("Prvo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    if (dodaj_u_stablo(&koren1, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
              broj);
      oslobodi_stablo(&koren1);
      exit(EXIT_FAILURE);
    }
    scanf("%d", &broj);
  }

  /* Ucitavaju se elementi drugog stabla */
  koren2 = NULL;
  printf("Drugo stablo: ");
  scanf("%d", &broj);
  while (broj != 0) {
    if (dodaj_u_stablo(&koren2, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
              broj);
      oslobodi_stablo(&koren2);
      exit(EXIT_FAILURE);
    }
    scanf("%d", &broj);
  }

  /* Poziva se funkcija koja ispituje identitet stabala i ispisuje
     se njen rezultat */
  if (identitet(koren1, koren2))
    printf("Stabla jesu identicna.\n");
  else
    printf("Stabla nisu identicna.\n");

  /* Oslobadja se memorija zauzeta stablima */
  oslobodi_stablo(&koren1);
  oslobodi_stablo(&koren2);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.20

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"
```

```c

/* Funkcija kreira novo stablo identicno stablu koje je dato
   korenom. Povratna vrednost funkcije je 0 ukoliko je kopiranje
   uspesno, odnosno 1 ukoliko je doslo do greske */
int kopiraj_stablo(Cvor * koren, Cvor ** duplikat)
{
  /* Izlaz iz rekurzije */
  if (koren == NULL) {
    *duplikat = NULL;
    return 0;
  }

  /* Duplira se koren stabla i postavlja da bude koren novog stabla
   */
  *duplikat = napravi_cvor(koren->broj);
  if (*duplikat == NULL) {
    return 1;
  }

  /* Rekurzivno se duplirju levo i desno podstablo i njihove adrese
     se cuvaju redom u pokazivacima na levo i desno podstablo korena
     duplikata */
  int kopija_levo = kopiraj_stablo(koren->levo, &(*duplikat)->levo);
  int kopija_desno =
      kopiraj_stablo(koren->desno, &(*duplikat)->desno);

  /* Ako je uspesno duplirano i levo i desno podstablo */
  if (kopija_levo == 0 && kopija_desno == 0)
    /* Uspesno je duplirano i celo stablo */
    return 0;
  /* Inace, prijavljuje se da je doslo do greske */
  return 1;
}

/* Funkcija izracunava uniju dva skupa predstavljena stablima -
   rezultujuci skup tj. stablo se dobija modifikacijom prvog
   stabla. Povratna vrednost funkcije je 0 ukoliko je kreiranje
   unije uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_uniju(Cvor ** adresa_korena1, Cvor * koren2)
{
  /* Ako drugo stablo nije prazno */
  if (koren2 != NULL) {
    /* 1. Dodaje se njegov koren u prvo stablo */
    if (dodaj_u_stablo(adresa_korena1, koren2->broj) == 1) {
      return 1;
    }

    /* 2. Rekurzivno se racuna unija levog i desnog podstabla
       drugog stabla sa prvim stablom */
    int unija_levo = kreiraj_uniju(adresa_korena1, koren2->levo);
    int unija_desno = kreiraj_uniju(adresa_korena1, koren2->desno);
```

```
58      /* Ako je unija podstabala uspesno kreirana */
        if (unija_levo == 0 && unija_desno == 0)
60        /* Uspesno je kreirana i unija stabala */
          return 0;
62
        /* U suprotnom se prijavljuje da je doslo do greske */
64      return 1;
    }
66
    /* Ako je drugo stablo prazno, nista se ne preduzima */
68  return 0;
}
70
/* Funkcija izracunava presek dva skupa predstavljana stablima -
72     rezultujuci skup tj. stablo se dobija modifikacijom prvog
       stabla. Povratna vrednost funkcije je 0 ukoliko je kreiranje
74     preseka uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_presek(Cvor ** adresa_korena1, Cvor * koren2)
76 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
78  if (*adresa_korena1 == NULL)
      return 0;
80
    /* Inace, kreira se presek levog i desnog podstabla sa drugim
82     stablom, tj. iz levog i desnog podstabla prvog stabla brisu se
       svi oni elementi koji ne postoje u drugom stablu */
84  int presek_levo =
        kreiraj_presek(&(*adresa_korena1)->levo, koren2);
86  int presek_desno =
        kreiraj_presek(&(*adresa_korena1)->desno, koren2);
88  if (presek_levo == 0 && presek_desno == 0) {
      /* Ako se koren prvog stabla ne nalazi u drugom stablu tada se
90       on uklanja iz prvog stabla */
      if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) == NULL)
92      obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

94    /* U ovom slucaju je presek stabala uspesno kreiran */
      return 0;
96  }
    /* Inace, prijavljuje se da je doslo do greske */
98  return 1;
}
100
/* Funkcija izracunava razliku dva skupa predstavljana stablima -
102    rezultujuci skup tj. stablo se dobija modifikacijom prvog
       stabla. Povratna vrednost funkcije je 0 ukoliko je kreiranje
104    razlike uspesno, odnosno 1 ukoliko je doslo do greske */
int kreiraj_razliku(Cvor ** adresa_korena1, Cvor * koren2)
106 {
    /* Ako je prvo stablo prazno, tada je i rezultat prazno stablo */
108 if (*adresa_korena1 == NULL)
      return 0;
```

```
110
   /* Inace, kreira se razlika levog i desnog podstabla sa drugim
112      stablom, tj. iz levog i desnog podstabla prvog stabla se brisu
         svi oni elementi koji postoje i u drugom stablu */
114   int razlika_levo =
          kreiraj_razliku(&(*adresa_korena1)->levo, koren2);
116   int razlika_desno =
          kreiraj_razliku(&(*adresa_korena1)->desno, koren2);
118   if (razlika_levo == 0 && razlika_desno == 0) {
        /* Ako se koren prvog stabla nalazi i u drugom stablu tada se
120         on uklanja se iz prvog stabla */
        if (pretrazi_stablo(koren2, (*adresa_korena1)->broj) != NULL)
122       obrisi_element(adresa_korena1, (*adresa_korena1)->broj);

124     /* Razlika stabala je uspesno kreirana */
        return 0;
126   }

128   /* Inace, prijavljuje se da je doslo do greske */
      return 1;
130 }

132 int main()
    {
134   Cvor *skup1;
      Cvor *skup2;
136   Cvor *pomocni_skup = NULL;
      int n;
138
      /* Ucitavaju se elementi prvog skupa */
140   skup1 = NULL;
      printf("Prvi skup: ");
142   while (scanf("%d", &n) != EOF) {
        if (dodaj_u_stablo(&skup1, n) == 1) {
144       fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
          oslobodi_stablo(&skup1);
146       exit(EXIT_FAILURE);
        }
148   }

150   /* Ucitavaju se elementi drugog skupa */
      skup2 = NULL;
152   printf("Drugi skup: ");
      while (scanf("%d", &n) != EOF) {
154     if (dodaj_u_stablo(&skup2, n) == 1) {
          fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n", n);
156       oslobodi_stablo(&skup2);
          exit(EXIT_FAILURE);
158     }
      }
160
      /* Kreira se unija skupova. Pre svega, napravi se kopija prvog
```

```
    skupa kako bi se polazni skup mogao iskoristiti i za preostale
    operacije */
  if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    exit(EXIT_FAILURE);
  }
  if (kreiraj_uniju(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
    exit(EXIT_FAILURE);
  }
  printf("Unija: ");
  ispisi_stablo_infiksno(pomocni_skup);
  putchar('\n');

  /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
     prethodne operacije */
  oslobodi_stablo(&pomocni_skup);

  /* Kreira se presek skupova. Prvo se napravi kopija prvog skupa
     kako bi se polazni skup mogao iskoristiti i za preostale
     operacije */
  if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    exit(EXIT_FAILURE);
  }
  if (kreiraj_presek(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
    exit(EXIT_FAILURE);
  }
  printf("Presek: ");
  ispisi_stablo_infiksno(pomocni_skup);
  putchar('\n');

  /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
     prethodne operacije */
  oslobodi_stablo(&pomocni_skup);

  /* Kreira se razlika skupova. Prvo se napravi kopija prvog skupa
     kako bi se polazni skup mogao iskoristiti i za preostale
     operacije */
  if (kopiraj_stablo(skup1, &pomocni_skup) == 1) {
    oslobodi_stablo(&skup1);
    oslobodi_stablo(&pomocni_skup);
    exit(EXIT_FAILURE);
  }
  if (kreiraj_razliku(&pomocni_skup, skup2) == 1) {
    oslobodi_stablo(&pomocni_skup);
    oslobodi_stablo(&skup2);
```

```
214     exit(EXIT_FAILURE);
    }
216   printf("Razlika: ");
    ispisi_stablo_infiksno(pomocni_skup);
218   putchar('\n');

220   /* Oslobadja se memorija zauzeta pomocnim skupom sa rezultatom
        prethodne operacije */
222   oslobodi_stablo(&pomocni_skup);

224   /* Oslobadja se memorija zauzeta polaznim skupovima */
    oslobodi_stablo(&skup1);
226   oslobodi_stablo(&skup2);

228   exit(EXIT_SUCCESS);
}
```

### Rešenje 1.21

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   /* Ukljucuje se biblioteka za rad sa stablima */
5  #include "stabla.h"

7  #define MAX 50

9  /* Funkcija koja obilazi stablo sa leva na desno i smesta vrednosti
      cvorova u niz. Povratna vrednost funkcije je broj vrednosti koje
11    su smestene u niz */
   int kreiraj_niz(Cvor * koren, int a[])
13 {
     int r, s;
15
     /* Stablo je prazno - u niz je smesteno 0 elemenata */
17   if (koren == NULL)
       return 0;
19
     /* Dodaju se u niz elementi iz levog podstabla */
21   r = kreiraj_niz(koren->levo, a);

23   /* Tekuca vrednost promenljive r je broj elemenata koji su
        upisani u niz i na osnovu nje se moze odrediti indeks novog
25      elementa */

27   /* Smesta se vrednost iz korena */
     a[r] = koren->broj;
29
```

```
  /* Dodaju se elementi iz desnog podstabla */
  s = kreiraj_niz(koren->desno, a + r + 1);

  /* Racuna se indeks na koji treba smestiti naredni element */
  return r + s + 1;
}

/* Funkcija sortira niz tako sto najpre elemente niza smesti u
   stablo, a zatim kreira novi niz prolazeci kroz stablo sa leva na
   desno. Povratna vrednost funkcije je 0 ukoliko je niz uspesno
   kreiran i sortiran, a 1 ukoliko je doslo do greske.

   Ovaj nacin sortiranja je primer sortiranja koje nije "u mestu"
   kao sto je to slucaj sa ostalim opisanim algoritmima sortiranja
   jer se sortiranje vrsi u pomocnoj dinamickoj strukturi, a ne
   razmenom elemenata niza. */
int sortiraj(int a[], int n)
{
  int i;
  Cvor *koren;

  /* Kreira se stablo smestanjem elemenata iz niza u stablo */
  koren = NULL;
  for (i = 0; i < n; i++) {
    if (dodaj_u_stablo(&koren, a[i]) == 1) {
      oslobodi_stablo(&koren);
      return 1;
    }
  }
  /* Infiksnim obilaskom stabla elementi iz stabla se prepisuju u
     niz a */
  kreiraj_niz(koren, a);

  /* Stablo vise nije potrebno pa se oslobadja zauzeta memorija */
  oslobodi_stablo(&koren);

  /* Vraca se indikator uspesnog sortiranja */
  return 0;
}

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

  /* Ucitavaju se dimenzija i elementi niza */
  printf("n: ");
  scanf("%d", &n);
  if (n < 0 || n > MAX) {
    printf("Greska: Pogresna dimenzija niza!\n");
    exit(EXIT_FAILURE);
  }
```

```
  printf("a: ");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Poziva se funkcija za sortiranje */
  if (sortiraj(a, n) == 0) {
    /* Ako je niz uspesno sortiran, ispisuje se rezultujuci niz */
    for (i = 0; i < n; i++)
      printf("%d ", a[i]);
    printf("\n");
  } else {
    /* Inace, obavestava se korisnik da je doslo do greske */
    printf("Greska: Problem prilikom sortiranja niza!\n");
  }

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.22

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* a) Funkcija koja izracunava broj cvorova stabla */
int broj_cvorova(Cvor * koren)
{
  /* Ako je stablo prazno, broj cvorova je 0 */
  if (koren == NULL)
    return 0;

  /* U suprotnom je broj cvorova stabla jednak zbiru broja cvorova
     u levom podstablu, broja cvorova u desnom podstablu i 1, zato
     sto treba racunati i koren */
  return broj_cvorova(koren->levo) + broj_cvorova(koren->desno) + 1;
}

/* b) Funkcija koja izracunava broj listova stabla */
int broj_listova(Cvor * koren)
{
  /* Ako je stablo prazno, broj listova je nula */
  if (koren == NULL)
    return 0;

  /* Proverava se da li je tekuci cvor list */
```

```
28    if (koren->levo == NULL && koren->desno == NULL)
        /* Ako jeste vraca se 1 - ova vrednost ce kasnije zbog
30         rekurzivnih poziva uvecati broj listova za 1 */
      return 1;
32
    /* U suprotnom se prebrojavaju listovi koje se nalaze u
34       podstablima */
    return broj_listova(koren->levo) + broj_listova(koren->desno);
36  }

38  /* c) Funkcija koja stampa pozitivne vrednosti listova stabla */
    void pozitivni_listovi(Cvor * koren)
40  {
    /* Slucaj kada je stablo prazno */
42  if (koren == NULL)
      return;
44
    /* Ako je cvor list i sadrzi pozitivnu vrednost */
46  if (koren->levo == NULL && koren->desno == NULL
        && koren->broj > 0)
48    /* Stampa se */
      printf("%d ", koren->broj);
50
    /* Nastavlja se sa stampanjem pozitivnih listova u podstablima */
52  pozitivni_listovi(koren->levo);
    pozitivni_listovi(koren->desno);
54  }

56  /* d) Funkcija koja izracunava zbir cvorova stabla */
    int zbir_svih_cvorova(Cvor * koren)
58  {
    /* Ako je stablo prazno, zbir cvorova je 0 */
60  if (koren == NULL)
      return 0;
62
    /* Inace, zbir cvorova stabla izracunava se kao zbir korena i
64       svih elemenata u podstablima */
    return koren->broj + zbir_svih_cvorova(koren->levo) +
66       zbir_svih_cvorova(koren->desno);
    }
68
    /* e) Funkcija koja izracunava najveci element stabla */
70  Cvor *najveci_element(Cvor * koren)
    {
72  /* Ako je stablo prazno, obustavlja se pretraga */
    if (koren == NULL)
74    return NULL;

76  /* Zbog prirode pretrazivackog stabla, vrednosti vece od korena
       se nalaze u desnom podstablu */
78
    /* Ako desnog podstabla nema */
```

```
  if (koren->desno == NULL)
    /* Najveca vrednost je koren */
    return koren;

  /* Inace, najveca vrednost se trazi desno */
  return najveci_element(koren->desno);
}

/* f) Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
    return 0;

  /* Izracunava se dubina levog podstabla */
  int dubina_levo = dubina_stabla(koren->levo);

  /* Izracunava se dubina desnog podstabla */
  int dubina_desno = dubina_stabla(koren->desno);

  /* Dubina stabla odgovara vecoj od dubina podstabala - 1 se
     dodaje jer se racuna i koren stabla */
  return dubina_levo >
      dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
}

/* g) Funkcija koja izracunava broj cvorova na i-tom nivou stabla */
int broj_cvorova_na_itom_nivou(Cvor * koren, int i)
{
  /* Ako je stablo prazno, broj cvorova je 0 */
  if (koren == NULL)
    return 0;

  /* Ako se stiglo do trazenog nivoa, vraca se 1. Ova vrednost ce
     kasnije zbog rekurzivnih poziva uvecati broj cvorova za 1. */
  if (i == 0)
    return 1;

  /* Inace, nastavlja se prebrojavanje na nivou nize i u levom i u
     desnom postablu */
  return broj_cvorova_na_itom_nivou(koren->levo, i - 1)
      + broj_cvorova_na_itom_nivou(koren->desno, i - 1);
}

/* h) Funkcija koja ispisuje sve elemente na i-tom nivou */
void ispis_nivo(Cvor * koren, int i)
{
  /* Ako je stablo prazno, nista se ne ispisuje */
  if (koren == NULL)
    return;
```

```c
  /* Ako se stiglo do trazenog nivoa, ispisuje se vrednost */
  if (i == 0) {
    printf("%d ", koren->broj);
    return;
  }
  /* Inace, spustanje se nastavlja za jedan nivo nize i u levom i u
     desnom podstablu */
  ispis_nivo(koren->levo, i - 1);
  ispis_nivo(koren->desno, i - 1);
}

/* i) Funkcija koja izracunava maksimalnu vrednost na i-tom nivou
   stabla */
Cvor *najveci_element_na_itom_nivou(Cvor * koren, int i)
{
  /* Ako je stablo prazno, obustavlja se pretraga */
  if (koren == NULL)
    return NULL;

  /* Ako se stiglo do trazenog nivoa, takodje se prekida pretraga */
  if (i == 0)
    return koren;

  /* Pronalazi se maksimum na i-tom nivou levog podstabla */
  Cvor *a = najveci_element_na_itom_nivou(koren->levo, i - 1);

  /* Pronalazi se maksimum na i-tom nivou desnog podstabla */
  Cvor *b = najveci_element_na_itom_nivou(koren->desno, i - 1);

  /* Trazi se i vraca maksimum izracunatih vrednosti */
  if (a == NULL && b == NULL)
    return NULL;
  if (a == NULL)
    return b;
  if (b == NULL)
    return a;
  /* Ako su obe vrednosti razlicite od NULL, veca od vrednosti se
     nalazi u b cvoru jer je stablo pretrazivacko */
  return b;
}

/* j) Funkcija koja izracunava zbir cvorova na i-tom nivou */
int zbir_cvorova_na_itom_nivou(Cvor * koren, int i)
{
  /* Ako je stablo prazno, zbir je 0 */
  if (koren == NULL)
    return 0;

  /* Ako se stiglo do trazenog nivoa, vraca se vrednost */
  if (i == 0)
    return koren->broj;
```

```c
  /* Inace, spustanje se nastavlja za jedan nivo nize i
     izracunavaju se sume levog i desnog podstabla */
  return zbir_cvorova_na_itom_nivou(koren->levo, i - 1)
      + zbir_cvorova_na_itom_nivou(koren->desno, i - 1);
}

/* k) Funkcija koja izracunava zbir svih vrednosti u stablu koje su
   manje ili jednake od date vrednosti x */
int zbir_manjih_od_x(Cvor * koren, int x)
{
  /* Ako je stablo prazno, zbir je 0 */
  if (koren == NULL)
    return 0;

  /* Ako je vrednost u korenu manja od trazene vrednosti, zbog
     prirode pretrazivackog stabla treba obici i levo i desno
     podstablo */
  if (koren->broj <= x)
    return koren->broj + zbir_manjih_od_x(koren->levo, x) +
        zbir_manjih_od_x(koren->desno, x);

  /* Inace, racuna se samo suma vrednosti iz levog podstabla jer
     medju njima jedino moze biti onih koje zadovoljavaju uslov */
  return zbir_manjih_od_x(koren->levo, x);
}

int main(int argc, char **argv)
{
  /* Analiziraju se argumenti komandne linije */
  if (argc != 3) {
    fprintf(stderr,
            "Greska: Program se poziva sa: ./a.out nivo
    broj_za_pretragu\n");
    exit(EXIT_FAILURE);
  }
  int i = atoi(argv[1]);
  int x = atoi(argv[2]);

  /* Kreira se stablo uz proveru uspesnosti dodavanja novih
     vrednosti */
  Cvor *koren = NULL;
  int broj;
  while (scanf("%d", &broj) != EOF) {
    if (dodaj_u_stablo(&koren, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
              broj);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Ispisuju se rezultati rada funkcija */
```

```
  printf("Broj cvorova: %d\n", broj_cvorova(koren));
  printf("Broj listova: %d\n", broj_listova(koren));
  printf("Pozitivni listovi: ");
  pozitivni_listovi(koren);
  printf("\n");
  printf("Zbir cvorova: %d\n", zbir_svih_cvorova(koren));
  if (najveci_element(koren) == NULL)
    printf("Najveci element: ne postoji\n");
  else
    printf("Najveci element: %d\n", najveci_element(koren)->broj);

  printf("Dubina stabla: %d\n", dubina_stabla(koren));

  printf("Broj cvorova na %d. nivou: %d\n", i,
         broj_cvorova_na_itom_nivou(koren, i));
  printf("Elementi na %d. nivou: ", i);
  ispis_nivo(koren, i);
  printf("\n");
  if (najveci_element_na_itom_nivou(koren, i) == NULL)
    printf("Nema elemenata na %d. nivou!\n", i);
  else
    printf("Maksimalni element na %d. nivou: %d\n", i,
           najveci_element_na_itom_nivou(koren, i)->broj);

  printf("Zbir elemenata na %d. nivou: %d\n", i,
         zbir_cvorova_na_itom_nivou(koren, i));
  printf("Zbir elemenata manjih ili jednakih od %d: %d\n", x,
         zbir_manjih_od_x(koren, x));

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.23

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
```

```
12      return 0;

14    /* Izracunava se dubina levog podstabla */
      int dubina_levo = dubina_stabla(koren->levo);
16
      /* Izracunava se dubina desnog podstabla */
18    int dubina_desno = dubina_stabla(koren->desno);

20    /* Dubina stabla odgovara vecoj od dubina podstabala uvecanoj za
         1, jer se racuna i koren stabla. */
22    return dubina_levo >
          dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
24  }

26  /* Funkcija koja ispisuje sve elemente na i-tom nivou */
    void ispisi_nivo(Cvor * koren, int i)
28  {
      /* Ako nema vise cvorova, nema ni spustanja niz stablo */
30    if (koren == NULL)
        return;
32
      /* Ako se stiglo do trazenog nivoa, ispisuje se vrednost. */
34    if (i == 0) {
        printf("%d ", koren->broj);
36      return;
      }
38    /* Inace, vrsi se spustanje za jedan nivo nize i u levom i u
         desnom podstablu */
40    ispisi_nivo(koren->levo, i - 1);
      ispisi_nivo(koren->desno, i - 1);
42  }

44  /* Funkcija koja ispisuje stablo po nivoima */
    void ispisi_stablo_po_nivoima(Cvor * koren)
46  {
      int i;
48
      /* Prvo se izracunava dubina stabla */
50    int dubina;
      dubina = dubina_stabla(koren);
52
      /* Zatim se ispisuje nivo po nivo stabla */
54    for (i = 0; i < dubina; i++) {
        printf("%d. nivo: ", i);
56      ispisi_nivo(koren, i);
        printf("\n");
58    }
    }
60
    int main(int argc, char **argv)
62  {
      Cvor *koren;
```

```
  int broj;

  /* Citaju se vrednosti sa ulaza i dodaju se u stablo uz proveru
     uspesnosti dodavanja */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    if (dodaj_u_stablo(&koren, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
              broj);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Ispisuje se stablo po nivoima */
  ispisi_stablo_po_nivoima(koren);

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.25

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja izracunava dubinu stabla */
int dubina_stabla(Cvor * koren)
{
  /* Dubina praznog stabla je 0 */
  if (koren == NULL)
    return 0;

  /* Izracunava se dubina levog podstabla */
  int dubina_levo = dubina_stabla(koren->levo);

  /* Izracunava se dubina desnog podstabla */
  int dubina_desno = dubina_stabla(koren->desno);

  /* Dubina stabla odgovara vecoj od dubina podstabala uvecanoj za
     1, jer se racuna i koren. */
  return dubina_levo >
      dubina_desno ? dubina_levo + 1 : dubina_desno + 1;
```

```c
}

/* Funkcija koja racuna broj cvorova koji ispunjavaju uslov za AVL
   stablo */
int avl(Cvor * koren)
{
  int dubina_levo, dubina_desno;

  /* Ako je stablo prazno, zaustavlja se brojanje */
  if (koren == NULL) {
    return 0;
  }

  /* Izracunava se dubina levog podstabla korena */
  dubina_levo = dubina_stabla(koren->levo);

  /* Izracunava se dubina desnog podstabla korena */
  dubina_desno = dubina_stabla(koren->desno);

  /* Ako je uslov za AVL stablo ispunjen */
  if (abs(dubina_desno - dubina_levo) <= 1) {
    /* Racuna se broj AVL cvorova u levom i desnom podstablu i
       uvecava za jedan iz razloga sto koren ispunjava uslov */
    return 1 + avl(koren->levo) + avl(koren->desno);
  } else {
    /* Inace, racuna se samo broj AVL cvorova u podstablima */
    return avl(koren->levo) + avl(koren->desno);
  }
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int broj;

  /* Ucitavaju se vrednosti sa ulaza i dodaju u stablo uz proveru
     uspesnosti dodavanja */
  koren = NULL;
  while (scanf("%d", &broj) != EOF) {
    if (dodaj_u_stablo(&koren, broj) == 1) {
      fprintf(stderr, "Greska: Neuspelo dodavanje broja %d.\n",
              broj);
      oslobodi_stablo(&koren);
      exit(EXIT_FAILURE);
    }
  }

  /* Racuna se i ispisuje broj AVL cvorova */
  printf("%d\n", avl(koren));

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);
```

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

### Rešenje 1.26

NAPOMENA: *Rešenje koristi biblioteku za rad sa binarnim pretraživačkim stablima iz zadatka 1.14.*

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

/* Ukljucuje se biblioteka za rad sa stablima */
#include "stabla.h"

/* Funkcija koja kreira stablo prema zadatoj slici. Povratna
   vrednost funkcije je 0 ako je stablo uspesno kreirano, odnosno 1
   ukoliko je doslo do greske */
int kreiraj_hip(Cvor ** adresa_korena)
{
  /* Stablo se proglasava praznim */
  *adresa_korena = NULL;

  /* Dodaje se cvor po cvor uz proveru uspesnosti dodavanja */
  if (((*adresa_korena) = napravi_cvor(100)) == NULL)
    return 1;
  if (((*adresa_korena)->levo = napravi_cvor(19)) == NULL)
    return 1;
  if (((*adresa_korena)->levo->levo = napravi_cvor(17)) == NULL)
    return 1;
  if (((*adresa_korena)->levo->levo->levo =
       napravi_cvor(2)) == NULL)
    return 1;
  if (((*adresa_korena)->levo->levo->desno =
       napravi_cvor(7)) == NULL)
    return 1;
  if (((*adresa_korena)->levo->desno = napravi_cvor(3)) == NULL)
    return 1;
  if (((*adresa_korena)->desno = napravi_cvor(36)) == NULL)
    return 1;
  if (((*adresa_korena)->desno->levo = napravi_cvor(25)) == NULL)
    return 1;
  if (((*adresa_korena)->desno->desno = napravi_cvor(1)) == NULL)
    return 1;

  /* Vraca se indikator uspesnog kreiranja */
  return 0;
}

/* Funkcija proverava da li je zadato binarno stablo celih
   pozitivnih brojeva hip. Ideja koja ce biti implementirana u
```

```c
   osnovi ima pronalazenje maksimalne vrednosti levog i maksimalne
   vrednosti desnog podstabla. Ako je vrednost u korenu veca od
   izracunatih vrednosti, uoceni fragment stabla zadovoljava uslov
   za hip. Zato ce funkcija vracati maksimalne vrednosti iz uocenog
   podstabala ili vrednost -1 ukoliko se zakljuci da stablo nije
   hip. */
int hip(Cvor * koren)
{
  int max_levo, max_desno;

  /* Prazno sablo je hip. Kao rezultat se vraca 0, kao najmanji
     pozitivan broj */
  if (koren == NULL) {
    return 0;
  }
  /* Ukoliko je stablo list... */
  if (koren->levo == NULL && koren->desno == NULL) {
    /* Vraca se njegova vrednost */
    return koren->broj;
  }

  /* Inace, proverava se svojstvo za levo podstablo */
  max_levo = hip(koren->levo);

  /* Proverava se svojstvo za desno podstablo */
  max_desno = hip(koren->desno);

  /* Ako levo ili desno podstablo uocenog cvora nije hip, onda nije
     ni celo stablo */
  if (max_levo == -1 || max_desno == -1) {
    return -1;
  }

  /* U suprotonom proverava se da li svojstvo vazi za uoceni cvor */
  if (koren->broj > max_levo && koren->broj > max_desno) {
    /* Ako vazi, vraca se vrednost korena */
    return koren->broj;
  }

  /* U suprotnom se zakljucuje da stablo nije hip */
  return -1;
}

int main(int argc, char **argv)
{
  Cvor *koren;
  int hip_indikator;

  /* Kreira se stablo prema zadatoj slici */
  if (kreiraj_hip(&koren) == 1) {
    fprintf(stderr, "Greska: Neuspesno kreiranje hipa.\n");
    oslobodi_stablo(&koren);
```

```
    exit(EXIT_FAILURE);
  }

  /* Poziva se funkcija kojom se proverava da li je stablo hip */
  hip_indikator = hip(koren);

  /* Ispisuje se rezultat */
  if (hip_indikator == -1) {
    printf("Zadato stablo nije hip!\n");
  } else {
    printf("Zadato stablo je hip!\n");
  }

  /* Oslobadja se memorija zauzeta stablom */
  oslobodi_stablo(&koren);

  exit(EXIT_SUCCESS);
}
```