Univerzitet u Beogradu
Matematički fakultet

Milena, Jelena, Ana, Mirko, Anđelka, Nina

Zbirka programa

Beograd, 2015.

Predgovor

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

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

Zahvaljujemo se recenzentima na ..., kao i studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli u obličavanju ovog materijala.

Autori

Sadržaj

Glava 1

Uvodni zadaci

1.1 Podela koda po datotekama

1.2 Algoritmi za rad sa bitovima

Zadatak 1.1

(a) Napisati funkciju `print_bits` koja štampa bitove u binarnom zapisu celog broja x.

(b) Napisati program koja testira funkciju `print_bits` za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

Test 1

```
Ulaz:   0x7F
Izlaz:  0000 0000 0000 0000 0000 0000 0111 1111
```

Test 2

```
Ulaz:   0x80
Izlaz:  0000 0000 0000 0000 0000 0000 1000 0000
```

Test 3

```
Ulaz:   0x00FF00FF
Izlaz:  0000 0000 1111 1111 0000 0000 1111 1111
```

Test 4

```
Ulaz:   0xFFFFFFFF
Izlaz:  1111 1111 1111 1111 1111 1111 1111 1111
```

Test

```
Ulaz:   0xABCDE123
Izlaz:  1010 1011 1100 1101 1110 0001 0010 0011
```

Zadatak 1.2 Napisati funkciju koja broji bitove postavljene na `1` u zapisu broja x. Napisati program koji testira tu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

Test 1

```
Ulaz:   0x7F
Izlaz:  7
```

Test 2

```
Ulaz:   0x80
Izlaz:  1
```

Test 3

```
Ulaz:   0x00FF00FF
Izlaz:  16
```

Test 4

```
Ulaz:   0xFFFFFFFF
Izlaz:  32
```

Test 4

```
Ulaz:   0xABCDE123
Izlaz:  17
```

Zadatak 1.3

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

(b) Napisati program koji testira prethodno napisane funkcije tako što prikazuje binarnu reprezentaciju brojeva koji se dobijaju nakon poziva funkcije `najveci`, ondosno `najmanji` za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

Test 1

```
Ulaz:   0x7F
Izlaz:
 Najveci:
 1111 1110 0000 0000 0000 0000 0000 0000
 Najmanji:
 0000 0000 0000 0000 0000 0000 0111 1110
```

Test 2

```
Ulaz:   0x80
Izlaz:
 Najveci:
 1000 0000 0000 0000 0000 0000 0000 0000
 Najmanji:
 0000 0000 0000 0000 0000 0000 0000 0001
```

Test 3

```
Ulaz:   0x00FF00FF
Izlaz:
 Najveci:
 1111 1111 1111 1111 0000 0000 0000 0000
 Najmanji:
 0000 0000 0000 0000 1111 1111 1111 1111
```

Test 4

```
Ulaz:   0xFFFFFFFF
Izlaz:
 Najveci:
 1111 1111 1111 1111 1111 1111 1111 1111
 Najmanji:
 1111 1111 1111 1111 1111 1111 1111 1111
```

Test 4

```
Ulaz:   0xABCDE123
Izlaz:
 Najveci:
 1111 1111 1111 1111 1000 0000 0000 0000
 Najmanji:
 0000 0000 0000 0001 1111 1111 1111 1111
```

Zadatak 1.4 Napisati program za rad sa bitovima.

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

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

(c) Napisati funkciju koja određuje broj koji se dobija kada se n bitova datog broja, počevši od pozicije p i vraća ih kao bitove najmanje težine rezultata.

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

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

(f) Napisati program koji testira prethodno napisane funkcije.

Program treba da testira prethodno napisane funkcije nad neoznačenim celim brojem koji se unosi sa standardnog ulaza. *Napomena: Pozicije se broje počev od pozicije najnižeg bita, pri čemu se broji od nule .*

Zadatak 1.5

(a) Napisati funkciju funkciju koja određuje broj koji se dobija rotiranjem u levo datog celog broja. Napomena: Rotiranje podrazumeva pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najviše težine pomera na mesto najmanje težine.

(b) Napisati funkciju koja određuje broj koji se dobija rotiranjem u desno datog celog neoznačenog broja.

(c) Napisati funkciju koja određuje broj koji se dobija rotiranjem u desno datog celog broja.

(d) Napisati program koji testira prethodno napisane funkcije za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu.

Zadatak 1.6 Napisati funkciju `mirror` koja određuje ceo broj čiji binarni zapis je slika u ogledalu binarnog zapisa argumenta funkcije. Napisati i program koji testira datu funkciju za brojeve koji se sa standardnog ulaza zadaju u heksadekasnom formatu, tako što najpre ispisuje binarnu reprezentaciju unetog broja, a potom i binarnu reprezentaciju broja dobijenog nakon poziva funkcije `mirror` za uneti broj.

Zadatak 1.7 Napisati funkciju `int Broj01(unsigned int n)` koja za dati broj `n` vraća `1` ako u njegovom binarnom zapisu ima više jednica nego nula, a inače vraća `0`. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1

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

Test 2

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

Test 3

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

Test 4

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

Zadatak 1.8 Napisati funkciju koja broji koliko se puta kombinacija `11` (dve uzastopne jedinice) pojavljuje u binarnom zapisu celog neoznačenog broja x. Tri uzastopne jedinice se broje dva puta. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1

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

Test 2

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

Test 3

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

Test 4

```
Ulaz:   1111111115
Izlaz:  6
```

Zadatak 1.9 Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama i, j. Pozicije i, j se učitavaju kao parametri komandne linije. Smatrati da je krajnji desni bit binarne reprezentacije `0`-ti bit. Pri rešavanju nije dozvoljeno koristiti pomoćni niz niti aritmetičke operatore +,-,/,*,%.

Test 1

```
Poziv:  ./a.out 1 2
Ulaz:   11
Izlaz:  13
```

Test 2

```
Poziv: ./a.out 1 2
Ulaz:   1024
Izlaz:  1024
```

Test 3

```
Poziv: ./a.out 12 12
Ulaz:   12345
Izlaz:  12345
```

Zadatak 1.10 Napisati funkciju koja na osnovu neoznačenog broja x formira nisku s koja sadrži heksadekadni zapis broja x, koristeći algoritam za brzo prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksadekadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1

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

Test 2

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

Test 3

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

Zadatak 1.11 Napisati funkciju koja za dva data neoznačena broja x i y invertuje u podatku x one bitove koji se poklapaju sa odgovarajućim bitovima u broju y. Ostali bitovi ostaju nepromenjeni. Napisati program koji tu funkciju testira za brojeve koji se zadaju sa standardnog ulaza.

Test 1

```
Ulaz:   123 10
Izlaz:  4294967285
```

Test 2

```
Ulaz:  3251 0
Izlaz: 4294967295
```

Test 3

```
Ulaz:   12541 1024
Izlaz:  4294966271
```

Zadatak 1.12 Napisati funkciju koja računa koliko petica bi imao ceo neoznačen broj x u oktalnom zapisu. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1

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

Test 2

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

Test 3

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

Milena: Naredne zadatke prebaciti da budu nakon rekurzije.

Zadatak 1.13 Napisati rekurzivnu funkciju vraća broj bitova koji su postavljeni na 1, u binarnoj reprezentaciji njenog celobrojnog argumenta. Napisati program koji testira napisanu funkciju za brojeve koji se učitavaju sa standardnog ulaza zadati u heksadekadnom formatu.

Test 1

```
Ulaz:   0x7F
Izlaz:  7
```

Test 2

```
Ulaz:   0x80
Izlaz:  1
```

Test 3

```
Ulaz:   0x00FF00FF
Izlaz:  16
```

Test 4

```
Ulaz:   0xFFFFFFFF
Izlaz:  32
```

Zadatak 1.14

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

Test 1

```
Ulaz:       10
Izlaz:      00000000000000000000000000001010
```

Zadatak 1.15 Nina: Ovo je prebaceno iz poglavlja sa pokazivacima. Napisati rekurzivnu funkciju za određivanje najveće cifre u oktalnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: binarne cifre grupisati u podgrupe od po tri cifre, počev od bitova najmanje težine.

Test 1

```
Ulaz:   5
Izlaz:  5
```

Test 2

```
Ulaz:   125
Izlaz:  7
```

Test 3

```
Ulaz:   8
Izlaz:  1
```

Test 4

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

Zadatak 1.16 Nina: Ovo je prebaceno iz poglavlja sa pokazivacima. Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. Uputstvo: binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.

Test 1

```
Ulaz:   5
Izlaz:  5
```

Test 2

```
Ulaz:   16
Izlaz:  1
```

Test 3

```
Ulaz:   18
Izlaz:  2
```

Test 4

```
Ulaz:   165
Izlaz:  10
```

1.3 Rekurzija

Zadatak 1.17 Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva. Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenziju n ($0 < n \leq 100$) celobrojong niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom.

Zadatak 1.18 Napisati rekurzivnu funkciju koja izračunava x^k, za dati realni broj x i prirodan broj k. Napisati program koji testira napisanu funkciju za vrednosti koje se unose sa standardnog ulaza.

Zadatak 1.19 Koristeći uzajamnu (posrednu) rekurziju napisati naredne dve funkcije:

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

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

Zadatak 1.20 Napisati repno-rekurzivnu funkciju koja izračunava faktorijel broja n. Napisati program koji testira napisanu funkciju za poizvoljan broj n ($n \leq 12$) unet sa standardnog ulaza.

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

$$F(0) = 0$$

$$F(1) = 1$$

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

(a) Napisati rekurzivnu funkciju koja računa n-ti element u nizu F.

(b) Napisati rekurzivnu funkciju koja računa n-ti element u nizu F ali tako da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkcije za poizvoljan broj n ($n \in \mathbb{N}$) unet sa standardnog ulaza.

Zadatak 1.22 Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa $\{1, 2, ..., n\}$. Napisati program koji testira napisanu funkciju za poizvoljan prirodan broj n ($n \leq 50$) unet sa standardnog ulaza.

Zadatak 1.23 Paskalov trougao se dobija tako što mu je svako polje (izuzev jedinica po krajevima) zbir jednog polja levo i jednog polja iznad.

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

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

(b) Napisati rekurzivnu funkciju koja izračunava vrednost polja `(i, j)`. Milena: dodati sta je i a sta j tj odakle se broji

Zadatak 1.24 Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju, za niz koji se unosi sa standardnog ulaza. Niz neće imati više od 256 elemenata, i njegovi elementi se unose sve do kraja ulaza.

Test 1

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

Test 2

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

Test 3

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

Test 4

```
Ulaz:   5
Izlaz:  5
```

Zadatak 1.25 Napisati rekurzivnu funkciju koja izračunava skalarni proizvod dva data vektora. Napisati program koji testira ovu funkciju, za nizove koji se unose sa standardnog ulaza. Nizovi neće imati više od 256 elemenata. Prvo se unosi dimenzija nizova, a zatim i sami njihovi elementi.

Test 1

```
Ulaz:    3 1 2 3 1 2 3
Izlaz:   14
```

Test 2

```
Ulaz:   2 3 5 2 6
Izlaz:   36
```

Test 3

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

Zadatak 1.26 Napisati rekurzivnu funkciju koja sabira dekadne cifre datog celog broja x. Napisati program koji testira ovu funkciju, za broj koji se unosi sa standardnog ulaza.

Test 1

```
Ulaz:    123
Izlaz:   6
```

Test 2

```
Ulaz:    23156
Izlaz:   17
```

Test 3

```
Ulaz:    1432
Izlaz:   10
```

Test 4

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

Test 5

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

Zadatak 1.27 Napisati rekurzivnu funkciju koja računa broj pojavljivanja elementa x u nizu a dužine n. Napisati program koji testira ovu funkciju, za x i niz koji se unose sa standardnog ulaza. Niz neće imati više od 256 elemenata. Prvo se unosi x, a zatim elementi niza sve do kraja ulaza.

Test 1

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

Test 2

```
Ulaz:    11 3 2 11 14 11 43 1
Izlaz:   2
```

Test 3

```
Ulaz:  1 3 21 5 6
Izlaz:  0
```

Zadatak 1.28 Napisati rekurzivnu funkciju koja ispituje da li je data niska palindrom. Napisati program koji testira ovu funkciju. Pretposatviti da niska neće neće imati više od 31 karaktera, i da se unosi sa standardnog ulaza.

Test 1

```
Ulaz:    programiranje
Izlaz:   ne
```

Test 2

```
Ulaz:    anavolimilovana
Izlaz:   da
```

Test 3

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

Test 4

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

Test

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

Zadatak 1.29 Napisati rekurzivnu funkciju kojom se proverava da li su tri zadata broja uzastopni članovi niza. Potom, napisati program koji je testira. Sa standardnog ulaza se unose najpre tri tražena broja, a zatim elementi niza, sve do kraja ulaza. Pretpostaviti da neće biti uneto više od 256 brojeva.

Test 1

```
Ulaz:    1 2 3 4 1 2 3 4 5
Izlaz:   da
```

Test 2

```
Ulaz:    1 2 3 11 1 2 4 3 6
Izlaz:   ne
```

Test 3

```
Ulaz:    1 2 3 1 2
Izlaz:   ne
```

Zadatak 1.30 Napisati rekurzivnu funkciju koja prikazuje sve varijacije sa ponavljanjem dužine n skupa $\{a, b\}$, i program koji je testira, za n koje se unosi sa standardnog ulaza.

Test 1

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

Zadatak 1.31 *Hanojske kule*: Data su tri vertikalna štapa, na jednom se nalazi n diskova poluprečnika 1,2,3,... do n, tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, a preostali štap se koristi kao pomoćni štap prilikom premeštanja.
Napisati program koji za proizvoljnu vrednost n, koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

Zadatak 1.32 *Modifikacija Hanojskih kula*: Data su četiri vertikalna štapa, na jednom se nalazi n diskova poluprečnika 1,2,3,... do n, tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala tri štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, premestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg, pri čemu se preostala dva štapa koriste kao pomoćni štapovi prilikom premeštanja.
Napisati program koji za proizvoljnu vrednost n, koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

1.4 Rešenja

Rešenje 1.1

Rešenje 1.2

Rešenje 1.3

Rešenje 1.4

Rešenje 1.5

Rešenje 1.6

Rešenje 1.7

Rešenje 1.8

Rešenje 1.9

Rešenje 1.10

Rešenje 1.11

Rešenje 1.12

Rešenje 1.13

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n"); /* Da li komentari rade ććžš*/
  return 0;
}
```

Rešenje 1.15

Rešenje 1.16

Rešenje 1.17

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n"); /* Da li komentari rade ććžš*/
  return 0;
}
```

Rešenje 1.18

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n"); /* Da li komentari rade ććžš*/
  return 0;
}
```

Glava 2

Pokazivači

2.1 Pokazivačka aritmetika

Zadatak 2.1 Milen: ovako definisan zadatak zahteva dva programa kao resenja, a ne jedan sa definisane dve funkcije. Za dati celobrojni niz dimenzije n, napisati funkciju koja obrće njegove elemente:

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju niza n ($0 < n \leq 100$), a zatim elemente niza. Prikazati sadržaj niza posle poziva funkcije za obrtanje elemenata niza.

Test 1

```
Ulaz:  3
       1 -2 3
Izlaz: 3 -2 1
```

Test 2

```
Ulaz:  0
Izlaz: Greska: neodgovarajuca dimenzija niza.
```

Zadatak 2.2 Dat je niz realnih brojeva dimenzije n.

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

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

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

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

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

Test 1

```
Ulaz: 3
      -1.1 2.2 3.3
Izlaz: zbir = 4.400
       proizvod = -7.986
       min = -1.100
       max = 3.300
```

Zadatak 2.3 Korišćenjem pokazivačke sintakse, napisati funkciju koja vrednosti elemenata u prvoj polovini niza povećava za jedan, a u drugoj polovini smanjuje za jedan. Ukoliko niz ima neparan broj elemenata, onda vrednost srednjeg elementa niza ostaviti nepromenjenim. Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati dimenziju n ($0 < n \leq 100$) celobrojong niza, a zatim i elemente niza. Na standardni izlaz ispisati rezultat primene napisane funkcije nad učitanim nizom. Jelena: Sta kazete na to da prekoracenja dimenzije niza u razlicitim zadacima razlicito obradjujemo. Na primer, mozemo da unosimo dimenziju niza sve dok se ne unese broj koji je u odgovarajucem opsegu, ili mozemo da dimenziju postavimo na 1 ako je korisnik uneo broj manji od 1, a na MAX ako je korisnik uneo broj veci od MAX, itd?

Test 1

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

Test 2

```
Ulaz: 4
      4 -3 2 -1
Izlaz: 5 -2 1 -2
```

Test 3

```
Ulaz: 0
Izlaz: Greska: neodgovarajuca dimenzija niza.
```

Test 4

```
Ulaz: 101
Izlaz: Greska: neodgovarajuca dimenzija niza.
```

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

(a) korišćenjem indeksne sintakse,

(b) korišćenjem pokazivačke sintakse.

Jelena: Da li je ok da ovaj zadatak pod a i b resim na nacin na koji sam resila, odnosno, da jedno od ta dva resenja iskomentarisem? Milena: Meni se cini da je bolje bez komentarisanja, vec da su oba prisutna.

Test 1

```
Poziv: ./a.out prvi 2. treci -4
Izlaz: 5
       0 ./a.out
       1 prvi
       2 2.
       3 treci
       4 -4
       . p 2 -
```

Test 2

```
Poziv: ./a.out
Izlaz: 1
       0 ./a.out
       .
```

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

Test 1

```
Poziv:  ./a.out programiranje anavolimilovana topot ana anagram t
Izlaz:  4
```

Test 2

```
Poziv: ./a.out a b 11 212
Izlaz: 4
```

Test 3

```
Poziv: ./a.out
Izlaz: 0
```

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

Test 1

```
Poziv:    ./a.out ulaz.txt 1
ulaz.txt: Ovo je sadrzaj datoteke i u njoj ima reci koje imaju
          1 karakter
Izlaz:    3
```

Test 2

```
Poziv:  ./a.out ulaz.txt
Izlaz:  Greska: Nedovoljan broj argumenata komandne linije.
        Program se poziva sa ./a.out ime_dat br_karaktera.
```

Test 3

```
Poziv:    ./a.out ulaz.txt 2
(ne postoji datoteka ulaz.txt)
Izlaz: Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

Zadatak 2.7 Napisati program koji kao prvi argument komandne linije prihvata

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

Test 1

```
Poziv:    ./a.out ulaz.txt ke -s
ulaz.txt: Ovo je sadrzaj datoteke i u njoj ima reci koje se
          zavrsavaju na ke
Izlaz:    2
```

Test 2

```
Poziv:    ./a.out ulaz.txt sa -p
ulaz.txt: Ovo je sadrzaj datoteke i u njoj ima reci koje
          pocinju sa sa
Izlaz:    3
```

Test 3

```
Poziv:  ./a.out ulaz.txt sa -p
(ne postoji datoteka ulaz.txt)
Izlaz:  Greska: Neuspesno otvaranje datoteke ulaz.txt.
```

Test 3

```
Poziv:  ./a.out ulaz.txt
Izlaz:  Greska: Nedovoljan broj argumenata komandne linije.
        Program se poziva sa ./a.out ime_dat suf/pref -s/-p.
```

2.2 Višedimenzioni nizovi

Zadatak 2.8 Data je kvadratna matrica dimenzije n.

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

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

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

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

Test 1

```
Ulaz:  3 1 -2 3 4 -5 6 7 -8 9
Izlaz: 1 -2 3
       4 -5 6
       7 -8 9
       trag = 5
       euklidska norma = 16.88
       vandijagonalna norma = 11
```

Test 2

```
Ulaz:  0
Izlaz: Greska: neodgovarajuca dimenzija matrice.
```

Zadatak 2.9 Date su dve kvadratne matrice istih dimenzija n.

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

(b) Napisati funkciju koja izračunava zbir matrica.

(c) Napisati funkciju koja izračunava proizvod matrica.

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

Test 1

```
Ulaz:   3
        1 2 3 1 2 3 1 2 3
        1 2 3 1 2 3 1 2 3
Izlaz:  da
        Zbir matrica je:
        2 4 6
        2 4 6
        2 4 6
        Proizvod matrica je:
        6 12 18
        6 12 18
        6 12 18
```

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

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

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

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

(d) Napisati funkciju koja određuje refleksivno zatvorenje relacije (najmanju refleksivnu relaciju koja sadrži datu).

(e) Napisati funkciju koja određuje simetrično zatvorenje relacije (najmanju simetričnu relaciju koja sadrži datu).

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

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

Test 1

```
Poziv: ./a.out ulaz.txt
ulaz.txt:  4
           1 0 0 0
           0 1 1 0
           0 0 1 0
           0 0 0 0
Izlaz:     Refleksivnost: ne
           Simetricnost: ne
           Tranzitivnost: da
           Refleksivno zatvorenje:
           1 0 0 0
           0 1 1 0
           0 0 1 0
           0 0 0 1
           Simetricno zatvorenje:
           1 0 0 0
           0 1 1 0
           0 1 1 0
           0 0 0 0
           Refleksivno-tranzitivno zatvorenje:
           1 0 0 0
           0 1 1 0
           0 0 1 0
           0 0 0 0
```

Zadatak 2.11 Data je kvadratna matrica dimenzije n.

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

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

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

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

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati elemente celobrojne kvadratne matrice čija se dimenzija n ($0 < n \leq 32$) zadaje kao argument komandne linije. Na standardni izlaz ispisati najveći element matrice na sporednoj dijagonali, indeks kolone koja sadrži najmanji element, indeks vrste koja sadrži najveći element i broj negativnih elemenata učitane matrice.

Test 1

```
Poziv:  ./a.out 3
Ulaz:   1 2 3
        -4 -5 -6
        7 8 9
Izlaz:  7 2 2 3
```

Test 2

```
Poziv:  ./a.out 4
Ulaz:   -1 -2 -3 -4
        -5 -6 -7 -8
        -9 -10 -11 -12
        -13 -14 -15 -16
Izlaz:  -4 3 0 16
```

Test 3

```
Poziv: ./a.out
Izlaz: Greska: Nedovoljan broj argumenata komandne linije.
       Program se poziva sa ./a.out dim_matrice.
```

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

Test 1

```
Ulaz:  4
       1 0 0 0
       0 1 0 0
       0 0 1 0
       0 0 0 1
Izlaz: da
```

Test 2

```
Ulaz:  3
       1 2 3
       5 6 7
       1 4 2
Izlaz: ne
```

Test 3

```
Ulaz:  33
Izlaz: Greska: neodgovarajuca dimenzija matrice.
```

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

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

(b) Napsiati funkciju koja na standardni izlaz spiralno ispisuje elemente matrice.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati dimenzije matrice n ($0 < n \leq 10$) i m ($0 < n \leq 10$), a zatim i elemente matrice

(pozivom gore napisane funkcije). Na standardni izlaz spiralno ispisati elemente učitane matrice.

Test 1

```
Ulaz:  3 3
       1 2 3
       4 5 6
       7 8 9
Izlaz: 1 2 3 6 9 8 7 4 5
```

Test 2

```
Ulaz:  3 4
       1 2 3 4
       5 6 7 8
       9 10 11 12
Izlaz: 1 2 3 4 8 12 11 10 9 5 6 7
```

Test 3

```
Ulaz:  11 4
Izlaz: Greska: neodgovarajuce dimenzije matrice.
```

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

Test 1

```
Ulaz:  3
       1 2 3
       4 5 6
       7 8 9
       8
Izlaz: 510008400 626654232 743300064
       1154967822 1419124617 1683281412
       1799927244 2211595002 2623262760
```

2.3 Dinamička alokacija memorije

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

Test 1

```
Ulaz:  3
       1 -2 3
Izlaz: 3 -2 1
```

Test 2

```
Ulaz:  -1
Izlaz: malloc(): neuspela alokacija memorije.
```

Zadatak 2.16 Napisati program koji sa standardnog ulaza učitava niz celih brojeva. Brojevi se unose sve dok se ne unese nula. Ne praviti nikakve pretpostavke

o dimenziji niza. Na standardni izlaz ispisati ovaj niz brojeva u obrnutom poretku. Zadatak uraditi na dva načina:

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

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

Test 1

```
Ulaz:  1 -2 3 -4 0
Izlaz: -4 3 -2 1
```

Test 2

```
Ulaz:  0
Izlaz:
```

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

Test 1

```
Ulaz:  Jedan Dva
Izlaz: JedanDva
```

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

Test 1

```
Ulaz:  2 3
       1.2 2.3 3.4
       4.5 5.6 6.7
Izlaz: 6.80
```

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

(a) Napisati funkciju koja vrši učitavanje matrice sa standardnog ulaza.

(b) Napisati funkciju koja ispisuje elemente ispod glavne dijagonale matrice (uključujući i glavnu dijagonalu).

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

Test 1

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

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

Test 1

```
Ulaz:  Unesite dimenzije matrice:
       2 3
       Unesite elemente matrice:
       1 2 3
       4 5 6
Izlaz: Kolona pod rednim brojem 3 ima najveci zbir.
```

Zadatak 2.21 Data je kvadratna realna matrica dimenzije n.

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

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

Test 1

```
Poziv: ./a.out matrica.txt
matrica.txt:  3
              1.1 -2.2 3.3
              -4.4 5.5 -6.6
              7.7 -8.8 9.9
Izlaz: Zbir apsolutnih vrednosti ispod sporedne dijagonale je 25.30.
       Transformisana matrica je:
       1.10 -1.10 1.65
       -8.80 5.50 -3.30
       15.40 -17.60 9.90
```

Zadatak 2.22 Petar sakuplja sličice igrača za predstojeće Svetsko prvenstvo u fudbalu. U datoteci „slicice.txt“ se nalaze informacije o sličicama koje mu nedostaju

u formatu: `redni_broj_sličice ime_reprezentacije_kojoj_sličica_pripada`. Pomozite Petru da otkrije koliko mu sličica ukupno nedostaje, kao i da pronađe ime reprezentacije čijih sličica ima najmanje. Dobijene podatke ispisati na standardni izlaz. Napomena: za realokaciju memorije koristiti `realloc()` funkciju. Jelena: treba dodati test primer.

Zadatak 2.23 U datoteci „temena.txt“ se nalaze tačke koje predstavljaju temena nekog n-tougla. Napisati program koji na osnovu sadržaja datoteke na standardni izlaz ispisuje o kom n-touglu je reč, a zatim i vrednosti njegovog obima i površine. Pretpostavka je da će mnogougao biti konveksan. Jelena: treba dodati test primer.

Zadatak 2.24 Napisati program koji na osnovu dve matrice dimenzija $m \times n$ formira matricu dimenzije $2 \cdot m \times n$ tako što naizmenično kombinuje jednu vrstu prve matrice i jednu vrstu druge matrice. Matirce su zapisane u datoteci „matrice.txt“. U prvom redu se nalaze dimenzije matrica m i n, u narednih m redova se nalaze vrste prve matrice, a u narednih m redova vrste druge matrice. Rezultujuću matricu ispisati na standardni izlaz. Jelena: treba dodati test primer.

Zadatak 2.25 Na ulazu se zadaje niz celih brojeva čiji se unos završava nulom. Napisati funkciju koja od zadatog niza formira matricu tako da prva vrsta odgovara unetom nizu, a svaka naredna se dobija cikličkim pomeranjem elemenata niza za jednu poziciju ulevo. Napisati program koji testira ovu funkciju. Sa standardnog ulaza se prvo unosi dimenzija matrice, a zatim redom elementi matrice. Rezultujuću matricu ispisati na standardni izlaz. Jelena: treba dodati test primer.

2.4 Pokazivači na funkcije

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

Test 1

```
Poziv: ./a.out sin
Ulaz: Unesite krajeve intervala:
      -0.5 1
      Koliko tacaka ima na ekvidistantnoj mrezi (ukljucujuci krajeve
   intervala)?
      4
Izlaz:      x         sin(x)
      ------------------------
      | -0.50000 | -0.47943 |
      |  0.00000 |  0.00000 |
      |  0.50000 |  0.47943 |
      |  1.00000 |  0.84147 |
      ------------------------
```

Test 2

```
Poziv: ./a.out cos
Ulaz: Unesite krajeve intervala:
      0 2
      Koliko tacaka ima na ekvidistantnoj mrezi (ukljucujuci krajeve
   intervala)?
      4
Izlaz:      x         cos(x)
       ------------------------
       |  0.00000 |  1.00000 |
       |  0.66667 |  0.78589 |
       |  1.33333 |  0.23524 |
       |  2.00000 | -0.41615 |
       ------------------------
```

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

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

Test 1

```
Ulaz:   tan 1.570795 10000
Izlaz:  -10134.5
```

Test 2

```
Ulaz:   log 0 1000000
Izlaz:  -13.81551
```

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

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

Vrednost h se izračunava po formuli $h = (b - a)/n$, dok se vrednosti n, a i b unose

sa standardnog ulaza kao i ime funkcije iz zaglavlja `math.h`. Na standardni izlaz ispisati vrednost integrala. Jelena: treba dodati test primer.

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

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

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

2.5 Rešenja

Rešenje 2.1

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

#define MAX 100

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

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

/* Funkcija obrce elemente niza koriscenjem pokazivacke
   sintakse. Umesto "void obrni_niz(int *a, int n)" potpis
   metode bi mogao da bude i "void obrni_niz(int a[], int n)".
   U oba slucaja se argument funkcije "a" tumaci kao pokazivac,
   ili tacnije, kao adresa prvog elementa niza. U odnosu na
   njega se odredjuju adrese ostalih elemenata u nizu */
void obrni_niz_v2(int *a, int n)
{
  /* Pokazivaci na elemente niza a */
  int *prvi, *poslednji;


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

```
    *prvi = *poslednji;
    *poslednji = t;
  }
}

/* Funkcija obrce elemente niza koriscenjem pokazivacke
   sintakse - modifikovano koriscenje pokazivaca */
void obrni_niz_v3(int *a, int n)
{
  /* Pokazivaci na elemente niza a */
  int *prvi, *poslednji;

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

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

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

int main()
{
  /* Deklaracija niza a od najvise MAX elemenata */
  int a[MAX];

  /* Broj elemenata niza a */
  int n;

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

  /* Unosimo dimenziju niza */
  scanf("%d", &n);

  /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
  if(n <= 0 || n > MAX) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
    exit(EXIT_FAILURE);
  }

  /* Unosimo elemente niza */
  for(p = a; p - a < n; p++)
    scanf("%d", p);

```

```
  obrni_niz_v1(a,n);
  // obrni_niz_v2(a,n);
  // obrni_niz_v3(a,n);

  /* Prikazujemo sadrzaj niza nakon obrtanja */
  for(p = a; p - a < n; p++)
    printf("%d ", *p);
  printf("\n");

  return 0;
}
```

Rešenje 2.2

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

#define MAX 100

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

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

  return s;
}

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

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

  return p;
}

/* Funkcija racuna najmanji element niza */
double min(double *a, int n)
{
  /* Za najmanji element se najpre postavlja prvi element */
  double min = a[0];
  int i;

  /* Ispitujemo da li se medju ostalim elementima niza
     nalazi najmanji */
  for(i = 1; i < n; i++)
    if ( a[i] < min )
      min = a[i];

   return min;
```

```
}

/* Funkcija racuna najveci element niza */
double max(double *a, int n)
{
  /* Za najveci element se najpre postavlja prvi element */
  double max = *a;

  /* Ispitujemo da li se medju ostalim elementima niza
     nalazi najveci */
  for(a++, n--; n > 0; a++, n--)
    if (*a > max)
      max = *a;

  return max;
}


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

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

  /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
  if(n <= 0 || n > MAX) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
    exit(EXIT_FAILURE);
  }

  /* Unosimo elemente niza */
  for(i = 0; i < n; i++)
    scanf("%lf", a + i);

  /* Testiramo definisane funkcije */
  printf("zbir = %5.3f\n", zbir(a, n));
  printf("proizvod = %5.3f\n", proizvod(a, n));
  printf("min = %5.3f\n", min(a, n));
  printf("max = %5.3f\n", max(a, n));

  return 0;
}
```

Rešenje 2.3

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

/* Funkcija povecava za jedan sve elemente u prvoj polovini niza
   a smanjuje za jedan sve elemente u drugoj polovini niza.
   Ukoliko niz ima neparan broj elemenata, srednji element
   ostaje nepromenjen */
```

```
void povecaj_smanji (int *a , int n)
{
  int *prvi = a;
  int *poslednji = a+n-1;

  while( prvi < poslednji ){

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

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

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

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

void povecaj_smanji_sazetije(int *a , int n)
{
  int *prvi = a;
  int *poslednji = a+n-1;

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

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

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

  /* Proveravamo da li je prekoraceno ogranicenje dimenzije */
  if(n <= 0 || n > MAX) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija niza.\n");
    exit(EXIT_FAILURE);
  }

  /* Unosimo elemente niza */
  for(p = a; p - a < n; p++)
    scanf("%d", p);

  povecaj_smanji(a,n);
  /*  povecaj_smanji_sazetije(a,n); */

```

```
  /* Prikaz niza nakon modifikacije */
  for(p = a; p - a < n; p++)
    printf("%d ", *p);
  printf("\n");

  return 0;
}
```

Rešenje 2.4

```
#include <stdio.h>

/* Argumenti funkcije main mogu da budu broj argumenta komandne
   linije (int argc) i niz arugmenata komandne linije
   (niz niski) (char *argv[]  <=>  char** argv) */
int main(int argc, char *argv[])
{
  int i;

  /* Ispisujemo broj argumenata komandne linije */
  printf("%d\n", argc);

  /* Ispisujemo argumente komandne linije */
  /* koristeci indeksnu sintaksu */
  for(i=0; i<argc; i++) {
    printf("%d %s\n", i, argv[i]);
  }

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


  /* Nakon ove petlje "argc" ce biti jednako nuli a "argv" ce
     pokazivati na polje u memoriji koje se nalazi iza
     poslednjeg argumenta komandne linije. Kako smo u
     promenljivoj "i" sacuvali vrednost broja argumenta
     komandne linije to sada mozemo ponovo da postavimo
     "argv" da pokazuje na nulti argument komandne linije */
  argv = argv - i;
  argc = i;

  /* Ispisujemo 0-ti karakter svakog od argumenata komandne linije
   */

  /* koristeci indeksnu sintaksu */
  for(i=0; i<argc; i++)
    printf("%c ", argv[i][0]);
  printf("\n");

  /* koristeci pokazivacku sintaksu */

  for (i=0 ; i<argc; i++ )
    printf("%c ", **argv++);
```

```
  return 0;
}
```

Rešenje 2.5

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

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

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

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

  printf("%d\n", n);
  return 0;
}
```

Rešenje 2.6

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

#define MAX_KARAKTERA 100

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

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

```
21    /* Ako korisnik nije uneo trazene argumente,
23       prijavljujemo gresku */
      if(argc < 3) {
25      printf("Greska: ");
      printf("Nedovoljan broj argumenata komandne linije.\n");
      printf("Program se poziva sa %s ime_dat br_karaktera.\n",
27                                                    argv[0]);
        exit(EXIT_FAILURE);
29    }

31    /* Otvaramo datoteku sa imenom koje se zadaje kao prvi
         argument komandne linije. */
33    in = fopen(*(argv+1), "r");
      if(in == NULL){
35      fprintf(stderr, "Greska: ");
        fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
37                                                argv[1]);
        exit(EXIT_FAILURE);
39    }

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

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

49    printf("%d\n", br);

51    /* Zatvaramo datoteku */
      fclose(in);
53    return 0;
}
```

Rešenje 2.7

```
1  #include<stdio.h>
   #include<stdlib.h>
3
   #define MAX_KARAKTERA 100
5
   /* Funkcija strcpy() iz standardne biblioteke */
7  void kopiranje_niske(char *dest, char *src)
   {
9    int i;
     for (i = 0; *(src+i); i++)
11     *(dest+i) = *(src+i);
   }
13
   /* Funkcija strcmp() iz standardne biblioteke */
15 int poredjenje_niski(char *s, char *t)
   {
17   int i;
```

```
  for (i = 0; *(s+i) == *(t+i); i++)
    if(*(s+i) == '\0')
      return 0;
  return *(s+i) - *(t+i);
}

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

/* Funkcija ispituje da li je niska zadata drugim argumentom
   funkcije sufiks niske zadate prvi argumentom funkcije */
int sufiks_niske(char *niska, char *sufiks) {
  if(duzina_niske(sufiks) <= duzina_niske(niska) &&
     poredjenje_niski(niska + duzina_niske(niska) -
                 duzina_niske(sufiks), sufiks) == 0)
    return 1;
  return 0;
}

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

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

  FILE *in;
  int br = 0, i = 0, n;
  char rec[MAX_KARAKTERA];

  in = fopen(*(argv+1), "r");
  if(in == NULL) {
```

```
    fprintf(stderr, "Greska: ");
    fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
                                                argv[1]);
    exit(EXIT_FAILURE);
  }

  /* Proveravamo kojom opcijom je pozvan program a zatim
     ucitavamo reci iz datoteke brojimo koliko reci
   zadovoljava trazeni uslov */
  if(!(poredjenje_niski(*(argv + 3), "-s")))
    while(fscanf(in, "%s", rec) != EOF)
      br += sufiks_niske(rec, *(argv+2));
  else if (!(poredjenje_niski(*(argv+3), "-p")))
    while(fscanf(in, "%s", rec) != EOF)
    br += prefiks_niske(rec, *(argv+2));

  printf("%d\n", br);
  fclose(in);
  return 0;
}
```

Rešenje 2.8

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

#define MAX 100

/* Deklarisemo funkcije koje cemo kasnije da definisemo */
double euklidska_norma( int M[][MAX], int n);
int trag(int M[][MAX], int n);
int gornja_vandijagonalna_norma(int M[][MAX], int n);

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

  /* Unosimo dimenziju kvadratne matrice */
  scanf("%d",&n);

  /* Proveravamo da li je prekoraceno ogranicenje */
  if( n > MAX || n <= 0) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
  fprintf(stderr, "matrice.\n");
    exit(EXIT_FAILURE);
  }

  /* Popunjavamo vrstu po vrstu matrice */
  for(i = 0; i<n; i++)
    for (j=0 ; j<n; j++)
      scanf("%d",&A[i][j]);

  /* Ispis elemenata matrice koriscenjem indeksne sintakse.
```

```
     Ispis vrsimo vrstu po vrstu */
  for(i = 0; i<n; i++) {
    /* Ispisujemo elemente i-te vrste */
    for ( j=0 ; j<n; j++)
      printf("%d ", A[i][j]);
    printf("\n");
  }

  /* Ispis elemenata matrice koriscenjem pokazivacke sintakse.
     Kod ovako definisane matrice, elementi su uzastopno
   smesteni u memoriju, kao na traci. To znaci da su svi
   elementi prve vrste redom smesteni jedan iza drugog. Odmah
   iza poslednjeg elementa prve vrste smesten je prvi element
   druge vrste za kojim slede svi elementi te vrste
   i tako dalje redom */
  /*
  for( i = 0; i<n; i++) {
    for ( j=0 ; j<n; j++)
      printf("%d ", *(*(A+i)+j));
    printf("\n");
  }
  */

  int tr = trag(A,n);
  printf("trag = %d\n",tr);

  printf("euklidska norma = %.2f\n",euklidska_norma(A,n));
  printf ("vandijagonalna norma = %d\n",
                      gornja_vandijagonalna_norma(A,n));

  return 0;
}

/* Definisemo funkcije koju smo ranije deklarisali */

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

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

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

  return sqrt(norma);
}
```

```
90 /* Funkcija izracunava gornju vandijagonalnu normu matrice */
   int gornja_vandijagonalna_norma(int M[][MAX], int n)
92 {
     int norma =0;
94   int i,j;

96   for(i=0 ;i<n; i++) {
       for(j = i+1; j<n; j++)
98       norma += abs(M[i][j]);
     }
100
     return norma;
102 }
```

Rešenje 2.9

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

4  #define MAX 100

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

12   for(i=0; i<n; i++)
       for(j=0; j<n; j++)
14       scanf("%d", &m[i][j]);
   }
16
   /* Funkcija ispisuje elemente kvadratne matrice dimenzije n na
18    standardni izlaz */
   void ispisi_matricu(int m[][MAX], int n) {
20   int i, j;

22   for(i=0; i<n; i++) {
       for(j=0; j<n; j++)
24       printf("%d ", m[i][j]);
       printf("\n");
26   }
   }
28
   /* Funkcija proverava da li su zadate kvadratne matrice a i b
30    dimenzije n jednake */
   int jednake_matrice(int a[][MAX], int b[][MAX], int n) {
32   int i, j;

34   for(i=0; i<n; i++)
       for(j=0; j<n; j++)
36     /* Nasli smo elemente na istim pozicijama u matricama
          koji se razlikuju */
38       if(a[i][j]!=b[i][j])
```

```
      return 0;

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

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

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

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

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

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

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

  /* Dimenzija matrica */
  int n;
  int i, j;

  /* Ucitavamo dimenziju kvadratnih matrica i proveravamo njenu
     korektnost */
  scanf("%d", &n);

  /* Proveravamo da li je prekoraceno ogranicenje */
  if( n > MAX || n <= 0) {
    fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
  fprintf(stderr, "matrica.\n");
    exit(EXIT_FAILURE);
  }

  /* Ucitavamo matrice */
  ucitaj_matricu(a, n);
```

```
  ucitaj_matricu(b, n);

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

  /* Ispisujemo rezultat */
  if(jednake_matrice(a, b, n) == 1)
    printf("da\n");
  else
    printf("ne\n");

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

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

  return 0;
}
```

Rešenje 2.10

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

#define MAX 64

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

  /* Obilazimo glavnu dijagonalu matrice. Za elemente na glavnoj
     dijagonali vazi da je indeks vrste jednak indeksu kolone */
  for(i=0; i<n; i++) {
    if(m[i][i] != 1)
      return 0;
  }

  return 1;
}

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

  /* Prepisujemo vrednosti elemenata matrice pocetne matrice */
  for(i=0; i<n; i++)
```

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

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

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

  /* Obilazimo elemente ispod glavne dijagonale matrice i
     uporedjujemo ih sa njima simetricnim elementima */
  for(i=0; i<n; i++)
    for(j=0; j<i; j++)
      if(m[i][j] != m[j][i])
      return 0;

  return 1;
}

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

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

  /* Odredjujemo simetricno zatvorenje matrice */
  for(i=0; i<n; i++)
    for(j=0; j<n; j++)
      if(zatvorenje[i][j] == 1)
      zatvorenje[j][i] = 1;
}


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

```
90    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
92        /* Pokusavamo da pronadjemo element koji narusava
         * tranzitivnost */
94        for(k=0; k<n; k++)
          if(m[i][k] == 1 && m[k][j] == 1 && m[i][j] == 0)
96          return 0;

98    return 1;
}
100

102 /* Funkcija odredjuje refleksivno-tranzitivno zatvorenje
       zadate relacije koriscenjem Varsalovog algoritma */
104 void tran_zatvorenje(int m[][MAX], int n, int zatvorenje[][MAX])
{
106   int i, j, k;

108   /* Kopiramo pocetnu matricu u matricu rezultata */
      for(i=0; i<n; i++)
110       for(j=0; j<n; j++)
          zatvorenje[i][j] = m[i][j];
112
      /* Primenom Varsalovog algoritma odredjujemo
114      refleksivno-tranzitivno zatvorenje matrice */
      for(k=0; k<n; k++)
116     for(i=0; i<n; i++)
          for(j=0; j<n; j++)
118         if((zatvorenje[i][k] == 1) && (zatvorenje[k][j] ==1)
                                    && (zatvorenje[i][j] == 0))
120       zatvorenje[i][j] = 1;
}
122
/* Funkcija ispisuje elemente matrice */
124 void pisi_matricu(int m[][MAX], int n)
{
126   int i, j;

128   for(i=0; i<n; i++) {
        for(j=0; j<n; j++)
130       printf("%d ", m[i][j]);
        printf("\n");
132   }
}
134
int main(int argc, char* argv[])
136 {
      FILE* ulaz;
138   int m[MAX][MAX];
      int pomocna[MAX][MAX];
140   int n, i, j, k;

142  /* Ako korisnik nije uneo trazene argumente,
         prijavljujemo gresku */
144   if(argc < 2) {
```

```
    printf("Greska: ");
146 printf("Nedovoljan broj argumenata komandne linije.\n");
    printf("Program se poziva sa %s ime_dat.\n", argv[0]);
148   exit(EXIT_FAILURE);
    }
150
    /* Otvaramo datoteku za citanje */
152 ulaz = fopen(argv[1], "r");
    if(ulaz == NULL) {
154   fprintf(stderr, "Greska: ");
      fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
156                                              argv[1]);
      exit(EXIT_FAILURE);
158 }

160 /* Ucitavamo dimenziju matrice */
    fscanf(ulaz, "%d", &n);
162
    /* Proveravamo da li je prekoraceno ogranicenje */
164 if( n > MAX || n <= 0) {
      fprintf(stderr, "Greska: neodgovarajuca dimenzija ");
166   fprintf(stderr, "matrice.\n");
      exit(EXIT_FAILURE);
168 }

170 /* Ucitavamo element po element matrice */
    for(i=0; i<n; i++)
172   for(j=0; j<n; j++)
        fscanf(ulaz, "%d", &m[i][j]);
174
    /* Ispisujemo trazene vrednosti */
176 printf("Refleksivnost: %s\n",
            refleksivnost(m, n) == 1 ? "da" : "ne");
178
    printf("Simetricnost: %s\n",
180             simetricnost(m, n) == 1 ? "da" : "ne");

182 printf("Tranzitivnost: %s\n",
            tranzitivnost(m, n) == 1 ? "da" : "ne");
184
    printf("Refleksivno zatvorenje:\n");
186 ref_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
188
    printf("Simetricno zatvorenje:\n");
190 sim_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
192
    printf("Refleksivno-tranzitivno zatvorenje:\n");
194 tran_zatvorenje(m, n, pomocna);
    pisi_matricu(pomocna, n);
196
    /* Zatvaramo datoteku */
198 fclose(ulaz);

200 return 0;
```

```
}
```

Rešenje 2.11

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

#define MAX 32

int max_sporedna_dijagonala(int m[][MAX], int n)
{
  int i, j;
  /* Trazimo najveci element na sporednoj dijagonali. Za
     elemente sporedne dijagonale vazi da je zbir indeksa vrste
   i indeksa kolone jednak n-1. Za pocetnu vrednost maksimuma
   uzimamo element u gornjem desnom uglu */
  int max_na_sporednoj_dijagonali = m[0][n-1];
  for(i=1; i<n; i++)
    if(m[i][n-1-i] > max_na_sporednoj_dijagonali)
      max_na_sporednoj_dijagonali = m[i][n-1-i];

  return max_na_sporednoj_dijagonali;
}

/* Funkcija izracunava indeks kolone najmanjeg elementa */
int indeks_min(int m[][MAX], int n)
{
  int i, j;
  /* Za pocetnu vrednost minimuma uzimamo element u gornjem
     levom uglu */
  int min=m[0][0], indeks_kolone=0;

  for(i=0; i<n; i++)
    for(j=0; j<n; j++)
      /* Ako je tekuci element manji od minimalnog */
      if(m[i][j]<min) {
      /* cuvamo njegovu vrednost */
      min=m[i][j];
      /* i cuvamo indeks kolone u kojoj se nalazi */
      indeks_kolone=j;
      }
  return indeks_kolone;
}

/* Funkcija izracunava indeks vrste najveceg elementa */
int indeks_max(int m[][MAX], int n) {
  int i, j;
  /* Za maksimalni element uzimamo gornji levi ugao */
  int max=m[0][0], indeks_vrste=0;

  for(i=0; i<n; i++)
    for(j=0; j<n; j++)
      /* Ako je tekuci element manji od minimalnog */
      if(m[i][j]>max) {
      /* cuvamo njegovu vrednost */
```

```
      max=m[i][j];
      /* i cuvamo indeks vrste u kojoj se nalazi */
      indeks_vrste=i;
      }
  return indeks_vrste;
}

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

  int broj_negativnih=0;

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

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

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

  /* Ucitavamo vrednost dimenzije i proveravamo njenu
     korektnost */
  n = atoi(argv[1]);

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

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

  int max_sd = max_sporedna_dijagonala(m, n);
  int i_min = indeks_min(m, n);
  int i_max = indeks_max(m, n);
  int bn = broj_negativnih(m, n);

  /* Ispisujemo rezultat */
  printf("%d %d %d %d\n", max_sd, i_min, i_max, bn);
```

```
109   /* Prekidamo izvrsavanje programa */
      return 0;
111 }
```

Rešenje 2.12

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

#define MAX 32

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

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

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

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

/* Funkcija proverava da li je zadata matrica ortonormirana */
int ortonormirana(int m[][MAX], int n)
{
  int i, j, k;
  int proizvod;

  /* Proveravamo uslov normiranosti, odnosno da li je proizvod
     svake vrste matrice sa samom sobom jednak jedinici */
  for(i=0; i<n; i++) {

    /* Izracunavamo skalarni proizvod vrste sa samom sobom */
    proizvod = 0;

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

    /* Ako proizvod bar jedne vrste nije jednak jedinici, odmah
       zakljucujemo da matrica nije normirana */
    if(proizvod!=1)
```

```
      return 0;
  }

  /* Proveravamo uslov ortogonalnosti, odnosno da li je proizvod
     dve bilo koje razlicite vrste matrice jednak nuli */
  for(i=0; i<n-1; i++) {
    for(j=i+1;j<n; j++) {

      /* Izracunavamo skalarni proizvod */
      proizvod = 0;

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

      /* Ako proizvod dve bilo koje razlicite vrste nije jednak
         nuli, odmah zakljucujemo da matrica nije ortogonalna */
      if(proizvod!=0)
      return 0;
    }
  }

  /* Ako su oba uslova ispunjena, vracamo jedinicu kao
     rezultat */
  return 1;
}

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

  /* Ucitavamo vrednost dimenzije i proveravamo njenu
     korektnost */
  scanf("%d", &n);

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

  /* Ucitavamo matricu */
  ucitaj_matricu(A, n);

  /* Ispisujemo rezultat rada funkcije */
  if(ortonormirana(A,n))
    printf("da\n");
  else
    printf("ne\n");

  return 0;
}
```

Rešenje 2.13

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

#define MAX_V 10
#define MAX_K 10

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

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

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

  while( !krajIspisa(top, bottom, left, right) ) {
    /* Ispisuje se prvi red*/
    for(j=left; j<=right; j++)
      printf("%d ",a[top][j]);

    /* Spustamo prvi red */
    top++;

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

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

    /* Pomeramo desnu kolonu za naredni krug ispisa
       blize levom kraju */
    right--;

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

    /* Ispisujemo donju vrstu */
    for(j=right; j>=left; j-- )
      printf("%d ",a[bottom][j]);

    /* Podizemo donju vrstu za naredni krug ispisa */
    bottom--;

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

    /* Ispisujemo prvu kolonu*/
    for(i=bottom; i>=top; i-- )
```

```
      printf("%d ",a[i][left]);

    /* Pripremamo levu kolonu za naredni krug ispisa */
    left++;
  }
  putchar('\n');
}

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

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

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

  /* Ucitaj broj vrsta i broj kolona matrice */
  scanf("%d",&n);
  scanf("%d", &m);

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

  ucitaj_matricu(a, n, m);
  ispisi_matricu_spiralno(a, n, m);

  return 0;
}
```

Rešenje 2.14

Rešenje 2.15

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

/* NAPOMENA: Primer demonstrira dinamicku alokaciju niza od n
   elemenata. Dovoljno je alocirati n * sizeof(T) bajtova, gde
   je T tip elemenata niza. Povratnu adresu malloc()-a treba
   pretvoriti iz void * u T *, kako bismo dobili pokazivac
   koji pokazuje na prvi element niza tipa T. Na dalje se
   elementima moze pristupati na isti nacin kao da nam
   je dato ime niza (koje se tako i ponasa - kao pokazivac
   na element tipa T koji je prvi u nizu) */
int main()
```

```
{
  int *p = NULL;
  int i, n;

  /* Unosimo dimenziju niza. Ova vrednost nije ogranicena
     bilo kakvom konstantom, kao sto je to ranije bio slucaj
     kod staticke alokacije gde je dimenzija niza bila unapred
     ogranicena definisanim prostorom. */
  scanf("%d", &n);

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

  /* Od ovog trenutka pokazivac "p" mozemo da koristimo kao da
     je ime niza, odnosno i-tom elementu se moze pristupiti
     sa p[i] */

  /* Unosimo elemente niza */
  for (i = 0; i < n; i++)
    scanf("%d", &p[i]);

  /* Ispisujemo elemente niza unazad */
  for (i = n - 1; i >= 0; i--)
    printf("%d ", p[i]);
  printf("\n");

  /* Oslobadjamo prostor */
  free(p);

  return 0;
}
```

Rešenje 2.16

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

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

  /* Velicina alocirane memorije  */
  int alocirano;

  /* Broj elemenata niza */
  int n;

  /* Broj koji se ucitava sa ulaza */
  int x;
```

```
18  int i;
    int* b = NULL;
20
    /* Inicijalizacija */
22  alocirano = n = 0;

24  /* Unosimo brojeve sa ulaza */
    scanf("%d", &x);
26
    /* Sve dok je procitani broj razlicit od nule... */
28  while(x!=0) {

30    /* Ako broj ucitanih elemenata niza odgovara broju
         alociranih mesta, za smestanje novog elementa treba
32       obezbediti dodatni prostor. Da se ne bi za svaki sledeci
         element pojedinacno alocirala memorija, prilikom
34       alokacije se vrsi rezervacija za jos KORAK dodatnih
         mesta za buduce elemente */
36    if(n == alocirano) {
        /* Povecava se broj alociranih mesta */
38      alocirano = alocirano + KORAK;

40      /* Vrsi se realokacija memorije sa novom velicinom */
        /*******************************************************/
42      /* Resenje sa funkcijom malloc() */
        /*******************************************************/
44      /* Vrsi se alokacija memorije sa novom velicinom, a adresa
           pocetka novog memorijskog bloka se cuva u
46         promenljivoj b */
        b = (int*) malloc (alocirano * sizeof(int));
48
        /* Ako prilikom alokacije dodje do neke greske */
50      if(b == NULL) {
          /* poruku ispisujemo na izlaz za greske */
52        fprintf(stderr, "malloc(): ");
          fprintf(stderr, "greska pri alokaciji memorije.\n");
54
          /* Pre kraja programa moramo svu dinamicki alociranu
56           memoriju da oslobodimo. U ovom slucaju samo memoriju
             na adresi a */
58        free(a);

60        /* Zavrsavamo program */
          exit(EXIT_FAILURE);
62      }

64      /* Svih n elemenata koji pocinju na adresi a prepisujemo
           na novu aderesu b */
66      for(i = 0; i < n; i++)
          b[i] = a[i];
68
        /* Posle prepisivanja oslobadjamo blok memorije sa pocetnom
70         adresom u a */
        free(a);
72
        /* Promenljivoj a dodeljujemo adresu pocetka novog, veceg
```

```
74          bloka koji je prilikom alokacije zapamcen u
            promenljivoj b */
76        a = b;

78        /*********************************************************/
          /* Resenje sa funkcijom realloc() */
80        /*********************************************************/
          /* Zbog funkcije realloc je neophodno da i u prvoj
82           iteraciji "a" bude inicijalizovano na NULL */
          /*
84        a = (int*) realloc(a,alocirano*sizeof(int));

86        if(a == NULL) {
            fprintf(stderr, "realloc(): ");
88          fprintf(stderr, "greska pri alokaciji memorije.\n");
            exit(EXIT_FAILURE);
90        }
          */
92      }

94      /* Smestamo element u niz */
        a[n++] = x;
96
        /* i ucitavamo sledeci element */
98      scanf("%d", &x);
    }
100
    /* Ispisujemo brojeve u obrnutom poretku */
102 for(n--; n>=0; n--)
      printf("%d ", a[n]);
104 printf("\n");

106 /* Oslobadjamo dinamicki alociranu memoriju */
    free(a);
108
    /* Program se zavrsava */
110 exit(EXIT_SUCCESS);
}
```

Rešenje 2.17

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4
#define MAX 1000
6
/* NAPOMENA: Primer demonstrira "vracanje nizova iz funkcije".
8   Ovako nesto se moze improvizovati tako sto se u funkciji
    dinamicki kreira niz potrebne velicine, popuni se potrebnim
10  informacijama, a zatim se vrati njegova adresa. Imajuci u
    vidu cinjenicu da dinamicki kreiran objekat ne nestaje
12  kada se izadje iz funkcije koja ga je kreirala, vraceni
    pokazivac se kasnije u pozivajucoj funkciji moze koristiti
14  za pristup "vracenom" nizu. Medjutim, pozivajuca funkcija
```

```
   ima odgovornost i da se brine o dealokaciji istog prostora */

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

  /* Proveravamo uspeh alokacije */
  if (p == NULL) {
    fprintf(stderr, "malloc(): ");
    fprintf(stderr, "greska pri alokaciji memorije.\n");
    exit(EXIT_FAILURE);
  }

  /* Kopiramo i nadovezujemo stringove */

  /* Resenje bez koriscenja biblioteckih funkcija */
  /*
  int i,j;
  for(i=j=0; s[j]!='\0'; i++, j++)
    p[i]=s[j];

  for(j=0; t[j]!='\0'; i++, j++)
    p[i]=t[j];

  p[i]='\0';
  */

  /* Resenje sa koriscenjem biblioteckih funkcija iz zaglavlja
     string.h */
  strcpy(p, s);
  strcat(p, t);

  /* Vracamo pokazivac p */
  return p;
}

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

  /* Ucitavamo dve niske koje cemo da nadovezemo */
  scanf("%s", s1);
  scanf("%s", s2);

  /* Pozivamo funkciju da nadoveze stringove */
  s = nadovezi(s1, s2);

  /* Prikazujemo rezultat */
  printf("%s\n", s);

  /* Oslobadjamo memoriju alociranu u funkciji nadovezi() */
  free(s);
```

```
72    return 0;
}
```

Rešenje 2.18

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

int main()
{
  int i,j;

  /* Pokazivac na dinamicki alociran niz pokazivaca na vrste
     matrice */
  double** A  = NULL;

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

   /* Trag matice */
  double trag = 0;

  /* Unosimo dimenzije matrice*/
  scanf("%d%d", &n, &m);

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

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

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

    if(A[i] == NULL) {
      /* Alokacija je neuspesna. Pre zavrsetka programa
         moramo da oslobodimo svih i-1 prethodno alociranih
         vrsta, i alociran niz pokazivaca */
        for( j=0; j<i; j++)
          free(A[j]);
        free(A);

        exit( EXIT_FAILURE);
    }
  }

  /* Unosimo sa standardnog ulaza brojeve u matricu.
     Popunjavamo vrstu po vrstu */
```

```
  for(i = 0; i < n; i++)
    for(j = 0; j < m; j++ )
      scanf("%lf", &A[i][j]);

  /* Racunamo trag matrice, odnosno sumu elemenata
     na glavnoj dijagonali */
  trag = 0.0;

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

  printf("%.2f\n", trag);

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

  /* Oslobadjamo memoriju za niz pokazivaca na vrste */
  free(A);

  return 0;
}
```

Rešenje 2.19

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

void ucitaj_matricu(int ** M, int n, int m)
{
  int i, j;

  /* Popunjavamo matricu vrstu po vrstu */
  for(i=0; i<n; i++)
    /* Popunjavamo i-tu vrstu matrice */
    for(j=0; j<m; j++)
      scanf("%d", &M[i][j]);
}

void ispisi_elemente_ispod_dijagonale(int ** M, int n, int m)
{
  int i, j;

  for(i=0; i<n; i++) {
    /* Ispisujemo elemente ispod glavne dijagonale matrice */
    for(j=0; j<=i; j++)
      printf("%d ", M[i][j]);
    printf("\n");
  }
}

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

```
31
   /* Unosimo dimenzije matrice */
33 scanf("%d %d",&n, &m);

35 /* Alociramo prostor za niz pokazivaca na vrste matrice */
   matrica = (int **) malloc(n * sizeof(int*));
37 if(matrica == NULL) {
     fprintf(stderr,"malloc(): Neuspela alokacija\n");
39   exit(EXIT_FAILURE);
   }
41
   /* Alociramo prostor za svaku vrstu matrice */
43 for(i=0; i<n; i++) {
     matrica[i] = (int*) malloc(m * sizeof(int));
45
     if(matrica[i] == NULL) {
47     fprintf(stderr,"malloc(): Neuspela alokacija\n");
       for(j=0; j<i; j++)
49       free(matrica[j]);
       free(matrica);
51     exit(EXIT_FAILURE);
     }
53 }

55 ucitaj_matricu(matrica, n, m);

57 ispisi_elemente_ispod_dijagonale(matrica, n, m);

59 /* Oslobadjamo dinamicki alociranu memoriju za matricu.
      Prvo oslobadjamo prostor rezervisan za svaku vrstu */
61 for( j=0; j<n; j++)
     free(matrica[j]);
63
   /* Zatim oslobadjamo memoriju za niz pokazivaca na vrste
65    matrice */
   free(matrica);
67
   return 0;
69 }
```

Rešenje 2.20

```
#include<stdio.h>
2
int main(){
4   printf("Hello pokazivaci!\n");
    return 0;
6 }
```

Rešenje 2.21

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

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

10  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
12  /* Ako je indeks vrste manji od indeksa kolone */
    if(i<j)
14    /* element se nalazi iznad glavne dijagonale
         pa ga polovimo */
16      a[i][j]/=2;
    else
18    /* Ako je indeks vrste veci od indeksa kolone */
      if(i>j)
20      /* element se nalazi ispod glavne dijagonale
          pa ga dupliramo*/
22      a[i][j] *= 2;
}
24
/* Funkcija izracunava zbir apsolutnih vrednosti elemenata ispod
26    sporedne dijagonale */
float zbir_ispod_sporedne_dijagonale(float** m, int n)
28 {
    int i, j;
30  float zbir=0;

32  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
34      /* Ukoliko je zbir indeksa vrste i indeksa kolone
           elementa veci od n-1, to znaci da se element nalazi
36         ispod sporedne dijagonale */
        if(i+j>n-1)
38      zbir+=fabs(m[i][j]);

40  return zbir;
}
42
/* Funkcija ucitava elemente kvadratne matrice dimenzije n
44    iz zadate datoteke */
void ucitaj_matricu(FILE* ulaz, float** m, int n) {
46  int i, j;

48  for(i=0; i<n; i++)
      for(j=0; j<n; j++)
50      fscanf(ulaz, "%f", &m[i][j]);
}
52
/* Funkcija ispisuje elemente kvadratne matrice dimenzije n
54    na standardni izlaz */
void ispisi_matricu(float** m, int n) {
56  int i, j;

58  for(i=0; i<n; i++){
      for(j=0; j<n; j++)
```

```
60        printf("%.2f ", m[i][j]);
      printf("\n");
62    }
}
64
/* Funkcija alocira memoriju za kvadratnu matricu dimenzije n */
66 float** alociraj_memoriju(int n) {
    int i, j;
68  float** m;

70  m = (float**) malloc(n * sizeof(float*));
    if(m == NULL) {
72    fprintf(stderr,"malloc(): Neuspela alokacija\n");
      exit(EXIT_FAILURE);
74  }

76  /* Za svaku vrstu matrice */
    for(i=0; i<n; i++) {
78    /* Alociramo memoriju */
      m[i] = (float*) malloc(n * sizeof(float));
80
      /* Proveravamo da li je doslo do greske pri alokaciji */
82    if(m[i] == NULL) {
        /* Ako jeste, ispisujemo poruku */
84      printf("malloc(): neuspela alokacija memorije!\n");

86      /* Oslobadjamo memoriju zauzetu do ovog koraka */
        for(j=0; j<i; j++)
88      free(m[i]);
        free(m);
90      exit(EXIT_FAILURE);
      }
92  }
    return m;
94 }

96 /* Funckija oslobadja memoriju zauzetu kvadratnom matricom
      dimenzije n */
98 void oslobodi_memoriju(float** m, int n)
{
100   int i;

102   for(i=0; i<n; i++)
        free(m[i]);
104   free(m);
}
106
int main(int argc, char* argv[])
108 {
    FILE* ulaz;
110 float** a;
    int n;
112
 /* Ako korisnik nije uneo trazene argumente,
114     prijavljujemo gresku */
    if(argc < 2) {
```

```
116     printf("Greska: ");
        printf("Nedovoljan broj argumenata komandne linije.\n");
118     printf("Program se poziva sa %s ime_dat.\n", argv[0]);
        exit(EXIT_FAILURE);
120   }

122   /* Otvaramo datoteku za citanje */
      ulaz = fopen(argv[1], "r");
124   if(ulaz == NULL) {
        fprintf(stderr, "Greska: ");
126     fprintf(stderr, "Neuspesno otvaranje datoteke %s.\n",
                                                argv[1]);
128     exit(EXIT_FAILURE);
      }
130
      /* citamo dimenziju matrice */
132   fscanf(ulaz, "%d", &n);

134   /* Alociramo memoriju */
      a = alociraj_memoriju(n);
136
      /* Ucitavamo elemente matrice */
138   ucitaj_matricu(ulaz, a, n);

140   float zbir = zbir_ispod_sporedne_dijagonale(a, n);

142   /* Pozivamo funkciju za modifikovanje elemenata */
      izmeni(a, n);
144
      /* Ispisujemo rezultat */
146   printf("Zbir apsolutnih vrednosti ispod sporedne dijagonale ");
      printf("je %.2f.\n", zbir);
148
      printf("Transformisana matrica je:\n");
150   ispisi_matricu(a,n);

152   /* Oslobadjamo memoriju */
      oslobodi_memoriju(a, n);
154
      /* Zatvaramo datoteku */
156   fclose(ulaz);

158   /* i prekidamo sa izvrsavanjem programa */
      return 0;
160 }
```

Rešenje 2.22

Rešenje 2.23

Rešenje 2.24

Rešenje 2.25

Rešenje 2.26

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

/* NAPOMENA:
   Zaglavlje math.h sadrzi deklaracije raznih matematickih
   funkcija. đIzmeu ostalog, to su ćsledee funkcije:
   double sin(double x);
   double cos(double x);
   double tan(double x);
   double asin(double x);
   double acos(double x);
   double atan(double x);
   double atan2(double y, double x);
   double sinh(double x);
   double cosh(double x);
   double tanh(double x);
   double exp(double x);
   double log(double x);
   double log10(double x);
   double pow(double x, double y);
   double sqrt(double x);
   double ceil(double x);
   double floor(double x);
   double fabs(double x);
*/

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

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

/* Umesto da koristimo stepenu funkciju iz zaglavlja
   math.h -> pow(a,2) ćpozivaemo čkorisniku sqr(a) */
double sqr (double a)
{
    return a*a;
}

```

```
int main(int argc, char *argv[])
{
  double a, b;
  int n;
  /* Imena funkicja koja ćemo navoditi su ćkraa ili čtano duga
     5 karaktera */
  char ime_fje[6];
  /* Pokazivac na funkciju koja ima jedan argument tipa double i
     povratnu vrednost istog tipa */
  double (*fp)(double);

   /* Ako korisnik nije uneo žtraene argumente,
     prijavljujemo šgreku */
  if(argc < 2) {
    printf("Greska: ");
  printf("Nedovoljan broj argumenata komandne linije.\n");
  printf("Program se poziva sa %s ime_funkcije iz math.h.\n",
                                                    argv[0]);
    exit(EXIT_FAILURE);
  }

  /* Niska ime_fje žsadri ime žtraene funkcije koja je navedena
     u komandnoj liniji */
  strcpy(ime_fje, argv[1]);

  /* Inicijalizujemo čpokaziva na funkciju koja treba da se
     tabelira */
  if(strcmp(ime_fje, "sin") == 0)
    fp=&sin;
  else if(strcmp(ime_fje, "cos") == 0)
    fp=&cos;
  else if(strcmp(ime_fje, "tan") == 0)
    fp=&tan;
  else if(strcmp(ime_fje, "atan") == 0)
    fp=&atan;
  else if(strcmp(ime_fje, "acos") == 0)
    fp=&acos;
  else if(strcmp(ime_fje, "asin") == 0)
    fp=&asin;
  else if(strcmp(ime_fje, "exp") == 0)
    fp=&exp;
  else if(strcmp(ime_fje, "log") == 0)
    fp=&log;
  else if(strcmp(ime_fje, "log10") == 0)
    fp=&log10;
  else if(strcmp(ime_fje, "sqrt") == 0)
    fp=&sqrt;
  else if(strcmp(ime_fje, "floor") == 0)
    fp=&floor;
  else if(strcmp(ime_fje, "ceil") == 0)
    fp=&ceil;
  else if(strcmp(ime_fje, "sqr") == 0)
    fp=&sqr;
  else {
    printf("Program jos uvek ne podrzava trazenu funkciju!\n");
    exit(EXIT_SUCCESS);
```

```
  }

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

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

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

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

  /* đProsleujemo funkciji tabela() funkciju zadatu kao
     argument komandne linije */
  tabela(a, b, n, fp);

  exit(EXIT_SUCCESS);
}
```

Rešenje 2.27

Rešenje 2.28

Rešenje 2.29

Glava 3

Algoritmi pretrage i sortiranja

3.1 Pretraživanje

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

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

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

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

Napisati i program koji generiše slučajni rastući niz dimenzije `n` (prvi argument komandne linije), i u njemu već napisanim funkcijama traži element `x` (drugi argument komandne linije). Potrebna vremena za izvršavanje ovih funkcija upisati u fajl `vremena.txt`.

Test 1

```
Poziv: ./a.out 1000000 235423
Izlaz:
Linearna pretraga
Element nije u nizu
---------------------------------
Binarna pretraga
Element nije u nizu
---------------------------------
Interpolaciona pretraga
Element nije u nizu
---------------------------------
```

Zadatak 3.2 Napisati rekurzivne funkcije algoritama linearne, binarne i interpolacione pretrage i program koji ih testira za brojeve koji se unose sa standardnog

ulaza. Pretpostaviti da niz brojeva koji se unosi neće biti duži od 1024 elemenata. Prvo se unosi broj koji se traži, a zatim sortirani elementi niza sve do kraja ulaza.

Test 1

```
Ulaz: 11 2 5 6 8 10 11 23
Izlaz:
  Linearna pretraga
  Pozicija elementa je 5.
  Binarna pretraga
  Pozicija elementa je 5.
  Interpolaciona pretraga
  Pozicija elementa je 5.
```

Test 2

```
Ulaz: 14 10 32 35 43 66 89 100
Izlaz:
  Linearna pretraga
  Element se ne nalazi u nizu.
  Binarna pretraga
  Element se ne nalazi u nizu.
  Interpolaciona pretraga
  Element se ne nalazi u nizu.
```

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

Test 1

```
Datoteka:
  20140003 Marina Petrovic
  20140012 Stefan Mitrovic
  20140032 Dejan Popovic
  20140049 Mirko Brankovic
  20140076 Sonja Stevanovic
  20140104 Ivan Popovic
  20140187 Vlada Stankovic
  20140234 Darko Brankovic
Ulaz:
  20140076
  Popovic
Izlaz:
  Indeks: 20140076, Ime i prezime: Sonja Stevanovic
  Indeks: 20140032, Ime i prezime: Dejan Popovic
```

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

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

Test 1

```
Poziv: ./a.out dat.txt -x
Datoteka:
  12 53
  2.342 34.1
  -0.3 23
  -1 23.1
  123.5 756.12
Izlaz: -0.3 23
```

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

Test 1

```
Izlaz: 1.57031
```

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

Test 1

```
Ulaz:  -151 -44 5
       12 13 15
Izlaz: 2
```

Test 2

```
Ulaz:  -100 -15 -11
       -8 -7 -5
Izlaz: -1
```

Test 3

```
Ulaz:  -100 -15 0 13
       155 124 258
       315 516 7000
Izlaz: 3
```

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

Test 1

```
Ulaz:  151 44 5 -12 -13 -15
Izlaz: 3
```

Test 2

```
Ulaz:  100 55 15 0 -15 -124 -155
       -258 -315 -516 -7000
Izlaz: 4
```

Test 3

```
Ulaz:  100 15 11 8 7 5 4 3 2
Izlaz: -1
```

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

(a) Napisati funkciju, linearne složenosti, koja određuje logaritam pomeranjem broja udesno dok ne postane 0.

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

Tražene funkcije testirati programom koji broj učitava sa standardnog ulaza, a logaritam ispisuje na standardni izlaz.

Test 1

```
Ulaz:  10
Izlaz: 3 3
```

Test 2

```
Ulaz:  4
Izlaz: 2 2
```

Test 3

```
Ulaz:  17
Izlaz: 4 3
```

Test 4

```
Ulaz:  1031
Izlaz: 10 10
```

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

Test 1

```
Ulaz:
  2
  2 0 2 1
  1 2 2 2
Izlaz: 26.57
```

Zadatak 3.11 Napisati program u kome se prvo inicijalizuje statički niz struktura osoba sa članovima ime i prezime (uređen u rastućem poretku prezimena) sa manje od 10 elemenata, a zatim se učitava jedan karakter i pronalazi (bibliotečkom funkcijom `bsearch`) i štampa jedna struktura iz niza osoba čije prezime počinje tim karakterom. Ako takva osoba ne postoji, štampati -1 na standardni izlaz.

```
Osoba niz_osoba[]={{"Mika", "Antic"},
                   {"Dobrica", "Eric"},
```

```
        {"Desanka", "Maksimovic"},
        {"Dusko", "Radovic"},
        {"Ljubivoje", "Rsumovic"}};
```

Test 1

```
Ulaz:  R
Izlaz: Dusko Radovic
```

3.2 Sortiranje

Zadatak 3.12 U datom nizu brojeva pronaći dva broja koja su na najmanjem rastojanju. Niz se zadaje sa standardnog ulaza, sve do kraja ulaza, i neće sadržati više od 256 i manje od 2 elemenata. Na izlaz ispisati njihovu razliku. Uputstvo: prvo sortirati niz.

Test 1

```
Ulaz:   23 64 123 76 22 7
Izlaz:  1
```

Test 2

```
Ulaz:   21 654 65 123 65 12 61
Izlaz:  0
```

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

Test 1

```
Ulaz:   anagram ramgana
Izlaz:  jesu
```

Test 2

```
Ulaz:   anagram anagrm
Izlaz:  nisu
```

Zadatak 3.14 Napisati program koji pronalazi broj koji se najviše puta pojavljivao u datom nizu. Niz se zadaje sa standardnog ulaza sve do kraja ulaza, i neće biti duži od 256 i kraći od jednog elemenata. Uputstvo: prvo sortirati niz, a zatim naći najdužu sekvencu jednakih elemenata.

Test 1

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

Test 2

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

Zadatak 3.15 Napisati funkciju koja proverava da li u datom nizu postoje dva elementa kojima je zbir zadati ceo broj. Napisati i program koji testira ovu funkciju. U programu se prvo učitava broj, a zatim i niz (pretpostaviti da za niz neće biti uneto više od 256 brojeva). Elementi niza se unose sve do kraja ulaza. Uputstvo: prvo sortirati niz.

Test 1

```
Ulaz:   34 134 4 1 6 30 23
Izlaz:  da
```

Test 2

```
Ulaz:   12 53 1 43 3 56 13
Izlaz:  ne
```

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

Test 1

```
Ulaz:   3 6 7 11 14 35 0 3 5 8 0
Izlaz:  3 3 5 6 7 8 11 14 35
```

Test 2

```
Ulaz:   1 4 7 0 9 11 23 54 75 0
Izlaz:  1 4 7 9 11 23 54 75
```

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

Test 1

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

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

Izlazna datoteka (ceo-tok.txt):

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

Zadatak 3.18 Napraviti biblioteku `sort.h` i `sort.c` koja implementira algoritme sortiranja nizova celih brojeva. Biblioteka treba da sadrži `selection`, `merge`, `quick`, `bubble`, `insertion` i `shell sort`. Upotrebiti biblioteku kako bi se napravilo poređenje efikasnosti različitih algoritama sortiranja. Efikasnost meriti na slučajno generisanim nizovima, na već sortiranim nizovima i na naopako sortiranim nizovima. Izbor algoritma, veličine i početnog rasporeda elemenata niza birati kroz argumente komandne linije. Vreme meriti programom `time`. Analizirati porast vremena sa porastom dimenzije `n`.

Upotreba programa 1

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

Upotreba programa 2

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

Upotreba programa 3

```
Poziv: time ./a.out 100000 -s
Izlaz:
  real    0m0.071s
  user    0m0.068s
  sys     0m0.000s
```

Zadatak 3.19 Napisati funkcije koje sortiraju niz struktura tačaka na osnovu sledećih kriterijuma:

(a) njihovog rastojanja od koordinatnog početka,

(b) `x` koordinata tačaka,

(c) `y` koordinata tačaka.

Napisati program koji učitava niz tačaka iz datoteke čije se ime zadaje kao drugi argument komandne linije, i u zavisnosti od prisutnih opcija (prvi argument) u komandnoj liniji (`-o`, `-x` ili `-y`), sortira tačke po jednom od prethodna tri kriterijuma i rezultat upisuje u datoteku čije se ime zadaje kao treći argument komandne linije. U ulaznoj datoteci nije zadato više od 128 tačaka.

Test 1

```
Poziv:  ./a.out -x in.txt out.txt
Ulazna datoteka (in.txt):
  3 4
  11 6
  7 3
  2 82
  -1 6
Izlazna datoteka (out.txt):
  -1 6
  2 82
  3 4
  7 3
  11 6
```

Test 2

```
Poziv:  ./a.out -o in.txt out.txt
Ulazna datoteka (on.txt):
  3 4
  11 6
  7 3
  2 82
  -1 6
Izlazna datoteka (out.txt):
  3 4
  -1 6
  7 3
  11 6
  2 82
```

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

Test 1

```
Ulazna datoteka (biracki-spisak.txt):
  Andrija Petrovic
  Anja Ilic
  Aleksandra Cvetic
  Bojan Golubovic
  Dragan Markovic
  Filip Dukic
  Ivana Stankovic
  Ivana Markovic
  Lazar Micic
  Marija Stankovic
Izlaz: 3
```

Zadatak 3.21 Definisana je struktura podataka

```
typedef struct dete
{
      char ime[MAX_IME];
      char prezime[MAX_IME];
      unsigned godiste;
} Dete;
```

Napisati funkciju koja sortira niz dece po godištu, a kada su deca istog godišta, tada ih sortira leksikografski po prezimenu i imenu. Napisati program koji učitava podatke o deci koji se nalaze u datoteci, čije se ime zadaje kao prvi argument komandne linije, sortira ih i sortirani niz upisuje u datoteku čije se ime zadaje kao drugi argument komandne linije. Pretpostaviti da u ulaznoj datoteci nisu zadati podaci o više od 128 deteta.

Test 1

```
Poziv: ./a.out in.txt out.txt
Ulazna datoteka (in.out):
  Petar Petrovic 2007
  Milica Antonic 2008
  Ana Petrovic 2007
  Ivana Ivanovic 2009
  Dragana Markovic 2010
  Marija Antic 2007
Izlazna datoteka (out.txt):
  Marija Antic 2007
  Ana Petrovic 2007
  Petar Petrovic 2007
  Milica Antonic 2008
  Ivana Ivanovic 2009
  Dragana Markovic 2010
```

Zadatak 3.22 Napisati funkciju koja sortira niz niski po broju suglasnika u niski, ukoliko reči imaju isti broj suglasnika tada po dužini niske, a ukoliko su i

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

Test 1

```
Ulazna datoteka (niske.txt):
  ana petar andjela milos nikola aleksandar ljubica matej milica
Izlaz:
  ana matej milos petar milica nikola andjela ljubica aleksandar
```

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

Upotreba programa 1

```
Ulazna datoteka (artikli.txt):
  1001 Keks Jaffa 120
  2530 Napolitanke Bambi  230
  0023 Medeno_srce Pionir 150
  2145 Pardon Marbo 70
Interakcija programa:
  Asortiman:
  KOD                Naziv artikla     Ime proizvodjaca        Cena
          23           Medeno_srce               Pionir      150.00
        1001                 Keks                Jaffa      120.00
        2145               Pardon                Marbo       70.00
        2530           Napolitanke                Bambi      230.00
  ---------------------------
  - Za kraj za kraj rada kase, pritisnite CTRL+D!
  - Za nov racun unesite kod artikla!

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

    UKUPNO: 270.00 dinara.

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

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

    UKUPNO: 230.00 dinara.

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

  Kraj rada kase!
```

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

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

Test 1

```
Ulazna datoteka (aktivnosti.txt):
  Aleksandra Cvetic 4 6
  Bojan Golubovic 4 3
  Dragan Markovic 3 5
  Ivana Stankovic 3 1
  Marija Stankovic 1 3
  Ognjen Peric 1 2
  Uros Milic 2 5
  Andrija Petrovic 2 5
  Anja Ilic 3 1
  Lazar Micic 1 3
  Nenad Brankovic 2 4
Izlazna datoteka (dat1.txt):
  Studenti sortirani po imenu leksikografski rastuce:
  Aleksandra Cvetic  4  6
  Andrija Petrovic  2  5
  Anja Ilic  3  1
  Bojan Golubovic  4  3
  Dragan Markovic  3  5
  Ivana Stankovic  3  1
  Lazar Micic  1  3
  Marija Stankovic  1  3
  Nenad Brankovic  2  4
  Ognjen Peric  1  2
  Uros Milic  2  5
Izlazna datoteka (dat2.txt):
  Studenti sortirani po broju zadataka opadajuce,
  pa po duzini imena rastuce:
  Aleksandra Cvetic  4  6
  Uros Milic  2  5
  Dragan Markovic  3  5
  Andrija Petrovic  2  5
  Nenad Brankovic  2  4
  Lazar Micic  1  3
  Bojan Golubovic  4  3
  Marija Stankovic  1  3
  Ognjen Peric  1  2
  Anja Ilic  3  1
  Ivana Stankovic  3  1
Izlazna datoteka (dat3.txt):
  Studenti sortirani po prisustvu opadajuce,
  pa po broju zadataka,
  pa po prezimenima leksikografski opadajuce:
  Aleksandra Cvetic  4  6
  Bojan Golubovic  4  3
  Dragan Markovic  3  5
  Ivana Stankovic  3  1
  Anja Ilic  3  1
  Andrija Petrovic  2  5
  Uros Milic  2  5
  Nenad Brankovic  2  4
  Marija Stankovic  1  3
  Lazar Micic  1  3
  Ognjen Peric  1  2
```

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

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

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

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

Na standardni izlaz ispisati informacije o pesmama sortirane na opisan način.

(a) Uraditi zadatak uz pretpostavku da je maksimalna dužina imena izvođača 20 karaktera, a imena naslova pesme 50 kraraktera.

(b) Uraditi zadatak bez pravljenja pretpostavki o maksimalnoj dužini imena izvođača i naslova pesme.

Jelena: Kako ovde da prikažem rešenje pod a) i b)? Milena: Mislim da bi bilo najbolje razdvojiti kod po datotekama i napisati dve main funkcije. Na taj nacin onda nedupliramo dva programa, vec oba ukljucuju zajednicke delove, ukoliko ima dovoljno tih zajednickih delova. Ukoliko ih ima malo, onda svaki zadatak posebno.

Test 1

```
Poziv: ./a.out
Datoteka:  5
           Ana - Nebo, 2342
           Laza - Oblaci, 29
           Pera - Ptice, 327
           Jelena - Sunce, 92321
           Mika - Kisa, 5341
Izlaz:     Jelena - Sunce, 92321
           Mika - Kisa, 5341
           Ana - Nebo, 2342
           Pera - Ptice, 327
           Laza - Oblaci, 29
```

Test 2

```
Poziv: ./a.out -i
Datoteka:  5
           Ana - Nebo, 2342
           Laza - Oblaci, 29
           Pera - Ptice, 327
           Jelena - Sunce, 92321
           Mika - Kisa, 5341
Izlaz:     Ana - Nebo, 2342
           Jelena - Sunce, 92321
           Laza - Oblaci, 29
           Mika - Kisa, 5341
           Pera - Ptice, 327
```

Test 3

```
Poziv: ./a.out -n
Datoteka:  5
           Ana - Nebo, 2342
           Laza - Oblaci, 29
           Pera - Ptice, 327
           Jelena - Sunce, 92321
           Mika - Kisa, 5341
Izlaz:     Mika - Kisa, 5341
           Ana - Nebo, 2342
           Laza - Oblaci, 29
           Pera - Ptice, 327
           Jelena - Sunce, 92321
```

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

Test 1

```
Ulaz: U 2 U 0 U 6 U 4 N 1 U 8 N 2 N 5 U 2 N 3 N 5
Izlaz: 0 2 8 2 6
```

**** Zadatak 3.27** Šef u restoranu je neuredan i palačinke koje ispeče ne slaže redom po veličini. Konobar pre serviranja mora da sortira palačinke po veličini, a jedina operacija koju sme da izvodi je da obrne deo palačinki. Na primer, sledeća slika po kolonama predstavlja naslagane palačinke posle svakog okretanja. Na početku, palačinka veličine 2 je na dnu, iznad nje se redom nalaze najmanja, najveća, itd... Na slici crtica predstavlja mesto iznad koga će konobar okrenuti palačinke. Prvi potez konobara je okretanje palačinki veličine 5, 4 i 3 (prva kolona), i tada će veličine palačinki odozdo nagore biti 2, 1, 3, 4, 5 (druga kolona). Posle još dva okretanja, palačinke će biti složene.

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

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

3.3 Bibliotečke funkcije pretrage i sortiranja

Zadatak 3.28 Napisati program koji ilustruje upotrebu bibiliotečkih funkcija za pretraživanje i sortiranje nizova, i mogućnost zadavanja različitih kriterijuma sortiranja. Sa standardnog ulaza se unosi dimenzija niza celih brojeva (ne veća od 100), a potom i sami elementi niza. Upotrebom funkcije `qsort` sortirati niz u rastućem poretku, sa standardnog ulaza učitati broj koji se traži u nizu, pa zatim funkcijama `bsearch` i `lfind` utvrditi da li se zadati broj nalazi u nizu i na standardni izlaz ispisati odgovarajuću poruku.

Upotreba programa 1

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

Upotreba programa 2

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

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

Upotreba programa 1

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

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

(a) leksikografski,

(b) po dužini.

Niske se učitavaju iz fajla `niske.txt`, neće ih biti više od 1000, i svaka će biti dužine najviše 30 karaktera. Program prvo leksikografski sortira niz, primenjuje binarnu pretragu (`bsearch`) zarad traženja niske unete sa standardnog ulaza, a

potom linearnu pretragu koristeći funkciju `lfind`. Na kraju, niske bivaju sortirane po dužini. Rezultate svih sortiranja i pretraga ispisati na standardni izlaz.

Test 1

```
Ulazna datoteka (niske.txt):
  ana petar andjela milos nikola aleksandar ljubica matej milica
Interakcija programa:
  Leksikografski sortirane niske:
  aleksandar ana andjela ljubica matej milica milos nikola petar
  Uneti trazenu nisku: matej
  Niska "matej" je pronadjena u nizu na poziciji 4
  Niska "matej" je pronadjena u nizu na poziciji 4
  Niske sortirane po duzini:
  ana matej milos petar milica nikola andjela ljubica aleksandar
```

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

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

Test 1

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

Zadatak 3.33 Uraditi zadatak 3.13, ali korišćenjem bibliotečke `qsort` funkcije.

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

Test 1

```
Ulaz:   4 prog search sort search
Izlaz:  ima
```

Test 2

```
Ulaz:   3 test kol ispit
Izlaz:  nema
```

Zadatak 3.35 Datoteka `studenti.txt` sadrži spisak studenata. Za svakog studenta poznat je nalog na Alas-u (oblika npr. `mr97125`, `mm09001`), ime i prezime i broj poena. Napisati program koji sortira (korišćenjem funkcije `qsort`) studente po

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

Test 1

```
Poziv: ./a.out -n mm13321
Ulazna datoteka (studenti.txt):
  mr14123 Marko Antic 20
  mm13321 Marija Radic 12
  ml13011 Ivana Mitrovic 19
  ml13066 Pera Simic 15
  mv14003 Jovan Jovanovic 17
Izlazna datoteka (izlaz.txt):
  ml13011 Ivana Mitrovic 19
  ml13066 Pera Simic 15
  mm13321 Marija Radic 12
  mr14123 Marko Antic 20
  mv14003 Jovan Jovanovic 17
Izlaz:
  mm13321 Marija Radic 12
```

Zadatak 3.36 Definisana je struktura:

```
typedef struct { int dan; int mesec; int godina; } Datum;}
```

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

Test 1

```
Poziv: ./a.out datoteka.txt
Datoteka:           Ulaz:             Izlaz:
1.1.2013            13.12.2016        postoji
13.12.2016          10.5.2015         ne postoji
11.11.2011          5.2.2009          postoji
3.5.2015
5.2.2009
```

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

Test 1

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

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

3.4 Rešenja

Rešenje 3.1

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

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

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

/* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
   indeks pozicije nadjenog elementa ili -1, ako element nije
   pronadjen */
int binarna_pretraga(int a[], int n, int x)
{
```

```
29    int levi = 0;
      int desni = n - 1;
31    int srednji;
      /* Dokle god je indeks levi levo od indeksa desni */
33    while (levi <= desni) {
        /* Racunamo srednji indeks */
35      srednji = (levi + desni) / 2;
        /* Ako je srednji element veci od x, tada se x mora nalaziti
37         u levoj polovini niza */
        if (x < a[srednji])
39        desni = srednji - 1;
        /* Ako je srednji element manji od x, tada se x mora
41         nalaziti u desnoj polovini niza */
        else if (x > a[srednji])
43        levi = srednji + 1;
        else
45        /* Ako je srednji element jednak x, tada smo pronasli x na
             poziciji srednji */
47        return srednji;
      }
49    /* Ako nije pronadjen vracamo -1 */
      return -1;
51  }

53  /* Funkcija trazi u sortiranom nizu a[] duzine n broj x. Vraca
       indeks pozicije nadjenog elementa ili -1, ako element nije
55     pronadjen */
    int interpolaciona_pretraga(int a[], int n, int x)
57  {
      int levi = 0;
59    int desni = n - 1;
      int srednji;
61    /* Dokle god je indeks levi levo od indeksa desni...  */
      while (levi <= desni) {
63      /* Ako je element manji od pocetnog ili veci od poslednjeg
           clana u delu niza a[levi],...,a[desni] tada nije u tom
65         delu niza. Ova provera je neophodna, da se ne bi dogodilo
           da se prilikom izracunavanja indeksa srednji izadje izvan
67         opsega indeksa [levi,desni] */
        if (x < a[levi] || x > a[desni])
69        return -1;
        /* U suprotnom, x je izmedju a[levi] i a[desni], pa ako su
71         a[levi] i a[desni] jednaki, tada je jasno da je x jednako
           ovim vrednostima, pa vracamo indeks levi (ili indeks
73         desni, sve jedno je).Ova provera je neophodna, zato sto
           bismo inace prilikom izracunavanja srednji imali deljenje
75         nulom. */
        else if (a[levi] == a[desni])
77        return levi;
        /* Racunamo srednji indeks */
79      srednji =
            levi +
81          ((double) (x - a[levi]) / (a[desni] - a[levi])) *
            (desni - levi);
83      /* Napomena: Indeks srednji je uvek izmedju levi i desni,
           ali ce verovatno biti blize trazenoj vrednosti nego da
```

```
85        smo prosto uvek uzimali srednji element. Ovo se moze
          porediti sa pretragom recnika: ako neko trazi rec na
87        slovo 'B', sigurno nece da otvori recnik na polovini, vec
          verovatno negde blize pocetku. */
89      /* Ako je srednji element veci od x, tada se x mora nalaziti
           u levoj polovini niza */
91      if (x < a[srednji])
          desni = srednji - 1;
93      /* Ako je srednji element manji od x, tada se x mora
           nalaziti u desnoj polovini niza */
95      else if (x > a[srednji])
          levi = srednji + 1;
97      else
          /* Ako je srednji element jednak x, tada smo pronasli x na
99           poziciji srednji */
          return srednji;
101   }
      /* Ako nije pronadjen vracamo -1 */
103   return -1;
  }
105
  /* Funkcija main */
107 int main(int argc, char **argv)
  {
109   int a[MAX];
      int n, i, x;
111   struct timespec time1, time2, time3, time4, time5, time6;
      FILE *f;
113
      /* Provera argumenata komandne linije */
115   if (argc != 3) {
        fprintf(stderr,
117             "koriscenje programa: %s dim_niza trazeni_br\n",
                argv[0]);
119     exit(EXIT_FAILURE);
      }
121
      /* Dimenzija niza */
123   n = atoi(argv[1]);
      if (n > MAX || n <= 0) {
125     fprintf(stderr, "Dimenzija niza neodgovarajuca\n");
        exit(EXIT_FAILURE);
127   }

129   /* Broj koji se trazi */
      x = atoi(argv[2]);
131
      /* Elemente niza odredjujemo slucajno, tako da je svaki
133      sledeci veci od prethodnog. srandom() funkcija obezbedjuje
         novi seed za pozivanje random() funkcije. Kako nas niz ne
135      bi uvek isto izgledao seed smo postavili na tekuce vreme u
         sekundama od Nove godine 1970. random()%100 daje brojeve
137      izmedju 0 i 99 */
      srandom(time(NULL));
139   for (i = 0; i < n; i++)
        a[i] = i == 0 ? random() % 100 : a[i - 1] + random() % 100;
```

```
141
    /* Lineara pretraga */
143 printf("Linearna pretraga\n");
    /* Racunamo vreme proteklo od Nove godine 1970 */
145 clock_gettime(CLOCK_REALTIME, &time1);
    /* Pretrazujemo niz */
147 i = linearna_pretraga(a, n, x);
    /* Racunamo novo vreme i razlika predstavlja vreme utroseno za
149    lin pretragu */
    clock_gettime(CLOCK_REALTIME, &time2);
151 if (i == -1)
      printf("Element nije u nizu\n");
153 else
      printf("Element je u nizu na poziciji %d\n", i);
155 printf("---------------------------------\n");

157 /* Binarna pretraga */
    printf("Binarna pretraga\n");
159 clock_gettime(CLOCK_REALTIME, &time3);
    i = binarna_pretraga(a, n, x);
161 clock_gettime(CLOCK_REALTIME, &time4);
    if (i == -1)
163   printf("Element nije u nizu\n");
    else
165   printf("Element je u nizu na poziciji %d\n", i);
    printf("---------------------------------\n");
167
    /* Interpolaciona pretraga */
169 printf("Interpolaciona pretraga\n");
    clock_gettime(CLOCK_REALTIME, &time5);
171 i = interpolaciona_pretraga(a, n, x);
    clock_gettime(CLOCK_REALTIME, &time6);
173 if (i == -1)
      printf("Element nije u nizu\n");
175 else
      printf("Element je u nizu na poziciji %d\n", i);
177 printf("---------------------------------\n");

179 /* Upisujemo podatke o izvrsavanju programa u log fajl */
    if ((f = fopen("vremena.txt", "a")) == NULL) {
181   fprintf(stderr, "Neuspesno otvaranje log fajla.\n");
      exit(EXIT_FAILURE);
183 }

185 fprintf(f, "Dimenzija niza od %d elemenata.\n", n);
    fprintf(f, "\tLinearna pretraga:%10ld ns\n",
187         (time2.tv_sec - time1.tv_sec) * 1000000000 +
            time2.tv_nsec - time1.tv_nsec);
189 fprintf(f, "\tBinarna: %19ld ns\n",
            (time4.tv_sec - time3.tv_sec) * 1000000000 +
191         time4.tv_nsec - time3.tv_nsec);
    fprintf(f, "\tInterpolaciona: %12ld ns\n\n",
193         (time6.tv_sec - time5.tv_sec) * 1000000000 +
            time6.tv_nsec - time5.tv_nsec);
195
    fclose(f);
```

```
197
  return 0;
199 }
```

Rešenje 3.2

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

18 int bin_pretgraga_rek(int a[], int l, int d, int x)
{
20  int srednji;
  if (l > d)
22    return -1;
  /* Srednja pozicija na kojoj se trazi vrednost x */
24  srednji = (l + d) / 2;
  /* Ako je sredisnji element trazeni */
26  if (a[srednji] == x)
    return srednji;
28  /* Ako je trazeni broj veci od srednjeg, pretrazujemo desnu
     polovinu niza */
30  if (a[srednji] < x)
    return bin_pretgraga_rek(a, srednji + 1, d, x);
32  /* Ako je trazeni broj manji od srednjeg, pretrazujemo levu
     polovinu niza */
34  else
    return bin_pretgraga_rek(a, l, srednji - 1, x);
36 }

38
int interp_pretgraga_rek(int a[], int l, int d, int x)
40 {
  int p;
42  if (x < a[l] || x > a[d])
    return -1;
44  if (a[d] == a[l])
    return l;
46  /* Pozicija na kojoj se trazi vrednost x */
  p = l + (d - l) * (x - a[l]) / (a[d] - a[l]);
48  if (a[p] == x)
    return p;
```

```
  if (a[p] < x)
    return interp_pretgraga_rek(a, p + 1, d, x);
  else
    return interp_pretgraga_rek(a, l, p - 1, x);
}

#define MAX 1024

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

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

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

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

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

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

  return 0;
}
```

Rešenje 3.3

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

```c

#define MAX_STUDENATA 128
#define MAX_DUZINA 16

/* O svakom studentu imamo 3 informacije i njih objedinjujemo u
   strukturu kojom cemo predstavljati svakog studenta. */
typedef struct {
  /* Indeks mora biti tipa long jer su podaci u datoteci
     preveliki za int, npr. 20140123 */
  long indeks;
  char ime[MAX_DUZINA];
  char prezime[MAX_DUZINA];
} Student;

/* Ucitan niz studenata ce biti sortiran prema indeksu, jer cemo
   ih, redom, kako citamo smestati u niz, a u datoteci su vec
   smesteni sortirani rastuce prema broju indeksa. Iz tog
   razloga pretragu po indeksu cemo vrsiti binarnom pretragom,
   dok pretragu po prezimenu moramo vrsiti linearno, jer nemamo
   garancije da postoji uredjenje po prezimenu. */

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

int linearna_pretraga(Student a[], int n, char x[])
{
  int i;
  for (i = 0; i < n; i++)
    /* Poredimo prezime i-tog studenta i poslato x */
    if (strcmp(a[i].prezime, x) == 0)
```

```
      return i;
  return -1;
}

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

  /* Proveravamo da li nam je korisnik prilikom poziva prosledio
     ime_datoteke sa informacijama o studentima */
  if (argc != 2) {
    fprintf(stderr,
            "Greska: Program se poziva sa %s ime_datoteke\n",
            argv[0]);
    exit(EXIT_FAILURE);
  }

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

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

  /* Nakon citanja datoteka nam vise nije neophodna i odmah je
     zatvaramo */
  fclose(fin);

  printf("Unesite indeks studenta cije informacije zelite: ");
  scanf("%ld", &trazen_indeks);
  i = binarna_pretraga(dosije, br_studenata, trazen_indeks);
  if (i == -1)
    printf("Ne postoji student sa indeksom %ld\n",
```

```
            trazen_indeks);
  else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

  printf("Unesite prezime studenta cije informacije zelite: ");
  scanf("%s", trazeno_prezime);
  i = linearna_pretraga(dosije, br_studenata, trazeno_prezime);
  if (i == -1)
    printf("Ne postoji student sa prezimenom %s\n",
            trazeno_prezime);
  else
    printf("Indeks: %ld, Ime i prezime: %s %s\n",
            dosije[i].indeks, dosije[i].ime, dosije[i].prezime);

  return 0;
}
```

Rešenje 3.5

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

/* Struktura koja opisuje tacku u ravni */
typedef struct Tacka {
  float x;
  float y;
} Tacka;

/* Funkcija koja racuna rastojanje zadate tacke od koordinatnog
   pocetka (0,0) */
float rastojanje(Tacka A)
{
  return sqrt(A.x * A.x + A.y * A.y);
}

/* Funkcija koja pronalazi tacku najblizu koordinatnom pocetku u
   nizu zadatih tacaka t dimenzije n */
Tacka najbliza_koordinatnom(Tacka t[], int n)
{
  Tacka najbliza;
  int i;
  najbliza = t[0];
  for (i = 1; i < n; i++) {
    if (rastojanje(t[i]) < rastojanje(najbliza)) {
      najbliza = t[i];
    }
  }
  return najbliza;
}

/* Funkcija koja pronalazi tacku najblizu x osi u nizu zadatih
   tacaka t dimenzije n */
```

```
36 Tacka najbliza_x_osi(Tacka t[], int n)
   {
38
     Tacka najbliza;
40   int i;
     najbliza = t[0];
42   for (i = 1; i < n; i++) {
       if (fabs(t[i].x) < fabs(najbliza.x)) {
44       najbliza = t[i];
       }
46   }
     return najbliza;
48 }

50 /* Funkcija koja pronalazi tacku najblizu y osi u nizu zadatih
      tacaka t dimenzije n */
52 Tacka najbliza_y_osi(Tacka t[], int n)
   {
54   Tacka najbliza;
     int i;
56   najbliza = t[0];
     for (i = 1; i < n; i++) {
58     if (fabs(t[i].y) < fabs(najbliza.y)) {
         najbliza = t[i];
60     }
     }
62   return najbliza;
   }
64
   #define MAX 1024
66
   int main(int argc, char *argv[])
68 {
     FILE *ulaz;
70   Tacka tacke[MAX];
     Tacka najbliza;
72   int i, n;

74   /* Ocekujemo da korisnik unese barem ime izvrsne verzije
        programa i ime datoteke sa tackama */
76   if (argc < 2) {
       fprintf(stderr,
78             "koriscenje programa: %s ime_datoteke\n", argv[0]);
       return EXIT_FAILURE;
80   }

82   /* Otvaramo datoteku za citanje */
     ulaz = fopen(argv[1], "r");
84   if (ulaz == NULL) {
       fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
86             argv[1]);
       return EXIT_FAILURE;
88   }

90   /* Sve dok ima tacaka u datoteci, smestamo ih u niz sa
        tackama; i predstavlja indeks tekuce tacke */
```

```
  i = 0;
  while (fscanf(ulaz, "%f %f", &tacke[i].x, &tacke[i].y) == 2) {
    i++;
  }
  n = i;

  /* Proveravamo koji su dodatni argumenti komandne linije. Ako
     nema dodatnih argumenata */
  if (argc == 2)
    /* Trazimo najblizu tacku u odnosu na koordinatni pocetak */
    najbliza = najbliza_koordinatnom(tacke, n);
  /* Inace proveravamo koji je dodatni argument. Ako je u
     pitanju opcija -x */
  else if (strcmp(argv[2], "-x") == 0)
    /* Racunamo rastojanje u odnosu na x osu */
    najbliza = najbliza_x_osi(tacke, n);
  /* Ako je u pitanju opcija -y */
  else if (strcmp(argv[2], "-y") == 0)
    /* Racunamo rastojanje u odnosu na y osu */
    najbliza = najbliza_y_osi(tacke, n);
  else {
    /* Ako nije zadata opcija -x ili -y, ispisujemo obavestenje
       za korisnika i prekidamo izvrsavanje programa */
    fprintf(stderr, "Pogresna opcija\n");
    return EXIT_FAILURE;
  }

  /* Stampamo koordinate trazene tacke */
  printf("%g %g\n", najbliza.x, najbliza.y);

  fclose(ulaz);

  return 0;
}
```

Rešenje 3.6

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

/* Tacnost */
#define EPS 0.001

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

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

  /* Sve dok ne pronadjemo trazenu vrednost argumenta */
  while (1) {
    /* Pronalazimo sredinu intervala */
```

```
    s = (l + d) / 2;
    /* Ako je vrednost kosinusa u ovoj tacki manja od zadate
       tacnosti, prekidamo pretragu */
    if (fabs(cos(s)) < EPS) {
      break;
    }
    /* Ako je nula u levom delu intervala, nastavljamo pretragu
       na intervalu [l, s] */
    if (cos(l) * cos(s) < 0)
      d = s;
    else
      /* Inace, nastavljamo pretragu na intervalu [s, d] */
      l = s;
  }

  /* Stampamo vrednost trazene tacke */
  printf("%g\n", s);

  return 0;
}
```

Rešenje 3.12

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

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

  /* U svakoj iteraciji ove petlje se pronalazi najmanji element
     medju elementima a[i], a[i+1],...,a[n-1], i postavlja se na
     poziciju i, dok se element na poziciji i premesta na
     poziciju min, na kojoj se nalazio najmanji od gore
     navedenih elemenata. */
  for (i = 0; i < n - 1; i++) {
    /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
       nalazi najmanji od elemenata a[i],...,a[n-1]. */
    min = i;
    for (j = i + 1; j < n; j++)
      if (a[j] < a[min])
        min = j;
    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
       samo ako su (i) i min razliciti, inace je nepotrebno. */
    if (min != i) {
      pom = a[i];
      a[i] = a[min];
      a[min] = pom;
    }
  }
}
```

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


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

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

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

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

  return 0;
}
```

Rešenje 3.13

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

#define MAX_DIM 128

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

```
  }
}

/* Funkcija vraca: 1 - ako jesu anagrami; 0 - inace.
   pretpostavlja se da su niske s i t sortirane */
int anagrami(char s[], char t[], int n_s, int n_t)
{
  int i, n;

  /* Ako dve niske imaju razlicit broj elemenata onda nisu
     anagrami */
  if (n_s != n_t)
    return 0;

  n = n_s;

  /* Dve sortirane niske su anagrami akko su jednake */
  for (i = 0; i < n; i++)
    if (s[i] != t[i])
      return 0;
  return 1;
}

int main()
{
  char s[MAX_DIM], t[MAX_DIM];
  int n_s, n_t;

  /* Ucitavamo dve niske sa ulaza */
  printf("Unesite prvu nisku: ");
  scanf("%s", s);
  printf("Unesite drugu nisku: ");
  scanf("%s", t);

  /* Odredjujemo duzinu niski */
  n_s = strlen(s);
  n_t = strlen(t);

  /* Sortiramo niske */
  selectionSort(s, n_s);
  selectionSort(t, n_t);

  /* Proveravamo da li su niske anagrami */
  if (anagrami(s, t, n_s, n_t))
    printf("jesu\n");
  else
    printf("nisu\n");
  return 0;
}
```

Rešenje 3.14

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

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

/* Funkcija za odredjivanje onog elementa sortiranog niza koji
   se najvise puta pojavio u tom nizu */
int najvise_puta(int a[], int n)
{
  int i, j, br_pojava, i_max_pojava = -1, max_br_pojava = -1;
  /* Za i-ti element izracunavamo koliko se puta pojavio u nizu */
  for (i = 0; i < n; i = j) {
    br_pojava = 1;
    for (j = i + 1; j < n && a[i] == a[j]; j++)
      br_pojava++;
    /* Ispitujemo da li se do tog trenutka i-ti element pojavio
       najvise puta u nizu */
    if (br_pojava > max_br_pojava) {
      max_br_pojava = br_pojava;
      i_max_pojava = i;
    }
  }
  /* Vracamo element koji se najvise puta pojavio u nizu */
  return a[i_max_pojava];
}

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

  /* Ucitavaju se element niza sve do kraja ulaza */
  i = 0;
  printf("Unesite elemente niza: ");
  while (scanf("%d", &a[i]) != EOF)
    i++;

  /* Niz se sortira */
  selectionSort(a, i);

  /* Odredjuje se broj koji se najvise puta pojavio u nizu */
  printf("%d\n", najvise_puta(a, i));

  return 0;
```

```
}
```

Rešenje 3.15

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

/* Funkcija za sortiranje niza */
void selectionSort(int a[], int n)
{
  int i, j, min, pom;
  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++)
      if (a[j] < a[min])
        min = j;
    if (min != i) {
      pom = a[i];
      a[i] = a[min];
      a[min] = pom;
    }
  }
}

/* Funkcija za binarnu pretragu niza. funkcija vraca: 1 - ako se
   element x nalazi u nizu; 0 - inace. pretpostavlja se da je
   niz sortiran u rastucem poretku */
int binarna_pretraga(int a[], int n, int x)
{
  int levi = 0, desni = n - 1, srednji;

  while (levi <= desni) {
    srednji = (levi + desni) / 2;
    if (a[srednji] == x)
      return 1;
    else if (a[srednji] > x)
      desni = srednji - 1;
    else if (a[srednji] < x)
      levi = srednji + 1;
  }
  return 0;
}

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

  /* Ucitava se trazeni zbir */
  printf("Unesite trazeni zbir: ");
  scanf("%d", &zbir);

  /* Ucitavaju se elementi niza sve do kraja ulaza */
  i = 0;
  printf("Unesite elemente niza: ");
  while (scanf("%d", &a[i]) != EOF)
```

```
    i++;
  n = i;

  selectionSort(a, n);

  for (i = 0; i < n; i++)
    /* Za i-ti element niza binarno se pretrazuje da li se u
       ostatku niza nalazi element koji sabran sa njim ima
       ucitanu vrednost zbira */
    if (binarna_pretraga(a + i + 1, n - i - 1, zbir - a[i])) {
      printf("da\n");
      return 0;
    }
  printf("ne\n");
  return 0;
}
```

Rešenje 3.16

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

int merge(int *niz1, int dim1, int *niz2, int dim2, int *niz3,
          int dim3)
{
  int i = 0, j = 0, k = 0;
  /* U slucaju da je dimenzija treceg niza manja od neophodne,
     funkcija vraca -1 */
  if (dim3 < dim1 + dim2)
    return -1;

  /* Vrsi se ucesljavanje nizova sve dok se ne dodje do kraja
     jednog od njih */
  while (i < dim1 && j < dim2) {
    if (niz1[i] < niz2[j])
      niz3[k++] = niz1[i++];
    else
      niz3[k++] = niz2[j++];
  }
  /* Ostatak prvog niza prepisujemo u treci */
  while (i < dim1)
    niz3[k++] = niz1[i++];

  /* Ostatak drugog niza prepisujemo u treci */
  while (j < dim2)
    niz3[k++] = niz2[j++];
  return dim1 + dim2;
}

int main()
{
  int niz1[MAX_DIM], niz2[MAX_DIM], niz3[2 * MAX_DIM];
  int i = 0, j = 0, k, dim3;

  /* Ucitavaju se nizovi sa ulaza sve dok se ne unese nula.
```

```
     Pretpostavka je da na ulazu nece biti vise od MAX_DIM
     elemenata */
  printf("Unesite elemente prvog niza: ");
  while (1) {
    scanf("%d", &niz1[i]);
    if (niz1[i] == 0)
      break;
    i++;
  }
  printf("Unesite elemente drugog niza: ");
  while (1) {
    scanf("%d", &niz2[j]);
    if (niz2[j] == 0)
      break;
    j++;
  }

  dim3 = merge(niz1, i, niz2, j, niz3, 2 * MAX_DIM);

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

  return 0;
}
```

Rešenje 3.17

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

/* Programu su neophodni argumenti komandne linije */
int main(int argc, char *argv[])
{
  FILE *fin1 = NULL, *fin2 = NULL;
  FILE *fout = NULL;
  char ime1[11], ime2[11];
  char prezime1[16], prezime2[16];
  int kraj1 = 0, kraj2 = 0;

  if (argc < 3) {
    fprintf(stderr,
            "koriscenje programa: %s fajl1 fajl2\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  fin1 = fopen(argv[1], "r");
  if (fin1 == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n\n",
            argv[1]);
    exit(EXIT_FAILURE);
  }

```

```
  fin2 = fopen(argv[2], "r");
28 if (fin2 == NULL) {
    fprintf(stderr, "Neuspesno otvaranje datoteke %s\n\n",
30          argv[2]);
    exit(EXIT_FAILURE);
32 }

34 fout = fopen("ceo-tok.txt", "w");
   if (fout == NULL) {
36   fprintf(stderr,
           "Neuspesno otvaranje datoteke ceo-tok.txt za pisanje\n\n
     ");
38   exit(EXIT_FAILURE);
   }
40
   /* Citamo narednog studenta iz prve datoteke */
42 if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
     kraj1 = 1;
44
   /* Citamo narednog studenta iz druge datoteke */
46 if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
     kraj2 = 1;
48
   /* Sve dok nismo dosli do kraja neke datoteke */
50 while (!kraj1 && !kraj2) {
     if (strcmp(ime1, ime2) < 0) {
52     /* Ime i prezime iz prve datoteke je leksikografski
          ranije, upisujemo ga u izlaznu datoteku */
54     fprintf(fout, "%s %s\n", ime1, prezime1);
       /* Citamo narednog studenta iz prve datoteke */
56     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
         kraj1 = 1;
58   } else {
       /* Ime i prezime iz druge datoteke je leksikografski
60        ranije, upisujemo ga u izlaznu datoteku */
       fprintf(fout, "%s %s\n", ime2, prezime2);
62     /* Citamo narednog studenta iz druge datoteke */
       if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
64       kraj2 = 1;
     }
66 }

68 /* Ako smo iz prethodne petlje izasli zato sto se doslo do
      kraja druge datoteke, onda ima jos imena u prvoj datoteci,
70    i prepisujemo ih, redom, jer su vec sortirani po imenu. */
   while (!kraj1) {
72   fprintf(fout, "%s %s\n", ime1, prezime1);
     if (fscanf(fin1, "%s%s", ime1, prezime1) == EOF)
74     kraj1 = 1;
   }
76
   /* Ako smo iz prve petlje izasli zato sto se doslo do kraja
78    prve datoteke, onda ima jos imena u drugoj datoteci, i
      prepisujemo ih, redom, jer su vec sortirani po imenu. */
80 while (!kraj2) {
     fprintf(fout, "%s %s\n", ime2, prezime2);
```

```
    if (fscanf(fin2, "%s%s", ime2, prezime2) == EOF)
      kraj2 = 1;
  }

  /* Zatvaramo datoteke */
  fclose(fin1);
  fclose(fin2);
  fclose(fout);

  return 0;
}
```

Rešenje 3.18

```
/* Datoteka sort.h */
#ifndef __SORT_H__
#define __SORT_H__ 1

/* Selection sort */
void selectionsort(int a[], int n);
/* Insertion sort */
void insertionsort(int a[], int n);
/* Bubble sort */
void bubblesort(int a[], int n);
/* Shell sort */
void shellsort(int a[], int n);
/* Merge sort */
void mergesort(int a[], int l, int r);
/* Quick sort */
void quicksort(int a[], int l, int r);

#endif
```

```
/* Datoteka sort.c */

#include "sort.h"

void selectionsort(int a[], int n)
{
  int i, j;
  int min;
  int pom;

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

```
24      /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
           samo ako su (i) i min razliciti, inace je nepotrebno. */
26      if (min != i) {
          pom = a[i];
28        a[i] = a[min];
          a[min] = pom;
30      }
    }
32
}
34

36  /* Funkcija sortira niz celih brojeva metodom sortiranja
       umetanjem. Ideja algoritma je sledeca: neka je na pocetku
38     i-te iteracije niz prvih i elemenata (a[0],a[1],...,a[i-1])
       sortirano. U i-toj iteraciji zelimo da element a[i] umetnemo
40     na pravu poziciju medju prvih i elemenata tako da dobijemo
       niz duzine i+1 koji je sortiran. Ovo radimo tako sto i-ti
42     element najpre uporedimo sa njegovim prvim levim susedom
       (a[i-1]). Ako je a[i] vece, tada je on vec na pravom mestu, i
44     niz a[0],a[1],...,a[i] je sortiran, pa mozemo preci na
       sledecu iteraciju. Ako je a[i-1] vece, tada zamenjujemo a[i]
46     i a[i-1], a zatim proveravamo da li je potrebno dalje
       potiskivanje elementa u levo, poredeci ga sa njegovim novim
48     levim susedom. Ovim uzastopnim premestanjem se a[i] umece na
       pravo mesto u nizu. */
50  void insertionsort(int a[], int n)
    {
52    int i, j;

54    /* Na pocetku iteracije pretpostavljamo da je niz
         a[0],...,a[i-1] sortiran */
56    for (i = 1; i < n; i++) {

58      /* U ovoj petlji redom potiskujemo element a[i] u levo
           koliko je potrebno, dok ne zauzme pravo mesto, tako da
60         niz a[0],...a[i] bude sortiran. Indeks j je trenutna
           pozicija na kojoj se element koji umecemo nalazi. Petlja
62         se zavrsava ili kada element dodje do levog kraja (j==0)
           ili dok ne naidjemo na element a[j-1] koji je manji od
64         a[j]. */
      for (j = i; j > 0 && a[j] < a[j - 1]; j--) {
66      int temp = a[j];
        a[j] = a[j - 1];
68      a[j - 1] = temp;
      }
70    }
  }
72

74  /* Funkcija sortira niz celih brojeva metodom mehurica. Ideja
       algoritma je sledeca: prolazimo kroz niz redom poredeci
76     susedne elemente, i pri tom ih zamenjujuci ako su u pogresnom
       poretku. Ovim se najveci element poput mehurica istiskuje na
78     "povrsinu", tj. na krajnju desnu poziciju. Nakon toga je
```

```
   potrebno ovaj postupak ponoviti nad nizom a[0],...,a[n-2],
   tj. nad prvih n-1 elemenata niza bez poslednjeg koji je
   postavljen na pravu poziciju. Nakon toga se istu postupak
   ponavlja nad sve kracim i kracim prefiksima niza, cime se
   jedan po jedan istiskuju elemenenti na svoje prave pozicije. */
void bubblesort(int a[], int n)
{
  int i, j;
  int ind;

  for (i = n, ind = 1; i > 1 && ind; i--)

    /* Poput "mehurica" potiskujemo najveci element medju
       elementima od a[0] do a[i-1] na poziciju i-1 uporedjujuci
       susedne elemente niza i potiskujuci veci u desno */
    for (j = 0, ind = 0; j < i - 1; j++)
      if (a[j] > a[j + 1]) {
        int temp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = temp;

        /* Promenljiva ind registruje da je bilo premestanja.
           Samo u tom slucaju ima smisla ici na sledecu
           iteraciju, jer ako nije bilo premestanja, znaci da su
           svi elementi vec u dobrom poretku, pa nema potrebe
           prelaziti na kraci prefiks niza. Moglo je naravno i
           bez ovoga, algoritam bi radio ispravno, ali bi bio
           manje efikasan, jer bi cesto nepotrebno vrsio mnoga
           uporedjivanja, kada je vec jasno da je sortiranje
           zavrseno. */
        ind = 1;
      }
}

/* Selsort je jednostavno prosirenje sortiranja umetanjem koje
   dopusta direktnu razmenu udaljenih elemenata. Prosirenje se
   sastoji u tome da se kroz algoritam umetanja prolazi vise
   puta; u prvom prolazu, umesto koraka 1 uzima se neki korak h
   koji je manji od n (sto omogucuje razmenu udaljenih
   elemenata) i tako se dobija h-sortiran niz, tj. niz u kome su
   elementi na rastojanju h sortirani, mada susedni elementi to
   ne moraju biti. U drugom prolazu kroz isti algoritam sprovodi
   se isti postupak ali za manji korak h. Sa prolazima se
   nastavlja sve do koraka h = 1, u kome se dobija potpuno
   sortirani niz. Izbor pocetne vrednosti za h, i nacina
   njegovog smanjivanja menja u nekim slucajevima brzinu
   algoritma, ali bilo koja vrednost ce rezultovati ispravnim
   sortiranjem, pod uslovom da je u poslednjoj iteraciji h imalo
   vrednost 1. */
void shellsort(int a[], int n)
{
  int h = n / 2, i, j;
  while (h > 0) {
    /* Insertion sort sa korakom h */
    for (i = h; i < n; i++) {
      int temp = a[i];
```

```
      j = i;
      while (j >= h && a[j - h] > temp) {
        a[j] = a[j - h];
        j -= h;
      }
      a[j] = temp;
    }
    h = h / 2;
  }
}

#define MAX 1000000

/* Sortiranje ucesljavanjem */
void mergesort(int a[], int l, int d)
{
  int s;
  static int b[MAX];            /* Pomocni niz */
  int i, j, k;

  /* Izlaz iz rekurzije */
  if (l >= d)
    return;

  /* Odredjujemo sredisnji indeks */
  s = (l + d) / 2;

  /* Rekurzivni pozivi */
  mergesort(a, l, s);
  mergesort(a, s + 1, d);

  /* Inicijalizacija indeksa. Indeks i prolazi krozi levu
     polovinu niza, dok indeks j prolazi kroz desnu polovinu
     niza. Indeks k prolazi kroz pomocni niz b[] */
  i = l;
  j = s + 1;
  k = 0;

  /* "Ucesljavanje" koriscenjem pomocnog niza b[] */
  while (i <= s && j <= d) {
    if (a[i] < a[j])
      b[k++] = a[i++];
    else
      b[k++] = a[j++];
  }

  while (i <= s)
    b[k++] = a[i++];

  while (j <= d)
    b[k++] = a[j++];

  /* Prepisujemo "ucesljani" niz u originalni niz */
  for (k = 0, i = l; i <= d; i++, k++)
    a[i] = b[k];

```

```
}

/* Funkcija menja mesto i-tom i j-tom elementu niza a */
void swap(int a[], int i, int j)
{
  int tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}


/* Funkcija sortira deo niza brojeva a izmedju pozicija l i r */
void quicksort(int a[], int l, int r)
{
  int i, pivot_position;

  /* Izlaz iz rekurzije -- prazan niz */
  if (l >= r)
    return;


  /* Particionisanje niza. Svi elementi na pozicijama <=
     pivot_position (izuzev same pozicije l) su strogo manji od
     pivota. Kada se pronadje neki element manji od pivota,
     uvecava se pivot_position i na tu poziciju se premesta
     nadjeni element. Na kraju ce pivot_position zaista biti
     pozicija na koju treba smestiti pivot, jer ce svi elementi
     levo od te pozicije biti manji a desno biti veci ili
     jednaki od pivota. */
  pivot_position = l;
  for (i = l + 1; i <= r; i++)
    if (a[i] < a[l])
      swap(a, ++pivot_position, i);

  /* Postavljamo pivot na svoje mesto */
  swap(a, l, pivot_position);

  /* Rekurzivno sortiramo elemente manje od pivota */
  quicksort(a, l, pivot_position - 1);
  /* Rekurzivno sortiramo elemente vece pivota */
  quicksort(a, pivot_position + 1, r);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sort.h"

/* Maksimalna duzina niza */
#define MAX 1000000

int main(int argc, char *argv[])
{
  /*
     tip_sortiranja == 0 => selectionsort (default)
     tip_sortiranja == 1 => insertionsort (-i opcija komandne
```

```
14      linije) tip_sortiranja == 2 => bubblesort (-b opcija
        komandne linije) tip_sortiranja == 3 => shellsort (-s
16      opcija komandne linije) tip_sortiranja == 4 => mergesort
        (-m opcija komandne linije) tip_sortiranja == 5 =>
18      quicksort (-q opcija komandne linije) */
    int tip_sortiranja = 0;
20  /*
        tip_niza == 0 => slucajno generisani nizovi (default)
22      tip_niza == 1 => sortirani nizovi (-r opcija komandne
        linije) tip_niza == 2 => obrnuto soritrani nizovi (-o
24      opcija komandne linije) */
    int tip_niza = 0;
26
    /* Dimenzija niza koji se sortira */
28  int dimenzija;
    int i;
30  int niz[MAX];

32  if (argc < 2) {
      fprintf(stderr,
34            "Program zahteva bar 2 argumenta komandne linije!\n");
      exit(EXIT_FAILURE);
36  }

38  /* Ocitavamo opcije i argumente prilikom poziva programa */
    for (i = 1; i < argc; i++) {
40    /* Ako je u pitanju opcija... */
      if (argv[i][0] == '-') {
42      switch (argv[i][1]) {
        case 'i':
44        tip_sortiranja = 1;
          break;
46      case 'b':
          tip_sortiranja = 2;
48        break;
        case 's':
50        tip_sortiranja = 3;
          break;
52      case 'm':
          tip_sortiranja = 4;
54        break;
        case 'q':
56        tip_sortiranja = 5;
          break;
58      case 'r':
          tip_niza = 1;
60        break;
        case 'o':
62        tip_niza = 2;
          break;
64      default:
          printf("Pogresna opcija -%c\n", argv[i][1]);
66        return 1;
          break;
68      }
      }
```

```
70      /* Ako je u pitanju argument, onda je to duzina niza koji
           treba da se sortira */
72      else {
          dimenzija = atoi(argv[i]);
74        if (dimenzija <= 0 || dimenzija > MAX) {
            fprintf(stderr, "Dimenzija niza neodgovarajuca!\n");
76          exit(EXIT_FAILURE);
          }
78      }
    }
80
    /* Elemente niza odredjujemo slucajno, ali vodeci racuna o
82     tipu niza dobijenom iz komandni linije. srandom funkcija
       obezbedjuje novi seed za pozivanje random funkcije, i kako
84     nas niz ne bi uvek isto izgledao seed smo postavili na
       tekuce vreme u sekundama od Nove godine 1970. random()%100
86     daje brojeve izmedju 0 i 99 */
    srandom(time(NULL));
88  if (tip_niza == 0)
      for (i = 0; i < dimenzija; i++)
90      niz[i] = random();
    else if (tip_niza == 1)
92    for (i = 0; i < dimenzija; i++)
        niz[i] =
94          i == 0 ? random() % 100 : niz[i - 1] + random() % 100;
    else
96    for (i = 0; i < dimenzija; i++)
        niz[i] =
98          i == 0 ? random() % 100 : niz[i - 1] - random() % 100;

100 /* Ispisujemo elemente niza printf("Niz koji sortiramo
       je:\n"); for (i = 0; i < dimenzija; i++) printf("%d\n",
102    niz[i]); */


104
    /* Sortiramo niz na odgovarajuci nacin */
106 if (tip_sortiranja == 0)
      selectionsort(niz, dimenzija);
108 else if (tip_sortiranja == 1)
      insertionsort(niz, dimenzija);
110 else if (tip_sortiranja == 2)
      bubblesort(niz, dimenzija);
112 else if (tip_sortiranja == 3)
      shellsort(niz, dimenzija);
114 else if (tip_sortiranja == 4)
      mergesort(niz, 0, dimenzija - 1);
116 else
      quicksort(niz, 0, dimenzija - 1);
118
    /* Ispisujemo elemente niza */
120   /*****************************
      printf("Sortiran niz je:\n");
122   for (i = 0; i < dimenzija; i++)
        printf("%d\n", niz[i]);
124   *****************************/
```

```
126   return 0;
}
```

Rešenje 3.19

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

#define MAX_BR_TACAKA 128

typedef struct Tacka {
  int x;
  int y;
} Tacka;

/* Funkcija racuna rastojanje zadate tacke od koordinatnog
   pocetka (0,0) */
float rastojanje(Tacka A)
{

  return sqrt(A.x * A.x + A.y * A.y);
}

/* Funkcija koja sortira niz tacaka po rastojanju od
   koordinatnog pocetka */
void sortiraj_po_rastojanju(Tacka t[], int n)
{

  int min, i, j;
  Tacka tmp;

  for (i = 0; i < n - 1; i++) {
    min = i;

    for (j = i + 1; j < n; j++) {
      if (rastojanje(t[j]) < rastojanje(t[min])) {
        min = j;
      }
    }

    if (min != i) {
      tmp = t[i];
      t[i] = t[min];
      t[min] = tmp;
    }
  }
}

/* Funkcija koja sortira niz tacaka po vrednosti x koordinate */
void sortiraj_po_x(Tacka t[], int n)
{

  int min, i, j;
```

```
51    Tacka tmp;

53    for (i = 0; i < n - 1; i++) {
        min = i;
55
        for (j = i + 1; j < n; j++) {
57        if (abs(t[j].x) < abs(t[min].x)) {
            min = j;
59        }
        }
61
        if (min != i) {
63        tmp = t[i];
          t[i] = t[min];
65        t[min] = tmp;
        }
67    }
}
69
/* Funkcija koja sortira niz tacaka po vrednosti y koordinate */
71 void sortiraj_po_y(Tacka t[], int n)
{
73
    int min, i, j;
75  Tacka tmp;

77  for (i = 0; i < n - 1; i++) {
      min = i;
79
      for (j = i + 1; j < n; j++) {
81      if (abs(t[j].y) < abs(t[min].y)) {
          min = j;
83      }
      }
85
      if (min != i) {
87      tmp = t[i];
        t[i] = t[min];
89      t[min] = tmp;
      }
91  }
}
93



95
int main(int argc, char *argv[])
97 {

99  FILE *ulaz;
    FILE *izlaz;
101 Tacka tacke[MAX_BR_TACAKA];
    int i, n;
103
    /* Proveravamo broj argumenata komandne linije: ocekujemo ime
105    izvrsnog programa, opciju, ime ulazne datoteke i ime
       izlazne datoteke tj. ocekujemo 4 argumenta */
```

```
107
    if (argc != 4) {
109   fprintf(stderr,
              "Program se poziva sa: ./a.out opcija ulaz izlaz!\n");
111   return 0;
    }
113
    /* Otvaramo datoteku u kojoj su zadate tacke */
115 ulaz = fopen(argv[2], "r");
    if (ulaz == NULL) {
117   fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
              argv[2]);
119   return 0;
    }
121
    /* Otvaramo datoteku u koju treba upisati rezultat */
123 izlaz = fopen(argv[3], "w");
    if (izlaz == NULL) {
125   fprintf(stderr, "Greska prilikom otvaranja datoteke %s!\n",
              argv[3]);
127   return 0;
    }
129
    /* Sve dok ne stignemo do kraja ulazne datoteke ucitavamo
131     koordinate tacaka i smestamo ih na odgovarajucu poziciju
        odredjenu brojacem i; prilikom ucitavanja oslanjamo se na
133     svojstvo funkcije fscanf povratka EOF vrednosti kada stigne
        do kraja ulaza */
135
    i = 0;
137 while (fscanf(ulaz, "%d %d", &tacke[i].x, &tacke[i].y) != EOF) {
      i++;
139 }

141 /* Cuvamo broj procitanih tacaka */
    n = i;
143


145 /* Analiziramo zadatu opciju: kako ocekujemo da je argv[1]
       "-x" ili "-y" ili "-o" sigurni smo da je argv[1][0] crtica
147    (karakter -) i dalje proveravamo sta je na sledecoj
       poziciji tj. sta je argv[1][1] */
149
    switch (argv[1][1]) {
151 case 'x':
      /* Ako je u pitanju karakter x, pozivamo funkciju za
153       sortiranje po vrednosti x koordinate */
      sortiraj_po_x(tacke, n);
155   break;
    case 'y':
157   /* Ako je u pitanju karakter y, pozivamo funkciju za
         sortiranje po vrednosti y koordinate */
159   sortiraj_po_y(tacke, n);
      break;
161 case 'o':
      /* Ako je u pitanju karakter o, pozivamo funkciju za
```

```
163         sortiranje po udaljenosti od koorinatnog pocetka */
      sortiraj_po_rastojanju(tacke, n);
165   }

167   /* Upisujemo dobijeni niz u izlaznu datoteku */
      for (i = 0; i < n; i++) {
169     fprintf(izlaz, "%d %d\n", tacke[i].x, tacke[i].y);
      }
171
      /* Zatvaramo otvorene datoteke */
173   fclose(ulaz);
      fclose(izlaz);
175
      /* I zavrsavamo sa programom */
177   return 0;
}
```

Rešenje 3.20

```
#include <stdio.h>
2 #include <string.h>
#include <stdlib.h>
4
#define MAX 1000
6 #define MAX_DUZINA 16

8 typedef struct gr {
    char ime[MAX_DUZINA];
10  char prezime[MAX_DUZINA];
} Gradjanin;
12
/* Fja sortira niz gradjana rastuce po imenima */
14 void sort_ime(Gradjanin a[], int n)
{
16  int i, j;
    int min;
18  Gradjanin pom;

20  for (i = 0; i < n - 1; i++) {
      /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
22       nalazi najmanji od elemenata a[i].ime,...,a[n-1].ime. */
      min = i;
24    for (j = i + 1; j < n; j++)
        if (strcmp(a[j].ime, a[min].ime) < 0)
26        min = j;

28    /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
         samo ako su (i) i min razliciti, inace je nepotrebno. */
30    if (min != i) {
        pom = a[i];
32      a[i] = a[min];
        a[min] = pom;
34    }
    }
36 }
```

```
38 /* Fja sortira niz gradjana rastuce po prezimenima */
   void sort_prezime(Gradjanin a[], int n)
40 {

42   int i, j;
     int min;
44   Gradjanin pom;

46   for (i = 0; i < n - 1; i++) {
       /* Unutrasnja petlja pronalazi poziciju min, na kojoj se
48        nalazi najmanji od elemenata
          a[i].prezime,...,a[n-1].prezime. */
50     min = i;
       for (j = i + 1; j < n; j++)
52       if (strcmp(a[j].prezime, a[min].prezime) < 0)
           min = j;
54
       /* Zamena elemenata na pozicijama (i) i min. Ovo se radi
56        samo ako su (i) i min razliciti, inace je nepotrebno. */
       if (min != i) {
58       pom = a[i];
         a[i] = a[min];
60       a[min] = pom;
       }
62   }
   }
64
   /* Pretraga niza Gradjana */
66 int linearna_pretraga(Gradjanin a[], int n, Gradjanin * x)
   {
68   int i;
     for (i = 0; i < n; i++)
70     if (strcmp(a[i].ime, x->ime) == 0
           && strcmp(a[i].prezime, x->prezime) == 0)
72       return i;
     return -1;
74 }


76
   int main()
78 {
     Gradjanin spisak1[MAX], spisak2[MAX];
80   int isti_rbr = 0;
     int i, n;
82   FILE *fp = NULL;

84   /* Otvaranje datoteke */
     if ((fp = fopen("biracki-spisak.txt", "r")) == NULL) {
86     fprintf(stderr,
               "Neupesno otvaranje datoteke biracki-spisak.txt.\n");
88     exit(EXIT_FAILURE);
     }
90
     /* Citanje sadrzaja */
92   for (i = 0;
```

```
        fscanf(fp, "%s %s", spisak1[i].ime,
94              spisak1[i].prezime) != EOF; i++)
      spisak2[i] = spisak1[i];
96  n = i;

98  /* Zatvaranje datoteke */
    fclose(fp);
100
    sort_ime(spisak1, n);
102
    /************************************************
104   printf("Biracki spisak [uredjen prema imenima]:\n");
      for(i=0; i<n; i++)
106     printf(" %d. %s %s\n",i,spisak1[i].ime, spisak1[i].prezime);
    ************************************************/
108
    sort_prezime(spisak2, n);
110
    /************************************************
112   printf("Biracki spisak [uredjen prema prezimenima]:\n");
      for(i=0; i<n; i++)
114     printf(" %d. %s %s\n",i,spisak2[i].ime, spisak2[i].prezime);
    ************************************************/
116
    /* Linearno pretrazivanje nizova */
118 for (i = 0; i < n; i++)
      if (i == linearna_pretraga(spisak2, n, &spisak1[i]))
120     isti_rbr++;

122 /* Alternativno (efikasnije) resenje */
    /************************************************
124   for(i=0; i<n ;i++)
        if( strcmp(spisak2[i].ime, spisak1[i].ime) == 0 &&
126         strcmp(spisak1[i].prezime, spisak2[i].prezime)==0)
    isti_rbr++;
128 ************************************************/

130 /* Ispis rezultata */
    printf("%d\n", isti_rbr);
132
    exit(EXIT_SUCCESS);
134 }
```

Rešenje 3.22

```
#include <stdio.h>
2 #include <string.h>
#include <ctype.h>
4
#define MAX_BR_RECI 128
6 #define MAX_DUZINA_RECI 32


8
/* Funkcija koja izracunava broj suglasnika u reci */
10 int broj_suglasnika(char s[])
```

```
{
  char c;
  int i;
  int suglasnici = 0;
  /* Obilazimo karakter po karakter zadate niske */
  for (i = 0; s[i]; i++) {
    /* Ako je u pitanju slovo */
    if (isalpha(s[i])) {
      /* Pretvaramo ga u veliko da bismo mogli da pokrijemo
         slucaj i malih i velikih suglasnika */
      c = toupper(s[i]);
      /* Ukoliko slovo nije samoglasnik */
      if (c != 'A' && c != 'E' && c != 'I' && c != 'O'
          && c != 'U') {
        /* Uvecavamo broj suglasnika */
        suglasnici++;
      }
    }
  }
  /* Vracamo izracunatu vrednost */
  return suglasnici;
}

/* Funkcija koja sortira reci po zadatom kriterijumu - OPREZ:
   informacija o duzini reci se mora proslediti zbog pravilnog
   upravljanja memorijom */
void sortiraj_reci(char reci[][MAX_DUZINA_RECI], int n)
{
  int min, i, j, broj_suglasnika_j, broj_suglasnika_min,
      duzina_j, duzina_min;
  char tmp[MAX_DUZINA_RECI];
  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i; j < n; j++) {
      /* Prvo uporedjujemo broj suglasnika */
      broj_suglasnika_j = broj_suglasnika(reci[j]);
      broj_suglasnika_min = broj_suglasnika(reci[min]);
      if (broj_suglasnika_j < broj_suglasnika_min)
        min = j;
      else if (broj_suglasnika_j == broj_suglasnika_min) {
        /* Zatim, reci imaju isti broj suglasnika uporedjujemo
           duzine */
        duzina_j = strlen(reci[j]);
        duzina_min = strlen(reci[min]);

        if (duzina_j < duzina_min)
          min = j;
        else
          /* A ako reci imaju i isti broj suglasnika i iste
             duzine, uporedjujemo ih leksikografski */
        if (duzina_j == duzina_min
              && strcmp(reci[j], reci[min]) < 0)
          min = j;
      }
    }
    if (min != i) {
```

```
      strcpy(tmp, reci[min]);
      strcpy(reci[min], reci[i]);
      strcpy(reci[i], tmp);
    }
  }
}

int main()
{

  FILE *ulaz;
  int i = 0, n;

  /* Niz u kojem ce biti smestane reci. Prvi broj oznacava broj
     reci, a drugi maksimalnu duzinu pojedinacne reci */
  char reci[MAX_BR_RECI][MAX_DUZINA_RECI];

  /* Otvaramo datoteku niske.txt za citanje */
  ulaz = fopen("niske.txt", "r");
  if (ulaz == NULL) {
    fprintf(stderr,
            "Greska prilikom otvaranja datoteke niske.txt!\n");
    return 0;
  }

  /* Sve dok mozemo da procitamo sledecu rec */
  while (fscanf(ulaz, "%s", reci[i]) != EOF) {
    /* Proveravamo da li smo ucitali najvise dozvoljenih reci i
       ako jesmo, prekidamo ucitavanje */
    if (i == MAX_BR_RECI)
      break;
    /* Pripremamo brojac za narednu iteraciju */
    i++;
  }

  /* n je duzina naseg niza reci i predstavlja poslednju
     vrednost koriscenog brojaca */
  n = i;
  /* Pozivamo funkciju za sortiranje reci - OPREZ: nacin
     prosledjivanja niza reci */
  sortiraj_reci(reci, n);

  /* Ispisujemo sortirani niz reci */
  for (i = 0; i < n; i++) {
    printf("%s ", reci[i]);
  }
  printf("\n");

  /* Zatvaramo datoteku */
  fclose(ulaz);

  /* I zavrsavamo sa programom */
  return 0;
}
```

Rešenje 3.23

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

#define MAX_ARTIKALA 100000

typedef struct art {
  long kod;
  char naziv[20];
  char proizvodjac[20];
  float cena;
} Artikal;

int binarna_pretraga(Artikal a[], int n, long x)
{
  int levi = 0;
  int desni = n - 1;

  /* Dokle god je indeks levi levo od indeksa desni */
  while (levi <= desni) {
    /* Racunamo sredisnji indeks */
    int srednji = (levi + desni) / 2;
    /* Ako je sredisnji element veci od x, tada se x mora
       nalaziti u levoj polovini niza */
    if (x < a[srednji].kod)
      desni = srednji - 1;
    /* Ako je sredisnji element manji od x, tada se x mora
       nalaziti u desnoj polovini niza */
    else if (x > a[srednji].kod)
      levi = srednji + 1;
    else
      /* Ako je sredisnji element jednak x, tada smo pronasli x
         na poziciji srednji */
      return srednji;
  }
  /* Ako nije pronadjen vracamo -1 */
  return -1;
}

void selection_sort(Artikal a[], int n)
{
  int i, j;
  int min;
  Artikal pom;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++)
      /* Sortiranje vrsimo po kodovima i onda i poredjenje
         vrsimo nad kodovoima */
      if (a[j].kod < a[min].kod)
        min = j;

    if (min != i) {
```

```
55        pom = a[i];
          a[i] = a[min];
57        a[min] = pom;
        }
59    }
  }
61
  int main()
63{
    Artikal asortiman[MAX_ARTIKALA];
65  long kod;
    int i, n;
67  float racun;

69  FILE *fp = NULL;

71  /* Otvaranje datoteke */
    if ((fp = fopen("artikli.txt", "r")) == NULL) {
73    fprintf(stderr,
              "Neuspesno otvaranje datoteke artikli.txt.\n");
75    exit(EXIT_FAILURE);
    }
77
    /* Ucitavanje artikala */
79  i = 0;
    while (fscanf(fp, "%ld %s %s %f", &asortiman[i].kod,
81                asortiman[i].naziv, asortiman[i].proizvodjac,
                  &asortiman[i].cena) == 4)
83    i++;

85  /* Zatvaranje datoteke */
    fclose(fp);
87
    n = i;
89
    /* Sortiracemo celokupan asortiman prodavnice prema kodovima
91     jer ce pri kucanju racuna prodavac unositi kod artikla.
       Prilikom kucanja svakog racuna pretrazuje se asortiman, da
93     bi se utvrdila cena artikla. Kucanje racuna obuhvata vise
       pretraga asortimana i u interesu nam je da ta operacija
95     bude sto efikasnija. Zelimo da koristimo algoritam binarne
       pretrage priliko pretrazivanje po kodu artikla. Iz tog
97     razloga, potrebno je da nam asortiman bude sortiran po
       kodovima i to cemo uraditi primenom selection sort
99     algoritma. Sortiramo samo jednom na pocetku, ali zato posle
       brzo mozemo da pretrazujemo prilikom kucanja proizvoljno
101    puno racuna. Vreme koje se utrosi na sortiranje na pocetku
       izvrsavanja programa, kasnije se isplati jer za brojna
103    trazenja artikla mozemo umesto linearne da koristimo
       efikasniju binarnu pretragu. */
105
    selection_sort(asortiman, n);
107
    printf
109     ("Asortiman:\nKOD                 Naziv artikla     Ime
     proizvodjaca       Cena\n");
```

```
  for (i = 0; i < n; i++)
    printf("%10ld %20s %20s %12.2f\n", asortiman[i].kod,
           asortiman[i].naziv, asortiman[i].proizvodjac,
           asortiman[i].cena);

  kod = 0;
  while (1) {
    printf("----------------------------\n");
    printf("- Za kraj za kraj rada kase, pritisnite CTRL+D!\n");
    printf("- Za nov racun unesite kod artikla!\n\n");
    if (scanf("%ld", &kod) == EOF)
      break;

    racun = 0;
    while (1) {
      if ((i = binarna_pretraga(asortiman, n, kod)) == -1) {
        printf
            ("\tGRESKA: Ne postoji proizvod sa trazenim kodom!\n");
      } else {
        printf("\tTrazili ste:\t%s %s %12.2f\n",
               asortiman[i].naziv, asortiman[i].proizvodjac,
               asortiman[i].cena);

        racun += asortiman[i].cena;
      }

      printf("Unesite kod artikla [ili 0 za prekid]: \t");
      scanf("%ld", &kod);

      if (kod == 0)
        break;
    }

    printf("\n\tUKUPNO: %.2lf dinara.\n\n", racun);
  }

  printf("Kraj rada kase!\n");

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.25

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

#define MAX 500

/* Struktura koja nam je neophodna za sve informacije o
   pojedinacnom studentu */
typedef struct {
  char ime[20];
  char prezime[25];
  int prisustvo;
```

```
  int zadaci;
} Student;

/* Funkcija kojom sortiramo niz struktura po prezimenu
   leksikografski rastuce */
void sort_ime_leksikografski(Student niz[], int n)
{
  int i, j;
  int min;
  Student pom;

  for (i = 0; i < n - 1; i++) {
    min = i;
    for (j = i + 1; j < n; j++)
      if (strcmp(niz[j].ime, niz[min].ime) < 0)
        min = j;

    if (min != i) {
      pom = niz[min];
      niz[min] = niz[i];
      niz[i] = pom;
    }
  }
}

/* Funkcija kojom sortiramo niz struktura po ukupnom broju
   uradjenih zadataka opadajuce, ukoliko neki studenti imaju
   isti broj uradjenih zadataka sortiraju se po duzini imena
   rastuce. */
void sort_zadatke_pa_imena(Student niz[], int n)
{
  int i, j;
  int max;
  Student pom;
  for (i = 0; i < n - 1; i++) {
    max = i;
    for (j = i + 1; j < n; j++)
      if (niz[j].zadaci > niz[max].zadaci)
        max = j;
      else if (niz[j].zadaci == niz[max].zadaci
               && strlen(niz[j].ime) < strlen(niz[max].ime))
        max = j;
    if (max != i) {
      pom = niz[max];
      niz[max] = niz[i];
      niz[i] = pom;
    }
  }
}

 /* Funkcija kojom sortiramo niz struktura po broju casova na
    kojima su bili opadajuce, a ukoliko * neki studenti imaju
    isti broj casova, sortiraju se opadajuce po broju uradjenih
    zadataka, * a ukoliko se i po broju zadataka poklapaju
    sortirati ih po prezimenu opadajuce. */
void sort_prisustvo_pa_zadatke_pa_prezimena(Student niz[], int n)
```

```
{
70   int i, j;
   int max;
72   Student pom;
   for (i = 0; i < n - 1; i++) {
74     max = i;
     for (j = i + 1; j < n; j++)
76       if (niz[j].prisustvo > niz[max].prisustvo)
           max = j;
78       else if (niz[j].prisustvo == niz[max].prisustvo
                  && niz[j].zadaci > niz[max].zadaci)
80         max = j;
       else if (niz[j].prisustvo == niz[max].prisustvo
82                  && niz[j].zadaci == niz[max].zadaci
                    && strcmp(niz[j].prezime, niz[max].prezime) > 0)
84         max = j;
     if (max != i) {
86       pom = niz[max];
       niz[max] = niz[i];
88       niz[i] = pom;
     }
90   }
}
92


94
int main(int argc, char *argv[])
96{
   Student praktikum[MAX];
98   int i, br_studenata = 0;

100  FILE *fp = NULL;

102  /* Otvaranje datoteke za citanje */
   if ((fp = fopen("aktivnost.txt", "r")) == NULL) {
104    fprintf(stderr,
             "Neupesno otvaranje datoteke aktivnost.txt.\n");
106    exit(EXIT_FAILURE);
   }
108
   /* Ucitavanje sadrzaja */
110  for (i = 0;
        fscanf(fp, "%s%s%d%d", praktikum[i].ime,
112             praktikum[i].prezime, &praktikum[i].prisustvo,
             &praktikum[i].zadaci) != EOF; i++);
114
   fclose(fp);
116  br_studenata = i;

118  /* Kreiramo prvi spisak studenata na kom su sortirani
      leksikografski po imenu rastuce */
120  sort_ime_leksikografski(praktikum, br_studenata);
   /* Otvaranje datoteke za pisanje */
122  if ((fp = fopen("dat1.txt", "w")) == NULL) {
     fprintf(stderr, "Neupesno otvaranje datoteke dat1.txt.\n");
124    exit(EXIT_FAILURE);
```

```
  }
  fprintf
      (fp,
       "Studenti sortirani po imenu leksikografski rastuce:\n");
  for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
  fclose(fp);

  /* Na drugom su sortirani po ukupnom broju uradjenih zadataka
     opadajuce, ukoliko neki studenti imaku isti broj uradjenih
     zadataka sortiraju se po duzini imena rastuce. */
  sort_zadatke_pa_imena(praktikum, br_studenata);
  /* Otvaranje datoteke za pisanje */
  if ((fp = fopen("dat2.txt", "w")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke dat2.txt.\n");
    exit(EXIT_FAILURE);
  }
  fprintf(fp,
          "Studenti sortirani po broju zadataka opadajuce,\n");
  fprintf(fp, "pa po duzini imena rastuce:\n");
  for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
  fclose(fp);

  /* Na trecem spisku su sortirani po broju casova na kojima su
     bili opadajuce, a ukoliko neki studenti imaju isti broj
     casova, sortiraju se opadajuce po broju uradjenih zadataka,
     a ukoliko se i po broju zadataka poklapaju sortirati ih po
     prezimenu opadajuce. */
  sort_prisustvo_pa_zadatke_pa_prezimena(praktikum,
                                         br_studenata);
  /* Otvaranje datoteke za pisanje */
  if ((fp = fopen("dat3.txt", "w")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke dat3.txt.\n");
    exit(EXIT_FAILURE);
  }
  fprintf(fp, "Studenti sortirani po prisustvu opadajuce,\n");
  fprintf(fp, "pa po broju zadataka,\n");
  fprintf(fp, "pa po prezimenima leksikografski opadajuce:\n");
  for (i = 0; i < br_studenata; i++)
    fprintf(fp, "%s %s  %d  %d\n", praktikum[i].ime,
            praktikum[i].prezime, praktikum[i].prisustvo,
            praktikum[i].zadaci);
  fclose(fp);

  return 0;
}
```

Rešenje 3.26

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

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

/* Struktura koja opisuje jednu pesmu */
typedef struct {
  char *izvodjac;
  char *naslov;
  int broj_gledanja;
} Pesma;

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

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

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

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

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

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


int main(int argc, char *argv[])
{
  FILE *ulaz;
  Pesma *pesme;                 /* Pokazivac na deo memorije za
                                   cuvanje pesama */
  int alocirano_za_pesme;       /* Broj mesta alociranih za
                                   pesme */
  int i;                        /* Redni broj pesme cije se
                                   informacije citaju */
  int n;                        /* Ukupan broj pesama */
  int j, k;
  char c;
  int alocirano;                /* Broj mesta alociranih za
                                   propratne informacije o
```

```
58                                      pesmama */
   int broj_gledanja;
60
   /* Pripremamo datoteku za citanje */
62 ulaz = fopen("pesme_bez_pretpostavki.txt", "r");
   if (ulaz == NULL) {
64   printf
         ("Problem sa citanjem datoteke pesme_bez_pretpostavki.txt!\n
     ");
66   return 0;
   }
68
   /* Citamo informacije o pesmama */
70 pesme = NULL;
   alocirano_za_pesme = 0;
72 i = 0;

74 while (1) {

76   /* Proveravamo da li smo stigli do kraja datoteke */
     c = fgetc(ulaz);
78   if (c == EOF) {
       /* Ako smo dobili kao rezultat EOF, jesmo, nema vise
80        sadrzaja za citanje */
       break;
82   } else {
       /* Ako nismo, vracamo procitani karakter nazad */
84     ungetc(c, ulaz);
     }
86

88   /* Proveravamo da li imamo dovoljno memorije za citanje nove
        pesme */
90   if (alocirano_za_pesme == i) {

92     /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
          alociramo novih KORAK mesta */
94     alocirano_za_pesme += KORAK;
       pesme =
96         (Pesma *) realloc(pesme,
                             alocirano_za_pesme * sizeof(Pesma));
98
       /* Proveravamo da li je nova memorija uspesno realocirana */
100    if (pesme == NULL) {
         /* Ako nije ... */
102      /* Ispisujemo obavestenje */
         printf("Problem sa alokacijom memorije!\n");
104
         /* I oslobadjamo svu memoriju zauzetu do ovog koraka */
106      for (k = 0; k < i; k++) {
           free(pesme[k].izvodjac);
108        free(pesme[k].naslov);
         }
110      free(pesme);
         return 0;
112    }
```

```
114     }
116     /* Ako jeste, nastavljamo sa citanjem pesama ... */
        /* Citamo ime izvodjaca */
118
        j = 0;                          /* Oznacava poziciju na koju
120                                        treba smestiti procitani
                                           karakter */
122     alocirano = 0;                  /* Oznacava broj alociranih
                                           mesta */
124     pesme[i].izvodjac = NULL;       /* Memorija koju mozemo
                                           koristiti za smestanje
126                                        procitanih karaktera */

128     /* Sve dok ne stignemo do prve beline u liniji - beline koja
           se nalazi nakon imena izvodjaca - citamo karaktere iz
130        datoteke */
        while ((c = fgetc(ulaz)) != ' ') {
132
          /* Proveravamo da li imamo dovoljno memorije za smestanje
134           procitanog karaktera */
          if (j == alocirano) {
136
            /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
138              alociramo novih KORAK mesta */
            alocirano += KORAK;
140         pesme[i].izvodjac =
                (char *) realloc(pesme[i].izvodjac,
142                              alocirano * sizeof(char));

144         /* Proveravamo da li je nova alokacija uspesna */
            if (pesme[i].izvodjac == NULL) {
146           /* Ako nije... */
              /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
148           for (k = 0; k < i; k++) {
                free(pesme[k].izvodjac);
150             free(pesme[k].naslov);
              }
152           free(pesme);
              /* I prekidamo sa izvrsavanjem programa */
154           return 0;
            }
156
          }
158       /* Ako imamo dovoljno memorije, smestamo procitani
             karakter */
160       pesme[i].izvodjac[j] = c;
          j++;
162       /* I nastavljamo sa citanjem */
        }
164
        /* Upisujemo terminirajucu nulu na kraju reci */
166     pesme[i].izvodjac[j] = '\0';

168
```

```
    /* Citamo - */
170 fgetc(ulaz);

172 /* Citamo razmak */
    fgetc(ulaz);
174


176 /* Citamo naslov pesme */
    j = 0;                      /* Oznacava poziciju na koju
178                                treba smestiti procitani
                                   karakter */
180 alocirano = 0;              /* Oznacava broj alociranih
                                   mesta */
182 pesme[i].naslov = NULL;     /* Memorija koju mozemo
                                   koristiti za smestanje
184                                procitanih karaktera */

186 /* Sve dok ne stignemo do zareza - zareza koji se nalazi
       nakon naslova pesme - citamo karaktere iz datoteke */
188
    while ((c = fgetc(ulaz)) != ',') {
190   /* Proveravamo da li imamo dovoljno memorije za smestanje
         procitanog karaktera */
192   if (j == alocirano) {
        /* Ako nemamo, ako smo potrosili svu alociranu memoriju,
194          alociramo novih KORAK mesta */
        alocirano += KORAK;
196     pesme[i].naslov =
            (char *) realloc(pesme[i].naslov,
198                          alocirano * sizeof(char));

200     /* Proveravamo da li je nova alokacija uspesna */
        if (pesme[i].naslov == NULL) {
202       /* Ako nije... */
          /* Oslobadjamo svu memoriju zauzetu do ovog koraka */
204       for (k = 0; k < i; k++) {
            free(pesme[k].izvodjac);
206         free(pesme[k].naslov);
          }
208       free(pesme[i].izvodjac);
          free(pesme);
210
          /* I prekidamo izvrsavanje programa */
212       return 0;
        }
214
      }
216   /* Ako imamo dovoljno memorije, smestamo procitani
         karakter */
218   pesme[i].naslov[j] = c;
      j++;
220   /* I nastavljamo dalje sa citanjem */
    }
222 /* Upisujemo terminirajucu nulu na kraju reci */
    pesme[i].naslov[j] = '\0';
224
```

```
    /* Citamo razmak */
    fgetc(ulaz);

    /* Citamo broj gledanja */

    broj_gledanja = 0;

    /* Sve dok ne stignemo do znaka za novi red - kraja linije -
       citamo karaktere iz datoteke */
    while ((c = fgetc(ulaz)) != '\n') {
      broj_gledanja = broj_gledanja * 10 + (c - '0');
    }
    pesme[i].broj_gledanja = broj_gledanja;

    /* Prelazimo na citanje sledece pesme */
    i++;

  }

  /* Cuvamo informaciju o broju pesama koje smo procitali */
  n = i;

  /* Zatvaramo datoteku jer nam nece vise trebati */
  fclose(ulaz);

  /* Analiziramo argumente komandne linije */
  if (argc == 1) {

    /* Nema dodatnih opcija - sortiramo po broju gledanja */
    qsort(pesme, n, sizeof(Pesma), &uporedi_gledanost);
  } else {

    if (argc == 2 && strcmp(argv[1], "-n") == 0) {
      /* Sortiramo po naslovu */
      qsort(pesme, n, sizeof(Pesma), &uporedi_naslove);
    } else {
      if (argc == 2 && strcmp(argv[1], "-i") == 0) {
        /* Sortiramo po izvodjacu */
        qsort(pesme, n, sizeof(Pesma), &uporedi_izvodjace);
      } else {
        printf("Nedozvoljeni argumenti!\n");
        free(pesme);
        return 0;
      }
    }
  }

  /* Ispisujemo rezultat */
  for (i = 0; i < n; i++) {
    printf("%s - %s, %d\n", pesme[i].izvodjac, pesme[i].naslov,
           pesme[i].broj_gledanja);
  }

  /* Oslobadjamo memoriju */
  for (i = 0; i < n; i++) {
```

```
    free(pesme[i].izvodjac);
    free(pesme[i].naslov);
  }
  free(pesme);

  /* Prekidamo izvrsavanje programa */
  return 0;
}
```

Rešenje 3.28

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

#define MAX 100

/* Program demonstrira upotrebu funkcija qsort() i bsearch().
   Ove funkcije su definisane u standardnoj biblioteci i za
   njihovo koriscenje dovoljno je ukljuciti zaglavlje stdlib.h.
   Funkcija qsort() vrsi sortiranje niza metodom sortiranja
   razdvajanjem. Deklaracija ove funkcije je sledeca:

   void qsort(void *b, int n, int s, int(*comp)(const void *,
   const void *));

   Prvi argument je adresa pocetka niza koji se sortira. S
   obzirom da se ne zna tacan tip elemenata niza, koristi se
   genericki pokazivac (void *). Drugi argument je broj
   elemenata niza, a treci velicina svakog od elemenata niza.
   Poslednji argument je pokazivac na funkciju poredjenja. Ova
   funkcija treba da prihvata adrese elemenata niza koji se
   porede, i da vraca >0 ako je prvi element veci, <0 ako je
   prvi element manji, a vraca 0 ako su elementi koji se porede
   jednaki. Na ovaj nacin se moze sortirati bilo koji niz,
   dovoljno je da je na neki nacin funkcijom poredjenja
   definisan potpuni poredak medju elementima niza. Argumenti
   funkcije poredjenja su takodje genericki pokazivaci, opet
   zato sto ne znamo tacno kog su tipa elementi niza. Ovi
   pokazivaci su jos kvalifikovani kljucnom recju const.

   Funkcija bsearch() vrsi pretrazivanje sortiranog niza metodom
   binarne pretrage. Funkcija ima sledecu deklaraciju:

   void *bsearch(const void *x, const void *b, int n, int s, int
   (*comp)(const void *, const void *));

   Prvi argument je pokazivac na podatak koji se trazi u nizu.
   Drugi argument je adresa pocetka niza, treci velicina niza, a
   cetvrti velicina elementa niza. Poslednji argument je
   pokazivac na funkciju poredjenja koja definise poredak u
   skladu sa kojim je sortiran niz. Funkcija vraca adresu
   elementa u nizu koji je jednak trazenom elementu, ili NULL
   ukoliko element nije pronadjen.
```

```
45
     Osim ovih funkcija, postoji funkcija za linearnu pretragu.
47
     void *lfind(const void *x, void *b, int *n, int s, int
49   (*comp)(const void *, const void *));

51   ima potpuno iste argumente kao i bsearch() -- jedina razlika
     je u tome sto se kao treci argument ne predaje duzina niza vec
53   adresa celobrojne promenljive u kojoj se nalazi duzina niza.
     Funkcija za uporedjivanja elemenata na koju pokazuje pokazivac
55   comp treba da zadovoljava ista pravila kao i kod prethodnih
     funkcija. Medjutim, s obzirom da se kod linearne pretrage
57   koristi iskljucivo poredjenje na jednakost, dovoljno je da
     funkcija za uporedjivanje vraca 0, ako su objekti koji se
59   uporedjuju jednaki, a razlicitu vrednost od nule u suprotnom.
     Funkcija vraca pokazivac na pronadjeni element ili NULL u
61   slucaju neuspesne pretrage. */

63 /* NAPOMENA: Pokazivac na void je pokazivac koji moze sadrzati
     adresu bilo kog podatka u memoriji. Svaki pokazivac se moze
65   bez konverzije dodeliti ovom pokazivacu, kao i obrnuto. Moze
     se koristiti za cuvanje adrese podatka za koji unapred ne
67   znamo kog ce biti tipa. Njegova glavna karakteristika je da
     se ne moze dereferencirati, zato sto se ne zna kog je tipa
69   ono na sta on pokazuje. Programer mora na drugi nacin da
     utvrdi kog je tipa podatak na toj adresi, pa da najpre
71   konvertuje void pokazivac u odgovarajuci tip pokazivaca, a
     zatim da ga tako konvertovanog dereferencira. */
73
   /* Funkcija poredi dva cela broja */
75 int compare_int(const void *a, const void *b)
   {
77   /* Konvertujemo void pokazivace u int pokazivace koje zatim
        dereferenciramo, dobijamo int-ove koje oduzimamo i razliku
79      vracamo. */

81   /* Zbog moguceg prekoracenja opsega celih brojeva necemo ih
        oduzimati return *((int *)a) - *((int *)b); */
83
     int b1 = *((int *) a);
85   int b2 = *((int *) b);

87   if (b1 > b2)
       return 1;
89   else if (b1 < b2)
       /* Ovo uredjenje favorizujemo jer zelimo rastuci poredak */
91     return -1;
     else
93     return 0;
   }
95
   int compare_int_desc(const void *a, const void *b)
97 {
     /* Za obrnuti poredak mozemo samo oduzimati a od b */
99   /* return *((int *)b) - *((int *)a); */
```

```
101   /* Ili samo promeniti znak vrednosti koju koju vraca prethodna
          funkcija */
103   return -compare_int(a, b);
}
105
/* Test program */
107 int main()
{
109   size_t n;
      int i, x;
111   int a[MAX], *p = NULL;

113   /* Unosimo dimenziju */
      printf("Uneti dimenziju niza: ");
115   scanf("%ld", &n);
      if (n > MAX)
117     n = MAX;

119   /* Unosimo elemente niza */
      printf("Uneti elemente niza:\n");
121   for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
123
      /* Sortiramo niz celih brojeva */
125   qsort(a, n, sizeof(int), &compare_int);

127   /* Prikazujemo sortirani niz */
      printf("Sortirani niz u rastucem poretku:\n");
129   for (i = 0; i < n; i++)
        printf("%d ", a[i]);
131   putchar('\n');

133   /* Pretrazivanje niza */
      /* Vrednost koju cemo traziti u nizu */
135   printf("Uneti element koji se trazi u nizu: ");
      scanf("%d", &x);
137
      printf("Binarna pretraga: \n");
139   p = bsearch(&x, a, n, sizeof(int), &compare_int);

141   if (p == NULL)
        printf("Elementa nema u nizu!\n");
143   else
        printf("Element je nadjen na poziciji %ld\n", p - a);
145
      printf("Linearna pretraga (lfind): \n");
147   p = lfind(&x, a, &n, sizeof(int), &compare_int);
      if (p == NULL)
149     printf("Elementa nema u nizu!\n");
      else
151     printf("Element je nadjen na poziciji %ld\n", p - a);

153   return 0;
}
```

Rešenje 3.29

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

#define MAX 100

/* Funkcija racuna broj delilaca broja x */
int no_of_deviders(int x)
{
  int i;
  int br;

  /* Ako je argument negativan broj menjamo mu znak */
  if (x < 0)
    x = -x;
  if (x == 0)
    return 0;
  if (x == 1)
    return 1;
  /* Svaki broj veci od 1 ima bar 2 delioca, (1 i samog sebe) */
  br = 2;
  /* Sve dok je */
  for (i = 2; i < sqrt(x); i++)
    if (x % i == 0)
      /* Ako i deli x onda su delioci: i, x/i */
      br += 2;
  /* Ako je broj bas kvadrat, onda smo iz petlje izasli kada je
     i bilo bas jednako korenu od x, tada x ima jos jednog
     delioca */
  if (i * i == x)
    br++;

  return br;
}

/* Funkcija poredjenja dva cela broja po broju delilaca */
int compare_no_deviders(const void *a, const void *b)
{
  int ak = *(int *) a;
  int bk = *(int *) b;
  int n_d_a = no_of_deviders(ak);
  int n_d_b = no_of_deviders(bk);

  if (n_d_a > n_d_b)
    return 1;
  else if (n_d_a < n_d_b)
    return -1;
  else
    return 0;
}

/* Test program */
int main()
```

```
{
  size_t n;
  int i;
  int a[MAX];

  /* Unosimo dimenziju */
  printf("Uneti dimenziju niza: ");
  scanf("%ld", &n);
  if (n > MAX)
    n = MAX;

  /* Unosimo elemente niza */
  printf("Uneti elemente niza:\n");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  /* Sortiramo niz celih brojeva prema broju delilaca */
  qsort(a, n, sizeof(int), &compare_no_deviders);

  /* Prikazujemo sortirani niz */
  printf
      ("Sortirani niz u rastucem poretku prema broju delilaca:\n");
  for (i = 0; i < n; i++)
    printf("%d ", a[i]);
  putchar('\n');

  return 0;
}
```

Rešenje 3.30

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>
#define MAX_NISKI 1000
#define MAX_DUZINA 30

/* Naredne dve funkcije bice koriscenje za sortiranje niza
   nizova karaktera. Svaka od njih ce biti pozivana za
   poredjenje dva elementa takvog niza, tj dva niza karaktera.
   Prilikom poziva ove funkcije za poredjenje, npr i-tog i j-tog
   elementa slale bi se, kao i inace adrese elemenata. Kako
   adresa niza i samo ime niza imaju istu vrednost, pa i slanje
   &niske[i] u pozivu funkcije je isto kao da se salje niske[i]
   i nije neophodno dalje dereferenciranje da bi se doslo do
   i-te niske niza. */

/* Funkcija koju koristimo za poredjenje na dve niske iz niza a
   i b su const void pokazivaci, ali kako mi znamo da cemo ovu
   funkciju koristiti za poredjenje dve niske iz niza
   eksplicitno im menjamo tipove u (char *) To je neophodno jer
   poredimo niske leksikografski i funkciji strcmp moramo
   proslediti bas char* pokazivace, da bi uporedila 2 niske. */
int poredi_leksikografski(const void *a, const void *b)
```

```
{
26   return strcmp((char *) a, (char *) b);
}
28
/* Funkcija koju cemo prosledjivati za sortiranje niski po
30    duzini. Dakle a i b ce biti uvek adrese niski koje se porede.
    Menjamo im, eksplicitno, tip u char* jer nam treba duzina
32    svake od niski, a to cemo dobiti pozivom funkcije strlen koja
    ocekuje bas char * pokazivac. */
34 int poredi_duzine(const void *a, const void *b)
{
36   return strlen((char *) a) - strlen((char *) b);
}
38
int main()
40 {
   int i;
42   size_t n;
   FILE *fp = NULL;
44   char niske[MAX_NISKI][MAX_DUZINA];
   char *p = NULL;
46   char x[MAX_DUZINA];

48   /* Otvaranje datoteke */
   if ((fp = fopen("niske.txt", "r")) == NULL) {
50     fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
     exit(EXIT_FAILURE);
52   }

54   /* Citanje sadrzaja datoteke */
   for (i = 0; fscanf(fp, "%s", niske[i]) != EOF; i++);
56
   /* Zatvaranje datoteke */
58   fclose(fp);
   n = i;
60
   /* Sortiramo niske leksikografski, tako sto biblioteckoj
62      funkciji qsort prosledjujemo funkciju kojom se zadaje
      kriterijum poredjenja 2 niske po duzini */
64   qsort(niske, n, MAX_DUZINA * sizeof(char),
        &poredi_leksikografski);
66
   printf("Leksikografski sortirane niske:\n");
68   for (i = 0; i < n; i++)
     printf("%s ", niske[i]);
70   printf("\n");

72   /* Unosimo trazeni nisku */
   printf("Uneti trazenu nisku: ");
74   scanf("%s", x);

76   /* Binarna pretraga */
   /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
78      jer nam je niz sortiran leksikografski. */
   p = bsearch(&x, niske, n, MAX_DUZINA * sizeof(char),
80              &poredi_leksikografski);
```

```
82    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
84             p, (p - (char *) niske) / MAX_DUZINA);
      else
86      printf("Niska nije pronadjena u nizu\n");

88    /* Linearna pretraga */
      /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
90       jer nam je niz sortiran leksikografski. */
      p = lfind(&x, niske, &n, MAX_DUZINA * sizeof(char),
92              &poredi_leksikografski);

94    if (p != NULL)
        printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
96             p, (p - (char *) niske) / MAX_DUZINA);
      else
98      printf("Niska nije pronadjena u nizu\n");

100   /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
         funkciju poredjenja */
102   qsort(niske, n, MAX_DUZINA * sizeof(char), &poredi_duzine);

104   printf("Niske sortirane po duzini:\n");
      for (i = 0; i < n; i++)
106     printf("%s ", niske[i]);
      printf("\n");
108
      exit(EXIT_SUCCESS);
110 }
```

Rešenje 3.31

```
#include <stdio.h>
2 #include <stdlib.h>
#include <string.h>
4 #include <search.h>
#define MAX_NISKI 1000
6 #define MAX_DUZINA 30

8 /* Fje poredjenja za qsort */
int poredi_leksikografski(const void *a, const void *b)
10 {
  return strcmp(*(char **) a, *(char **) b);
12 }

14 int poredi_duzine(const void *a, const void *b)
{
16   return strlen(*(char **) a) - strlen(*(char **) b);
}
18
/* Fja poredjenja za bsearch */
20 int poredi_leksikografski_b(const void *a, const void *b)
{
22   return strcmp((char *) a, *(char **) b);
```

```
}

int main()
{
  int i;
  size_t n;
  FILE *fp = NULL;
  char *niske[MAX_NISKI];
  char **p = NULL;
  char x[MAX_DUZINA];

  /* Otvaranje datoteke */
  if ((fp = fopen("niske.txt", "r")) == NULL) {
    fprintf(stderr, "Neupesno otvaranje datoteke niske.txt.\n");
    exit(EXIT_FAILURE);
  }

  /* Citanje sadrzaja datoteke */
  i = 0;
  while (fscanf(fp, "%s", x) != EOF) {
    /* Alociranje dovoljne memorije */
    if ((niske[i] = malloc(strlen(x) * sizeof(char))) == NULL) {
      fprintf(stderr, "Greska pri alociranju niske\n");
      exit(EXIT_FAILURE);
    }
    /* Kopiranje procitane niske na svoje mesto */
    strcpy(niske[i], x);
    i++;
  }

  /* Zatvaranje datoteke */
  fclose(fp);
  n = i;

  /* Sortiramo niske leksikografski, tako sto biblioteckoj
     funkciji qsort prosledjujemo funkciju kojom se zadaje
     kriterijum poredjenja 2 niske po duzini */
  qsort(niske, n, sizeof(char *), &poredi_leksikografski);

  printf("Leksikografski sortirane niske:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  /* Unosimo trazeni nisku */
  printf("Uneti trazenu nisku: ");
  scanf("%s", x);

  /* Binarna pretraga */
  /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
     jer nam je niz sortiran leksikografski. */
  p = bsearch(&x, niske, n, sizeof(char *),
              &poredi_leksikografski_b);

  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
```

```
            *p, p - niske);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Linearna pretraga */
  /* Prosledjujemo pokazivac na funkciju poredi_leksikografski
     jer nam je niz sortiran leksikografski. */
  p = lfind(&x, niske, &n, sizeof(char *),
            &poredi_leksikografski_b);

  if (p != NULL)
    printf("Niska \"%s\" je pronadjena u nizu na poziciji %ld\n",
            *p, p - niske);
  else
    printf("Niska nije pronadjena u nizu\n");

  /* Sada ih sortiramo po duzini i ovaj put saljemo drugu
     funkciju poredjenja */
  qsort(niske, n, sizeof(char *), &poredi_duzine);

  printf("Niske sortirane po duzini:\n");
  for (i = 0; i < n; i++)
    printf("%s ", niske[i]);
  printf("\n");

  /* Oslobadjanje zauzete memorije */
  for (i = 0; i < n; i++)
    free(niske[i]);

  exit(EXIT_SUCCESS);
}
```

Rešenje 3.32

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

#define MAX 500

/* Struktura koja nam je neophodna za sve informacije o
   pojedinacnom studentu */
typedef struct {
  char ime[21];
  char prezime[21];
  int bodovi;
} Student;

/* Funkcija poredjenja koju cemo koristiti za sortiranje po
   broju bodova, a studente sa istim brojevem bodova dodatno
   sortiramo leksikografski po prezimenu */
int compare(const void *a, const void *b)
{
  Student *prvi = (Student *) a;
```

```
22    Student *drugi = (Student *) b;

24    if (prvi->bodovi > drugi->bodovi)
        return -1;
26    else if (prvi->bodovi < drugi->bodovi)
        return 1;
28    else
        /* Jednaki su po broju bodova, treba ih uporediti po
30         prezimenu */
        return strcmp(prvi->prezime, drugi->prezime);
32  }

34  /* Funkcija za poredjenje koje ce porediti samo po broju bodova
       Prvi parametar je ono sto trazimo u nizu, ovde je to broj
36     bodova, a drugi parametar ce biti element niza ciji se bodovi
       porede. */
38  int compare_za_bsearch(const void *a, const void *b)
    {
40    int bodovi = *(int *) a;
      Student *s = (Student *) b;
42    return s->bodovi - bodovi;
    }
44
    /* Funkcija za poredjenje koje ce porediti samo po prezimenu
46     Prvi parametar je ono sto trazimo u nizu, ovde je to prezime,
       * a drugi parametar ce biti element niza cije se prezime
48     poredi. */
    int compare_za_linearna_prezimena(const void *a, const void *b)
50  {
      char *prezime = (char *) a;
52    Student *s = (Student *) b;
      return strcmp(prezime, s->prezime);
54  }


56
    int main(int argc, char *argv[])
58  {
      Student kolokvijum[MAX];
60    int i;
      size_t br_studenata = 0;
62    Student *nadjen = NULL;
      FILE *fp = NULL;
64    int bodovi;
      char prezime[21];
66
      /* Ako je program pozvan sa nedovoljnim brojem argumenata
68       informisemo korisnika kako se program koristi i prekidamo
         izvrsavanje. */
70    if (argc < 2) {
        fprintf(stderr,
72              "Program se poziva sa:\n%s datoteka_sa_rezultatima\n",
                argv[0]);
74      exit(EXIT_FAILURE);
      }
76
      /* Otvaranje datoteke */
```

```
78  if ((fp = fopen(argv[1], "r")) == NULL) {
      fprintf(stderr, "Neupesno otvaranje datoteke %s\n", argv[1]);
80    exit(EXIT_FAILURE);
    }
82
    /* Ucitavanje sadrzaja */
84  for (i = 0;
         fscanf(fp, "%s%s%d", kolokvijum[i].ime,
86             kolokvijum[i].prezime,
               &kolokvijum[i].bodovi) != EOF; i++);
88
    /* Zatvaranje datoteke */
90  fclose(fp);
    br_studenata = i;
92
    /* Sortiramo niz studenata po broju bodova, pa unutar grupe
94     studenata sa istim brojem bodova sortiranje se vrsi po
       prezimenu */
96  qsort(kolokvijum, br_studenata, sizeof(Student), &compare);

98  printf("Studenti sortirani po broju poena opadajuce, ");
    printf("pa po prezimenu rastuce:\n");
100 for (i = 0; i < br_studenata; i++)
      printf("%s %s  %d\n", kolokvijum[i].ime,
102          kolokvijum[i].prezime, kolokvijum[i].bodovi);

104 /* Pretrazivanje studenata po broju bodova se vrsi binarnom
       pretragom jer je niz sortiran po broju bodova. */
106 printf("Unesite broj bodova: ");
    scanf("%d", &bodovi);
108
    nadjen =
110     bsearch(&bodovi, kolokvijum, br_studenata, sizeof(Student),
              &compare_za_bsearch);
112
    if (nadjen != NULL)
114   printf
        ("Pronadjen je student sa unetim  brojem bodova: %s %s %d\n"
      ,
116      nadjen->ime, nadjen->prezime, nadjen->bodovi);
    else
118   printf("Nema studenta sa unetim brojem bodova\n");

120 /* Pretraga po prezimenu se mora vrsiti linearnom pretragom
       jer nam je niz sortiran po bodovima, globalno gledano. */
122 printf("Unesite prezime: ");
    scanf("%s", prezime);
124
    nadjen =
126     lfind(prezime, kolokvijum, &br_studenata, sizeof(Student),
            &compare_za_linearna_prezimena);
128
    if (nadjen != NULL)
130   printf
        ("Pronadjen je student sa unetim prezimenom: %s %s %d\n",
132      nadjen->ime, nadjen->prezime, nadjen->bodovi);
```

```
  else
134   printf("Nema studenta sa unetim prezimenom\n");

136 return 0;
}
```

Rešenje 3.33

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

5 #define MAX 128

7 /* Funkcija poredi dva karaktera */
int uporedi_char(const void *pa, const void *pb)
9 {
  return *(char *) pa - *(char *) pb;
11 }

13 /* Funkcija vraca: 1 - ako jesu anagrami 0 - inace */
int anagrami(char s[], char t[], int n_s, int n_t)
15 {
  /* Ako dve niske imaju razlicitu duzinu => nisu anagrami */
17   if (n_s != n_t)
    return 0;
19
  /* Sortiramo niske */
21   qsort(s, strlen(t) / sizeof(char), sizeof(char),
        &uporedi_char);
23   qsort(t, strlen(t) / sizeof(char), sizeof(char),
        &uporedi_char);
25
  /* Ako su niske nakon sortiranja iste => jesu anagrami, u
27     suprotnom, nisu */
  return !strcmp(s, t);
29 }

31 int main()
{
33   char s[MAX], t[MAX];

35   /* Unose se dve niske sa ulaza */
  printf("Unesite dve niske: ");
37   scanf("%s %s", s, t);

39   /* Ispituje se da li su niske anagrami */
  if (anagrami(s, t, strlen(s), strlen(t)))
41     printf("jesu\n");
  else
43     printf("nisu\n");

45   return 0;
}
```

Rešenje 3.34

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

#define MAX 10
#define MAX_DUZINA 32

/* Funkcija poredi dve niske (stringove) */
int uporedi_niske(const void *pa, const void *pb)
{
  return strcmp((char *) pa, (char *) pb);
}

int main()
{
  int i, n;
  char S[MAX][MAX_DUZINA];

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

  printf("Unesite niske:\n");
  for (i = 0; i < n; i++)
    scanf("%s", S[i]);

  /* Sortiramo niz niski */
  qsort(S, n, MAX_DUZINA * sizeof(char), &uporedi_niske);

  /*******************************
  printf("Sortirane niske su:\n");
  for(i = 0; i < n; i++)
    printf("%s ", S[i]);
  *******************************/

  /* Ako postoje dve iste niske u nizu, onda ce one nakon
     sortiranja niza biti jedna do druge */
  for (i = 0; i < n - 1; i++)
    if (strcmp(S[i], S[i + 1]) == 0) {
      printf("ima\n");
      return 0;
    }

  printf("nema\n");
  return 0;
}
```

Rešenje 3.35

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

typedef struct student {
```

```
  char nalog[8];
  char ime[21];
  char prezime[21];
  int poeni;
} Student;


/* Funkcija poredi studente prema broju poena, rastuce */
int uporedi_poeni(const void *a, const void *b)
{

  Student s = *(Student *) a;
  Student t = *(Student *) b;
  return s.poeni - t.poeni;
}

/* Funkcija poredi studente prvo prema godini, zatim prema smeru
   i na kraju prema indeksu */
int uporedi_nalog(const void *a, const void *b)
{
  Student s = *(Student *) a;
  Student t = *(Student *) b;
  /* Za svakog studenta iz naloga izdvajamo godinu upisa, smer i
     broj indeksa */
  int godina1 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
  int godina2 = (t.nalog[2] - '0') * 10 + t.nalog[3] - '0';
  char smer1 = s.nalog[1];
  char smer2 = t.nalog[1];
  int indeks1 =
      (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
      s.nalog[6] - '0';
  int indeks2 =
      (t.nalog[4] - '0') * 100 + (t.nalog[5] - '0') * 10 +
      t.nalog[6] - '0';
  if (godina1 != godina2)
    return godina1 - godina2;
  else if (smer1 != smer2)
    return smer1 - smer2;
  else
    return indeks1 - indeks2;
}

int uporedi_bsearch(const void *a, const void *b)
{
  /* Nalog studenta koji se trazi */
  char *nalog = (char *) a;
  /* Kljuc pretrage */
  Student s = *(Student *) b;

  int godina1 = (nalog[2] - '0') * 10 + nalog[3] - '0';
  int godina2 = (s.nalog[2] - '0') * 10 + s.nalog[3] - '0';
  char smer1 = nalog[1];
  char smer2 = s.nalog[1];
  int indeks1 =
      (nalog[4] - '0') * 100 + (nalog[5] - '0') * 10 + nalog[6] -
      '0';
```

```
  int indeks2 =
63      (s.nalog[4] - '0') * 100 + (s.nalog[5] - '0') * 10 +
      s.nalog[6] - '0';
65  if (godina1 != godina2)
    return godina1 - godina2;
67  else if (smer1 != smer2)
    return smer1 - smer2;
69  else
    return indeks1 - indeks2;
71 }

73 int main(int argc, char **argv)
{
75  Student *nadjen = NULL;
  char nalog_trazeni[8];
77  Student niz_studenata[100];
  int i = 0, br_studenata = 0;
79  FILE *in = NULL, *out = NULL;

81  /* Ako je broj argumenata komandne linije razlicit i od 2 i od
      3, korisnik nije ispravno pozvao program i prijavljujemo
83      gresku: */
  if (argc != 2 && argc != 3) {
85    fprintf(stderr,
            "Greska! Program se poziva sa: ./a.out -opcija (nalog)!\
     n");
87    exit(EXIT_FAILURE);
  }
89

  /* Otvaranje datoteke */
91  in = fopen("studenti.txt", "r");
  if (in == NULL) {
93    fprintf(stderr,
            "Greska prilikom otvarnja datoteke studenti.txt!\n");
95    exit(EXIT_FAILURE);
  }
97

  out = fopen("izlaz.txt", "w");
99  if (out == NULL) {
    fprintf(stderr,
101           "Greska prilikom otvaranja datoteke izlaz.txt!\n");
    exit(EXIT_FAILURE);
103 }

105 /* Ucitavamo studente iz ulazne datoteke sve do njenog kraja */
  while (fscanf
107        (in, "%s %s %s %d", niz_studenata[i].nalog,
         niz_studenata[i].ime, niz_studenata[i].prezime,
109      &niz_studenata[i].poeni) != EOF)
    i++;
111

  br_studenata = i;
113

  /* Ako je student uneo opciju -p, vrsi se sortiranje po
115     poenima */
  if (strcmp(argv[1], "-p") == 0)
```

```
117     qsort(niz_studenata, br_studenata, sizeof(Student),
              &uporedi_poeni);
119   /* A ako je uneo opciju -n, vrsi se sortiranje po nalogu */
      else if (strcmp(argv[1], "-n") == 0)
121     qsort(niz_studenata, br_studenata, sizeof(Student),
              &uporedi_nalog);
123
      /* Sortirani studenti se ispisuju u izlaznu datoteku */
125   for (i = 0; i < br_studenata; i++)
        fprintf(out, "%s %s %s %d\n", niz_studenata[i].nalog,
127             niz_studenata[i].ime, niz_studenata[i].prezime,
                niz_studenata[i].poeni);
129
      /* Ukoliko je u komandnoj liniji uz opciju -n naveden i nalog
131       studenta... */
      if (argc == 3 && (strcmp(argv[1], "-n") == 0)) {
133     strcpy(nalog_trazeni, argv[2]);

135     /* ... pronalazi se student sa tim nalogom... */
        nadjen =
137         (Student *) bsearch(nalog_trazeni, niz_studenata,
                                br_studenata, sizeof(Student),
139                             &uporedi_bsearch);

141     if (nadjen == NULL)
          printf("Nije nadjen!\n");
143     else
          printf("%s %s %s %d\n", nadjen->nalog, nadjen->ime,
145               nadjen->prezime, nadjen->poeni);
      }
147
      fclose(in);
149   fclose(out);
      return 0;
151 }
```

Rešenje 3.37

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

4 /* Funkcija koja ucitava elemente matrice a dimenzije nxm sa
     standardnog ulaza */
6 void ucitaj_matricu(int **a, int n, int m)
  {
8   printf("Unesite elemente matrice po vrstama:\n");
    int i, j;
10
    for (i = 0; i < n; i++) {
12    for (j = 0; j < m; j++) {
        scanf("%d", &a[i][j]);
14    }
    }
16 }
```

```c
/* Funkcija koja odredjuje zbir v-te vrste matrice a koja ima m
   kolona */
int zbir_vrste(int **a, int v, int m)
{
  int i, zbir = 0;

  for (i = 0; i < m; i++) {
    zbir += a[v][i];
  }

  return zbir;
}

/* Funkcija koja sortira vrste matrice na osnovu zbira
   koriscenjem selection algoritma */
void sortiraj_vrste(int **a, int n, int m)
{
  int i, j, min;

  for (i = 0; i < n - 1; i++) {
    min = i;

    for (j = i + 1; j < n; j++) {
      if (zbir_vrste(a, j, m) < zbir_vrste(a, min, m)) {
        min = j;
      }
    }
    if (min != i) {
      int *tmp;
      tmp = a[i];
      a[i] = a[min];
      a[min] = tmp;
    }

  }
}

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

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


/* Funkcija koja alocira memoriju za matricu dimenzija nxm */
int **alociraj_memoriju(int n, int m)
{
  int i, j;
```

```
74    int **a;

76    a = (int **) malloc(n * sizeof(int *));
      if (a == NULL) {
78      printf("Problem sa alokacijom memorije!\n");
        exit(EXIT_FAILURE);
80    }

82    /* Za svaku vrstu ponaosob */
      for (i = 0; i < n; i++) {
84
        /* Alociramo memoriju */
86      a[i] = (int *) malloc(m * sizeof(int));

88      /* Proveravamo da li je doslo do greske prilikom alokacije */
        if (a[i] == NULL) {
90        /* Ako jeste, ispisujemo poruku */
          printf("Problem sa alokacijom memorije!\n");
92
          /* I oslobadjamo memoriju zauzetu do ovog koraka */
94        for (j = 0; j < i; j++) {
            free(a[i]);
96        }
          free(a);
98        exit(EXIT_FAILURE);
        }
100   }

102   return a;
}
104
/* Funkcija koja oslobadja memoriju zauzetu matricom a dimenzije
106    nxm */
void oslobodi_memoriju(int **a, int n, int m)
108 {
      int i;
110
      for (i = 0; i < n; i++) {
112     free(a[i]);
      }
114   free(a);
}
116


118
int main(int argc, char *argv[])
120 {
      int **a;
122   int n, m;


124   /* Citamo dimenzije matrice */
      printf("Unesite dimenzije matrice: ");
126   scanf("%d %d", &n, &m);
128
```

```
130   /* Alociramo memoriju */
      a = alociraj_memoriju(n, m);
132
      /* Ucitavamo elemente matrice */
134   ucitaj_matricu(a, n, m);

136   /* Pozivamo funkciju koja sortira vrste matrice prema zbiru */
      sortiraj_vrste(a, n, m);
138
      /* Ispisujemo rezultujucu matricu */
140   printf("Sortirana matrica je:\n")
          ispisi_matricu(a, n, m);
142
      /* Oslobadjamo memoriju */
144   oslobodi_memoriju(a, n, m);

146   /* I prekidamo sa izvrsavanjem programa */
      return 0;
148 }
```

Glava 4

Dinamičke strukture podataka

4.1 Liste

Zadatak 4.1 Napisati program koji koristi 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.

(a) Definisati strukturu `Cvor` koja predstavlja čvor liste.

(b) Milena:Da li ovde treba dodati i funkciju koja kreira cvor? Nemam resenje kod sebe pa ne znam kako to vec ide, ali mislim da bi trebalo

(c) Napisati funkciju koja dodaje novi elemenat na početak liste.

(d) Napisati funkciju koja dodaje novi elemenat na kraj liste.

(e) Milena: Da li bi ovo trebalo izdvojiti u poseban zadatak? Nekako, ako dodajemo na pocetak i kraj, nemamo garanciju sortiranosti liste, tako da mi to nekako deluje da smo dva zadatka strpali u jedan. Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.

(f) Napisati funkciju koja oslobađa memoriju koju je zauzela lista.

(g) Milena: Ova funkcija bi mogla razlicito da se implementira sa pretpostavkom da je lista sortirana i da nije sortirana, i zato mi dodatno deluje da bi ta dva zadatka trebalo razdvojiti Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.

(h) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.

(i) Milena: Da li ovde nedostaje funckija koja oslobadja celu memoriju?

Sve funckije za rad sa listom najpre implementirati iterativno, a zatim i rekurzivno.

Zadatak 4.2 Napisati program koji koristi dvostruko povezanu listu za čuvanje celih brojeva koji se unose sa standardnog ulaza. Unošenje novih brojeva u listu se prekida učitavanjem kraja ulaza (EOF). Svako dodavanje novog broja u listu ispratiti ispisivanjem trenutnog sadržaja liste. I ovde isto mozda razdvojiti sortiranost od obične liste.

(a) Napisati funkciju koja dodaje novi elemenat na početak liste.

(b) Napisati funkciju koja dodaje novi elemenat na kraj liste.

(c) Napisati funkciju koja dodaje novi elemenat u listu tako da lista ostane rastuće sortirana.

(d) Napisati funkciju koja oslobađa memoriju koju je zauzela lista.

(e) Napisati funkciju koja pretražuje listu za elementom koji ima vrednost koja je argument funkcije.

(f) Napisati funkciju koja briše sve elemente u listi koji imaju vrednost koja je argument funkcije.

Sve funckije za rad sa listom implementirati iterativno.

Zadatak 4.3 Sadržaj datoteke je aritmetički izraz koji može sadržati zagrade {, [i (. Napisati program koji učitava sadržaj datoteke i korišćenjem steka utvrđuje da li su zagrade u aritmetičkom izrazu dobro uparene. Program štampa odgovarajuću poruku na standardni izlaz. Milena: promenjeni test primeri, voditi racuna u resenjima sta se stampa!

Test 1

```
Datoteka: {[23 + 5344] * (24 - 234)} - 23
Izlaz:  Zagrade su ispravno uparene.
```

Test 2

```
Datoteka: {[23 + 5] * (9 * 2)} - {23}
Izlaz:  Zagrade su ispravno uparene.
```

Test 3

```
Datoteka: {[2 + 54) / (24 * 87)} + (234 + 23)
Izlaz:  Zagrade nisu ispravno uparene.
```

Zadatak 4.4 Napisati program koji proverava ispravnost uparivanja etiketa u HTML datoteci. Ime datoteke se zadaje kao argument komandne linije . Milena: A sta ako se ne navede argument komandne linije? Uputstvo: za rešavanje problema koristiti stek implementiran preko listi čiji su čvorovi HTML etikete.

Test 1

```
Poziv: ./a.out datoteka.html
Datoteka.html:                              Izlaz:
<html>                                      Ispravno uparene etikete.
  <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>
```

Test 2

```
Poziv: ./a.out datoteka.html
Datoteka.html:                              Izlaz:
<html>                                      Neispravno uparene etikete.
  <head><title>Primer</title></head>
  <body>
</html>
```

Test 3

```
Poziv: ./a.out datoteka.html
Datoteka.html:                              Izlaz:
<html>                                      Neispravno uparene etikete.
  <head><title>Primer</title></head>
  <body>
  </body>
```

Zadatak 4.5 Milena: Problem sa ovim zadatkom je sto je program najpre na usluzi korisnicima, a zatim na usluzi sluzbeniku i to nekako zbunjuje u formulaicji. Formulacija mi nije bila jasna bez citanja resenja, pokusala sam da je preciziran, u nastavku je izmenjena formulacija.
Medjutim, ja i dalje nisam bas zadovoljna i zato predlazem da se formulacija izmeni tako da je program stalno na usluzi sluzbeniku. Program ucitva podatke o prijavljenim korisnicima iz datoteke. Sluzbenik odlucuje da li ce da obradjuje redom korisnike, ili ce u nekim situacijama da odlozi rad sa korisnikom i stavi ga na kraj reda. Program ga uvek pita da na osnovu jmbg-a i zahteva odluci da li ce ga staviti na kraj reda, ako hoce, on ide na kraj reda, ako nece, onda sluzbenik daje odgovor na zahtev i jmbg, zahtev i odgovor se upisuju u izlaznu datoteku.

Napisati program kojim se simulira rad jednog šaltera na kojem se prvo zakazuju termini, a potom službenik uslužuje korisnike redom, kako su se prijavljivali.

Korisnik se prijavljuje unošenjem svog `jmbg` broja (niska koja sadrži 13 karaktera) i zahteva (niska koja sadrži najviše 999 karaktera). Prijavljivanje korisnika se

prekida unošenjem karaktera za kraj ulaza (`EOF`).

Službenik redom proziva korisnike čitanjem njihovog `jmbg` broja, a zatim odlučuje da li korisnika vraća na kraj reda ili ga odmah uslužuje. Službeniku se postavlja pitanje `Da li korisnika vracate na kraj reda?` i ukoliko on da odgovor `Da`, korisnik se vraća na kraj reda. Ukoliko odgovor nije `Da`, tada službenik čita korisnikov zahtev. Posle svakog 10 uslуženog korisnika, službeniku se nudi mogućnost da prekine sa radom, nevezano od broja korisnika koji i dalje čekaju u redu.

Za čuvanje korisničkih zahteva koristiti red implementiran korišćenjem listi.

Zadatak 4.6 Milena: Dodati sta se desava ako nije zadat argument komandne linije ili ako datoteka ne postoji Napisati program koji prebrojava pojavljivanja etiketa HTML datoteke čije se ime zadaje kao argument komandne linije. Rezultat prebrojavanja ispisati na standardni izlaz. Etikete smeštati u listu, a za formiranje liste koristiti strukturu:

```
typedef struct _Element
{
  unsigned broj_pojavljivanja;
  char etiketa[20];
  struct _Element *sledeci;
} Element;
```

Test 1

```
Poziv: ./a.out datoteka.html
Datoteka.html:                                          Izlaz:
<html>                                                  a - 4
  <head><title>Primer</title></head>                    br - 1
  <body>                                                h1 - 2
    <h1>Naslov</h1>                                     body - 2
    Danas je lep i suncan dan. <br>                     title - 2
    A sutra ce biti jos lepsi.                          head - 2
    <a link="http://www.google.com"> Link 1</a>         html - 2
    <a link="http://www.math.rs"> Link 2</a>
  </body>
</html>
```

Zadatak 4.7 Milena: i ovde dodati sta ako nema argumenata i ako nema datoteka, kao i u svim ostalim zadacima, a ne bih stalno ovaj komentar ponavljala. Takodje, malo me muci u ovom zadatku sto nema neki smisao. Naime, ako se samo vrsi ucitavanje iz datoteka i ispisivanje, onda su ove liste zapravo visak jer isti rezultat moze da se dobije i bez koriscenja listi. Zato mi fali da program uradi nesto sto ne bi mogao da uradi bez koriscenja listi, npr da na osnovu unetog broja ispisuje svaki n-ti broj rezultujuce liste pa to u nekoj petlji da korisnik moze da ispisuje za razlicite unete n ili tako nesto...

Napisati program koji objedinjuje dve sortirane liste. Funkcija ne treba da kreira nove čvorove, već da samo postojeće čvorove preraspodeli. Prva lista se učitava iz datoteke koja 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

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz: 2 4 5 6 6 10 11 12 14 15
   16
```

Zadatak 4.8 Napisati funkciju koja formira listu studenata tako što se podaci o studentima učitavaju iz datoteke čije se ime zadaje kao argument komandne linije. U svakom redu datoteke nalaze se podaci o studentu i to broj indeksa, ime i prezime. Napisati rekurzivnu funkciju koja određuje da li neki student pripada listi ili ne. Ispisati zatim odgovarajuću poruku i rekurzivno osloboditi memoriju koju je data lista zauzimala. Student se traži na osnovu broja indeksa, koji se zadaje sa standardnog ulaza.

Test 1

```
Poziv: ./a.out studenti.txt
Datoteka:                Ulaz:       Izlaz:
123/2014 Marko Lukic     3/2014      da: Ana Sokic
3/2014 Ana Sokic         235/2008    ne
43/2013 Jelena Ilic      41/2009     da: Marija Zaric
41/2009 Marija Zaric
13/2010 Milovan Lazic
```

Milena: Imamo dva zadatka sa labelom 608!

Zadatak 4.9 Neka su date dve jednostruko povezane liste L1 i L2. Napisati funkciju koja od tih lista formira novu listu L koja sadrži alternirajući raspoređene elemente lista L1 i L2 (prvi element iz L1, prvi element iz L2, drugi element L1, drugi element L2, itd). Ne formirati nove čvorove, već samo postojeće čvorove rasporediti u jednu listu. Prva lista se učitava iz datoteke koja 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. Milena: Sta ako je neka lita duza? To precizirati. I ovde me muci sto nedostaje neki smisao zadatku, nesto sto ne bi moglo da se uradi da nismo kristili liste.

Test 1

```
Poziv: ./a.out dat1.txt dat2.txt
dat1.txt: 2 4 6 10 15
dat2.txt: 5 6 11 12 14 16
Izlaz:  2 5 4 6 6 11 10 12 15 14
   16
```

Zadatak 4.10 Data je datotka brojevi.txt koja sadrži cele brojeve, po jedan u svakom redu.

(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 maximalan strogo rastući podniz.

Napisati program koji u datoteku `Rezultat.txt` upisuje nađeni strogo rastući podniz. Milena: I ovde me muci sto bi zadatak mogao da se resi i bez koriscenja listi...

Milena: Prirodni oblik testa ovde bi bio horizontalan, a ne ovako vertikalan.

Test 1

```
Ulaz:  brojevi.txt        Izlaz: Rezultat.txt
       43                              12
       12                              15
       15                              16
       16
       4
       2
       8
```

Zadatak 4.11 Grupa od n plesača na kostimima imaju brojeve od 1 do n, redom, u smeru kazaljke na satu. Plesači izvode svoju plesnu tačku tako što formiraju krug iz kog najpre izlazi k-ti plesač. Odbrojava se počevši od plesača označenog brojem 1 u smeru kretanja kazaljke na satu. Preostali plesači obrazuju manji krug iz kog opet izlazi k-ti plesač. Odbrojavanje 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: u implementaciji koristiti kružnu listu.

Test 1

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

Milena: Bilo bi lepo dodati i prethodni zadatak u kojem se smer izbacivanja stalno menja, tako da se onda korsti dvostruko povezana kružna lista.

4.2 Stabla

Zadatak 4.12 Napisati program za rad sa binarnim pretraživačkim stablima.

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

podstablo[1].

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

(c) Napisati funkciju `void dodaj_u_stablo(Cvor** koren, int broj)` koja u stablo na koje pokazuje argument `koren` dodaje ceo broj `broj`.

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

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

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

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

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

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

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

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

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

[1] U zadacima ove glave u kojima nije eksplicitno naglašen sadržaj čvorova stabla, podrazumevaće se ova struktura.

Test 1

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa): 7 2 1 9 32 18
Izlaz:
  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
```

Test 2

```
Poziv: ./a.out
Ulaz:
  Unesite brojeve (CRL+D za kraj unosa): 8 -2 6 13 24 -3
Izlaz:
  Infiksni ispis:  -3 -2 6 8 13 24
  Prefiksni ispis: 8 -2 -3 6 13 24
  Postfiksni ispis: -3 6 -2 24 13 8
  Trazi se broj: 6
  Broj se nalazi u stablu!
  Brise se broj: 14
  Rezultujuce stablo: -3 -2 6 8 13 24
```

Zadatak 4.13 Napisati program koji izračunava i na standardnom izlazu ispisuje broj pojavljivanja svake reči datoteke čije se ime zadaje kao argument komandne linije. Program realizovati korišćenjem binarnog pretraživackog stabla uređenog leksikografski prema rečima ne uzimajući u obzir razliku između malih i velikih slova. Ukoliko prilikom pokretanja programa korisnik ne navede ime ulazne datoteke ispisati poruku `Nedostaje ime ulazne datoteke!`.

Milena: dodati i test primer sa pokretanjem bez ulazne datoteke

Test 1

```
Poziv: ./a.out test.txt
Datoteka test.txt:
  Sunce utorak raCunar SUNCE
   programiranje jabuka
   PROGramiranje sunCE JABUka
Izlaz:
  jabuka: 2
  programiranje: 2
  racunar: 1
  sunce: 3
  utorak: 1
```

Test 2

```
Poziv: ./a.out suma.txt
Datoteka suma.txt:
  lipa zova hrast ZOVA breza LIPA
Izlaz:
  breza: 1
  hrast: 1
  lipa: 2
  zova: 2
```

Test 3

```
Poziv: ./a.out
Izlaz:
  Nedostaje ime ulazne datoteke!
```

Zadatak 4.14 U svakoj liniji datoteke čije se ime zadaje sa standardnog ulaza nalazi se ime osobe, prezime osobe i njen broj telefona, npr. `Pera Peric 064/123-4567`. Napisati program koji korišćenjem binarnog pretraživačkog stabla implementira mapu koja sadrži navedene informacije i koja će omogućiti pretragu brojeva telefona za zadata imena i prezimena. Imena i prezimena se unose sve do unosa reči `KRAJ`, a za svaki od unetih podataka ispisuje se ili broj telefona ili obaveštenje da traženi broj nije u imeniku. Može se pretpostaviti da imena, prezimena i brojevi telefona neće biti duži od 30 karaktera.

Upotreba programa 1

```
Poziv: ./a.out
Datoteka imenik.txt:
   Pera Peric 011/3240-987
   Marko Maric 064/1234-987
   Mirko Maric 011/589-333
   Sanja Savkovic 063/321-098
   Zika Zikic 021/759-858
Ulaz:
  Unesite ime datoteke: imenik.txt
  Unesite ime i prezime: Pera Peric
Izlaz:
  Broj je: 011/3240-987
Ulaz:
  Unesite ime i prezime: Marko Markovic
Izlaz:
  Broj nije u imeniku!
Ulaz:
  Unesite ime i prezime: KRAJ
```

Zadatak 4.15 U datoteci `prijemni.txt` nalaze se podaci o prijemnom ispitu učenika jedne osnovne škole tako što je u svakom redu navedeno ime i prezime učenika (niz najviše 50 karaktera), broj poena na osnovu uspeha (realan broj), broj poena na prijemnom ispitu iz matematike (realan broj) i broj poena na pri-

jemnom 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

```
Poziv: ./a.out
Datoteka 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 19 20 99
  2. Nadica Zec 54.2 10 15.8 80
  3. Marko Markovic 45.4 12.3 11 68.7
  4. Milan Jevremovic 35.2 1.3 9 45.5
  ----------------------------------
  5. Jovana Milic 23.3 2 5.6 30.9
```

*** Zadatak 4.16** Napisati program koji implementira podsetnik za rođendane. Informacije o rođendanima se nalaze u datoteci čije se ime zadaje kao argument komandne linije u formatu `Ime Prezime DD.MM.YYYY.` - za svaku osobu po jedna linija datoteke. Korisnik unosi datum u naznačenom formatu, a program pronalazi i ispisuje ime i prezime osobe čiji je rođendan zadatog datuma ili ime i prezime osobe koja prva sledeća slavi rođendan. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog pretraživačkog stabla i uređena po datumima. Može se pretpostaviti da će svi korišćeni datumi biti validni i u formatu `DD.MM.YYYY`.

Upotreba programa 1

```
Poziv: a.out
Datoteka rodjendani.txt:
  Marko Markovic 12.12.1990.
  Milan Jevremovic 04.06.1989.
  Maja Agic 23.04.2000.
  Nadica Zec 01.01.1993.
  Jovana Milic 05.05.1990.
Ulaz:
  Unesite datum: 23.04.
Izlaz:
  Slavljenik: Maja Agic
Ulaz:
  Unesite datum: 01.01.
Izlaz:
  Slavljenik: Nadica Zec
Ulaz:
  Unesite datum: 01.05.
Izlaz:
  Slavljeni: Jovana Milic 05.05.
Ulaz:
  Unesite datum: CTRL+D
```

Zadatak 4.17 Dva binarna stabla su identična ako su ista po strukturi i sadržaju tj. ako oba korena imaju isti sadržaj i identična odgovarajuća podstabla. Napistati funkciju `int identitet(Cvor* koren1, Cvor* koren2)` koja proverava da li su binarna stabla `koren1` i `koren2` koja sadrže cele brojeve identična, a zatim i glavni program koji testira njen rad. Elemente pojedinačnih stabla unositi sa standardnog ulaza sve do pojave broja 0.

Test 1

```
Poziv: ./a.out
Ulaz:
  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
Izlaz:
  Stabla jesu identicna.
```

Test 2

```
Poziv: ./a.out
Ulaz:
  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
Izlaz:
  Stabla nisu identicna.
```

*** Zadatak 4.18** Napisati program koji za dva binarna pretraživačka stabla čiji se elementi zadaju sa standardnog ulaza, sve do kraja ulaza, ispisuje uniju, presek i razliku stabla. Unija dva stabala je stablo koje sadrži vrednosti iz oba stabla

uračunata tačno po jednom. Presek dva stabala je stablo koje sadrži vrednosti koje se pojavljuju i u prvom i u drugom stablu. Razlika dva stabla je stablo koje sadrži sve vrednosti prvog stabla koje se ne pojavljuju u drugom stablu.

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 1 7 8 9 2 2
  Drugo stablo: 3 9 6 11 1
Izlaz:
  Unija: 1 2 3 6 7 8 9 11
  Presek: 1 9
  Razlika: 2 7 8
```

Zadatak 4.19 Napisati funkciju `void sortiraj(int a[], int n)` koja sortira niz celih brojeva `a` dimenzije `n` korišćenjem binarnog pretraživačkog stabla. Napisati i program koji sa standardnog ulaza učitava ceo broj `n` manji od 50 i niz `a` celih brojeva dužine `n`, poziva funkciju `sortiraj` i rezultat ispisuje na standardnom izlazu.

Test 1

```
Poziv: ./a.out
Ulaz:
  n: 7
  a: 1 11 8 6 37 25 30
Izlaz:
  1 6 8 11 25 30 37
```

Test 2

```
Poziv: ./a.out
Ulaz:
  n: 55
Izlaz:
  Greska: pogresna dimenzija niza!
```

Zadatak 4.20 Dato je binarno pretraživačko stablo celih brojeva.

(a) Napisati funkciju koja izračunava broj čvorova stabla.

(b) Napisati funkciju koja izračunava broj listova stabla.

(c) Napisati funkciju koja štampa pozitivne vrednosti listova stabla.

(d) Napisati funkciju koja izračunava zbir čvorova stabla.

(e) Napisati funkciju koja izračunava najveći element stabla.

(f) Napisati funkciju koja izračunava dubinu stabla.

(g) Napisati funkciju koja izračunava broj čvorova na i-tom nivou stabla.

(h) Napisati funkciju koja ispisuje sve elemente na i-tom nivou stabla.

(i) Napisati funkciju koja izračunava maksimalnu vrednost na i-tom nivou stabla.

(j) Napisati funkciju koja izračunava zbir čvorova na i-tom nivou stabla.

(k) Napisati funkciju koja izračunava zbir svih vrednosti stabla koje su manje ili jednake od date vrednosti x.

Napisati program koji testira prethodne funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza, a vrednosti parametara i i x pročitati kao argumente komandne linije.

Test 2

```
Poziv: ./a.out 2 15
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  broj cvorova: 10
  broj listova: 4
  pozitivni listovi: 2 4 13 30
  zbir cvorova: 108
  najveci element: 30
  dubina stabla: 5
  broj cvorova na 2. nivou: 3
  elementi na 2. nivou: 3 12 30
  maksimalni na 2. nivou: 30
  zbir na 2. nivou: 45
  zbir elemenata manjih ili jednakih od 15: 7
```

Zadatak 4.21 Dato je binarno pretraživačko stablo celih brojeva.

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

(b) Napisati funkciju koja pronalazi čvor u stablu sa najmanjom sumom 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 gorenavedene funkcije. Stablo formirati na osnovu vrednosti koje se unose sa standardnog ulaza, sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 3 2 4 30 12 14 13
Izlaz:
  Cvor sa maksimalnim desnim proizvodom: 10
  Cvor sa najmanjom levom sumom: 2
  Putanja do najdubljeg cvora: 10 15 12 14 13
  Putanja do najmanjeg cvora: 10 5 3 2
```

Zadatak 4.22 Napisati program koji ispisuje sadržaj binarnog pretraživačkog stabla po nivoima.

Test 1

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

*** Zadatak 4.23** 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.

Test 1

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 14 30 1 0
Izlaz:
  Stabla su slicna kao u ogledalu
    .
```

Test 2

```
Poziv: ./a.out
Ulaz:
  Prvo stablo: 11 20 5 3 0
  Drugo stablo: 8 20 15 0
Izlaz:
  Stabla nisu slicna kao u
   ogledalu.
```

Zadatak 4.24 AVL-stablo je binarno stablo pretrage kod koga apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju `int avl(Cvor* koren)` koja izračunava broj čvorova stabla sa korenom `koren` koji ispunjavaju uslov za AVL stablo. Napisati zatim i glavni program koji ispisuje rezultat `avl` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  10 5 15 2 11 16 1 13
Izlaz:
  7
```

Test 2

```
Poziv: ./a.out
Ulaz:
  16 30 40 24 10 18 45 22
Izlaz:
  6
```

Zadatak 4.25 Binarno stablo se naziva HEAP ako je kompletno (svaki njegov čvor, izuzev listova, ima i levog i desnog potomka) i za svaki čvor u stablu važi da je njegova vrednost veća od vrednosti svih ostalih čvorova u njegovim podstablima. Napisati funkciju `int heap(Cvor* koren)` koja proverava da li je dato binarno stablo celih brojeva HEAP. Napisati zatim i glavni program koji ispisuje rezultat `heap` funkcije za stablo čiji se elementi unose sa standardnog ulaza sve do kraja ulaza.

Test 1

```
Poziv: ./a.out
Ulaz:
  100 19 36 17 3 25 1 2 7
Izlaz:
  Stablo je heap.
```

4.3 Rešenja

Rešenje 4.1

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

```
int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.2

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.8

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.4

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.5

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.6

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.7

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.8

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.9

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.10

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.11

```
#include<stdio.h>

int main(){
  printf("Hello bitovi!\n");
  return 0;
}
```

Rešenje 4.12

Rešenje 4.13

Glava 5

Ispitni rokovi

5.1 Programiranje 2, praktični deo ispita, jun 2015.

Zadatak 5.1

Kao argument komandne linije zadaje se ime ulazne datoteke u kojoj se nalaze niske. U prvoj liniji datoteke nalazi se informacija o broju niski, a zatim u narednim linijama po jedna niska ne duža od `50` karaktera.

Napisati program u kojem se dinamički alocira memorija za zadati niz niski, a zatim se na standardnom izlazu u redosledu suprotnom od redosleda čitanja ispisuju sve niske koje počinju velikim slovom.

U slučaju pojave bilo kakve greške na standardnom izlazu ispisati vrednost `-1` i prekinuti izvršavanje programa.

Test 1

```
ž
Sadraj datoteke:
 5
 Programiranje
 Matematika
 12345
 dInAmiCnArEc
 Ispit
Izlaz:
 Ispit
 Matematika
 Programiranje
```

Test 2

```
ž
Sadraj datoteke:
  2
  maksimalano
  poena
Izlaz:
```

Test 3

```
Problem:
  datoteka
   ne postoji
Izlaz:
  -1
```

Zadatak 5.2

Data je biblioteka za rad sa binarnim pretrazivačkim stablima čiji čvorovi sadrže cele brojeve. Napisati funkciju `int sumirajN (Cvor * koren, int n)` koja izračunava zbir svih čvorova koji se nalaze na n-tom nivou stabla (koren se nalazi na nultom nivou, njegova deca na prvom nivou i tako redom). Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa pretraživačkim stablima.

Napisati program koji sa standardnog ulaza učitava najpre prirodan broj n, a potom i brojeve sve do pojave nule koje smešta u stablo i ispisuje rezultat pozivanja funkcije `prebrojN` za broj n i tako kreirano stablo. U slučaju greške na standardni izlaz za grešku ispisati `-1`.

Test 1

```
Ulaz:
 2 8 10 3 6 14 13 7 4 0
Izlaz:
 20
```

Test 2

```
Ulaz:
 0 50 14 5 2 4 56 8 52 7 1 0
Izlaz:
 50
```

Zadatak 5.3 Sa standardnog ulaza učitava se broj vrsta i broj kolona celobrojne matrice A, a zatim i elementi matrice A. Napisati program koji će ispisati indeks kolone u kojoj se nalazi najviše negativnih elemenata. Ukoliko postoji više takvih kolona, ispisati indeks prve kolone. Može se pretpostaviti da je broj vrsta i broj kolona manji od `50`. U slučaju greške ispisati vrednost `-1` na standardni izlaz za greške.

Test 1

```
Ulaz:
  4
  5
  1  2  3  4  5
 -1  2 -3  4 -5
 -5 -4 -3 -2  1
 -1  0  0  0  0
Izlaz:
 0
```

Test 2

```
Ulaz:
 2
 3
 0 0 -5
 1 2 -4
Izlaz:
```

Test 3

```
Ulaz:
 -2
Izlaz (na stderr):
 -1
```

5.2 Programiranje 2, praktični deo ispita, jul 2015.

Zadatak 5.4

Napisati program koji kao prvi arugment komandne linije prima ime dokumenta u kome treba prebrojati sva pojavljivanja tražene niske (bez preklapanja) koja se navodi kao drugi argument komandne linije (iskoristiti funkciju standardne biblioteke `strstr`). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Pretpostaviti da linije datoteke neće biti duže od `127` karaktera.
Potpis funkcije strstr:
`char *strstr(const char *haystack, const char *needle);`
Funkcija traži prvo pojavljivanje podniske needle u nisci haystack, i vraća pokazivač na početak podniske, ili NULL ako podniska nije pronađena.

Test 1

```
Poziv:  ./a.out fajl.txt test
Datoteka:   Ovo je test primer.
        U njemu se rec test
    javlja
        vise puta. testtesttest
Izlaz:  5
```

Test 2

```
Poziv:   ./a.out
Izlaz  (na stderr):  -1
```

Test 3

```
Poziv:   ./a.out fajl.txt foo
Datoteka:   (ne postoji)
Izlaz (na stderr):    -1
```

Test 4

```
Poziv:  ./a.out fajl.txt .
Datoteka:   (prazna)
Izlaz:    0
```

Zadatak 5.5

Na početku datoteke "trouglovi.txt" nalazi se broj trouglova čije su koordinate temena zapisane u nastavku datoteke. Napisati program koji učitva trouglove, i ispisuje ih na standardni izlaz sortirane po površini opadajuće (koristiti Heronov obrazac: $P = \sqrt{s * (s-a) * (s-b) * (s-c)}$, gde je s poluobim trougla). U slučaju bilo kakve greške ispisati `-1` na standardni izlaz za greške. Ne praviti nikave pretpostavke o broju trouglova u datoteci, i proveriti da li je datoteka ispravno zadata.

Test 1

```
Datoteka:  4
           0 0 0 1.2 1 0
           0.3 0.3 0.5 0.5 0.9 1
           -2 0 0 0 0 1
           2 0 2 2 -1 -1
Izlaz:     2 0 2 2 -1 -1
           -2 0 0 0 0 1
           0 0 0 1.2 1 0
           0.3 0.3 0.5 0.5 0.9 1
```

Test 2

```
Datoteka:  3
           1.2 3.2 1.1
    4.3
Izlaz:     -1
```

Test 3

```
Datoteka:   (nema datoteke)
Izlaz:     -1
```

Test 4

```
Datoteka:   0
Izlaz:
```

Zadatak 5.6 Data je biblioteka za rad sa binarnim pretraživačkim stablima celih brojeba. Napisati funkciju
`int f3(Cvor *koren, int n)`
koja u datom stablu prebrojava čvorove na n-tom nivou, koji imaju tačno jednog potomka. Pretpostaviti da se koren nalazi na nivou `0`. Ispravnost napisane funkcije testirati na osnovu zadate `main` funkcije i biblioteke za rad sa stablima.

Test 1

```
Ulaz:
 1 5 3 6 1 4 7 9
Izlaz:
 1
```

Test 2

```
Ulaz:
 2 5 3 6 1 0 4 7 9
Izlaz:
 2
```

Test 3

```
Ulaz:
 0 4 2 5
Izlaz:
 0
```

Test 4

```
Ulaz:
  3
Izlaz:
 0
```

Test 5

```
Ulaz:
  -1 4 5 1 7
Izlaz:
 0
```

5.3 Rešenja

Rešenje 5.1

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 50

void greska(){
    printf("-1\n");
    exit(EXIT_FAILURE);
}

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

    FILE* ulaz;
    char** linije;
    int i, j, n;

    /* Proveravamo argumente komandne linije.
  */
    if(argc!=2){
        greska();
    }

    /* Otvaramo datoteku čije ime je navedeno kao argument komandne
    linije neposredno nakon imena programa koji se poziva. */
    ulaz=fopen(argv[1], "r");
    if(ulaz==NULL){
        greska();
    }

    /* čUitavamo broj linija. */
    fscanf(ulaz, "%d", &n);

    /* Alociramo memoriju na osnovu čuitanog broja linija.*/
    linije=(char**)malloc(n*sizeof(char*));
```

```
    if(linije==NULL){
35      greska();
    }
37  for(i=0; i<n; i++){
        linije[i]=malloc(MAX*sizeof(char));
39      if(linije[i]==NULL){
            for(j=0; j<i; j++){
41              free(linije[j]);
            }
43          free(linije);
            greska();
45      }
    }
47
    /* čUitavamo svih n linija iz datoteke. */
49  for(i=0; i<n; i++){
        fscanf(ulaz, "%s", linije[i]);
51  }

53  /* Ispisujemo u ćodgovarajuem poretku čuitane linije koje
    zadovoljavaju kriterijum. */
    for(i=n-1; i>=0; i--){
55      if(isupper(linije[i][0])){
            printf("%s\n", linije[i]);
57      }
    }
59
    /* đOslobaamo memoriju koju smo čdinamiki alocirali. */
61  for(i=0; i<n; i++){
        free(linije[i]);
63  }

65  free(linije);

67  /* Zatvaramo datoteku. */
    fclose(ulaz);
69
    /* šZavravamo sa programom. */
71  return 0;

73 }
```

Rešenje 5.2

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


4
int sumirajN (Cvor * koren, int n){
6   if(koren==NULL){
            return 0;
8   }

10  if(n==0){
        return koren->broj;
```

```
    }

    return sumirajN(koren->levo, n-1) + sumirajN(koren->desno, n-1);
}


int main(){
    Cvor* koren=NULL;
    int n;
    int nivo;

    /* Čitamo vrednost nivoa */
    scanf("%d", &nivo);


    while(1){

        /* Čitamo broj sa standardnog ulaza */
        scanf("%d", &n);

        /* Ukoliko je korisnik uneo 0, prekidamo dalje čitanje. */
        if(n==0){
            break;
        }

        /* A ako nije, dodajemo procitani broj u stablo. */
        dodaj_u_stablo(&koren, n);

    }

    /* Ispisujemo rezultat rada žtraene funkcije */
    printf("%d\n", sumirajN(koren,nivo));

    /* đOslobaamo memoriju */
    oslobodi_stablo(&koren);


    /* Prekidamo šizvravanje programa */
    return 0;
}
```

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

Cvor* napravi_cvor(int b ) {
    Cvor* novi = (Cvor*) malloc(sizeof(Cvor));
    if( novi == NULL)
        return NULL;

    /* Inicijalizacija polja novog čvora */
    novi->broj = b;
    novi->levo = NULL;
    novi->desno = NULL;

    return novi;
```

```
}

void oslobodi_stablo(Cvor** adresa_korena)  {
      /* Prazno stablo i nema šta da se đoslobaa */
    if( *adresa_korena == NULL)
        return;

    /* Rekurzivno đoslobaamo najpre levo, a onda i desno podstablo*/
    if( (*adresa_korena)->levo )
        oslobodi_stablo(&(*adresa_korena)->levo);
    if( (*adresa_korena)->desno)
        oslobodi_stablo(&(*adresa_korena)->desno);

    free(*adresa_korena);
    *adresa_korena =NULL;
}


void proveri_alokaciju( Cvor* novi) {
    if( novi == NULL) {
        fprintf(stderr, "Malloc greska za nov cvor!\n");
        exit(EXIT_FAILURE);
    }
}

void dodaj_u_stablo(Cvor** adresa_korena, int broj) {
    /* Postojece stablo je prazno*/
    if( *adresa_korena == NULL){
        Cvor* novi = napravi_cvor(broj);
        proveri_alokaciju(novi);
        *adresa_korena = novi;  /*  Kreirani čvor novi će biti od
   sada koren stabla*/
        return;
    }

    /* Brojeve šsmetamo u đureeno binarno stablo, pa
    ako je broj koji ubacujemo manji od broja koji je u korenu  */
    if( broj < (*adresa_korena)->broj)         /* dodajemo u levo
   podstablo */
        dodaj_u_stablo(&(*adresa_korena)->levo, broj);
    /* ako je broj manji ili jednak od broja koji je u korenu stabla
    , dodajemo nov čvor desno od korena */
    else
        dodaj_u_stablo(&(*adresa_korena)->desno, broj);
}
```

```
#ifndef __STABLA_H__
#define __STABLA_H__ 1

/* Struktura kojom se predstavlja čvor drveta */
typedef struct dcvor{
    int broj;
    struct dcvor* levo, *desno;
} Cvor;

```

```
/* Funkcija alocira prostor za novi čvor drveta, inicijalizuje polja
    strukture i ćvraa čpokaziva na nov čvor */
Cvor* napravi_cvor(int b );

/* đOslobaamo čdinamiki alociran prostor za stablo
 * Nakon đoslobaanja se u ćpozivajuoj funkciji koren
 * postavljana NULL, jer je stablo prazno  */
void oslobodi_stablo(Cvor** adresa_korena);


/* Funkcija proverava da li je novi čvor ispravno alociran,
 * i nakon toga prekida program */
void proveri_alokaciju( Cvor* novi);


/* Funkcija dodaje nov čvor u stablo i
 * žaurira vrednost korena stabla u ćpozivajuoj funkciji.
 */
void dodaj_u_stablo(Cvor** adresa_korena, int broj);

#endif
```

Rešenje 5.3

```
#include <stdio.h>
#define MAX 50


int main(){
    int m[MAX][MAX];
    int v, k;
    int i, j;
    int max_broj_negativnih, max_indeks_kolone;
    int broj_negativnih;

    /* čUitavamo dimenzije matrice */
    scanf("%d", &v);
    scanf("%d", &k);

    if(v<0 || v>MAX || k<0 || k>MAX){
        fprintf(stderr, "-1\n");
        return 0;
    }

    /* čUitavamo elemente matrice */
    for(i=0; i<v; i++){
        for(j=0; j<k; j++){
            scanf("%d", &m[i][j]);
        }
    }

    /*Pronalazimo kolonu koja žsadri ćnajvei broj negativnih
    elemenata */
    max_indeks_kolone=0;

```

```
    max_broj_negativnih=0;
    for(i=0; i<v; i++){
        if(m[i][0]<0){
            max_broj_negativnih++;
        }

    }

    for(j=0; j<k; j++){
        broj_negativnih=0;
        for(i=0; i<v; i++){
            if(m[i][j]<0){
                broj_negativnih++;
            }
            if(broj_negativnih>max_broj_negativnih){
                max_indeks_kolone=j;
            }
        }

    }

    /* Ispisujemo žtraeni rezultat */
    printf("%d\n", max_indeks_kolone);

    /* šZavravamo program */
    return 0;
}
```

Rešenje 5.4

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 128

int main(int argc, char **argv) {
  FILE *f;
  int brojac = 0;
  char linija[MAX], *p;

  if (argc != 3) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  if ((f = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "-1\n");
    exit(EXIT_FAILURE);
  }

  while (fgets(linija, MAX, f) != NULL) {
    p = linija;
    while (1) {
      p = strstr(p, argv[2]);
      if (p == NULL)
```

```
    break;
27      brojac++;
        p = p + strlen(argv[2]);
29    }
  }
31
  fclose(f);
33
  printf("%d\n", brojac);
35
  return 0;
37}
```

Rešenje 5.5

Rešenje 5.6