II pismeni zadatak

1. Napisati program koji čita jednu reč teksta koju čine samo velika slova i ispisuje slovo koji se najčešće pojavljuje i koliko puta se pojavljuje. Ako se više slova najčešće pojavljuje, ispisuje se slovo koji je leksikografski manje. U jednoj liniji standardnog ulaza nalazi se jedna reč teksta sa ne više od 20 slova. U prvoj liniji standarnog izlaza prikazati slovo koje se najčešće pojavljuje, razmak i, potom koliko puta se to slovo pojavljuje.
   ULAZ IZLAZ
   GREEEKKK E 3
   ABBAC A 2
   ACDCEBB B 2
2. Napisati
   a) iterativnu funkciju int palindrom (char s[]) kojom se proverava da li je zadata niska s palindrom (za nisku kazemo da je palindrom ako se isto cita i sa leve i sa desne strane). Na primer: niska anavolimilovana je palindrom
   b) rekurzivnu funkciju za proveru da li je niska palindrom
   c) program koji testira rad funkcija pod a) i b)

3. Igra pikado: Prostor je podeljen u zone oblika kružnih prstenova čiji centar je u koordinatnom početku, pri čemu su širine prstenova međusobno različite. Napiši program koji za datu tačku određuje zonu kojoj pripada. Sa standardnog ulaza unosi se broj ***n*** (1 <= n <= 50000), a zatim i ***n*** realnih brojeva zaokruženih na dve decimale, svaki u posebnom redu, koji predstavljaju širine svih kružnih prstenova. Nakon toga se unosi broj ***m*** (1 <= m <= 50000) i zatim ***m*** parova koordinata tačaka (u svakom redu se nalaze dva realna broja zaokružena na dve decimale, razdvojena sa po jednim razmakom). Na standardni izlaz ispisati ***m*** linija. U svakoj liniji ispisati ili indeks zone (broje se od nule) kojoj tačka pripada ili nisku ***izvan*** ako je tačka izvan poslednje zone. Ako je tačka na granici dve zone, smatrati da pripada unutrašnjoj.

Ulaz	Izlaz
3 2.0 3.0 7.0 5 1.0 1.0 2.0 3.0 8.0 7.0 13.2 14.5 0.0 12.0	0 1 2 izvan 2

4. Dat je niz takmičara, za svakog takmičara poznato je njegovo ime i broj poena na takmičenju. Napisati program kojim se sortira niz učenika nerastuće po broju poena, a ako dva takmičara imaju isti broj poena, onda ih urediti po imenu u nerastućem poretku. U prvoj liniji standardnog ulaza nalazi se prirodan broj n (n <= 100). U sledećih n linija nalaze se redom elementi niza. Za svakog takmičara, u jednoj liniji, nalazi se odvojeni jednim blanko simbolom, njegovo ime (dužine najviše 20) i broj poena (prirodan broj iz intervala [0,300]) koje je takmičar osvojio. Na standardni izlaz ispisati elemente uređenog niza takmičara, za svakog takmičara u jednoj liniji prikazati njegovo ime i broj poena, odvojeni jednim blanko simbolom.

Ulaz	Izlaz
5 Magdalina 200 MarkoG 237 MarkoS 232 Mladen 300 Mihailo 212	Mladen 300 MarkoG 237 MarkoS 232 Mihailo 212 Magdalina 200

Resenje:

1.

2. Ideja - dovoljno je proveriti da li je prva polovina stringa jednaka obrnutoj drugoj polovini (pri čemu se u slučaju neparnog broja karaktera središnji karakter ne uračunava ni u jednu od dve polovine).

a) C rešenje

```
int palindrom(char s[])
{   int i, j;
  int sl = strlen(s);
   for(i=0, j=sl-1; i<j; i++, j--)
     if(s[i]!=s[j])       return 0;
   return 1;
}

/*GLAVNA FUNKCIJA*/

int main()
{  char s[Nmax];
 printf("Unesite nisku za koju proveravate da li je palindrom:\n");
 scanf("%s", s);
 if(palindrom(s))    printf("Niska %s je palindrom\n");
 else    printf("Niska %s nije palindrom\n");
 return 0;
}
```

a) C++ resenje

```
bool palindrom2(string s)

{

    int n = s.size();

    for(int i = 0; i < n/2; i++)

        if (s[i] != s[n-1-i])

            return false;

    return true;

}
```

C++ rešenja sa bibliotečkim funkcijama

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

bool palindrom1 (string s)
{
    string s_kopija;
      s_kopija=s;
      reverse(s_kopija.begin(), s_kopija.end());
      if (s==s_kopija) return true;
      else return false;
}



bool palindrom2(string s)
{
```

```
   string s_kopija;
    s_kopija=s;
    reverse(s_kopija.begin(), s_kopija.end());
    if (s. compare(s_kopija)==0)        return true;
    else return false;
}

bool palindrom3(string s)
{
    if (equal(s.begin(), s.end(), s.rbegin()))       return true;
    else return false;
}

bool palindrom4(string s)
{
    if (s == string(s.rbegin(), s.rend()))
       return true;
    else return false;
}

int main()
{
    string s;
   cin >> s;

    if (palindrom1(s))
        cout << "Palindrom" << endl;
    else cout << "Nije palindrom" << endl;

    if (palindrom2(s))
        cout << "Palindrom" << endl;
    else cout << "Nije palindrom" << endl;

    if (palindrom3(s))
        cout << "Palindrom" << endl;
    else cout << "Nije palindrom" << endl;

     if (palindrom4(s))
        cout << "Palindrom" << endl;
    else cout << "Nije palindrom" << endl;

    return 0;
}
```

Kraći kod (ali ne obavezno i efikasno rešenje) moguće je postići primenom bibliotečkih funkcija.
Na primer, primenom bibliotečkih funkcija implementiramo proveru da li je polazni niz jednak nizu koji se dobija njegovim obrtanjem.

Postoji nekoliko načina da se obrne niska *s* u jeziku C++. Jedan je da se napravi njena kopija *t*, a onda da se upotrebi funkcija *reverse(t.begin(), t.end())*
Vremenska složenost je proporcionalna polovini broja znakova u nisci t s obzirom da reverse vrši trampu tj. swap karaktera s početka i kraja stringa.

ALI , još bolji način je da se upotrebi konstruktor kojem se prosleđuju reverzni iteratori (oni string obilaze sdesna na levo) tj. *string(s.rbegin(), s.rend())*

Obrtanje stringa na načine koje smo prikazali uzrokuje izgradnju novog stringa u memoriji, što je memorijski nepotrebno zahtevno i time su prethodna rešenja zasnovana na bibliotečkoj funkcionalnosti neefikasnija nego ona ručno implementirana.

U jeziku C++, provera se može obraditi upotrebom funkcije *equal* koja proverava da li su dve segmenta elemenata niza jednaka. Argumenti ove funkcije su iteratori koji određuju prvi segment (iterator na njegov prvi element i iterator koji ukazuje neposredno iza njegovog poslednjeg elementa) i iterator koji ukazuje na prvi element drugog segmenta. Drugi iterator može biti i reverzni, čime se postiže da se prilikom poređenja drugi segment obilazi unazad, s desna na levo. Time se uslov palindroma svodi na ***equal(s.begin(), s.end(), s.rbegin())***
ili još bolje na ***equal(s.begin(), next(s.begin(), s.size() / 2),s.rbegin())***
Ovo rešenje je veoma efikasno.

PODSEĆANJE:

1.1 string konstruktor (C++ string - kreiranje)

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{

  string s0 ("Pera Mika Laza");


  // konstruktori
   string s1;
  string s2 (s0);
  string s3 (s0, 6, 5);

  string s4 ("Ana voli Milovana");
  string s5 ("Ana voli Milovana", 5);

  string s6a (10, 'x');
  string s6b (10, 46);      // ASCII code za '.' je 46

  string s7 (s0.begin(), s0.begin()+7);


  cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;

  cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6a: " << s6a;

  cout << "\ns6b: " << s6b << "\ns7: " << s7 << '\n';

  return 0;

}
```

IZLAZ

```
s1:
s2: Pera Mika Laza
s3: ika L
s4: Ana voli Milovana
s5: Ana v
s6a: xxxxxxxxxx
s6b: ..........
s7: Pera Mi
```

b)
Rekurzivne pomoćne (recursive helper) funkcije su vrlo korisne za nalaženje rekurzivnih rešenje, naročito kada se rešavaju problemi koji koriste nizove i stringove. Praksa je pri rekurzivnom programiranju da se definiše drugi potprogram (funkcija) koji prima dodatne parametra. Takva funkcija je poznata kao rekurzivna pomoćna funkcija.

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

```
/* pomocna funkcija, rekurzivna,
  koja proverava da li je palindrom deo stringa s od slova na koje pokazuje l do slova na koje pokazuje d
*/

int palindrompom(char *s, char *l, char *d){
    /* izlaz iz rekurzije:     string duzine 0 i string duzine 1 jeste palindrom
     * do stringa duzine 0 cemo doci  ako krenemo od stringa parne duzine  koji jeste palindrom
     (pogledati rekurzivni korak:
     *  ako su pocetno i krajnje slovo jednaki, l se uveca, a d se smanji  pa kad dodjemo do stringa duzine 0
     oni ce se ukrstiti (d pokazuje levo od l i d-l bude jednako -1)

     * d-l je oduzimanje pokazivaca koje ima smisla samo ako oba pokazivaca pokazuju na elemente istog niza, sto
ovde jeste slucaj i rezultat je razlika indeksa elemenata na koje oni pokazujU  */

    if(d-l<=0)       return 1;

    /* rekurzivni korak: string je palindrom ako su mu pocetno i krajnje slovo jednaki i ako je string bez ta dva slova
takodje palindrom*/

    return *l==*d && palindrompom(s,l+1,d-1);
}



/* funkcija poziva pomocnu funkciju: string je palindrom, ako je palindrom "od pocetka do kraja" */

int palindrom(char *s){
    return palindrompom(s,s, s+strlen(s)-1);
}



int main(){
   char s[100];
   printf("Uneti string:\n");   scanf("%s", s);
   if(palindrom(s))      printf("jeste palindrom\n");
   else
      printf("nije palindrom\n");

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

II resenje

Problem proveravanja da li je neki string palandrom može se podeliti u dva podproblema:

1. Provera da li je prvi znak jednak poslednjem znaku stringa
2. Ignoriši dva krajnja znaka stringa i proveri da li je preostali tekst palindrom.

Drugi podproblem je isti kao što je početni problem samo je manje veličine. Postoje dva osnovna slučaja:

1. Dva krajnja znaka nisu jednaka
2. String je dužine 0 ili 1.

Programsko rešenje ovog problema dato je u funkciji **Palindrom**
Metod substr() u liniji 8 kreira novi string jednak početnom stringu, sem što nema dva znaka (pocetak i kraj). Njegova provera da li je palindorm je identična proveri početnog stringa.

```
bool palindromRek(string s) {
    int n=s.length();
```

```
    if (n <= 1) // izlaz iz rekurzije
      return true;
    else if (s[0]!= s[n-1])// izlaz iz rekurzije
      return false;
    else
      return palindromRek(s.substr(1, n - 2));

 }
```

III resenje

```
#include <stdio.h>
#define MAXREC 21     /* Ogranicava duzinu unete reci */
   int palindrom();      /* F-ja koja proverava da li je uneta rec palindrom */

main()
{
  char rec[MAXREC];     /* Uneta rec je niz karaktera */
  char *s, *t;          /* Promenljiva s je pokazivac na prvo, promenljiva t pokazuje na poslednje slovo unete reci */

   printf("Unesite neku rec:\n");
   gets(rec);     /* Ucitavamo unetu rec */
   s = t = rec;        /* U pocetku i s i t pokazuju na prvi karakter stringa */
   while(*t != '\0')   /* Dok se ne dodje do znaka za kraj stringa */
     t++;              /* povecavamo t */

   t--;                /* Zatim ga smanjimo za 1 kako bo pokazivalo na */
                       /* poslednje slovo u unete reci */

   if (palindrom(rec, s, t) == 1)   /* U zavisnosti od vrednosti koju f-ija */
     printf("Jeste palindrom");     /* palindrom() vraca (1 ako jeste, a 0 */
   else                             /* ako uneta rec nije palindrom), */
     printf("Nije palindrom");      /* ispisuje se odgoverajuca poruka */
}

/* Rekurzivna funkcija proverava da li je uneta rec palindrom */
int palindrom(char rec[], char *s, char *t)
{
   while(s < t)     /* Dovoljno je da uporedimo levu i desnu "polovinu" reci */
    if(*s != *t)   /* Ako se bilo koja dva simetricna slova razlikuju */
       return 0;    /* f-ija vraca nulu */
      else                          /* U ostalim slucajevima funkcija poziva */
       palindrom(rec, ++s, --t);   /* samu sebe, ali se s pomera na */
                                   /* sledece, a t na prethodno slovo */
   if(s >= t)     /* Ako su se s i t "susreli", tj. ako je uneta rec */
     return 1;    /* palindrom, f-ija vraca 1 */
}
```

3. Tačka pripada nekoj zoni ako i samo ako je njeno rastojanje od koordinatnog početka (a njega možemo izračunati korišćenjem Euklidskog rastojanja, tj. Pitagorine teoreme, kao u zadatku ZBIRKA BUBBLE BEE ->Aritmetika/01 Formule/01 Poznate formule/04 rastojanje_tacaka)) strogo veće od unutrašnjeg, a manje ili jednako spoljašnjem prečniku te zone.

Unutrašnji prečnik neke zone jednak je zbiru širina svih zona pre nje, ne uključujući njenu širinu, a spoljašnji prečnik jednak je zbiru širina svih zona pre nje, uključujući i njenu širinu. Važi da je unutrašnji prečnik neke zone jednak spoljašnjem prečniku prethodne zone (osim kod zone 0, koja ima unutrašnji prečnik nula).

Dakle, biće sasvim dovoljno da izračunamo spoljašnje prečnike svih zona, a oni se mogu izračunati kao parcijalni zbirovi širina zona. Njih možemo računati inkrementalno, dodajući svaku novu učitanu širinu na tekući zbir (slično kao što smo u zadatku ZBIRKA BUBBLE BEE 03 Petlje/01 serije_brojeva/08 rekurentne_serije/06 prefiks_najveceg_zbira računali zbirove svih prefiksa) ili, u jeziku C++ bibliotečkom funkcijom `partial_sum`. Kada je poznat niz spoljašnjih

prečnika zona zona u kojem se tačka nalazi se može naći tako što se nađe prvi spoljašnji prečnik zone koji je veći ili jednak rastojanju tačke od koordinatnog početka.
To možemo uraditi binarnom pretragom, bilo ručno implementiranom (kao, na primer, u zadatku ZBIRKA BUBBLE BEE01 /02 prvi_veci_poslednji_manji)), bilo pomoću bibliotečkih funkcija `lower_bound` u jeziku C++.

Resenje 01 //staticki alociran niz zona, linearna pretraga

```
#include <iostream>
#include <algorithm>
#include <cmath>


#define Nmax 50000
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  // ucitavamo n sirina zona i izracunavamo precnik svake od njih
  int n;
  cin >> n;
  double zone [Nmax];

  // poluprecnik prve zone jednak je njenoj sirini

  cin >> zone[0];
  for (int i = 1; i < n; i++) {
    // poluprecnik svake sledece zone jednak je ukupnoj sirini svih zona, ukljucujuci i tu zonu

    double d; cin >> d;
    zone[i] = zone[i - 1] + d;
  }

  // ucitavamo m tacaka
   int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo koordinate tacke (x, y)
    double x, y;
    cin >> x >> y;

    // rastojanje tacke (x, y) od koordinatnog pocetka
    double r = sqrt(x*x + y*y);

    // pronalazimo prvu zonu takvu da joj je poluprecnik veci ili jednak r
    int z;
    for (z = 0; z < n; z++)
      if (zone[z] >= r)  break;

    // ako takva ne postoji, tacka je izvan svih zona
    if (z == n)
      cout << "izvan" << endl;
    else
      cout << z << endl;
  }
  return 0;
}
```

Resenje 02 (upotreba binarne pretrage)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;
const int Nmax=50000;
int main() {
  ios_base::sync_with_stdio(false);


  // ucitavamo n sirina zona i izracunavamo poluprecnik svake od njih
  int n;
  cin >> n;

  double zone[Nmax];
  // poluprecnik prve zone jednak je njenoj sirini
  cin >> zone[0];

  for (int i = 1; i < n; i++) {
    // poluprecnik svake sledece zone jednak je ukupnoj sirini svih zona, ukljucujuci i tu zonu

    double d; cin >> d;
    zone[i] = zone[i - 1] + d;
  }

  // ucitavamo m tacaka
  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    double x, y;
    cin >> x >> y;

    // rastojanje tacke (x, y) od koordinatnog pocetka
    double r = sqrt(x*x + y*y);

    // pronalazimo prvu zonu takvu da joj je poluprecnik veci ili jednak r
    // [0, l) sadrzi elemente manje od r
    // [l, d) sadrzi jos neispitane elemente
    // [d, n) sadrzi elemente vece ili jednake r

    int l = 0, d = n;

    // dok svi elementi ne postanu ispitani tj. dok je [l, d) neprazan
    while (l < d) {
      // nalazimo srednju tacku u [l, d)
      int s = l + (d - l) / 2;

      if (zone[s] >= r)
          // svi elementi iz [s, n) su veci ili jednaki r
          d = s;
      else
          // svi elementi iz [0, s] su manji od r
          l = s + 1;
     }

    // prva zona ciji je poluprecnik veci ili jednak r je d
```

```
    // ako takva ne postoji, tacka je izvan svih zona
    if (d == n)
      cout << "izvan" << endl;
    else
      cout << d << endl;
  }
  return 0;
}
```

Resenje 03 (upotreba STL + vector)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;
int main() {
  ios_base::sync_with_stdio(false);

  // ucitavamo n sirina zona i izracunavamo precnik svake od njih
  int n;
  cin >> n;
  vector<double> zone(n);

  // poluprecnik prve zone jednak je njenoj sirini
  cin >> zone[0];

  for (int i = 1; i < n; i++) {

    // poluprecnik svake sledece zone jednak je ukupnoj sirini svih zona

    // zakljucno sa njom

    double d; cin >> d;

    zone[i] = zone[i - 1] + d;

  }


  // ucitavamo m tacaka

  int m;

  cin >> m;

  for (int i = 0; i < m; i++) {

    // ucitavamo koordinate tacke (x, y)

    double x, y;

    cin >> x >> y;

    // rastojanje tacke (x, y) od koordinatnog pocetka

    double r = sqrt(x*x + y*y);


    // pronalazimo prvu zonu takvu da joj je poluprecnik veci ili jednak r

    int z;

    for (z = 0; z < zone.size(); z++)

      if (zone[z] >= r)
```

```
            break;

    // ako takva ne postoji, tacka je izvan svih zona
    if (z == zone.size())
      cout << "izvan" << endl;
    else
      cout << z << endl;
  }
  return 0;
}
```

Resenje 04 (BINARNA PRETRAGA + STL + vector)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  // ucitavamo n sirina zona i izracunavamo poluprecnik svake od njih
  int n;
  cin >> n;
  vector<double> zone(n);
  // poluprecnik prve zone jednak je njenoj sirini
  cin >> zone[0];
  for (int i = 1; i < n; i++) {
    // poluprecnik svake sledece zone jednak je ukupnoj sirini svih zona
    // zakljucno sa njom
    double d; cin >> d;
    zone[i] = zone[i - 1] + d;
  }

  // ucitavamo m tacaka
  int m;
```

```
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo koordinate tacke (x, y)
    double x, y;
    cin >> x >> y;
    // rastojanje tacke (x, y) od koordinatnog pocetka
    double r = sqrt(x*x + y*y);

    // pronalazimo prvu zonu takvu da joj je poluprecnik veci ili jednak r
    // [0, l) sadrzi elemente manje od r
    // [l, d) sadrzi jos neispitane elemente
    // [d, n) sadrzi elemente vece ili jednake r
    int l = 0, d = n;
    // dok svi elementi ne postanu ispitani tj. dok je [l, d) neprazan
    while (l < d) {
      // nalazimo srednju tacku u [l, d)
      int s = l + (d - l) / 2;

      if (zone[s] >= r)
          // svi elementi iz [s, n) su veci ili jednaki r
          d = s;
      else
          // svi elementi iz [0, s] su manji od r
          l = s + 1;
    }
    // prva zona ciji je poluprecnik veci ili jednak r je d

    // ako takva ne postoji, tacka je izvan svih zona
    if (d == zone.size())
      cout << "izvan" << endl;
    else
      cout << d << endl;
  }
  return 0;
}
```

Resenje 05

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  // ucitavamo niz sirina zona
  int n;
  cin >> n;
  vector<double> zone(n);
  for (int i = 0; i < n; i++)
    cin >> zone[i];

  // izracunavamo poluprecnike zona (kao zbir sirina svih zona zakljucno sa tom)
  partial_sum(zone.begin(), zone.end(), zone.begin());

  // ucitavamo m tacaka
  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo koordinate tacke (x, y)
    double x, y;
    cin >> x >> y;
    // rastojanje tacke (x, y) od koordinatnog pocetka
    double r = sqrt(x*x + y*y);

    // pronalazimo prvu zonu takvu da joj je poluprecnik veci ili jednak r
    auto it = lower_bound(zone.begin(), zone.end(), r);

    // ako takva ne postoji, tacka je izvan svih zona
    if (it == zone.end())
      cout << "izvan" << endl;
```

```
    else
      cout << distance(zone.begin(), it) << endl;
  }
  return 0;
}
```

4.

Prvo pitanje koje je potrebno razrešiti je kako čuvati podatke o takmičarima. Najprirodnije rešenje da se podaci o takmičaru pamte u strukturi Takmicar čiji su elementi ime takmičara (podatak tipa string) i broj poena (podatak tipa int). Za čuvanje informacija o svim takmičarima možemo onda upotrebiti niz ili vektor tj. listu struktura. Druge mogućnosti su da se podaci čuvaju u dva niza (nizu imena i nizu poena) ili da se umesto struktura koriste parovi tj. torke (tipovi pair ili tuple u jeziku C++).
Sortiranje je moguće izvršiti na razne načine. Jedan od načina je da upotrebiti bibliotečku funkciju sortiranja (funkciju sort u jeziku C++, bez obzira na to da li se koristi vektor ili niz).

Funkciji sortiranja je potrebno dostaviti i funkciju poređenja kojom se zapravo određuje redosled elemenata nakon sortiranja. Telo te funkcije poređenja vrši višekriterijumsko (leksikografsko) poređenje uređenih dvojki podataka, slično onom koje smo videli u zadacima [Punoletstvo](Zbirka/02 Grananje/04 Slozeno grananje/01 leksikografsko/01 punoletstvo), [Pobednik u dve discipline](Zbirka/02 Grananje/04 Slozeno grananje/01 leksikografsko/02 pobednik_u_dve_discipline) ili [Pobednik u tri discipline](Zbirka/03 Petlje/01 serije_brojeva/05 maksimum_minumum/10 pobednik_u_tri_discipline). Prvo se poredi broj poena i ako je broj poena prvog takmičara manji funkcija poređenja vraća false (jer on treba da ide iza drugog takmičara), ako je veći vraća se true (jer on treba da ide ispred drugog takmičara), a ako su brojevi poena jednaki, prelazi se na poređenje imena. Za to se može uporediti bibliotečko abecedno (leksikografsko) poređenje niski (slično kao u zadatku [Leksikografski minimum](../../../01 karakteri_i_niske/04 leksikografski_minimum)).
U jeziku C++ postoji nekoliko načina da definišemo funkciju poređenja. Prvi je da se definiše klasična (globalna) funkcija koja prima dve strukture (podatke o dva takmičara) i koja vrača vrednost tipa bool i to vrednost true ako prvi takmičar treba da bude ispred drugog u konačnom sortiranom nizu, tj. vrednost false u suprotnom. Efikasnosti radi, strukture koje se porede ima smisla preneti preko konstantnih referenci. Ta funkcija dobija svoje ime (na primer, poredi) i to ime se onda navodi kao treći parametar u pozivu bibliotečke funkcije sort (nakon dva iteratora koji određuju raspon koji se sortira).

```
bool porediTakmicare(const Takmicar& a, const Takmicar& b) {
  ...
}
sort(a.begin(), a.end(), porediTakmicare);
```

Drugi način je da se umesto imenovane funkcije, funkcija poređenja definiše kao anonimna funkcija tj. kao lambda-izraz. Parametri i povratna vrednost te funkcije su isti kao u slučaju imenovane funkcije, pri čemu se povratni tip ne mora eksplicitno navesti. Pošto funkcija poređenja u ovom slučaju ne mora da koristi druge podatke osim podataka o takmičarima koji se porede, kao parametre zatvorenja te funkcije (parametri koji se navode u zagradama []) nema potrebe navoditi ništa. Anonimna funkcija se navodi direktno u pozivu funkcije sort kao njen treći argument.

```
sort(a.begin(), a.end(),
    [](const Takmicar& a, const Takmicar& b) {
        ...
    });
```

Treći način je da se definiše struktura (ili klasa) koja implementira funkcionalnost poređenja dva takmičara u okviru svoje metode operator(). Paramteri i povratna vrednost te funkcije je ista kao i u prethodnim slučajevima, a u pozivu funkcije sort navodi se jedan objekat te strukture (ili klase).

```
struct PoredjenjeTakmicara {
  bool operator() (const Takmicar& a, const Takmicar& b) {
     ...
  }
}

sort(a.begin(), a.end(), PoredjenjeTakmicara());
```

Prirodno je da takav objekat "upoređivač" bude objekat posebne strukture ili klase, a moguće je dodati ga i nekoj već postojećoj strukturi (na primer, strukturi Takmicar).

Četvrti način je da se u samoj strukturi koja čuva podatke o takmičaru definiše metoda pod imenom operator<kojom se obezbeđuje da se strukture A i B mogu porediti pomoću A < B. Ta metoda ima jedan parametar koji predstavlja drugog takmičara, dok je prvi takmičar ona struktura na kojoj se taj metod poziva.

```
struct Takmicar {
    string ime;
    int brojPoena;

  bool operator<(const Takmicar& A) {
     ...
  }
};

sort(a.begin(), a.end());
```

Resenje 01

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
struct Takmicar {
    string ime;
    int brojPoena;
};
int main() {
  int n;
  cin >> n;
  vector<Takmicar> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i].ime >> a[i].brojPoena;
```

```
  sort(a.begin(), a.end(),
      [] (const Takmicar& A, const Takmicar& B) {
            if (A.brojPoena > B.brojPoena)
              return true;
            if (A.brojPoena < B.brojPoena)
              return false;
            return A.ime <= B.ime;
      });

  for(int i = 0; i < n; i++)
    cout << a[i].ime << " " << a[i].brojPoena << endl;
    return 0;
}
```

Resenje 02

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;

struct Takmicar {
  string ime;
  int brojPoena;
};

struct PoredjenjeTakmicara {
  bool operator() (const Takmicar& A, const Takmicar& B) {
    return A.brojPoena > B.brojPoena ||
          (A.brojPoena == B.brojPoena && A.ime <= B.ime);
 }
};

int main() {
   int n;
  cin >> n;
  vector<Takmicar> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i].ime >> a[i].brojPoena;
  sort(a.begin(), a.end(), PoredjenjeTakmicara());
  for(int i = 0; i < n; i++)
    cout << a[i].ime << " " << a[i].brojPoena << endl;
    return 0;}
```

Resenje 03

```
#include <iostream>
#include <algorithm>
#include <string>

using namespace std;

struct Takmicar {
    string ime;
    int brojPoena;

  bool operator<(const Takmicar& A) {
    if (brojPoena > A.brojPoena)
      return true;
    if (brojPoena < A.brojPoena)
      return false;
    return ime <= A.ime;
}
};

int main() {
  int n;
  cin >> n;

  Takmicar a[200];
  for(int i = 0; i < n; i++)
    cin >> a[i].ime >> a[i].brojPoena;

  sort(a, a+n);

  for(int i = 0; i < n; i++)
    cout << a[i].ime << " " << a[i].brojPoena << endl;
  return 0;
}
```

Resenje 04

```
#include <iostream>
#include <algorithm>
#include <string>
#include <utility>

using namespace std;

int main() {

  int n;
  cin >> n;
  vector<pair<int, string>> a(n);

  for(int i = 0; i < n; i++) {
    string ime; int brojPoena;
    cin >> ime >> brojPoena;
    a[i] = make_pair(-brojPoena, ime);
   }

  sort(a.begin(), a.end());

  for(int i = 0; i < n; i++)
    cout << a[i].second << " " << a[i].first << endl;
```

```
  return 0;
}
```

Resenje 05

```
#include <iostream>
#include <algorithm>
#include <string>

using namespace std;

struct Takmicar {
    string ime;
    int brojPoena;
};

bool uporedi(Takmicar A, Takmicar B) {
  if (A.brojPoena > B.brojPoena)
    return true;
  if (A.brojPoena < B.brojPoena)
    return false;

  return A.ime <= B.ime;

}


int main() {
  int n;
  cin >> n;

  Takmicar a[200];
  for(int i = 0; i < n; i++)
    cin >> a[i].ime >> a[i].brojPoena;

//implementacija sortiranjam slozenost O(n^2)

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


  for(int i = 0; i < n; i++)
    cout << a[i].ime << " " << a[i].brojPoena << endl;

  return 0;
}
```