Name : Ritabroto Ganguly Roll: 001910501090

OOP Assignment 7

1. Create an array of student objects (containing roll, name, name and score) whose size may vary dynamically once objects are added or removed, randomly elements may be accessed, one can find number of objects in the list, one can find the student with highest score, find the students with a substring in their name and also without a substring in the name. Take the help of suitable STL classes.

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

class student{
  static int rolls;
  int roll;
  string name;
  int score;
  public:
  student(const string& name,const int score):name(name),score(score),roll(rolls)
{rolls++;}
  int get_score() const {return score;}
  string get_name() const {return name;}
  int get_roll() const {return roll;}
  void set_score(int x) {score = x;}
  static bool find_max(const student& a,const student& b){return a.score<b.score;}
  friend ostream& operator<<(ostream& o,const student& s){
    o<<s.roll<<" "<<s.name<<" "<<s.score<<endl;
    return o;
  }
};
int student::rolls = 1;

class string_matcher{
   const string pattern;
   const bool with;//boolean indicating if name with pattern is to be found or
without the pattern
   public:
   string_matcher(const string& pattern,bool with):pattern(pattern),with(with){}
   bool operator()(const student& stu) const {
      string name = stu.get_name();
      if(name.size()<pattern.size())
         return !with;
```

```
    int i = 0;
    while(i <= name.size()-pattern.size()){
      int flag = 1;
      for(int j = 0; j < pattern.size(); j++){
        if(name.at(i) != pattern.at(j)){
          flag = 0;
          if(j==0)
            i++;
          break;
        }
        i++;
      }
      if(flag)
        return with;
    }
    return !with;
  }
};

int main()
{
 const string names[] =
{"Ganguly","Banerjee","Ganguly","Chakrabarti","Chatterjee"};
 vector<student> students;
 for(int i = 0; i < 5; i++){
    students.push_back(student(names[i],(i+1)));
 }

 vector<student>::iterator it =
max_element(students.begin(),students.end(),student::find_max);
 if(it!=students.end())
   cout<<"Max Scorer is: "<<*it;

 vector<student>::iterator start = students.begin();
 while(start!=students.end()){
    start = find_if(start,students.end(),string_matcher("jee",true));
    if(start!=students.end()){
      cout<<*start;
      start++;
    }
 }
}
```

```
Max Scorer is: 5 Chatterjee 5
2 Banerjee 2
5 Chatterjee 5
```

2. Create an array of student objects where along with the support mentioned in Q.1, one can remove an object with specific roll, sort the collection in the descending order and show the same; two student collections can also be combined. Take the help of suitable STL class.

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

class student{
  static int rolls;
  int roll;
  string name;
  int score;
  public:
  student(const string& name,const int score):name(name),score(score),roll(rolls)
{rolls++;}
  int get_score() const {return score;}
  string get_name() const {return name;}
  int get_roll() const {return roll;}
  void set_score(int x) {score = x;}
  static bool find_max(const student& a,const student& b){return a.score<b.score;}
  friend ostream& operator<<(ostream& o,const student& s){
    o<<s.roll<<" "<<s.name<<" "<<s.score<<endl;
    return o;
  }
};
int student::rolls = 1;

class string_matcher{
   const string pattern;
   const bool with;//boolean indicating if name with pattern is to be found or
without the pattern
   public:
   string_matcher(const string& pattern,bool with):pattern(pattern),with(with){}
   bool operator()(const student& stu) const {
      string name = stu.get_name();
      if(name.size()<pattern.size())
         return !with;
      int i = 0;
      while(i <= name.size()-pattern.size()){
         int flag = 1;
         for(int j = 0; j < pattern.size(); j++){
            if(name.at(i) != pattern.at(j)){
              flag = 0;
              if(j==0)
```

```
            i++;
          break;
        }
        i++;
      }
      if(flag)
        return with;
    }
    return !with;
  }
};

class remove_by_roll{
  int roll;
  public:
  remove_by_roll(int roll):roll(roll){}
  bool operator()(const student& s) const {
    return roll==s.get_roll();
  }
};

class sort_desc_by{
  int choice;
  public:
  sort_desc_by(int c):choice(c){}
  bool operator()(const student& a,const student& b) const {
    switch(choice){
      case 1: return a.get_roll()>b.get_roll();
      case 2: return a.get_name()>b.get_name();
      case 3: return a.get_score()>b.get_score();
      default : cout<<"Invalid input to sort_desc_by"<<endl;exit(0);
    }
  }
};

void print_students(const vector<student>& x){
  vector<student>::const_iterator i = x.cbegin();
  for(;i!=x.cend();i++)
    cout<<*i;
}

int main()
{
 const string names[] = {"Ganguly","Banerjee","Ganguly","Chakrabarti","Pal"};
 vector<student> students;
 for(int i = 0; i < 5; i++){
    students.push_back(student(names[i],(i+1)));
 }
```

```
 print_students(students);
 cout<<"------------------------"<<endl;

 vector<student>::iterator it =
max_element(students.begin(),students.end(),student::find_max);
 if(it!=students.end())
    cout<<"Max Scorer is: "<<*it;
 cout<<"------------------------"<<endl;

 vector<student>::iterator start = students.begin();
 while(start!=students.end()){
      start = find_if(start,students.end(),string_matcher("Ganguly",false));
      if(start!=students.end()){
        cout<<*start;
        start++;
      }
 }
 cout<<"------------------------"<<endl;
 sort(students.begin(),students.end(),sort_desc_by(2));
 print_students(students);
 cout<<"------------------------"<<endl;

 vector<student> students2;
 for(int i = 0; i < 5; i++){
    students2.push_back(student("name"+to_string((i+1)),(i+1)));
 }
 print_students(students2);
 cout<<"------------------------"<<endl;

 vector<student> students3;
 for(int i=0;i<students.size();i++)
    students3.push_back(students[i]);
 for(int i=0;i<students2.size();i++)
    students3.push_back(students2[i]);
 print_students(students3);
 cout<<"sizeof students3: "<<students3.size()<<endl;
 cout<<"------------------------"<<endl;
 cout<<"Input roll to delete: ";int x;
 cin>>x;
 it = remove_if(students3.begin(),students3.end(),remove_by_roll(x));//logical
deletion
 if(it!=students3.end()){//actual deletion
      students3.erase(it,students3.end());
 }

 cout<<"sizeof students3: "<<students3.size()<<endl;
 print_students(students3);
}
```

```
1 Ganguly 1
2 Banerjee 2
3 Ganguly 3
4 Chakrabarti 4
5 Pal 5
-------------------------
Max Scorer is: 5 Pal 5
-------------------------
2 Banerjee 2
4 Chakrabarti 4
5 Pal 5
-------------------------
5 Pal 5
1 Ganguly 1
3 Ganguly 3
4 Chakrabarti 4
2 Banerjee 2
-------------------------
6 name1 1
7 name2 2
8 name3 3
9 name4 4
10 name5 5
-------------------------
5 Pal 5
1 Ganguly 1
3 Ganguly 3
4 Chakrabarti 4
2 Banerjee 2
6 name1 1
7 name2 2
8 name3 3
9 name4 4
10 name5 5
sizeof students3: 10
-------------------------
Input roll to delete: 6
sizeof students3: 9

5 Pal 5
1 Ganguly 1
3 Ganguly 3
4 Chakrabarti 4
2 Banerjee 2
7 name2 2
8 name3 3
9 name4 4
10 name5 5
```

3. Students come to mark sheet collection desk and are served in first come first served basis. Implement the scenario. Take the help of suitable STL class.

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

class marksheet{
   static int rolls;
   int roll;
   string name;
   int score;
   public:
   marksheet(const string& name,const int score):roll(rolls),name(name),score(score)
{rolls++;}
   friend ostream& operator<<(ostream& o,const marksheet& m){
      o<<m.roll<<" "<<m.name<<" "<<m.score<<endl;
      return o;
   }
   bool operator==(const int r) const {return roll==r;}
};
int marksheet::rolls = 1;

int main()
{

 vector<marksheet> marksheets;
 for(int i=0;i<100;i++)//filling marksheets upto 100 rolls
    marksheets.push_back(marksheet("name"+to_string((i+1)),(i+1)));

 queue<int> q;
 int x;
 do{
   cout<<"Enter roll (enter 0 to cancel): ";
   cin>>x;
   if(x>100 || x<0){//max roll 100 (programmer decided)
     cout<<"Enter a valid roll"<<endl;
     continue;
   }
   if(x!=0)
     q.push(x);
 }while(x>0);

 while(!q.empty()){
    vector<marksheet>::iterator it =
find(marksheets.begin(),marksheets.end(),q.front());
```

```
    q.pop();
    if(it!=marksheets.end())
      cout<<*it;
    else
      cout<<"Marksheet Not Found!"<<endl;
  }
 }
```

```
Enter roll (enter 0 to cancel): 1
Enter roll (enter 0 to cancel): 2
Enter roll (enter 0 to cancel): 3
Enter roll (enter 0 to cancel): 9
Enter roll (enter 0 to cancel): 6
Enter roll (enter 0 to cancel): 0
1 name1 1
2 name2 2
3 name3 3
9 name9 9
6 name6 6
```

4. Maintain a container of students where they are kept in the descending order of their scores. Take the help of suitable STL class.

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

class student{
   static int rolls;
   int roll;
   string name;
   public:
   student(const string& name):name(name),roll(rolls){rolls++;}
   friend ostream& operator<<(ostream& o,const student& s) {
      o<<s.roll<<" "<<s.name;
      return o;
   }
};
int student::rolls = 1;

int main()
```

```
{
 multimap<int,student,greater<int> > m;
 int x = 0;
 do{
   string name;
   cout<<"Enter name: ";
   getline(cin,name);
   cout<<"Enter score: ";
   cin>>x;
   m.insert(pair<int,student>(x,student(name)));
   cout<<"Enter another? (yes=1,no=0) ";
   cin>>x;
   if(x!=0)
     cin.ignore();
 }while(x>0);

 for(multimap<int,student,greater<int> >::iterator it = m.begin();it!=m.end();it++)
    cout<<it->second<<" "<<it->first<<endl;
}
```

```
Enter name: rg
Enter score: 12
Enter another? (yes=1,no=0) 1
Enter name: ag
Enter score: 15
Enter another? (yes=1,no=0) 1
Enter name: sg
Enter score: 11
Enter another? (yes=1,no=0) 0
2 ag 15
1 rg 12
3 sg 11
```

5. Store the roll and score of the students in a map in the sorted order of roll. One should be able to retrieve the score for a given roll. Take the help of suitable STL class.

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

class student{
   int score;
   string name;
   public:
   student(const string& name,const int score):name(name),score(score){}
   string get_name() const {return name;}
   int get_score() const {return score;}
   friend ostream& operator<<(ostream& o,const student& s) {
      o<<s.name<<" "<<s.score<<endl;
      return o;
   }
};

int main()
{
 map<int,student> m;
 int x = 0;
 do{
   string name;int score;
   cout<<"Enter name: ";
   getline(cin,name);
   cout<<"Enter score: ";
   cin>>score;
   cout<<"Enter roll: ";
   cin>>x;

   if(!m.insert(pair<int,student>(x,student(name,score))).second)
     cout<<"Student with roll "<<x<<" is already present"<<endl;

   cout<<"Enter another? (yes=1,no=0) ";
   cin>>x;
   if(x!=0)
     cin.ignore();
 }while(x>0);

do{
 cout<<"Enter roll to search: ";cin>>x;
 if(m.find(x)!=m.end())
   cout<<"Score for roll "<<x<<" is "<<m.at(x).get_score()<<endl;
```

```
 else
   cout<<"Student with roll "<<x<<" not found!"<<endl;

 cout<<"Search another? (yes=1,no=0) ";cin>>x;
}while(x>0);

 for(map<int,student>::iterator it = m.begin();it!=m.end();it++)
    cout<<it->first<<" "<<it->second;
}
```

```
Enter name: rg
Enter score: 12
Enter roll: 1
Enter another? (yes=1,no=0) 1
Enter name: ag
Enter score: 13
Enter roll: 5
Enter another? (yes=1,no=0) 1
Enter name: sg
Enter score: 1333
Enter roll: 3
Enter another? (yes=1,no=0) 0
Enter roll to search: 5
Score for roll 5 is 13
Search another? (yes=1,no=0) 0
1 rg 12
3 sg 1333
5 ag 13
```