Assignment 7

Q1. 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.

```
#define pb emplace_back

#include <bits/stdc++.h>

using namespace std;


class Student {

public:

    int roll;

    string name;

    int score;


    Student(int r, const string& n, int s) : roll(r), name(n), score(s) {}

};


class StudentList {

private:

    vector<Student> students;


public:

    void addStudent(const Student& student) {

        students.pb(student);

    }
```

```
void removeStudent(int roll) {
    for(auto it = students.begin(); it != students.end(); ++it) {
        if (it->roll == roll) {
            students.erase(it);
            break;
        }
    }
}

Student* getStudent(int roll) {
   for(auto it = students.begin(); it != students.end(); ++it) {
        if (it->roll == roll) {
            return &(*it);
        }
    }
    return nullptr;
}

size_t getSize() const {
    return students.size();
}

Student getHighestScoreStudent() const {
    int max=0,idx=-1;
    for(int i=0;i<students.size();i++){
        if(students[i].score>max){
            max=students[i].score;
```

```
                idx=i;
            }
        }
        return students[idx];
    }

    vector<Student> findStudentsWithSubstring(const string& substring) const {
        vector<Student> result;
        for(int i=0;i<students.size();i++){
            if(students[i].name.find(substring)!=string::npos){
                result.pb(students[i]);
            }
        }
        return result;
    }

    vector<Student> findStudentsWithoutSubstring(const string& substring) const {
        vector<Student> result;
        for(int i=0;i<students.size();i++){
            if(students[i].name.find(substring)==string::npos){
                result.pb(students[i]);
            }
        }
        return result;
    }
};
```

```
int main() {
    StudentList list;
    int choice;
    do {
        cout << "\nMenu:\n";
        cout << "1. Add Student\n";
        cout << "2. Remove Student\n";
        cout << "3. Get Student\n";
        cout << "4. Get Total Students\n";
        cout << "5. Get Highest Score Student\n";
        cout << "6. Find Students With Substring\n";
        cout << "7. Find Students Without Substring\n";
        cout << "8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int roll, score;
                string name;
                cout << "Enter roll number: ";
                cin >> roll;
                if(list.getStudent(roll)) {
                    cout << "Student with roll number " << roll << " already exists." << endl;
                    break;
                }
                cout << "Enter name: ";
    cin >> name;
    cout << "Enter score: ";
    cin >> score;
    list.addStudent(Student(roll, name, score));
    break;
}
case 2: {
    int roll;
    cout << "Enter roll number to remove: ";
    cin >> roll;
    if (!list.getStudent(roll)) {
        cout << "Student with roll number " << roll << " does not exist." << endl;
        break;
    }
    list.removeStudent(roll);
    break;
}
case 3: {
    int roll;
    cout << "Enter roll number to get: ";
    cin >> roll;
    Student* student = list.getStudent(roll);
    if (student) {
        cout << "Student found: " << student->name << " with score " << student->score << endl;
    } else {
        cout << "Student not found." << endl;
```

```
            }
            break;
        }
        case 4: {
            cout << "Total students: " << list.getSize() << endl;
            break;
        }
        case 5: {
            Student highest = list.getHighestScoreStudent();
            cout << "Highest scoring student: " << highest.name << " with score " << highest.score <<
endl;
            break;
        }
        case 6: {
            string substring;
            cout << "Enter substring to search: ";
            cin >> substring;
            vector<Student> withSubstring = list.findStudentsWithSubstring(substring);
            if(withSubstring.size()==0){
                cout << "No student found with substring '" << substring << "'" << endl;
                break;
            }
            cout << "Students with substring '" << substring << "':" << endl;
            for (const auto& student : withSubstring) {
                cout << student.name << endl;
            }
            break;
        }

        case 7: {

            string substring;

            cout << "Enter substring to search: ";

            cin >> substring;

            vector<Student> withoutSubstring = list.findStudentsWithoutSubstring(substring);

            if(withoutSubstring.size()==0){

                cout << "No student found without substring '" << substring << "'" << endl;

                break;

            }

            cout << "Students without substring '" << substring << "':" << endl;

            for (const auto& student : withoutSubstring) {

                cout << student.name << endl;

            }

            break;

        }

        case 8: {

            cout << "Exiting..." << endl;

            break;

        }

        default: {

            cout << "Invalid choice. Please try again." << endl;

            break;

        }

    }

} while (choice != 8);
```

```
    return 0;
}
```

Q2. 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 <bits/stdc++.h>
using namespace std;
class Student {
public:
    int roll;
    string name;
    float marks;

    Student(int r, string n, float m) : roll(r), name(n), marks(m) {}

    void display() const {
        cout << "Roll: " << roll << ", Name: " << name << ", Marks: " << marks << endl;
    }
};


class StudentCollection {
private:
    vector<Student> students;

public:
    void addStudent(const Student &student) {
```

```
    students.push_back(student);
}

void removeStudentByRoll(int roll) {
    for(auto it = students.begin(); it != students.end(); ++it) {
        if (it->roll == roll) {
            students.erase(it);
            break;
        }
    }
}

void sortStudents() {
    for(int i=0;i<students.size();i++){
        bool swapped=false;
        for(int j=0;j<students.size()-i-1;j++){
            if(students[j].marks<students[j+1].marks){
                swap(students[j],students[j+1]);
                swapped=true;
            }
        }
    if(!swapped) break;
    }
}

void displayStudents() const {
    for (const auto &student : students) {
            student.display();
        }
    }

    void combineCollections(const StudentCollection &other) {
        students.insert(students.end(), other.students.begin(), other.students.end());
    }
};

int main() {
    StudentCollection collection1, collection2;
    int choice;
    do {
        cout << "\nMenu:\n";
        cout << "1. Add Student\n";
        cout << "2. Remove Student by Roll\n";
        cout << "3. Display Students\n";
        cout << "4. Sort Students\n";
        cout << "5. Combine Collections\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int roll;
                string name;
    float marks;
    cout << "Enter roll: ";
    cin >> roll;
    cout << "Enter name: ";
    cin >> name;
    cout << "Enter marks: ";
    cin >> marks;
    collection1.addStudent(Student(roll, name, marks));
    break;
}
case 2: {
    int roll;
    cout << "Enter roll to remove: ";
    cin >> roll;
    collection1.removeStudentByRoll(roll);
    break;
}
case 3:
    collection1.displayStudents();
    break;
case 4:
    collection1.sortStudents();
    cout << "Students sorted by marks in descending order.\n";
    //collection1.displayStudents();
    break;
case 5:
```

```
            int n;
            cout << "Enter number of students to add to second collection: ";
            cin >> n;
            for (int i = 0; i < n; i++) {
                int roll;
                string name;
                float marks;
                cout << "Enter roll: ";
                cin >> roll;
                cout << "Enter name: ";
                cin >> name;
                cout << "Enter marks: ";
                cin >> marks;
                collection2.addStudent(Student(roll, name, marks));
            }
            collection1.combineCollections(collection2);
            break;
        case 6:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

Q3. 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 <bits/stdc++.h>

using namespace std;

int main() {

    queue<string> students;

    string student;

    char choice;


    do {

        cout << "Enter student name: ";

        cin >> student;

        students.push(student);


        cout << "Do you want to add another student? (y/n): ";

        cin >> choice;

    } while (choice == 'y' || choice == 'Y');


    cout << "\nServing students in first come first served order:\n";

    while (!students.empty()) {

        cout << "Serving student: " << students.front() << endl;

        students.pop();

    }


    return 0;

}
```

Q4. Maintain a container of students where they are kept in the descending order of their scores. Take the help of suitable STL class.

```
#include <bits/stdc++.h>

using namespace std;

class Student {

public:

    string name;

    int score;


    Student(string n, int s) : name(n), score(s) {}


    bool operator<(const Student& other) const {

        return score > other.score;

    }

};


int main() {

    set<Student> students;

    students.insert(Student("Alice", 90));

    students.insert(Student("Bob", 85));

    students.insert(Student("Charlie", 95));


    for (const auto& student : students) {

        cout << "Name: " << student.name << ", Score: " << student.score << endl;

    }


    return 0;

}
```

Q5. 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 <bits/stdc++.h>

using namespace std;

struct DescendingComparator

{

    bool operator()(const int &a, const int &b) const

    {

        return a > b;

    }

};


int main() {

    map<int, vector<int>, DescendingComparator> studentScores;

    int roll, score;

    char choice;


    // Input student roll and score

    do {

        cout << "Enter roll number: ";

        cin >> roll;

        cout << "Enter score: ";

        cin >> score;

        studentScores[score].push_back(roll);


        cout << "Do you want to enter another student? (y/n): ";

        cin >> choice;
    } while (choice == 'y' || choice == 'Y');


    // Retrieve score for a given roll number

    cout << "Enter roll number to retrieve score: ";

    cin >> roll;

    for(auto it = studentScores.begin(); it != studentScores.end(); it++) {

        for(auto i = it->second.begin(); i != it->second.end(); i++) {

            if(*i == roll) {

                cout << "Score for roll number " << roll << " is " << it->first << endl;

                break;

            }

        }

    }


    return 0;

}
```