**Assignment 7**

Q1.

CODE:

#include<iostream>

#include<vector>

#include<string>

#include<algorithm>

using namespace std;

#define lroll StudentList::lastroll //macro to define number of students enrolled

//stores details of each student

class Student{

int roll;

string name;

int score;

public:

void getInfo(int r){

cout<<"\nEnter Name: ";

cin>>name;

cout<<"Enter Score: ";

cin>>score;

roll = r;

cout<<"Alotted Roll: "<<roll<<"\n";

}

void showInfo(){

cout<<"\nRoll: "<<roll<<"\n";

cout<<"Name: "<<name<<"\n";

cout<<"Score: "<<score<<"\n";

}

int returnScore() const{ //accessor function which returns score of student

return score;

}

string returnName() const{ //accessor function which returns name of student

return name;

}

};

//overloaded function object to compare score before finding max from the list

class findmax{

public:

bool operator()(const Student& s1, const Student& s2){

return (s1.returnScore() < s2.returnScore());

}

};

//overloaded function object to check if a substring is present in a string, if present return true

class checkWithSubString{

string s;

public:

checkWithSubString(string temp){

s = temp;

}

bool operator()( const Student& S){

int index;

index = S.returnName().find(s, 0);

if(index != string::npos)

return true;

return false;

}

};

//overloaded function object to check if a substring is present in a string, if not present return true

class checkWithoutSubString{

string s;

public:

checkWithoutSubString(string temp){

s = temp;

}

bool operator()( const Student& S){

int index;

index = S.returnName().find(s, 0);

if(index != string::npos)

return false;

return true;

}

};

//stores an array of student objects

class StudentList{

vector<Student> sl;

static int lastroll;

public:

void addStudent(){

Student s;

lroll++;

s.getInfo(lroll);

sl.push\_back(s);

}

void showTopperDetails(){

vector<Student>::iterator it;

//finds the reference for student with maximum score in the vector of students by using a function object

it = max\_element(sl.begin(), sl.end(), findmax());

(\*it).showInfo();

}

void showStudentByName(string s){

vector<Student>::iterator it;

//finds the reference to the student(s) which has the substring

it = find\_if(sl.begin(), sl.end(), checkWithSubString(s));

if(it != sl.end()){

cout<<"Match Found!\n\n";

(\*it).showInfo();

while(it!=sl.end()){

it = find\_if(it+1, sl.end(), checkWithSubString(s));

if(it != sl.end())

(\*it).showInfo();

}

}

else{

cout<<"Match Not Found!!\n";

return;

}

}

void showStudentWithoutName(string s){

vector<Student>::iterator it;

//finds the reference to the student(s) which does not has the substring

it = find\_if(sl.begin(), sl.end(), checkWithoutSubString(s));

if(it != sl.end()){

cout<<"The Students not having this substring are:\n";

(\*it).showInfo();

while(it!=sl.end()){

it = find\_if(it+1, sl.end(), checkWithoutSubString(s));

if(it != sl.end())

(\*it).showInfo();

}

}

else{

cout<<"Match Not Found!!\n";

return;

}

}

//returns current size of vector

int showsize(){

return sl.size();

}

};

int lroll=0;

int main(){

StudentList SL;

int ch;

string subs;

cout<<"\*\*\*\*STUDENT MANAGEMENT\*\*\*\*\n\n";

//menu of operations

do{

cout<<"\n1. Add Student.\n";

cout<<"2. Show Current Number of Students.\n";

cout<<"3. Show Topper Info.\n";

cout<<"4. Find Student By Name.\n";

cout<<"5. Find Student Without Name.\n";

cout<<"6. Exit.\n";

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

SL.addStudent();

break;

case 2:

cout<<"Current Registered Number of Students: "<<SL.showsize()<<"\n";

break;

case 3:

//checks if the vector is empty, prints appropriate message

if(!SL.showsize()){

cout<<"No Students Added!\n";

break;

}

cout<<"Student with Highest Score:\n";

SL.showTopperDetails();

break;

case 4:

cout<<"\nEnter Sub-String to be Searched: ";

cin>>subs;

SL.showStudentByName(subs);

break;

case 5:

cout<<"\nEnter Sub-String to be Searched: ";

cin>>subs;

SL.showStudentWithoutName(subs);

break;

case 6:

exit(0);

break;

default:

cout<<"Please enter a choice from the given menu!\n";

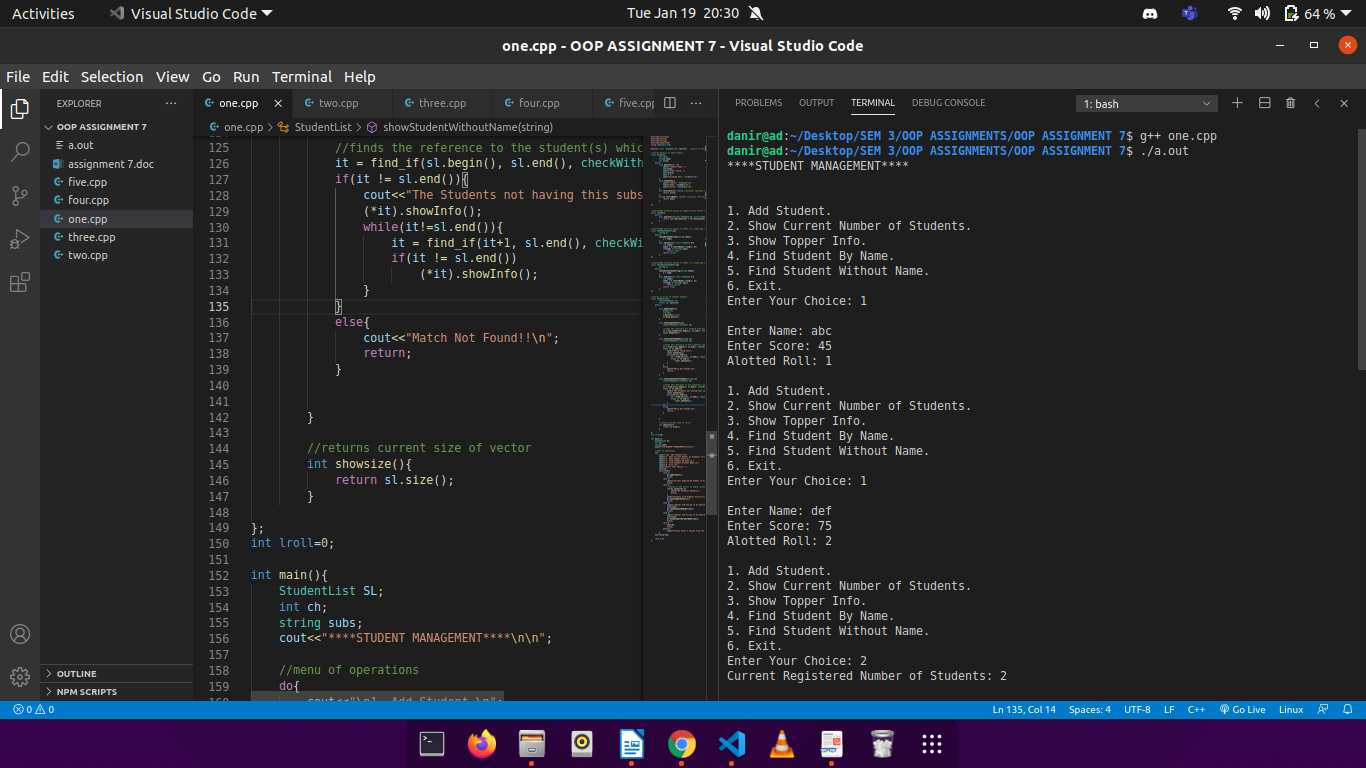
}

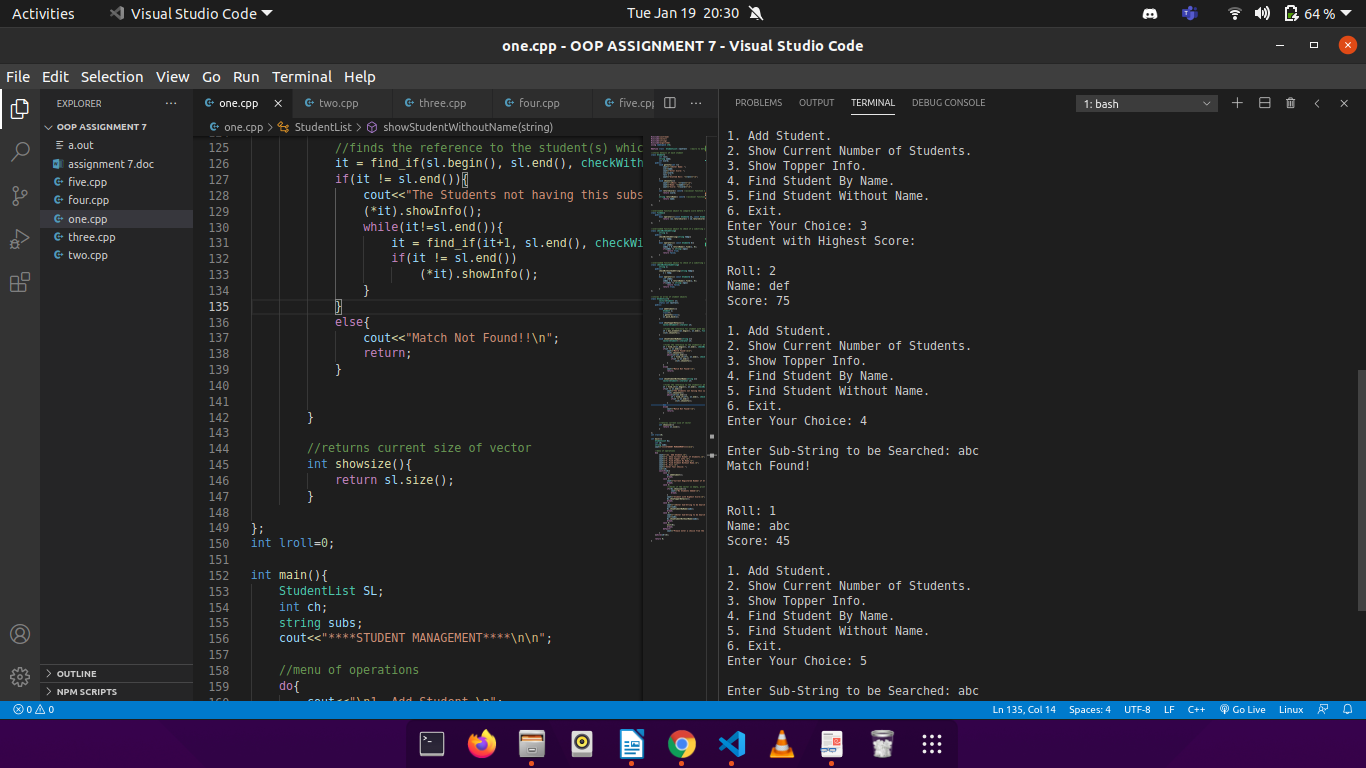
}while(ch!=6);

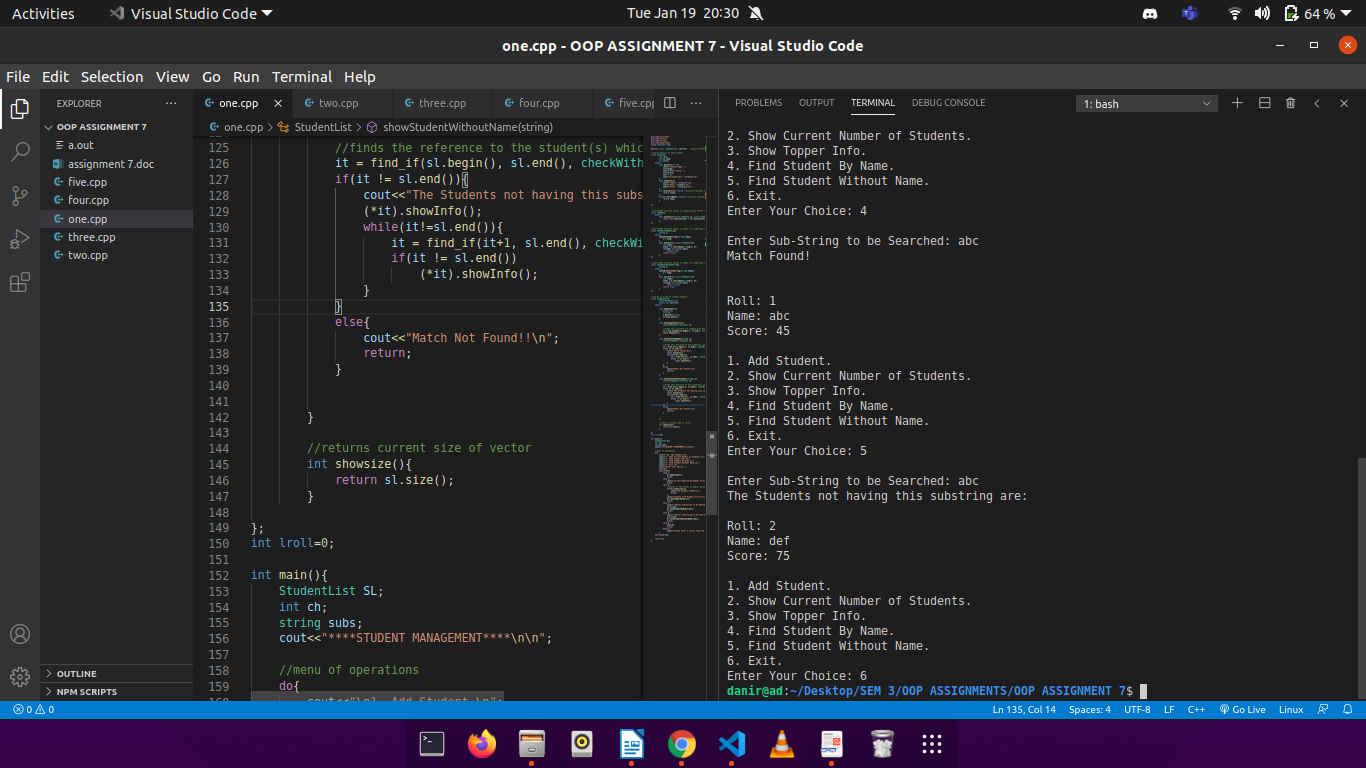
return 0;

}

OUTPUT:







Q2.

CODE:

#include<iostream>

#include<vector>

#include<string>

#include<algorithm>

using namespace std;

#define lroll1 StudentList::lastrollinlist1

#define lroll2 StudentList::lastrollinlist2

//stores details of each student

class Student{

int roll;

string name;

int score;

char section;

public:

void getInfo(int r, int s){

cout<<"\nEnter Name: ";

cin>>name;

cout<<"Enter Score: ";

cin>>score;

roll = r;

cout<<"Alotted Roll: "<<roll<<"\n";

if(s==1)

section='A';

else

section='B';

cout<<"Section: "<<section<<endl;

}

void showInfo(){

cout<<"\nRoll: "<<roll<<"\n";

cout<<"Name: "<<name<<"\n";

cout<<"Secton: "<<section<<"\n";

cout<<"Score: "<<score<<"\n";

}

int returnScore() const{ //accessor function which returns score of student

return score;

}

string returnName() const{ //accessor function which returns name of student

return name;

}

int returnRoll() const{ //accessor function which returns roll of student

return roll;

}

};

//overloaded function object to sort in according to score

class findmax{

public:

bool operator()( const Student& s1, const Student& s2){

return (s1.returnScore() < s2.returnScore());

}

};

//overloaded function object to check if a substring is present in a string, if present return true

class checkWithSubString{

string s;

public:

checkWithSubString(string temp){

s = temp;

}

bool operator()( const Student& S ){

int index;

index = S.returnName().find(s, 0);

if(index != string::npos)

return true;

return false;

}

};

//overloaded function object to check if a substring is present in a string, if not present return true

class checkWithoutSubString{

string s;

public:

checkWithoutSubString(string temp){

s = temp;

}

bool operator()(const Student& S){

int index;

index = S.returnName().find(s, 0);

if(index != string::npos)

return false;

return true;

}

};

//overloaded function object to find student with desired roll

class findStudentWithRoll{

int r;

public:

findStudentWithRoll(int roll){

r = roll;

}

bool operator()(const Student& s){

return (s.returnRoll() == r) ;

}

};

//overloaded function object compare students by roll

class sortByRoll{

public:

bool operator()(const Student& s1, const Student& s2){

return s1.returnRoll() > s2.returnRoll();

}

};

//overloaded function object to compare students by their score

class sortByScore{

public:

bool operator()(const Student& s1, const Student& s2){

return s1.returnScore() > s2.returnScore();

}

};

//stores arrays of student objects

class StudentList{

vector<Student> sl1;

vector<Student> sl2;

static int lastrollinlist1;

static int lastrollinlist2;

public:

void addStudent(int sec){

Student s;

if(sec==1){

lroll1++;

s.getInfo(lroll1, sec);

sl1.push\_back(s);

}

else{

lroll2++;

s.getInfo(lroll2, sec);

sl2.push\_back(s);

}

}

void showTopperDetails(){

vector<Student>::iterator it;

//finds the reference for student with maximum score in the vector of students by using a function object

it = max\_element(sl1.begin(), sl1.end(), findmax());

(\*it).showInfo();

}

void showStudentByName(string s){

vector<Student>::iterator it;

//finds the reference to the student(s) which has the substring

it = find\_if(sl1.begin(), sl1.end(), checkWithSubString(s));

if(it != sl1.end()){

cout<<"Match Found!\n\n";

(\*it).showInfo();

}

else{

cout<<"Match Not Found!!\n";

return;

}

}

void showStudentWithoutName(string s){

vector<Student>::iterator it;

//finds the reference to the student(s) which does not has the substring

it = find\_if(sl1.begin(), sl1.end(), checkWithoutSubString(s));

if(it != sl1.end()){

cout<<"The Students not having this substring are:\n";

(\*it).showInfo();

while(it!=sl1.end()){

it = find\_if(it+1, sl1.end(), checkWithoutSubString(s));

if(it != sl1.end())

(\*it).showInfo();

}

}

else{

cout<<"Match Not Found!!\n";

return;

}

}

//removes student with desired roll

void removeStudent(int r){

vector<Student>::iterator it;

//finds the reference to the student with the given roll

it = find\_if(sl1.begin(), sl1.end(), findStudentWithRoll(r));

if(it == sl1.end()){

cout<<"Student Not Found!\n";

return;

}

else{

//physical deletion of the student object from the vector

sl1.erase(it);

vector<Student>::iterator temp;

cout<<"The Current List is:\n";

for(temp = sl1.begin(); temp != sl1.end(); temp++){

(\*temp).showInfo();

cout<<"\n";

}

}

}

//sorts the students according to their roll or score according to choice

void sortListInDescending(int choice){

if(choice==0){

sort(sl1.begin(), sl1.end(), sortByRoll()); //calls the ovrloaded sortByRoll function object

}

else{

sort(sl1.begin(), sl1.end(), sortByScore()); //calls the ovrloaded sortByScore function object

}

vector<Student>::iterator temp;

cout<<"The Current List is:\n";

for(temp = sl1.begin(); temp != sl1.end(); temp++){

(\*temp).showInfo();

cout<<"\n";

}

}

//combines both the sections and appends it to a single vector

void combineSections(){

vector<Student> combined;

cout<<"The Combined List is:\n";

sort(sl1.begin(), sl1.end(), sortByScore());

sort(sl2.begin(), sl2.end(), sortByScore());

combined.reserve( sl1.size() + sl2.size() ); //adjusts the size of the vector to keep both the sections

combined.insert(combined.end(), sl1.begin(), sl1.end()); //students of section 1 are inserted

combined.insert(combined.end(), sl2.begin(), sl2.end()); //students of section 2 are inserted

vector<Student>::iterator temp;

for(temp = combined.begin(); temp < combined.end(); temp++){

(\*temp).showInfo();

}

cout<<"\n";

}

//returns current size of vector

int showsize(){

return sl1.size();

}

};

int lroll1=0;

int lroll2=0;

int main(){

StudentList SL;

int ch, roll, sortchoice, section;

string subs;

cout<<"\*\*\*\*STUDENT MANAGEMENT\*\*\*\*\n\n";

//menu

do{

cout<<"\n1. Add Student.\n";

cout<<"2. Show Current Number of Students.\n";

cout<<"3. Show Topper Info.\n";

cout<<"4. Find Student By Name.\n";

cout<<"5. Find Student Without Name.\n";

cout<<"6. Remove a Student.\n";

cout<<"7. Sort the List.\n";

cout<<"8. Combine Both Lists.\n";

cout<<"9. Exit.\n";

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

cout<<"In Section 1 or In Section 2: ";

cin>>section;

SL.addStudent(section);

break;

case 2:

cout<<"Current Registered Number of Students: "<<SL.showsize()<<"\n";

break;

case 3:

//check for empty vector

if(!SL.showsize()){

cout<<"No Students Added!\n";

break;

}

cout<<"Student with Highest Score:\n";

SL.showTopperDetails();

break;

case 4:

cout<<"\nEnter Sub-String to be Searched: ";

cin>>subs;

SL.showStudentByName(subs);

break;

case 5:

cout<<"\nEnter Sub-String to be Searched: ";

cin>>subs;

SL.showStudentWithoutName(subs);

break;

case 6:

//check for empty vector

if(!SL.showsize()){

cout<<"No Students Added!\n";

break;

}

cout<<"Enter Roll to be Removed: ";

cin>>roll;

SL.removeStudent(roll);

break;

case 7:

cout<<"Sort By Score(1) or Roll(0): ";

cin>>sortchoice;

SL.sortListInDescending(sortchoice);

break;

case 8:

SL.combineSections();

break;

case 9:

exit(0);

break;

default:

cout<<"Please enter a choice from the given menu!\n";

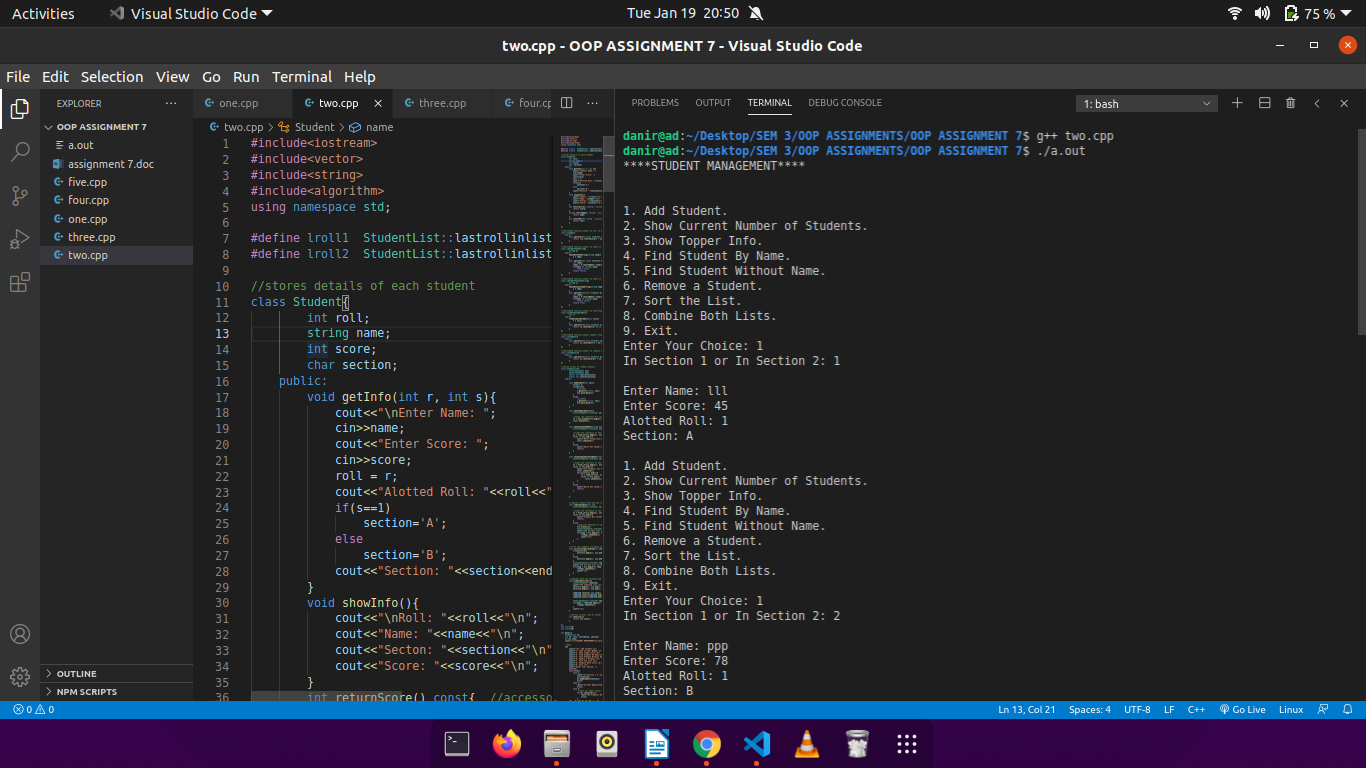
}

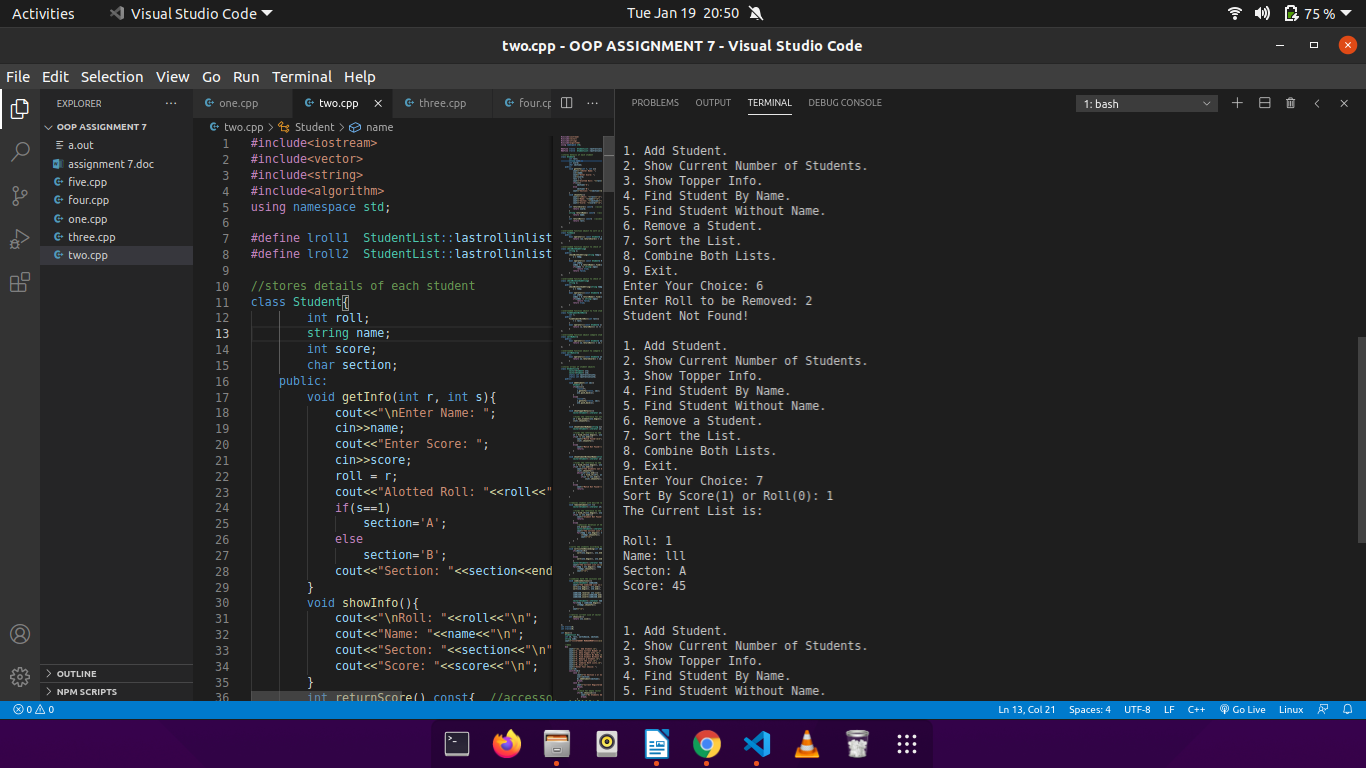
}while(ch!=9);

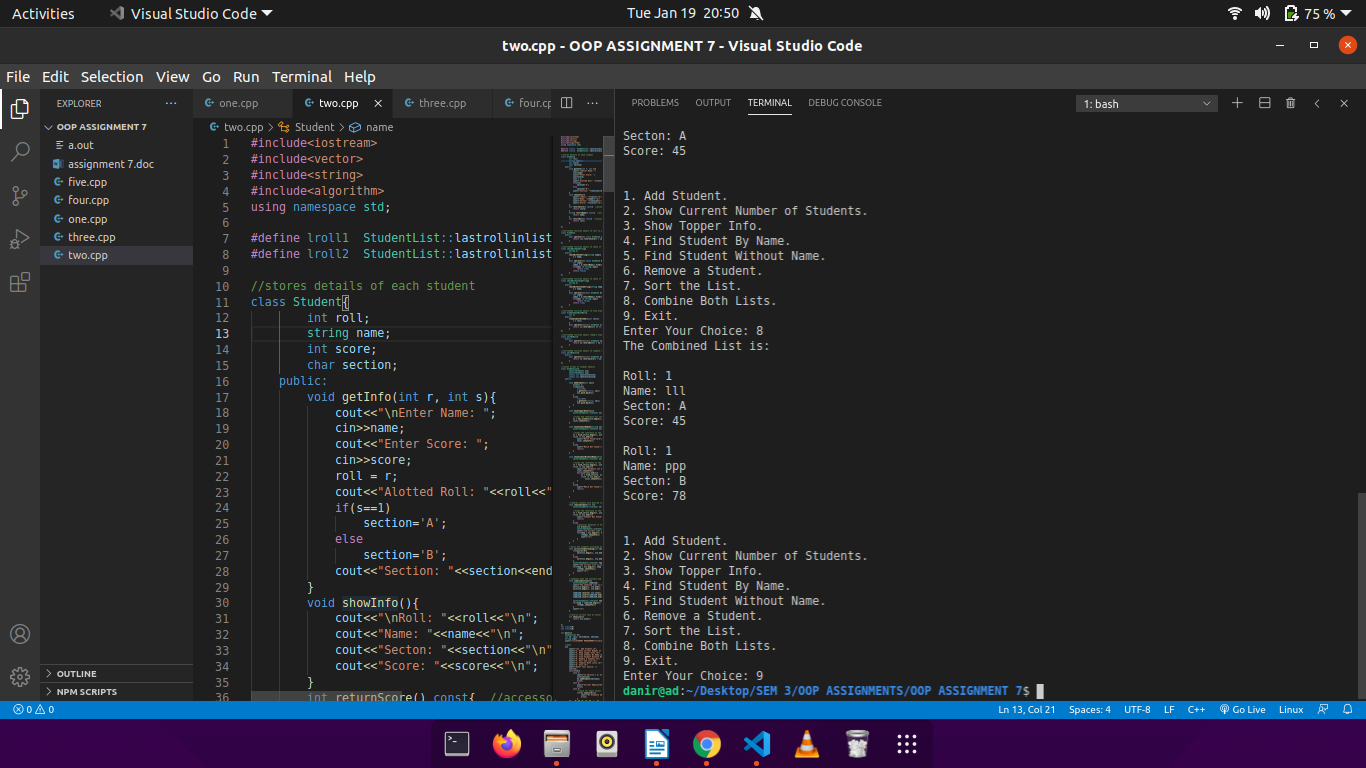
return 0;

}

OUTPUT:







Q3.

CODE:

#include<iostream>

#include<queue>

using namespace std;

#define lroll StudentQueue::lastroll

//stores student details

class Student{

int roll;

string name;

int score;

public:

void getInfo(int r){

cout<<"Enter Name: ";

cin>>name;

cout<<"Enter Score: ";

cin>>score;

roll = r;

cout<<"Alotted Roll: "<<roll<<"\n";

}

void displayInfo(){

cout<<"\t "<<roll;

cout<<"\t "<<name;

cout<<"\t "<<score<<"\n";

}

int returnScore() const{ //accessor function to return score of a student

return score;

}

};

//stores a queue of student objects

class StudentQueue{

queue<Student> Q\_stu;

static int lastroll;

public:

void addStudent(){

Student s;

lroll++;

s.getInfo(lroll);

Q\_stu.push(s);

}

void displayMarksheet(){

bool proceed = 1;

Student s;

cout<<"Marksheet of Student in order of earliest enrollment:\n";

while(proceed){

s = Q\_stu.front(); //gets the element at the top of the queue

cout<<"\n\tRoll\tName\tScore";

cout<<"\n.....................................\n";

s.displayInfo();

Q\_stu.pop(); //pops (or removes) the element at the top of the queue

cout<<"\nWant to see Marksheet of next enrolled Student? Press 1 to continue 0 for Top Menu.\n";

cin>>proceed;

//check for emoty queue

if(!Q\_stu.size()){

cout<<"No More Students Enrolled!\n";

return;

}

}

}

//returns current size of the queue

int returnSize(){

return Q\_stu.size();

}

};

int lroll=0;

int main(){

StudentQueue SQ;

int ch;

//menu

do{

cout<<"\n1. Add Student.\n";

cout<<"2. Display Marksheet of a Student.\n";

cout<<"3. Exit.\n";

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

SQ.addStudent();

break;

case 2:

//check for empty queue

if(!SQ.returnSize()){

cout<<"No Student Added!\n";

break;

}

SQ.displayMarksheet();

break;

case 3:

exit(0);

break;

default:

cout<<"Wrong Choice!\n";

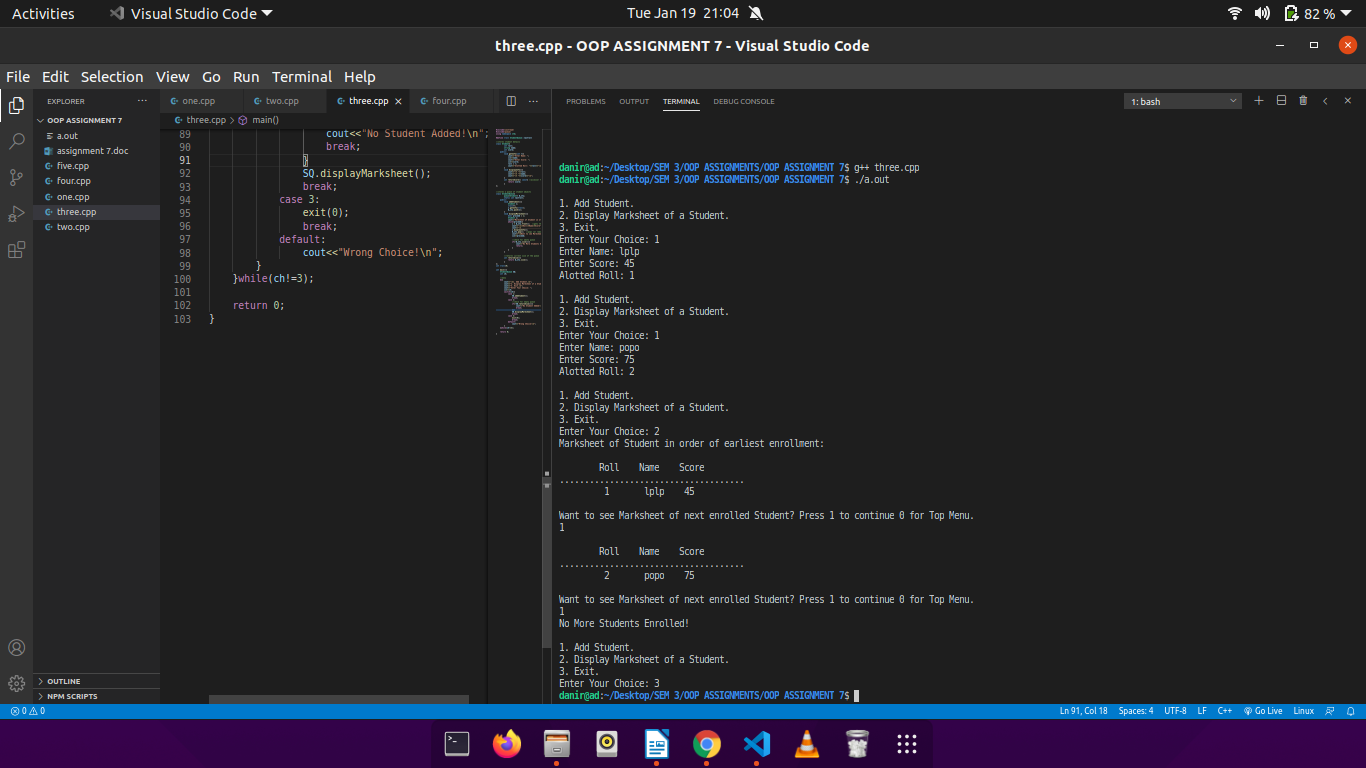
}

}while(ch!=3);

return 0;

}

OUTPUT:



Q4.

CODE:

#include<iostream>

#include<queue>

using namespace std;

#define lroll StudentPriorityQueue::lastroll

//stores student details

class Student{

int roll;

string name;

int score;

public:

void getInfo(int r){

cout<<"Enter Name: ";

cin>>name;

cout<<"Enter Score: ";

cin>>score;

roll = r;

cout<<"Alotted Roll: "<<roll<<"\n";

}

void displayInfo(){

cout<<"\nRoll: "<<roll<<"\n";

cout<<"Name: "<<name<<"\n";

cout<<"Score: "<<score<<"\n";

}

int returnScore() const{ //accessor function to return score of the student

return score;

}

};

//overloading the < operator to make score of students the priority

bool operator<(const Student& s1, const Student& s2){

return s1.returnScore() < s2.returnScore() ;

}

//stores a queue of student objects according to a priority

class StudentPriorityQueue{

priority\_queue<Student> PQ\_stu;

static int lastroll;

public:

void addStudent(){

Student s;

lroll++;

s.getInfo(lroll);

PQ\_stu.push(s);

}

void displayQueue(){

Student s;

cout<<"\nThe list is in descending order of score:\n";

while(!PQ\_stu.empty()){

s = PQ\_stu.top(); //gets the element with highest priority

s.displayInfo();

PQ\_stu.pop(); //removes the element with highest priority

}

cout<<"\nQueue Empty, Add More Students to continue!\n";

}

//returns current size of the queue

int returnSize(){

return PQ\_stu.size();

}

};

int lroll=0;

int main(){

StudentPriorityQueue SPQ;

int ch;

//menu

do{

cout<<"\n1. Add Student.\n";

cout<<"2. Display All Students according to Priority (highest score first).\n";

cout<<"3. Exit.\n";

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

SPQ.addStudent();

break;

case 2:

//check for empty queue

if(!SPQ.returnSize()){

cout<<"No Students Added!\n";

break;

}

SPQ.displayQueue();

break;

case 3:

exit(0);

break;

default:

cout<<"Wrong Choice!\n";

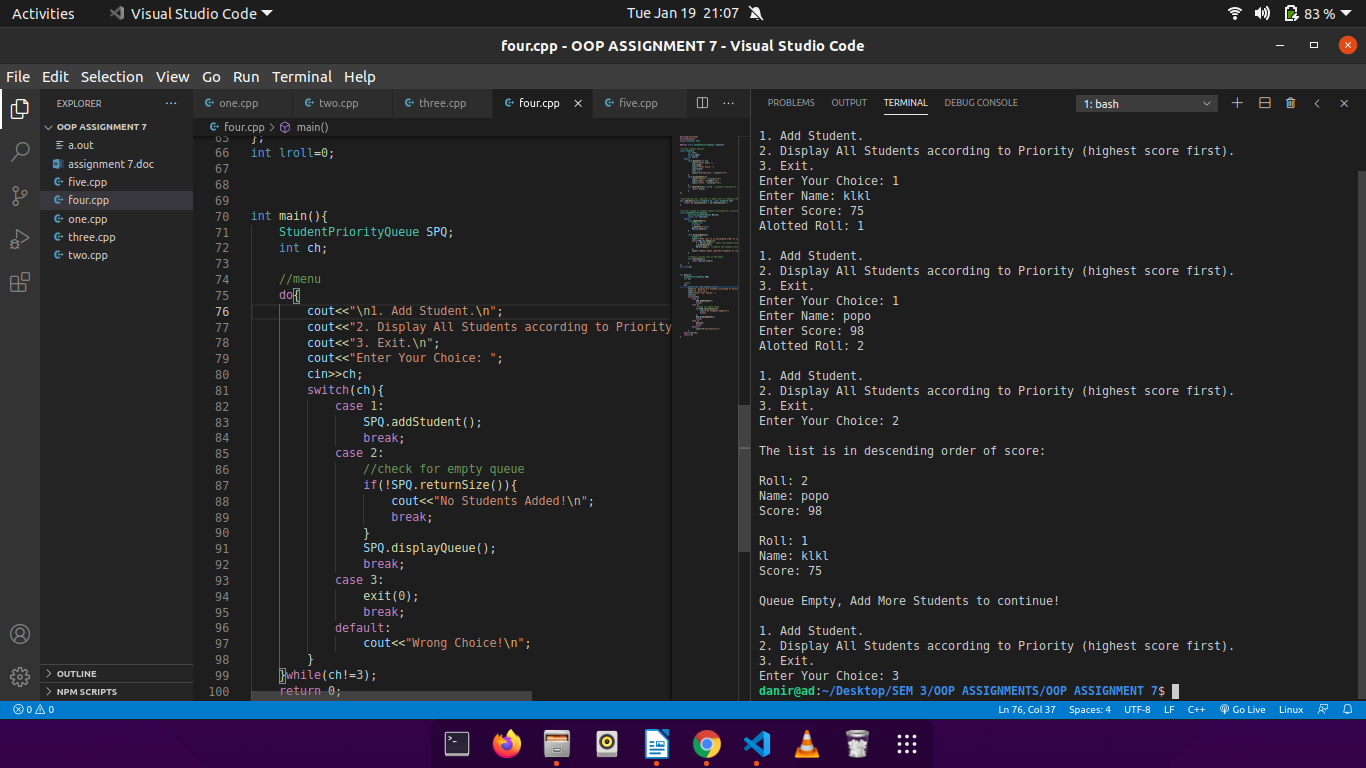
}

}while(ch!=3);

return 0;

}

OUTPUT:



Q5.

CODE:

#include<iostream>

#include<map>

#include<utility>

#include<vector>

#include<algorithm>

using namespace std;

vector<int> roll\_array;

//stores student details

class Student{

int roll;

string name;

int score;

public:

int getInfo(){

cout<<"Enter Roll: ";

cin>>roll;

vector<int>::iterator it;

it = find(roll\_array.begin(), roll\_array.end(), roll);

//check for duplicate roll

if(it == roll\_array.end())

roll\_array.push\_back(roll);

else{

return -1;

}

cout<<"Enter Name: ";

cin>>name;

cout<<"Enter Score: ";

cin>>score;

return roll;

}

void displayInfo(){

cout<<"Name: "<<name<<"\n";

cout<<"\tScore: "<<score<<"\n";

}

int returnRoll() const{ //accessor function to return roll of student

return roll;

}

int returnScore() const{ //accessor function to return score of student

return score;

}

};

//overloading the < operator to make rolls of student the keys of map and store in ascnding order

bool operator<(const Student& s1, const Student& s2){

return s1.returnRoll() < s2.returnRoll() ;

}

class StudentMap{

map<int, Student> M\_stu;

public:

void addStudent(){

Student s;

int r = s.getInfo();

//check for duplicate roll

if(r < 0){

cout<<"Roll Already Exists! Previous Data will be Lost!\n";

return;

}

M\_stu.insert(pair<int, Student>(r, s));

}

void displayAllStudents(){

map<int, Student>::iterator it;

cout<<"\nThe Student Map(with roll as key) looks like this:\n";

cout<<"\nRoll\tDetails\n\n";

for(it = M\_stu.begin(); it != M\_stu.end(); it++){

cout<<it->first<<"\t";

(it->second).displayInfo();

cout<<"\n";

}

cout<<"\n";

}

//returns current size of the map

int returnSize(){

return M\_stu.size();

}

};

int main(){

StudentMap SM;

int ch;

roll\_array.clear();

//menu

do{

cout<<"\n1. Add Student.\n";

cout<<"2. Display All Students in Ascending Order of Roll.\n";

cout<<"3. Exit.\n";

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

SM.addStudent();

break;

case 2:

//check for empty map

if(!SM.returnSize()){

cout<<"No Student Added!\n";

break;

}

SM.displayAllStudents();

break;

case 3:

exit(0);

break;

default:

cout<<"Wrong Choice!\n";

}

}while(ch!=3);

return 0;

}

OUTPUT:

