**Assignment 6**

Q1.

CODE:

#include<iostream>

#include<string.h>

#include<fstream>

using namespace std;

//book details

class book{

int serial;

int bookid;

char title[21];

char author[31];

char publisher[31];

int price;

public:

void getInfo(int id){

cout<<"Enter Book Details:\n";

cout<<"Title: ";

cin>>title;

cout<<"Author: ";

cin>>author;

cout<<"Publisher: ";

cin>>publisher;

cout<<"Price: ";

cin>>price;

bookid=id;

serial=1;

cout<<"Book-ID: "<<bookid<<endl;

}

void showInfo(){

cout<<"Title: "<<title<<endl;;

cout<<"Author: "<<author<<endl;

cout<<"Publisher: "<<publisher<<endl;

cout<<"Price: "<<price<<endl;

cout<<"Book-ID: "<<bookid<<endl;

cout<<"Serial: "<<serial<<endl;

}

int returnid(){ //returns book id

return bookid;

}

int returnserial(){ //returns current number of copies

return serial;

}

void updateserial(int news){

serial = serial + news;

}

friend class transactionfile;

};

//member details

class member{

protected:

int memberid;

char name[31];

char email[21];

char address[51];

int numberofbooks;

int borrowed[11];

public:

void getInfo(int id){

cout<<"Enter Member Details:\n";

cout<<"Name: ";

cin>>name;

cout<<"Email: ";

cin>>email;

cout<<"Address: ";

cin>>address;

memberid=id;

numberofbooks=0;

cout<<"Member Id: "<<memberid<<endl;

}

int returnmemid(){ //returns member id

return memberid;

}

int \* returnborrowedlist(){

return borrowed;

}

int returnbooksnumber(){

return numberofbooks;

}

friend class transactionfile;

};

//inherited from member

class student:public member{

public:

int checkIssue(){ //checks if anymore issue is possible

if(numberofbooks>2){

cout<<"NO MORE BOOKS CAN BE ISSUED!\n";

return false;

}

return true;

}

};

//inherited from member

class faculty:public member{

public:

int checkIssue(){ //checks if anymore issue is possible

if(numberofbooks>10){

cout<<"NO MORE BOOKS CAN BE ISSUED!\n";

return false;

}

return true;

}

};

//stores transaction details

class transaction{

int transactionid;

char date[11];

int memberid;

int bookid;

public:

int getInfo(int id){ //gets transaction details

cout<<"Enter Transaction Details:\n";

cout<<"Member-Id: ";

cin>>memberid;

cout<<"Book-Id: ";

cin>>bookid;

transactionid=id;

int flag=0;

//to check if book exists or not, displays error if doesn't

fstream f("Books", ios::in | ios::binary);

book b;

f.seekg((bookid-1)\*sizeof(book), ios::beg);

f.read((char\*)&b, sizeof(b));

if(b.returnid()==bookid)

flag=1;

f.seekg(0, ios::beg);

f.close();

if(flag==0){

cout<<"Book Not Available!\n";

return -1;

}

cout<<"Date: ";

cin>>date;

cout<<"Transaction ID: "<<transactionid<<endl;

return id;

}

int returnbookid(){

return bookid;

}

int returnmemid(){

return memberid;

}

int returntransactionid(){ //returns transaction id of the user

return transactionid;

}

};

//stores list of book objects in a file

class bookfile{

string fn;

static int nob;

public:

bookfile(string f){ //to update the id everytime the code is ran

fn = f;

ifstream fbook(fn, ios::in | ios::binary);

if(fbook.tellg()==0){

book b;

fbook.seekg((-1)\*sizeof(b), ios::end);

fbook.read((char\*)&b, sizeof(b));

bookfile::init(b.returnid());

}

else

bookfile::init(0);

fbook.seekg(0, ios::beg);

fbook.close();

}

void addBook(){ //adds a book record

bookfile::nob++;

book b;

b.getInfo(nob);

fstream f(fn, ios::app | ios::binary);

f.write((char\*)&b, sizeof(b));

f.close();

cout<<"Book Added!\n";

}

void addexisting(int id){ //updates stock of existing book

if(id>nob){

cout<<"Invalid ID!\n";

return;

}

int qnty;

int count=0;

cout<<"Enter Quantity to Stock Up: ";

cin>>qnty;

book b;

fstream f(fn, ios::in | ios::out | ios::binary);

f.seekg((id-1)\*sizeof(b), ios::beg);

f.read((char\*)&b, sizeof(b));

if(b.returnid()==id)

b.updateserial(qnty);

if(f.eof())

f.clear();

f.seekp((id-1)\*sizeof(b), ios::beg);

f.write((char\*)&b, sizeof(b));

f.seekg(0, ios::beg);

f.close();

cout<<"Stocked Up!\n";

}

//checks if book is in stock, displays error if it doesnt

void checkAvailable(int id){

book b;

fstream f(fn, ios::in | ios::binary);

f.seekg((id-1)\*sizeof(b), ios::beg);

f.read((char\*)&b, sizeof(b));

if(b.returnid()==id){

if(b.returnserial()<1){

cout<<"Book Not Available! Kindly Re-stock.\n";

}

else{

cout<<"Book Available! Copies Remmaining: "<<b.returnserial()<<endl;

}

}

if(f.eof())

f.clear();

if(id>nob || id<0){

cout<<"Invalid ID!\n";

return;

}

f.seekg(0, ios::beg);

f.close();

}

void showBookInfo(int id){ //display book info

book b;

fstream f(fn, ios::in | ios::binary);

f.seekg((id-1)\*sizeof(b), ios::beg);

f.read((char\*)&b, sizeof(b));

if(f.eof())

f.clear();

b.showInfo();

f.seekg(0, ios::beg);

f.close();

}

static void init(int a){ //to initialize static variable

bookfile::nob = a;

}

};

int bookfile::nob=0;

//stores list of student objects in a file

class studentfile{

string fn;

static int nos;

public:

studentfile(string f){ //to update the id everytime the code is ran

fn = f;

ifstream fstudent(fn, ios::in | ios::binary);

if(fstudent.tellg()==0){

student s;

fstudent.seekg((-1)\*sizeof(s), ios::end);

fstudent.read((char\*)&s, sizeof(s));

studentfile::init(s.returnmemid());

}

else

studentfile::init(0);

fstudent.seekg(0, ios::beg);

fstudent.close();

}

void addStudent(){ //adds student record

studentfile::nos++;

student s;

s.getInfo(nos);

fstream f(fn, ios::app | ios::binary);

f.write((char\*)&s, sizeof(s));

f.close();

cout<<"Student Added!\n";

}

void showborrowed(int id){ //displays borrow history of the student

fstream f(fn, ios::in | ios::binary);

student s;

int \*list;

f.seekg((id-1)\*sizeof(s), ios::beg);

f.read((char\*)&s, sizeof(s));

if(s.returnmemid()==id){

list = s.returnborrowedlist();

for(int i=0;i<s.returnbooksnumber();i++){

cout<<"Book ID: "<<list[i]<<"\n";

}

}

if(f.eof())

f.clear();

f.seekg(0, ios::beg);

f.close();

}

static void init(int a){ //initialize static variable

studentfile::nos = a;

}

};

int studentfile::nos=0;

//stores list of faculty objects in a file

class facultyfile{

string fn;

static int nof;

public:

facultyfile(string f){ //to update the id everytime the code is ran

fn = f;

ifstream ffaculty(fn, ios::in | ios::binary);

if(ffaculty.tellg()==0){

faculty ff;

ffaculty.seekg((-1)\*sizeof(ff), ios::end);

ffaculty.read((char\*)&ff, sizeof(ff));

facultyfile::init(ff.returnmemid());

}

else

facultyfile::init(0);

ffaculty.seekg(0, ios::beg);

ffaculty.close();

}

void addFaculty(){ //add faculty record

facultyfile::nof++;

faculty facl;

facl.getInfo(nof);

fstream f(fn, ios::app | ios::binary);

f.write((char\*)&facl, sizeof(facl));

f.close();

cout<<"Faculty Added!\n";

}

void showborrowed(int id){ //display borrow history

fstream f(fn, ios::in | ios::binary);

faculty fa;

int \*list;

f.seekg((id-1)\*sizeof(fa), ios::beg);

f.read((char\*)&fa, sizeof(fa));

if(fa.returnmemid()==id){

list = fa.returnborrowedlist();

for(int i=0;i<fa.returnbooksnumber();i++){

cout<<"Book ID: "<<list[i]<<"\n";

}

}

if(f.eof())

f.eof();

f.seekg(0, ios::beg);

f.close();

}

static void init(int a){ //initialize static variable

facultyfile::nof = a;

}

};

int facultyfile::nof=0;

//stores list of transaction objects in a file

class transactionfile{

string fn;

static int notra;

public:

transactionfile(string f){ //to update the id everytime the code is ran

fn = f;

ifstream ftransact(fn, ios::in | ios::binary);

if(ftransact.tellg()==0){

transaction t;

ftransact.seekg((-1)\*sizeof(t), ios::end);

ftransact.read((char\*)&t, sizeof(t));

transactionfile::init(t.returntransactionid());

}

else

transactionfile::init(0);

ftransact.seekg(0, ios::beg);

ftransact.close();

}

int addTansacction(){ //add transaction record

transactionfile::notra++;

transaction t;

int flag = t.getInfo(notra);

if(flag<0){

return 0;

}

fstream f(fn, ios::app | ios::binary);

f.write((char\*)&t, sizeof(t));

f.close();

return t.returntransactionid();

}

void studentissued(int transid){ //issue book to a student

int bId;

int mId;

fstream ftransact(fn, ios::in | ios::binary);

transaction t;

ftransact.seekg((transid-1)\*sizeof(ftransact));

ftransact.read((char\*)&t, sizeof(t));

if(t.returntransactionid()==transid){

bId=t.returnbookid();

mId=t.returnmemid();

}

if(ftransact.eof())

ftransact.clear();

ftransact.seekg(0, ios::beg);

ftransact.close();

fstream fstudent("Students", ios::in | ios::out | ios::binary);

student s;

fstudent.seekg((mId-1)\*sizeof(student), ios::beg);

fstudent.read((char\*)&s, sizeof(s));

if(fstudent.eof())

fstudent.clear();

fstudent.seekg(0, ios::beg);

fstream fbook("Books", ios::in | ios::out | ios::binary);

book b;

fbook.seekg((bId-1)\*sizeof(book), ios::beg);

fbook.read((char\*)&b, sizeof(b));

if(b.returnid()==bId){

cout<<s.numberofbooks<<"\n";

if(s.checkIssue()){

s.borrowed[s.numberofbooks]=bId;

s.numberofbooks++;

b.serial--;

}

else{

cout<<"MAX ISSUE LIMIT REACHED!\n";

return;

}

}

if(fbook.eof())

fbook.clear();

fstudent.seekp((mId-1)\*sizeof(student), ios::beg);

fstudent.write((char\*)&s, sizeof(s));

fbook.seekp((bId-1)\*sizeof(b), ios::beg);

fbook.write((char\*)&b, sizeof(b));

fbook.seekg(0, ios::beg);

fbook.close();

fstudent.close();

cout<<"Book Issued to Student!\n";

}

void studentreturned(int transid){ //return book from a student

int bId;

int mId;

fstream ftransact(fn, ios::in | ios::binary);

transaction t;

ftransact.seekg((transid-1)\*sizeof(ftransact));

ftransact.read((char\*)&t, sizeof(t));

if(t.returntransactionid()==transid){

bId=t.returnbookid();

mId=t.returnmemid();

}

if(ftransact.eof())

ftransact.clear();

ftransact.seekg(0, ios::beg);

ftransact.close();

fstream fstudent("Students", ios::in | ios::out | ios::binary);

student s;

fstudent.seekg((mId-1)\*sizeof(student), ios::beg);

fstudent.read((char\*)&s, sizeof(s));

if(fstudent.eof())

fstudent.clear();

fstudent.seekg(0, ios::beg);

fstream fbook("Books", ios::in | ios::out | ios::binary);

book b;

fbook.seekg((bId-1)\*sizeof(book), ios::beg);

fbook.read((char\*)&b, sizeof(b));

if(b.returnid()==bId){

if(s.returnbooksnumber()<1){

cout<<"No Books To Return!\n";

return;

}

s.numberofbooks--;

b.serial++;

}

if(fbook.eof())

fbook.clear();

fstudent.seekp((mId-1)\*sizeof(student), ios::beg);

fstudent.write((char\*)&s, sizeof(s));

fbook.seekp((bId-1)\*sizeof(b), ios::beg);

fbook.write((char\*)&b, sizeof(b));

fbook.seekg(0, ios::beg);

fbook.close();

fstudent.close();

cout<<"Book Returned Successfully\n";

}

void facultyissued(int transid){ //issue book to a faculty

int bId;

int mId;

fstream ftransact(fn, ios::in | ios::binary);

transaction t;

ftransact.seekg((transid-1)\*sizeof(ftransact));

ftransact.read((char\*)&t, sizeof(t));

if(t.returntransactionid()==transid){

bId=t.returnbookid();

mId=t.returnmemid();

}

if(ftransact.eof())

ftransact.clear();

ftransact.seekg(0, ios::beg);

ftransact.close();

fstream ffaculty("Faculties", ios::in | ios::out | ios::binary);

faculty f;

ffaculty.seekg((mId-1)\*sizeof(student), ios::beg);

ffaculty.read((char\*)&f, sizeof(f));

if(ffaculty.eof())

ffaculty.clear();

ffaculty.seekg(0, ios::beg);

fstream fbook("Books", ios::in | ios::out | ios::binary);

book b;

fbook.seekg((bId-1)\*sizeof(book), ios::beg);

fbook.read((char\*)&b, sizeof(b));

if(b.returnid()==bId){

cout<<f.numberofbooks<<"\n";

if(f.checkIssue()){

f.borrowed[f.numberofbooks]=bId;

f.numberofbooks++;

b.serial--;

}

else{

cout<<"MAX ISSUE LIMIT REACHED!\n";

return;

}

}

if(fbook.eof())

fbook.clear();

ffaculty.seekp((mId-1)\*sizeof(student), ios::beg);

ffaculty.write((char\*)&f, sizeof(f));

fbook.seekp((bId-1)\*sizeof(b), ios::beg);

fbook.write((char\*)&b, sizeof(b));

fbook.seekg(0, ios::beg);

fbook.close();

ffaculty.close();

cout<<"Book Issued to Faculty!\n";

}

void facultyreturned(int transid){ //return book from a faculty

int bId;

int mId;

fstream ftransact(fn, ios::in | ios::binary);

transaction t;

ftransact.seekg((transid-1)\*sizeof(ftransact));

ftransact.read((char\*)&t, sizeof(t));

if(t.returntransactionid()==transid){

bId=t.returnbookid();

mId=t.returnmemid();

}

if(ftransact.eof())

ftransact.clear();

ftransact.seekg(0, ios::beg);

ftransact.close();

fstream ffaculty("Faculties", ios::in | ios::out | ios::binary);

faculty f;

ffaculty.seekg((mId-1)\*sizeof(f), ios::beg);

ffaculty.read((char\*)&f, sizeof(f));

if(ffaculty.eof())

ffaculty.clear();

ffaculty.seekg(0, ios::beg);

fstream fbook("Books", ios::in | ios::out | ios::binary);

book b;

fbook.seekg((bId-1)\*sizeof(book), ios::beg);

fbook.read((char\*)&b, sizeof(b));

if(b.returnid()==bId){

cout<<f.numberofbooks<<"\n";

if(f.returnbooksnumber()<1){

cout<<"No Books To Return!\n";

return;

}

f.numberofbooks--;

b.serial++;

}

if(fbook.eof())

fbook.clear();

ffaculty.seekp((mId-1)\*sizeof(student), ios::beg);

ffaculty.write((char\*)&f, sizeof(f));

fbook.seekp((bId-1)\*sizeof(b), ios::beg);

fbook.write((char\*)&b, sizeof(b));

fbook.seekg(0, ios::beg);

fbook.close();

ffaculty.close();

cout<<"Book Returned Successfully!\n";

}

static void init(int a){ //initialize static variable

transactionfile::notra = a;

}

};

int transactionfile::notra=0;

int main(){

string bfile="Books";

string sfile="Students";

string ffile="Faculties";

string tfile="Transactions";

//objects for respecive file classes

bookfile bf(bfile);

studentfile sf(sfile);

facultyfile ff(ffile);

transactionfile tf(tfile);

int ch, id, transid, choice, memberflag=0, bookflag=0;

//menu

do{

cout<<"\n\n\*\*\*LIBRARY MANAGEMENT SYSTEM\*\*\*\n\n";

cout<<"1. Add New Book.\n";

cout<<"2. Stock up Existing Book.\n";

cout<<"3. Add Student.\n";

cout<<"4. Add Faculty.\n";

cout<<"5. Issue Book.\n";

cout<<"6. Return Book.\n";

cout<<"7. Check Book Availability.\n";

cout<<"8. Show 'Borrow History' of Member.\n";

cout<<"9. Show Book Info.\n";

cout<<"10. Exit.\n";

cout<<"\nEnter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

bf.addBook();

bookflag++;

break;

case 2:

cout<<"Enter Book-Id: ";

cin>>id;

bf.addexisting(id);

break;

case 3:

memberflag=1;

sf.addStudent();

break;

case 4:

memberflag=-1;

ff.addFaculty();

break;

case 5:

if(bookflag==0){

cout<<"NO BOOKS ADDED!!\n";

break;

}

transid = tf.addTansacction();

if(transid==0){

break;

}

cout<<"\n1. Issue For Student.\n";

cout<<"2. Issue For Faculty.\n";

cout<<"Enter Your Choice: ";

cin>>choice;

//menu for issue

switch(choice){

case 1:

tf.studentissued(transid);

break;

case 2:

tf.facultyissued(transid);

}

break;

case 6:

if(bookflag==0){

cout<<"NO BOOKS ADDED!!\n";

break;

}

transid = tf.addTansacction();

cout<<"\n1. Return For Student.\n";

cout<<"2. Return For Faculty.\n";

cout<<"Enter Your Choice: ";

cin>>choice;

//menu for return

switch(choice){

case 1:

tf.studentreturned(transid);

break;

case 2:

tf.facultyreturned(transid);

break;

}

break;

case 7:

cout<<"\nEnter Book-Id to be checked: ";

cin>>id;

bf.checkAvailable(id);

break;

case 8:

if(memberflag==0){

cout<<"NO MEMBERS ADDED!!\n";

break;

}

cout<<"Student(1) or Faculty(0): ";

int k;

cin>>k;

cout<<"Enter Member-Id: ";

int memid;

cin>>memid;

//checking if ny student / faculty has been entered brore, if not displaying error

if(k==1 && memberflag==1){

sf.showborrowed(memid);

}

else if(k==1 && memberflag==-1){

cout<<"NO STUDENT ADDED!\n";

}

else if(k==0 && memberflag==-1){

ff.showborrowed(memid);

}

else{

cout<<"NO FACULTY ADDED!\n";

}

break;

case 9:

cout<<"\nEnter Book-Id to be checked: ";

cin>>id;

bf.showBookInfo(id);

break;

case 10:

exit(0);

break;

default:

cout<<"INVALID CHOICE!\n";

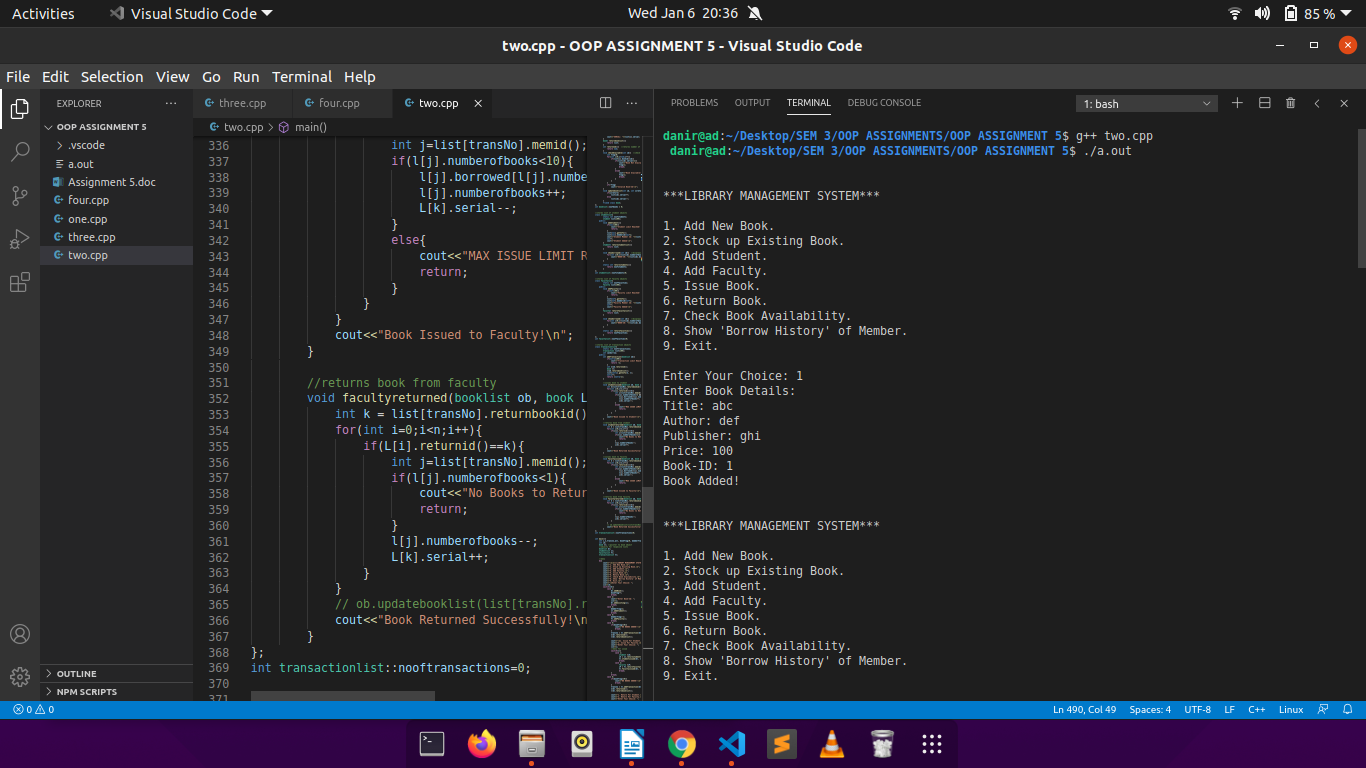
}

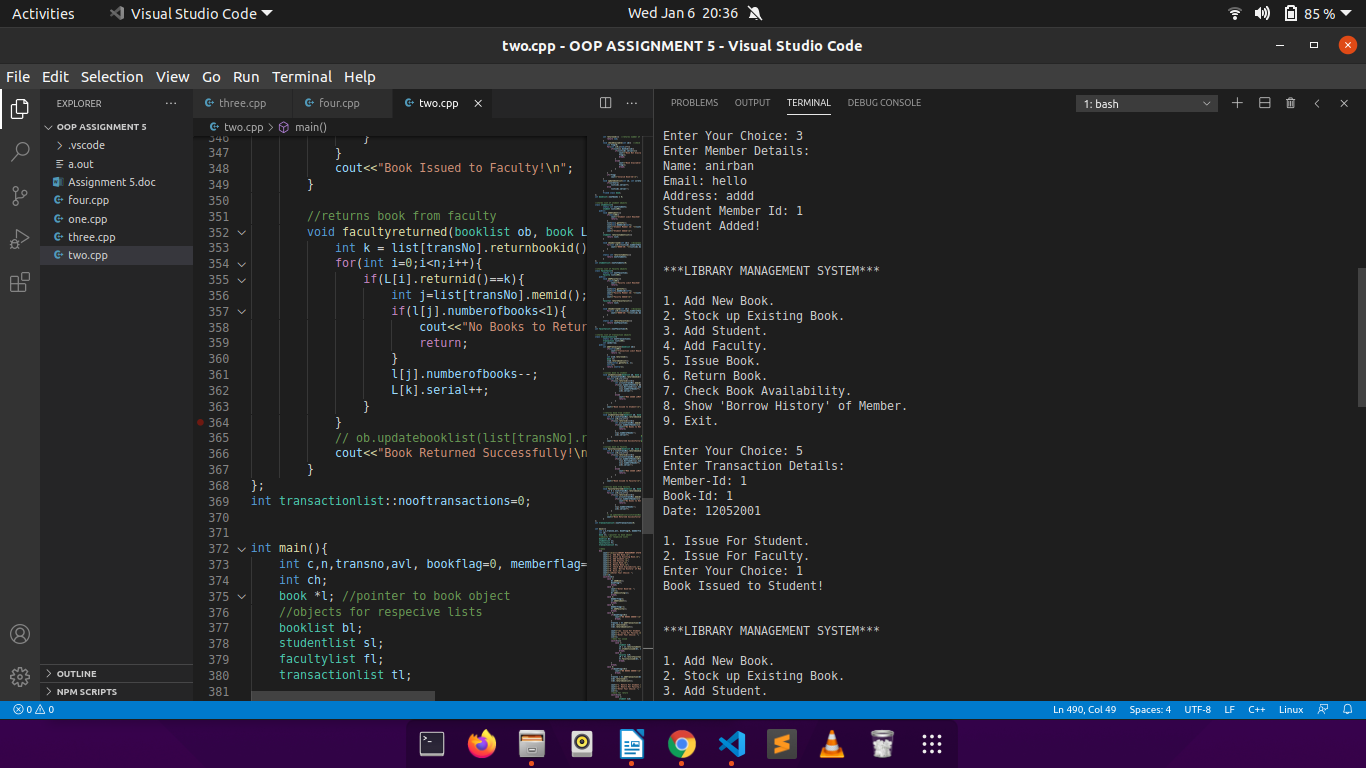
}while(ch!=10);

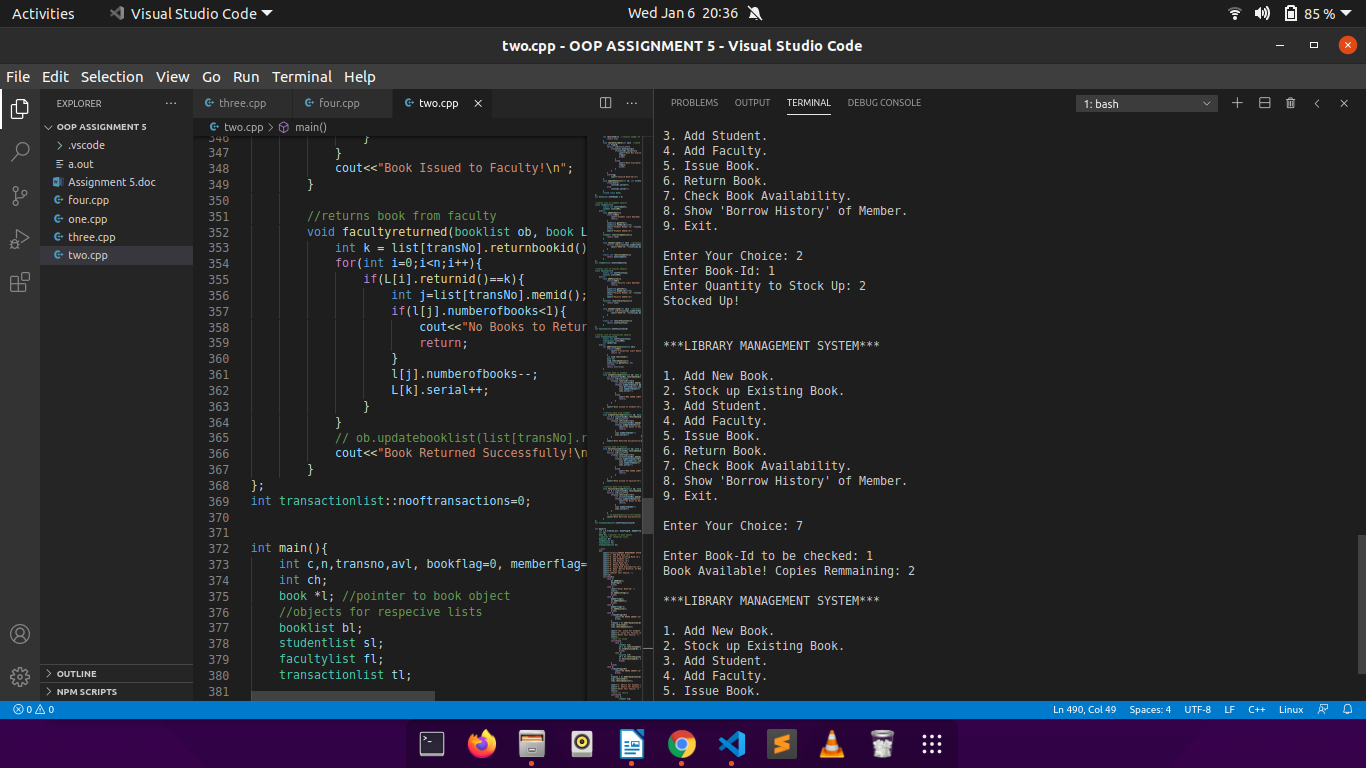
return 0;

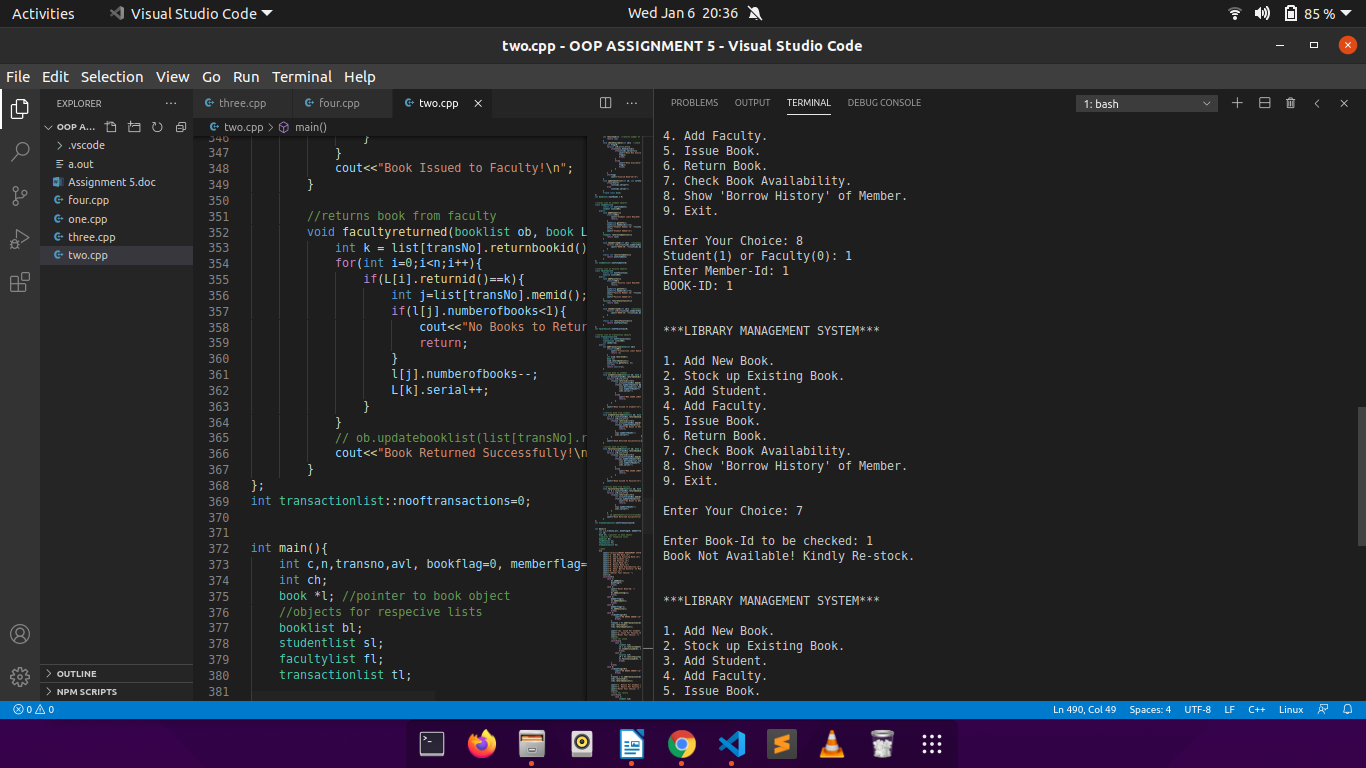
}

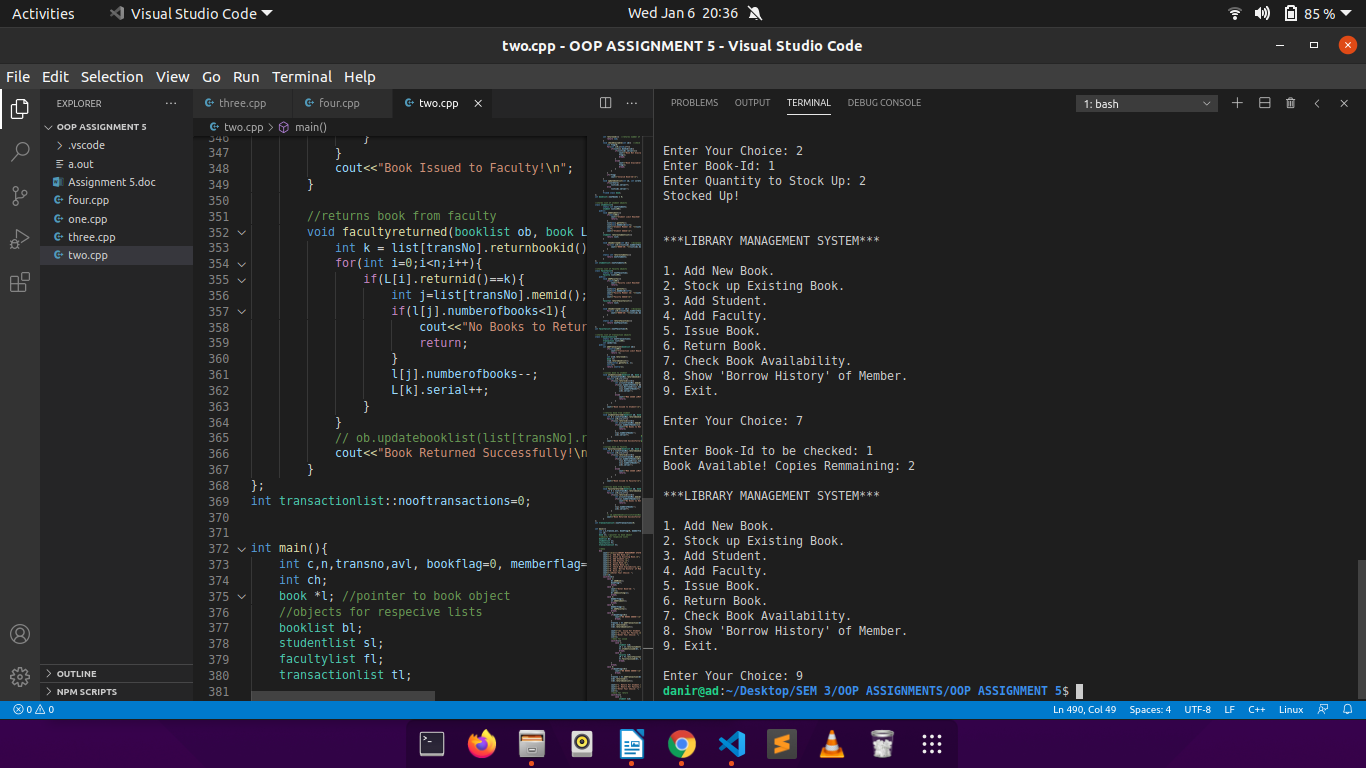
OUTPUT:

****









Q2.

CODE:

#include<iostream>

#include<fstream>

using namespace std;

//stores student details

class student{

int roll;

char name[31];

int score;

public:

void getData(int r) { //accepts student data

cout<<"Enter Name: ";

cin>>name;

cout<<"Enter Score: ";

cin>>score;

roll = r;

cout<<"Roll Alloted: "<<roll<<endl;

}

void showData(){ //prints student data

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

cout<<"\nName: "<<name;

cout<<"\nScore: "<<score;

}

};

//stores list of student objects

class list{

static int numberofstudents;

student array[100];

public:

void addStudent(){ //adds a student to the list

array[numberofstudents].getData(numberofstudents+1);

numberofstudents++;

}

void displayStudent(int r) throw(int){

if(r>100 || r<1 || r>numberofstudents) //throws error if roll goes out of bound

throw -1;

array[r-1].showData();

}

};

int list::numberofstudents=0; //static variable initialilised outside

int main(){

list ob;

int ch, roll;

do{

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

cout<<"2. Display Student.\n";

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

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch){

case 1:

ob.addStudent();

break;

case 2:

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

cin>>roll;

//try-catch block

try{

ob.displayStudent(roll);

}

catch(int i){

cout<<"INVALID ROLL NUMBER!\n";

}

break;

case 3:

exit(0);

break;

default:

cout<<"Invalid Choice!\n";

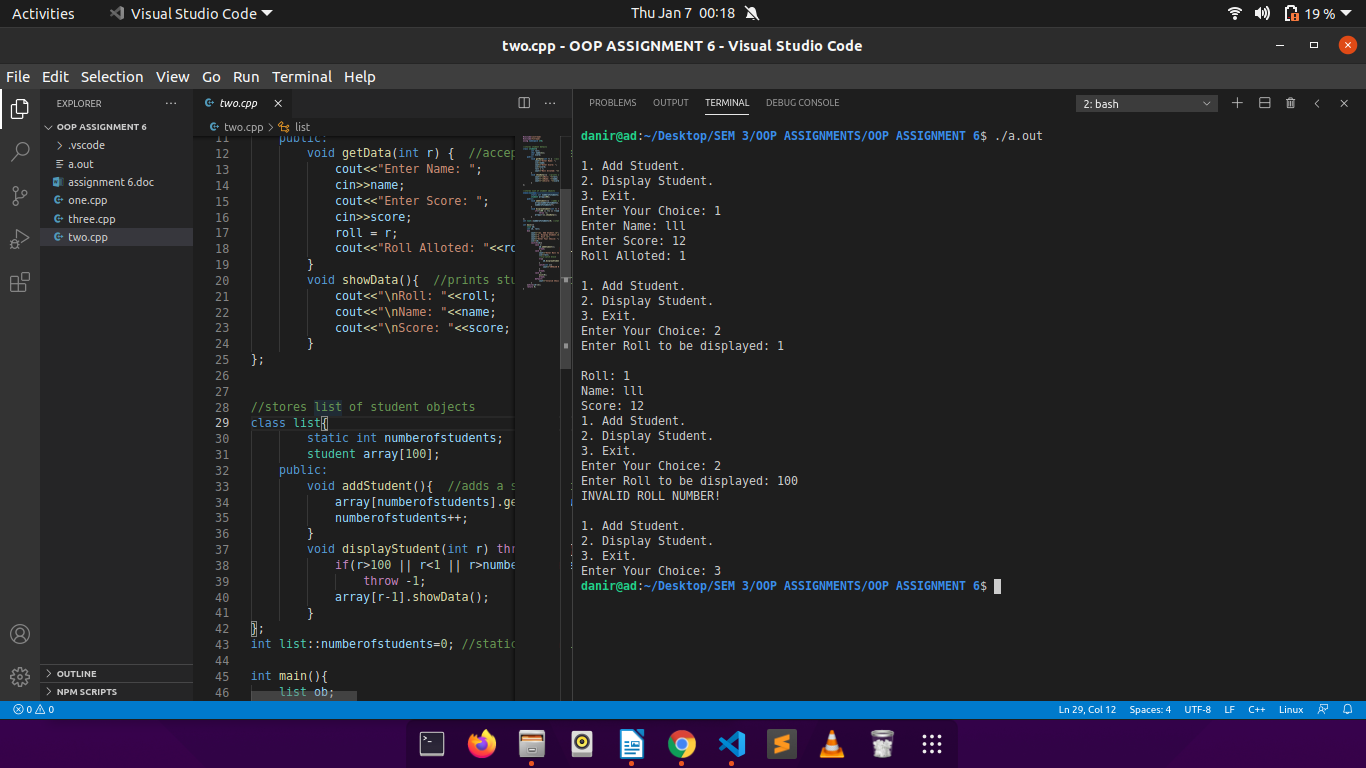
}

}while(ch!=3);

return 0;

}

OUTPUT:

****

Q3.

CODE:

#include<iostream>

using namespace std;

//class template for array of elements with type X

template < class X >

class Array{

X \*arr;

int size;

public:

Array(int s){ //constructor

if(s==0)

arr=NULL;

else

arr = new X(s);

size=s;

}

void getData(){

cout<<"Enter Data:\n";

for(int i=0;i<size;i++)

cin>>arr[i];

}

X findMax(){ //finds max of all elements

X max=arr[0];

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

if(arr[i]>max)

max=arr[i];

}

return max;

}

X findSum();

};

//function template defined explicitly for return type T

template < class T >

T Array < T > ::findSum(){ //returns sum of all elements

T sum=0;

for(int i=0;i<size;i++)

sum += arr[i];

return sum;

}

int main(){

Array < int > ob1(5); //array of 5 integers

ob1.getData();

cout<<"Max Element is: "<<ob1.findMax()<<"\n";

cout<<"Sum of elements is: "<<ob1.findSum()<<"\n";

Array < float > ob2(3); //array of 3 floats

ob2.getData();

cout<<"Max Element is: "<<ob2.findMax()<<"\n";

cout<<"Sum of elements is: "<<ob2.findSum()<<"\n";

Array < char > ob3(4); //array of 4 characters

ob3.getData();

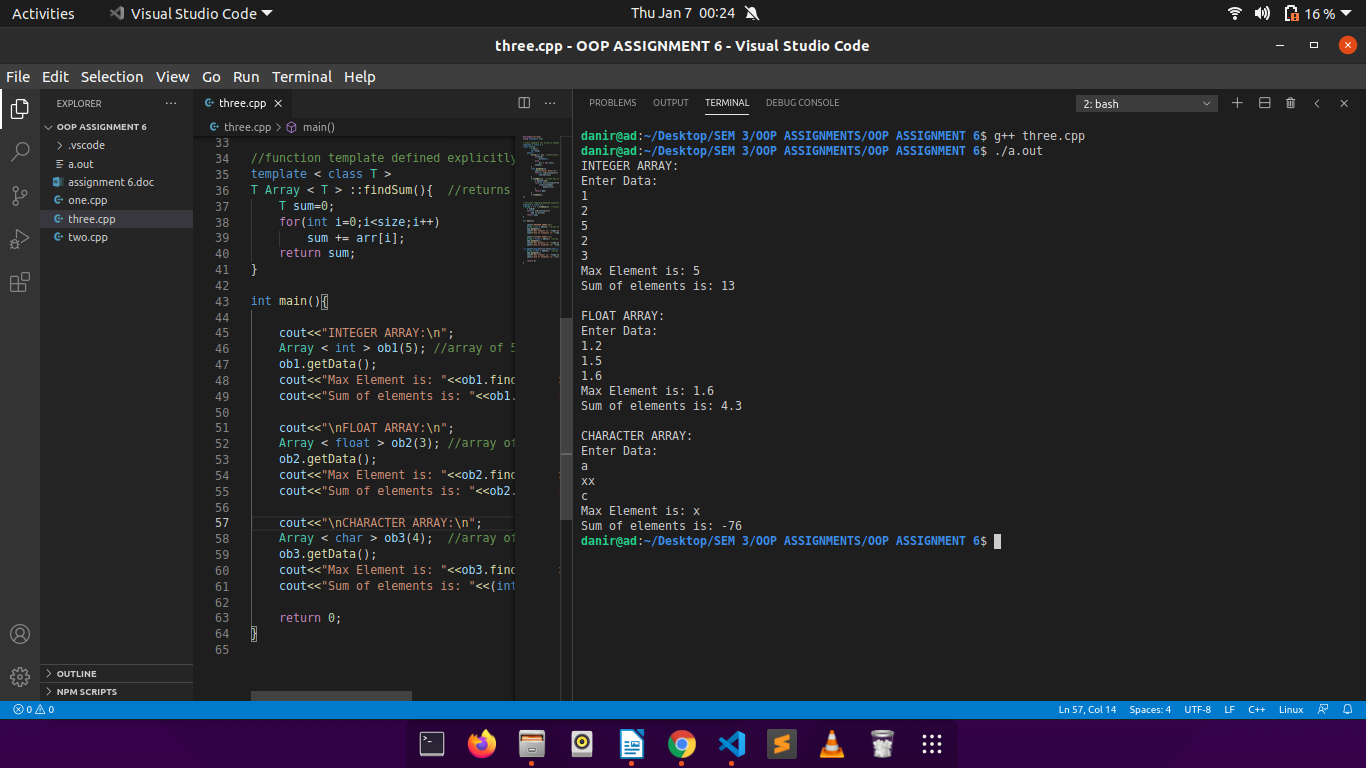
cout<<"Max Element is: "<<ob3.findMax()<<"\n";

cout<<"Sum of elements is: "<<(int)ob3.findSum()<<"\n";

return 0;

}

OUTPUT:

****