```
/*
   ASSIGNMENT 5                QUESTION 1
There are a number of students. For every student roll (unique), name is to be stored. For each subject, subject code and name is to be stored. A student can opt for a number of subjects. System should be able to maintain a student list, subject list and will be able to answer: i) which student has selected which subjects and ii) for the subjects who are the students.
Design the classes and implement. For lists consider memory data structure.
   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/
#include <bits/stdc++.h>
using namespace std;
class Subject;
class Student {
   public: string roll;
   string name;
   vector < Subject * > subjects;
   Student(string r, string n): roll(r),
   name(n) {}
   void addSubject(Subject * subject);
   void displaySubjects() const;
};
class Subject {
   public: string code;
   string name;
   vector < Student * > students;
   Subject(string c, string n): code(c),
   name(n) {}
   void addStudent(Student * student) {
      students.push_back(student);
   }
   void displayStudents() const {
      cout << " - Student Roll: Student Name\n";
      for (const auto & stu: students) {
         cout << " - " << stu -> roll << ": " << stu -> name << "\n";
      }
   }
};
void Student::addSubject(Subject * subject) {
   subjects.push_back(subject);
}
void Student::displaySubjects() const {
   cout << " - Subject Code: Subject Name\n";
   for (const auto & subj: subjects) {
      cout << " - " << subj -> code << ": " << subj -> name << "\n";
   }
}
```

```
class System {
    vector < Student * > students;
    vector < Subject * > subjects;
    public:
        void addStudent(string roll, string name) {
            students.push_back(new Student(roll, name));
        }
        void addSubject(string code, string name) {
            subjects.push_back(new Subject(code, name));
        }
        void enrollSubject(string roll, string code) {
            Student * student = nullptr;
            Subject * subject = nullptr;
            for (auto s: students) {
                if (s -> roll == roll) {
                    student = s;
                    break;
                }
            }
            for (auto subj: subjects) {
                if (subj -> code == code) {
                    subject = subj;
                    break;
                }
            }
            if (student && subject) {
                student -> addSubject(subject);
                subject -> addStudent(student);
                cout << "Enrolled " << student -> name << " in " << subject -> name <<
".\n";
            }
            else
                cout << "Student or subject not found.\n";
        }
        void displayStudents() const {
            for (const auto & student: students) {
                cout << "Student: " << student -> name << " (" << student -> roll << ")\n";
                student -> displaySubjects();
            }
        }
        void displayStuSub(const string & roll) const {
            Student * student = nullptr;
            for (auto s: students) {
                if (s -> roll == roll) {
                    student = s;
                    break;
                }
```

```
        }
        if (student)
            student -> displaySubjects();
        else
            cout << "Student not found\n";
    }
    void displaySubStu(const string & code) const {
        Subject * subject = nullptr;
        for (auto subj: subjects) {
            if (subj -> code == code) {
                subject = subj;
                break;
            }
        }
        if (subject)
            subject -> displayStudents();
        else
            cout << "Subject not found\n";
    }
};
class Menu {
    System system;
    public:
        void displayMenu() {
            int choice;
            cout << "Menu:\n";
            cout << "1. Add Student\n";
            cout << "2. Add Subject\n";
            cout << "3. Enroll Student in Subject\n";
            cout << "4. Display Students and their Enrollments\n";
            cout << "5. Display Students for a subject\n";
            cout << "6. Display Subjects for a student\n";
            cout << "7. Exit\n";
            do {
                cout << "Enter your choice: ";
                cin >> choice;
                switch (choice) {
                    case 1:
                        addStudent();
                        break;
                    case 2:
                        addSubject();
                        break;
                    case 3:
                        enrollStudent();
                        break;
                    case 4:
```

```
                system.displayStudents();
                break;
            case 6:
                displaySubjectsForStudent();
                break;
            case 5:
                displayStudentsForSubject();
                break;
            case 7:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 7);
}
private:
    void addStudent() {
        string roll, name;
        cout << "Enter roll number: ";
        cin >> roll;
        cout << "Enter name: ";
        cin.ignore();
        getline(cin, name);
        system.addStudent(roll, name);
        cout << "Student added.\n";
    }
    void addSubject() {
        string code, name;
        cout << "Enter subject code: ";
        cin >> code;
        cout << "Enter subject name: ";
        cin.ignore();
        getline(cin, name);
        system.addSubject(code, name);
        cout << "Subject added.\n";
    }
    void enrollStudent() {
        string roll, code;
        cout << "Enter student roll number: ";
        cin >> roll;
        cout << "Enter subject code: ";
        cin >> code;
        system.enrollSubject(roll, code);
    }
    void displaySubjectsForStudent() {
        string roll;
```

```
            cout << "Enter Student roll number: ";
            cin.ignore();
            getline(cin, roll);
            system.displayStuSub(roll);
        }
        void displayStudentsForSubject() {
            string code;
            cout << "Enter subject code: ";
            cin.ignore();
            getline(cin, code);
            system.displaySubStu(code);
        }
};
int main() {
    Menu menu;
    menu.displayMenu();
    return 0;
}
```

```
Menu:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Display Students and their Enrollments
5. Display Students for a subject
6. Display Subjects for a student
7. Exit
Enter your choice: 1
Enter roll number: 1
Enter name: anirudh
Student added.
Enter your choice: 2
Enter subject code: 102
Enter subject name: maths
Subject added.
Enter your choice: 4
Student: anirudh (1)
 - Subject Code: Subject Name
Enter your choice: 3
Enter student roll number: 1
Enter subject code: 102
Enrolled anirudh in maths.
Enter your choice: 4
Student: anirudh (1)
 - Subject Code: Subject Name
 - 102: maths
Enter your choice: 5
Enter subject code: 102
 - Student Roll: Student Name
 - 1: anirudh
Enter your choice: 6
Enter Student roll number: 23
Student not found
Enter your choice: 7
Exiting...
```

```
/*
   ASSIGNMENT 5                QUESTION 2
In a library, for each book book-id, serial number (denotes copy number of a book), title, author, publisher and price are stored. Book-id and serial number together will be a unique identifier for a book. Members are either students or faculty. Each member has a unique member-id. Name, e-mail, address are also to be stored. For any transaction (book issue or return), members are supposed to place transactions slip. Users will submit member-id, book-id, and serial number (only for book return). While processing a transaction, check the validity of the member. While issuing, availability of a copy of the book is to be checked. While returning a book, it is to be checked whether this copy was issued to the member or not. A student member can have 2 books issued at a point of time. For faculty members it is 10. Transaction information is to be stored like date of transaction, member-id, book-id, serial number, returned or not. An entry is made when a book is issued and updated when the book is returned.
Design the classes and implement. For lists consider memory data structure.

   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/
#include<iostream>
using namespace std;
class Book{
   int bookid;
   string title;
   string author;
   string publisher;
   double price;
   bool avail;
   public:
      Book(int b = -1, string t = "\0", string a = "\0", string p = "\0", double m =
0.0){
         bookid = b;
         title = t;
         author = a;
         publisher = p;
         price = m;
         avail = true;
      }
      int getbookid(){
         return bookid;
      }
      void changeavailablestatus(bool f){
         avail = f;
      }
      bool available(){
         return avail;
      }
      void displaybookdetails();
};
```

```
class BookCopy : public Book{
    int serialnum;
    public:
        BookCopy(int b = -1, int s = -1, string t = "\0", string a = "\0", string p = "\0",
double m = 0.0) : Book(b, t, a, p, m) {
            serialnum = s;
        }
        int getserialnum(){
            return serialnum;
        }
        void displaybookcopydetails();
};
class BookList{
    BookCopy list[200];                    //currently stroring 200 books.
    int count;
    public:
        BookList(){
            count = 0;
        }
        void addbooks(int, int, string, string, string, double);
        void showbooks();
        int checkavailability(int,int);
        void changestatus(int,int,bool);
};
class Member{
    int memberid;
    string name;
    string email;
    string address;
    public:
        Member(int u = -1, string n = "\0", string e = "\0", string a = "\0"){
            memberid = u;
            name = n;
            email = e;
            address = a;
        }
        int getmemberid(){
            return memberid;
        }
        void displaymemberdetails(){
            cout<<"Member Details are:- \n";
            cout<<"Member id:- "<<memberid<<endl;
            cout<<"Name:- "<<name<<endl;
            cout<<"Email:- "<<email<<endl;
            cout<<"Address:- "<<address<<endl;
        }
};
```

```
class Student : public Member{
   public:
      int bookids[2];                  //store books id taken by the student
      int serialnums[2];               //store serial nums of book taken by the student
      int count;
      Student(int u = -1, string n = "\0", string e = "\0", string a = "\0") :
Member(u,n,e,a){
         count = 0;
      }
      void displaystudentdetails();    //display all details of student.
      void addBook(int,int,BookList&);          //addsbook.
      void remBook(int,int,BookList&);
};
class Faculty : public Member{
   public:
      int bookids[10];                 //store books id taken by the faculty
      int serialnums[10];              //store serial nums of book taken by the facultty
      int count;
      Faculty(int u = -1, string n = "\0", string e = "\0", string a = "\0") :
Member(u,n,e,a){
         count = 0;
      }
      void displayfacultydetails();    //display all details of student.
      void addBook(int,int,BookList&);          //addsbook.
      void remBook(int,int,BookList&);
};
class StudentList{
   Student list[50];                   //currently storing 50 students.
   int count;
   public:
      StudentList(){
         count = 0;
      }
      void addStudent(int,string,string,string);
      int addstubooks(int,int,int,BookList&);                    //add books to the
student.
      int sturembooks(int,int,int,BookList&);
      int checkStudent(int);
};
class FacultyList{
   Faculty list[10];                   //currently storing 10 faculty.
   int count;
   public:
      FacultyList(){
         count = 0;
      }
      void addFaculty(int,string,string,string);
```

```
        int addfacbooks(int,int,int,BookList&);                         //add books to the
faculty.
        int facrembooks(int,int,int,BookList&);
        int checkfaculty(int);
};
class Transaction{
    int memberid;
    int bookid;
    int serialnum;
    bool returned;
    public:
        Transaction(int m = -1,int b = -1, int s = -1, string d = "\0"){
            memberid = m;
            bookid = b;
            serialnum = s;
        }
        int getbookid(){
            return bookid;
        }
        int getserialnum(){
            return serialnum;
        }
        int getmemberid(){
            return memberid;
        }
        void returnstatus(bool r){              //changes return status
            returned = r;
        }
        void displaytransaction();
};
class Issue : public Transaction{
    string date;            //format(dd/mm/yyyy)
    public:
        Issue(int m = -1,int b = -1, int s = -1, string d = "\0") : Transaction(m,b,s,d){
            date = d;
            returnstatus(false);
        }
        string getdate(){
            return date;
        }
};
class IssueList{
    Issue list[100];
    int count;
    public:
        IssueList(){
            count = 0;
```

```
        }
        void issuebook(int,int,int,string,BookList&,StudentList&,FacultyList&);
        void displayissuedbooks();
};
class Return : public Transaction{
    string date;            //format(dd/mm/yyyy)
    public:
        Return(int m = -1,int b = -1, int s = -1, string d = "\0") : Transaction(m,b,s,d){
            date = d;
            returnstatus(true);
        }
        string getdate(){
            return date;
        }
};
class ReturnList {
    Return list[100];
    int count;
    public:
        ReturnList(){
            count = 0;
        }
        void returnbook(int,int,int,string,BookList&,StudentList&,FacultyList&);
        void checkreturnstatus(int,int,int,BookList&,StudentList&,FacultyList&);
        void displayreturnedbooks();
};
class Library{            //will perform all the operations upfront.
    public:
    void performoperations();
};
int StudentList::checkStudent(int m){
    if(count == 0){
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            return 1;
        }
    }
    return 0;
}
int FacultyList::checkfaculty(int m){
    if(count == 0){
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
```

```
            return 1;
        }
    }
    return 0;
}
void Book::displaybookdetails(){
    cout<<"Book Details are:- \n";
    cout<<"Book id:- "<<bookid<<endl;
    cout<<"Title:- "<<title<<endl;
    cout<<"Author:- "<<author<<endl;
    cout<<"Publisher:- "<<publisher<<endl;
    cout<<"Price:- "<<price<<endl;
}
void BookCopy::displaybookcopydetails(){
    Book::displaybookdetails();
    cout<<"Serial number:- "<<serialnum<<endl;
}
void Student::addBook(int b,int s,BookList& t){
    if(count >= 2){
        cout<<"Can not issue more than 2 books.\n";
        return;
    }
    int k = t.checkavailability(b,s);
    if(k == 0){
        cout<<"Book not available\n";
    }
    for(int i = 0; i < count; i++){
        if(bookids[i] == b && serialnums[i] == s){
            cout<<"Book already issued.\n";
            return;
        }
    }
    bookids[count] = b;
    serialnums[count] = s;
    t.changestatus(b,s,false);
    count++;
}
void Student::remBook(int b,int s,BookList& t){
    if(count == 0){
        cout<<"No Books found\n";
        return ;
    }
    int k;
    for(int i = 0; i < count; i++){
        if(bookids[i] == b && serialnums[i] == s){
            k = i;
        }
```

```
    }
    for(int i = k; i < count-1;i++){
        bookids[i] = bookids[i+1];
        serialnums[i] = serialnums[i+1];
    }
    t.changestatus(b,s,true);
    count--;
    return ;
}
void Faculty::remBook(int b,int s,BookList& t){
    if(count == 0){
        cout<<"No Books found\n";
        return;
    }
    int k;
    for(int i = 0; i < count; i++){
        if(bookids[i] == b && serialnums[i] == s){
            k = i;
        }
    }
    for(int i = k; i < count-1;i++){
        bookids[i] = bookids[i+1];
        serialnums[i] = serialnums[i+1];
    }
    t.changestatus(b,s,true);
    count--;
}
void Student::displaystudentdetails(){
    displaymemberdetails();
    if(count != 0){
        cout<<"Books-id and its Serial number issued are:- \n";
        for(int i = 0; i < count; i++){
            cout<<bookids[i]<<" "<<serialnums[i]<<endl;
        }
    }
}
void Faculty::addBook(int b,int s,BookList& t){
    if(count >= 10){
        cout<<"Can not issue more than 10 books.\n";
        return;
    }
    int k = t.checkavailability(b,s);
    if(k == 0){
        cout<<"Book not available\n";
    }
    for(int i = 0; i < count; i++){
        if(bookids[i] == b && serialnums[i] == s){
```

```
            cout<<"Book already issued.\n";
            return;
        }
    }
    bookids[count] = b;
    serialnums[count] = s;
    t.changestatus(b,s,false);
    count++;
}
void Faculty::displayfacultydetails(){
    displaymemberdetails();
    if(count != 0){
        cout<<"Books-id and its Serial number issued are:- \n";
        for(int i = 0; i < count; i++){
            cout<<bookids[i]<<" "<<serialnums[i]<<endl;
        }
    }
}
void BookList::addbooks(int b, int s, string t, string a, string p, double m){
    BookCopy temp(b,s,t,a,p,m);
    if(count >= 100){
        cout<<"Books exceeded\n";
        return;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getbookid() == b && list[i].getserialnum() == s){
            cout<<"Book already exists\n";
            return;
        }
    }
    list[count] = temp;
    count++;
}
void BookList::showbooks(){
    if(count == 0){
        cout<<"No Books found\n";
        return;
    }
    for(int i = 0; i < count; i++){
        list[i].displaybookcopydetails();
    }
}
void StudentList::addStudent(int m,string n,string e,string a){
    Student temp(m,n,e,a);
    if(count >= 50){
        cout<<"Memberships are full\n";
        return;
```

```
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            cout<<"Already a member\n";
            return;
        }
    }
    list[count] = temp;
    count++;
}
int StudentList::addstubooks(int m,int b,int s,BookList& t){
    if(count == 0){
        cout<<"No Student found\n";
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            if(list[i].count >= 2){
                cout<<"Maximum books are issued\n";
                return 0;
            }
            list[i].addBook(b,s,t);
        }
    }
    return 1;
}
int StudentList::sturembooks(int m,int b,int s,BookList& t){
    if(count == 0){
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            list[i].remBook(b,s,t);
            return 1;
        }
    }
    return 0;
}
int FacultyList::facrembooks(int m,int b,int s,BookList& t){
    if(count == 0){
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            list[i].remBook(b,s, t);
            return 1;
        }
```

```
    }
    return 0;
}
void FacultyList::addFaculty(int m,string n,string e,string a){
    Faculty temp(m,n,e,a);
    if(count >= 10){
        cout<<"Memberships are full\n";
        return;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            cout<<"Already a member\n";
            return;
        }
    }
    list[count] = temp;
    count++;
}
int FacultyList::addfacbooks(int m,int b,int s,BookList& t){
    if(count == 0){
        cout<<"No Faculty found\n";
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getmemberid() == m){
            if(list[i].count >= 10){
                cout<<"Maximum books are issued\n";
                return 0;
            }
            list[i].addBook(b,s,t);
        }
    }
    return 1;
}
void Transaction::displaytransaction(){
    cout<<"Member-id:- "<<memberid<<endl;
    cout<<"Book-id:- "<<bookid<<endl;
    cout<<"Serial Number:- "<<serialnum<<endl;
}
int BookList::checkavailability(int b,int s){
    if(count == 0){
        cout<<"No Book Found\n";
        return 0;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getbookid() == b && list[i].getserialnum() == s){
            if(list[i].available() == true){
```

```
            return 1;
        }
        else{
            return 0;
        }
    }
    return 0;
}
void BookList::changestatus(int b,int s,bool f){
    if(count == 0){
        return;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getbookid() == b && list[i].getserialnum() == s){
            list[i].changeavailablestatus(f);
        }
    }
}
void IssueList::issuebook(int m,int b,int s,string d,BookList& t,StudentList& st,
FacultyList& fac){
    Issue temp(m,b,s,d);
    if(count >= 100){
        cout<<"Can not issue books\n";
        return;
    }
    int k = t.checkavailability(b,s);
    int r = st.checkStudent(m);
    int x = fac.checkfaculty(m);
    if(k == 0){
        cout<<"Book not available\n";
        return;
    }
    if(r == 0 && x == 0){
        cout<<"Member not found\n";
        return;
    }
    if(st.addstubooks(m,b,s,t)||fac.addfacbooks(m,s,b,t)){
        list[count] = temp;
        count++;
        return;
    }
}
void IssueList::displayissuedbooks(){
    if(count == 0){
        cout<<"No books issued\n";
        return;
```

```
    }
    cout<<"Issue List:- \n";
    for(int i = 0; i < count; i++){
        cout<<"Member-id:- "<<list[i].getmemberid()<<endl;
        cout<<"Book id:- "<<list[i].getbookid()<<endl;
        cout<<"Serial Number:- "<<list[i].getserialnum()<<endl;
        cout<<"Date:- "<<list[i].getdate()<<endl;
        cout<<"----------------"<<endl;
    }
    return;
}
void ReturnList::returnbook(int m,int b,int s,string d,BookList& t,StudentList& st,
FacultyList& fac){
    Return temp(m,b,s,d);
    if(count >= 100){
        cout<<"Book cant be returned now\n";
        return;
    }
    if(t.checkavailability(b,s) == 0){
        cout<<"Book wasnt issued\n";
        return;
    }
    int r = st.checkStudent(m);
    int x = fac.checkfaculty(m);
    if(r == 0 && x == 0){
        cout<<"Member not found\n";
        return;
    }
    if(st.sturembooks(m,b,s,t)||fac.facrembooks(m,b,s,t)){
        list[count] = temp;
        count++;
        return;
    }
}
void ReturnList::checkreturnstatus(int m,int b,int s,BookList& t,StudentList& st,
FacultyList& fac){
    if(count == 0){
        cout<<"Not returned\n";
        return;
    }
    for(int i = 0; i < count; i++){
        if(list[i].getserialnum() == s && list[i].getbookid() == b && list[i].getmemberid()
== m){
            if(st.sturembooks(m,b,s,t)||fac.facrembooks(m,b,s,t)){
                cout<<"Book Returned\n";
                return;
            }
```

```
            else{
                cout<<"Book not returned\n";
                return;
            }
        }
    }
    cout<<"No such book was issued\n";
}
void ReturnList::displayreturnedbooks(){
    if(count == 0){
        cout<<"No books returned\n";
        return;
    }
    cout<<"Return List:- \n";
    for(int i = 0; i < count; i++){
        cout<<"Member-id:- "<<list[i].getmemberid()<<endl;
        cout<<"Book id:- "<<list[i].getbookid()<<endl;
        cout<<"Serial Number:- "<<list[i].getserialnum()<<endl;
        cout<<"Date:- "<<list[i].getdate()<<endl;
        cout<<"----------------"<<endl;
    }
    return;
}
void Library::performoperations(){
    BookList bk;
    StudentList st;
    FacultyList fac;
    IssueList iss;
    ReturnList ret;
    int choice;
    cout<<"Enter number to perform following Operations:- \n";
    cout<<"1 -> Add Book\n";
    cout<<"2 -> Add Student Member\n";
    cout<<"3 -> Add Faculty Member\n";
    cout<<"4 -> Show Books\n";
    cout<<"5 -> Check Availabilty of a book\n";
    cout<<"6 -> Issue Book\n";
    cout<<"7 -> Return Book\n";
    cout<<"8 -> Check Issued Books\n";
    cout<<"9 -> Check Returned Book\n";
    cout<<"10 -> Check whether a book is returned or not.\n";
    cout<<"0 -> Exit\n";
    do{
        cout<<"Enter your choice :";
        cin>>choice;
        int b,m,s;
        string t,n,p,a,e,d;
```

```
double c;
switch(choice){
case 1:
    cout<<"Enter the details of Book:- \n";
    cout<<"Bookid:- ";cin>>b;
    cout<<"Serial Number:- ";cin>>s;
    cout<<"Title:- ";cin>>t;
    cout<<"Author:- ";cin>>a;
    cout<<"Publisher:- ";cin>>p;
    cout<<"Price:- ";cin>>c;
    bk.addbooks(b,s,t,a,p,c);
    break;
case 2:
    cout<<"Enter the details of member:- \n";
    cout<<"Memberid:- ";cin>>m;
    cout<<"Name:- ";cin>>n;
    cout<<"Email:- ";cin>>e;
    cout<<"Address:- ";cin>>a;
    st.addStudent(m,n,e,a);
    break;
case 3:
    cout<<"Enter the details of member:- \n";
    cout<<"Memberid:- ";cin>>m;
    cout<<"Name:- ";cin>>n;
    cout<<"Email:- ";cin>>e;
    cout<<"Address:- ";cin>>a;
    fac.addFaculty(m,n,e,a);
    break;
case 4:
    bk.showbooks();
    break;
case 5:
    cout<<"Enter Details of Book to check:- \n";
    cout<<"Bookid:- ";cin>>b;
    cout<<"Serial Number:- ";cin>>s;
    if(bk.checkavailability(b,s))
        cout<<"Available\n";
    else
        cout<<"Not Available\n";
    break;
case 6:
    cout<<"Enter the details to issue a book:- \n";
    cout<<"Memberid:- ";cin>>m;
    cout<<"Bookid:- ";cin>>b;
    cout<<"Serial Number:- ";cin>>s;
    cout<<"Date:- ";cin>>a;
    iss.issuebook(m,b,s,a,bk,st,fac);
```

```
            break;
        case 7:
            cout<<"Enter the details to return a book:- \n";
            cout<<"Memberid:- ";cin>>m;
            cout<<"Bookid:- ";cin>>b;
            cout<<"Serial Number:- ";cin>>s;
            cout<<"Date:- ";cin>>a;
            ret.returnbook(m,b,s,a,bk,st,fac);
            break;
        case 8:
            iss.displayissuedbooks();
            break;
        case 9:
            ret.displayreturnedbooks();
            break;
        case 10:
            cout<<"Enter the details to check whether a book is returned- \n";
            cout<<"Memberid:- ";cin>>m;
            cout<<"Bookid:- ";cin>>b;
            cout<<"Serial Number:- ";cin>>s;
            ret.checkreturnstatus(m,b,s,bk,st,fac);
            break;
        case 0:
            break;
        }
    }while(choice != 0);

    return;
}
int main(){
    Library lib;
    lib.performoperations();
    return 0;
}
```

```
Enter number to perform following Operations:-
1 -> Add Book
2 -> Add Student Member
3 -> Add Faculty Member
4 -> Show Books
5 -> Check Availabilty of a book
6 -> Issue Book
7 -> Return Book
8 -> Check Issued Books
9 -> Check Returned Book
10 -> Check whether a book is returned or not.
0 -> Exit
Enter your choice :1
Enter the details of Book:-
Bookid:- 100
Serial Number:- 201
Title:- maths
Author:- moody
Publisher:- AM
Price:- 1000
Enter your choice :2
Enter the details of member:-
Memberid:- 101
Name:- anirudh
Email:- iamanirudh.modi@gmail.com
Address:- barrackpore
Enter your choice :3
Enter the details of member:-
Memberid:- 910
Name:- sks
Email:- sks@gmail.com
Address:- jadavpur
Enter your choice :4
Book Details are:-
```

```
Enter your choice :3
Enter the details of member:-
Memberid:- 910
Name:- sks
Email:- sks@gmail.com
Address:- jadavpur
Enter your choice :4
Book Details are:-
Book id:- 100
Title:- maths
Author:- moody
Publisher:- AM
Price:- 1000
Serial number:- 201
Enter your choice :5
Enter Details of Book to check:-
Bookid:- 100
Serial Number:- 202
Not Available
Enter your choice :6
Enter the details to issue a book:-
Memberid:- 100
Bookid:- 100
Serial Number:- 201
Date:- 12/11/2024
Member not found
Enter your choice :8
No books issued
Enter your choice :6
Enter the details to issue a book:-
Memberid:- 101
```

```
/*
   ASSIGNMENT 5                QUESTION 3
```
Employees have unique emp-id, name, designation and basic pay. An employee is either a permanent one or contractual. For permanent employees salary is computed as basic pay+ hra (30% of basic pay) + da (80% of basic pay). For contractual employees it is basic pay + allowance (it is different for different contractual employees). An employee pointer may point to either of the two categories and accordingly the salary has to be created.
Design the classes and implement.
```
   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/
#include <bits/stdc++.h>
using namespace std;
class Employee {
   protected:
      string empId;
      string name;
      string designation;
      double basicPay;
   public:
      Employee(string id, string n, string d, double pay): empId(id),
      name(n),
      designation(d),
      basicPay(pay) {}
      virtual double computeSalary() = 0; // Pure virtual function
      virtual void display() {
         cout << "Employee ID: " << empId << "\nName: " << name <<
         "\nDesignation: " << designation << "\nBasic Pay: " << basicPay << endl;
      }
      virtual~Employee() {}
};
// Derived class for Permanent Employee
class PermanentEmployee: public Employee {
   public: PermanentEmployee(string id, string n, string d, double pay):
Employee(id, n, d, pay) {}
   double computeSalary() override {
      double hra = 0.30 * basicPay;
      double da = 0.80 * basicPay;
      return basicPay + hra + da;
   }
   void display() override {
      Employee::display();
      cout << "Salary: " << computeSalary() << endl;
   }
};
// Derived class for Contractual Employee
class ContractualEmployee: public Employee {
   double allowance;
```

```
    public:
        ContractualEmployee(string id, string n, string d, double pay, double allow):
Employee(id, n, d, pay), allowance(allow) {}
    double computeSalary() override {
        return basicPay + allowance;
    }
    void display() override {
        Employee::display();
        cout << "Allowance: " << allowance << "\nSalary: " << computeSalary() <<
endl;
    }
};
// Menu class for managing the menu options
class Menu {
    public: vector < Employee * > employees;
    void menu() {
        int ch;
        cout << "\nMenu Options:\n";
        cout << "1. Create New Employee\n";
        cout << "2. Display All Employees\n";
        cout << "3. Exit\n";
        do {
            cout << "Enter your choice: ";
            cin >> ch;
            while (ch < 1 || ch > 3) {
                cout << "Invalid choice, please enter a number between 1 and 3: ";
                cin >> ch;
            }
            switch (ch) {
                case 1: {
                    string empId, name, designation, type;
                    double basicPay, allowance;
                    cout << "Enter employee ID: ";
                    cin.ignore();
                    getline(cin, empId);
                    cout << "Enter employee name: ";
                    getline(cin, name);
                    cout << "Enter employee designation: ";
                    getline(cin, designation);
                    cout << "Enter employee type (permanent/contractual): ";
                    getline(cin, type);
                    cout << "Enter basic pay: ";
                    cin >> basicPay;
                    if (type == "permanent") {
                        employees.push_back(new PermanentEmployee(empId, name,
designation, basicPay));
                    } else if (type == "contractual") {
```

```
                cout << "Enter allowance: ";
                cin >> allowance;
                employees.push_back(new ContractualEmployee(empId, name,
designation, basicPay, allowance));
            } else {
                cout << "Invalid employee type entered.\n";
            }
            break;
        }
        case 2: {
            if (employees.empty()) {
                cout << "No employees to display.\n";
            } else {
                for (auto employee: employees) {
                    employee -> display();
                    cout << "-------------------------\n";
                }
            }
            break;
        }
        case 3: {
            for (auto employee: employees) {
                delete employee;
            }
            employees.clear();
            cout << "Exiting... Cleaning up resources.\n";
            break;
        }
      }
    } while (ch != 3);
  }
};

// Main function
int main() {
  Menu m;
  m.menu();
  return 0;
}
```

```
Menu Options:
1. Create New Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter employee ID: 101
Enter employee name: anirudh
Enter employee designation: ceo
Enter employee type (permanent/contractual): permanent
Enter basic pay: 10000
Enter your choice: 2
Employee ID: 101
Name: anirudh
Designation: ceo
Basic Pay: 10000
Salary: 21000
------------------------
Enter your choice: 1
Enter employee ID: 101
Enter employee name: jasnk
Enter employee designation: msaka
Enter employee type (permanent/contractual): conasj
Enter basic pay: 2200102
Invalid employee type entered.
Enter your choice: 3
```

```
/*
   ASSIGNMENT 5                QUESTION 4
Each cricketer has a name, date of birth and matches played. Cricketer may be a bowler or batsman. For a bowler, number of wickets taken, average economy is stored. For a batsman, total runs scored, average score is stored.  A double wicket pair is formed taking a bowler and a batsman. An all-rounder is both a bowler and batsman. Support must be there to show the details of a cricketer, bowler, batsmen, all-rounder and the pair.
Design the classes and implement.
   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/

#include <iostream>

#include <algorithm>

#include <string>

using namespace std;
#define ll long long

class Cricketer {
   protected:
      string name;
      string dob;
      int matches_played;
   public:
      Cricketer(const string n, const string d, int m) {
         name = n;
         dob = d;
         matches_played = m;
      }
      virtual void showDetails() {
         cout << "Name: " << name << ", DOB: " << dob << ", Matches played: " <<
matches_played << endl;
      }
};
class Batsman: virtual public Cricketer {
   protected:
      int totalRuns;
      float avgScore;
   public:
      Batsman(const string n,const string d, int m, int tRuns, float
aScore):Cricketer(n, d, m){
         totalRuns = tRuns;
         avgScore = aScore;
      }
      void showDetails() {
```

```
            Cricketer::showDetails();
            cout << "->Total Runs: " << totalRuns << ", Average score: " << avgScore <<
endl;
        }
};
class Bowler: virtual public Cricketer {
    protected:
        int totalWickets;
        float avgEconomy;
    public:
        Bowler(const string n,const string d,int m,int tWickets,float
aEconomy):Cricketer(n, d, m){
            totalWickets = tWickets;
            avgEconomy = aEconomy;
        }
        void showDetails() {
            Cricketer::showDetails();
            cout << "->Total Wickets: " << totalWickets << ", Average economy: " <<
avgEconomy << endl;
        }
};
class AllRounder: public Bowler, public Batsman {
    public:
        AllRounder(const string & name,const string & dob, int matchesPlayed, int
wicketsTaken, float averageEconomy, int totalRuns, float averageScore):
Cricketer(name, dob, matchesPlayed),Bowler(name, dob, matchesPlayed,
wicketsTaken, averageEconomy),Batsman(name, dob, matchesPlayed, totalRuns,
averageScore) {}
        void showDetails() {
            Cricketer::showDetails();
            cout << "Wickets: " << totalWickets << " Average economy: " << avgEconomy
<< endl;
            cout << "Total runs: " << totalRuns << " Average runs: " << avgScore <<
endl;
        }
};
class DoubleWicketPair {
    Bowler * bowler;
    Batsman * batsman;
    public:
        DoubleWicketPair(Bowler * bow, Batsman * bat) {
            this -> batsman = bat;
            this -> bowler = bow;
        }
    void showDetails() const {
        cout << "Double Wicket Pair Details:\n";
        cout << "Bowler:\n";
```

```
        bowler -> showDetails();
        cout << "Batsman:\n";
        batsman -> showDetails();
    }
};
int main() {
    Bowler bowler("James Anderson", "1982/07/30", 160, 640, 3.1f);
    Batsman batsman("Joe Root", "1990/12/30", 100, 8000, 51.2f);
    AllRounder allRounder("Ben Stokes", "1991/06/04", 90, 150, 4.5f, 4000, 36.7f);
    cout << "\nBowler Details:\n";
    bowler.showDetails();
    cerr << "\nBatsman Details:\n";
    batsman.showDetails();
    clog << "\nAll-Rounder Details:\n";
    allRounder.showDetails();
    cout << "\nDouble Wicket Pair:\n";
    DoubleWicketPair pair( & bowler, & batsman);
    pair.showDetails();
    return 0;
}
```

```
Bowler Details:
Name: James Anderson, DOB: 1982/07/30, Matches played: 160
->Total Wickets: 640, Average economy: 3.1

Batsman Details:
Name: Joe Root, DOB: 1990/12/30, Matches played: 100
->Total Runs: 8000, Average score: 51.2

All-Rounder Details:
Name: Ben Stokes, DOB: 1991/06/04, Matches played: 90
Wickets: 150 Average economy: 4.5
Total runs: 4000 Average runs: 36.7

Double Wicket Pair:
Double Wicket Pair Details:
Bowler:
Name: James Anderson, DOB: 1982/07/30, Matches played: 160
->Total Wickets: 640, Average economy: 3.1
Batsman:
Name: Joe Root, DOB: 1990/12/30, Matches played: 100
->Total Runs: 8000, Average score: 51.2
```