Assignment I: Library Management System

Student Name: Orlando Alvarado Vargas
Student ID: 2226968

Instructions:

You are required to implement a simple Library Management System using Object-Oriented Programming (OOP) principles in C++.
Your program should:

- Define classes for **Book**, **Member**, and **Library**.
- Allow adding new books and members to the library.
- Implement functionality for borrowing and returning books.
- Ensure that a book cannot be borrowed if it is already checked out.
- Display the list of available books and the borrowing status.
- Use appropriate methods and attributes for each class.
- Demonstrate encapsulation, inheritance (if applicable), and other OOP concepts.

Submission Requirements:

- Include your well-commented source code in the report.
- Provide sample test cases and their outputs.
- Briefly explain your design choices and how OOP principles are applied.
- Ensure your code is well-structured and follows best practices.

A Introduction

In this document I will explain how I did the Library Management System for the assignment in C++. With this system you can register books and members, manage the borrowing and returning of books, and displaying the overall status of the library, including the total number of books and members as well as the availability of materials.

B Class Definitions

First of all, I defined 3 classes with their own attributes and functions:

- **Member:** With a name, ID, and an indicator to know if the member currently has a borrowed book. The class provides methods to display user information, retrieve the member's ID, check borrowing status, and update the status when a book is borrowed or returned.
- **Book:** With a title, author, ID and an indicator to know if is borrowed. Its methods allow displaying book details, retrieving the book ID and title, checking its borrowing status, and updating the status when the book is loaned or returned.
- **Library:** Manages the lists of books and members and implements the system's main functions: adding new books and members, borrowing and returning items, and reporting the current state of the library.

B.1 Member Class

```
// Represents a library member with a unique incremental ID
class Member {
private:
    // ---------- Attributes ----------
    string name;        // Member's name
    int memberid;       // ID assigned to each member
    bool hasbook;       // Indicates if the member currently has a
       borrowed book
    static int nextid;  // Static counter for generating unique IDs
public:
    Member (string name, bool hasbook);  // Constructor
    void display();                      // Display member information
    int getid() const;                   // Returns member ID
    bool returnhasbook() const;          // Returns whether the
       member has a book
    void borrowbook();                   // Marks that the member has
       borrowed a book
    void returnbook();                   // Marks that the member has
       returned their book
};
```

B.2 Book Class

```
// Represents a book in the library with title, author, and unique ID
class Book {
private:
    // ---------- Attributes ----------
    string title;       // Book title
    string author;      // Book author
    int bookid;         // ID assigned to each book
    static int nextid;  // Static counter for generating book IDs
    bool borrowed;      // Indicates if the book is borrowed or not
public:
    Book (string title, string author, bool borrowed);  // Constructor
    void display();                         // Display book information
    int getid() const;                      // Returns the book ID
    string gettitle() const;                // Returns the book title
    bool getborrowed() const;               // Returns current borrow
       status
    void isborrowed(bool b);                // Sets borrow status of
       the book
};
```

B.3 Library Class

```
class Library {
public:
    //---------- Definition of vectors ----------
    vector<Book> booklist;
    vector<Member> memberlist;

    void borrow();                // Borrowing function
    void returnb();               // Returning function
    void addmember();             // Add a new member
```

```
    void addbook();                 // Add a new book
    void showstatus();              // Show current library status
};
```

C Method Implementations

This section describes the implementation of the constructors, methods, and accessors of the system's classes. A static counter was used in both the `Member` and `Book` classes to automatically assign unique incremental IDs, ensuring that each instance is consistently identifiable without requiring manual input.

C.1 Member Class

```
int Member::nextid = 1;   // Initialize static counter for
   incremental ID

Member::Member (string name, bool hasbook){
    this -> name = name;
    this -> hasbook = hasbook;
    memberid = nextid++;   // Assigns incremental ID
}
void Member::display(){
    cout << "Name: " << name << "       ID: " << memberid << "       
       Has a book?: " << (hasbook ? "Yes" : "No") << "\n";
}
int Member::getid() const{
    return memberid;
}
bool Member::returnhasbook() const{
    return hasbook;
}
void Member::borrowbook(){
    hasbook = true;
}
void Member::returnbook(){
    hasbook = false;
}
```

C.2 Book Class

```
int Book::nextid = 1;   // Initialize static counter for incremental
   ID

Book::Book (string title, string author, bool borrowed){
    this -> title = title;
    this -> author = author;
    this -> borrowed = borrowed;
    bookid = nextid++;   // Assigns incremental ID
}
void Book::display(){
    cout << "Title: " << title << "      Author: " << author << "       
       ID: " << bookid << "       Is borrowed?: " << (borrowed ?
       "Yes" : "No") << "\n";
}
int Book::getid() const{
    return bookid;
```

```
}
string Book::gettitle() const{
    return title;
}
bool Book::getborrowed() const{
    return borrowed;
}
void Book::isborrowed(bool b){
    borrowed = b;   // Changes borrow status of the book
}
```

C.3 Library Class

The Library class handles the main operations of the system. To keep things simple and avoid making the document overly extensive , I only show the borrowing method here. In this function, I use variables like *mi* and *bi* to check if the entered IDs are valid. The method also includes basic error messages to let the user know if a member or book doesn't exist, if a book is already borrowed, or if a member already has one. When everything is correct, it updates both the member and the book to reflect the loan. The return method works the same way, and the rest of the functions (like adding members, books, or showing the status) are much easier.

```
void Library::borrow(){
    int memberid, bookid;
    int mi = -1;   // Index of member
    int bi = -1;   // Index of book
    cout <<
       "\n----------------------BORROWING----------------------\n";
    cout << "\nEnter␣the␣following␣information:\n";
    cout << "Member's␣ID:␣";
    cin >> memberid;
    cout << "Book's␣ID:␣";
    cin >> bookid;

    // Search for the member by ID
    for (int i = 0; i < memberlist.size(); i++){
        if (memberlist[i].getid() == memberid){
            mi = i;
            break;
        }
    }
    // Search for the book by ID
    for (int i = 0; i < booklist.size(); i++){
        if (booklist[i].getid() == bookid){
            bi = i;
            break;
        }
    }
    // Error handling for invalid IDs
    if (mi == -1){
        cout << "\n#############␣␣␣␣␣I'm␣sorry.␣I␣couldn't␣find␣the␣
           member's␣ID␣in␣the␣system␣D:␣␣␣␣␣#############\n";
        return;
    }
    if (bi == -1){
        cout << "\n#############␣␣␣␣␣I'm␣sorry.␣I␣couldn't␣find␣the␣
```

```
            book's␣ID␣in␣the␣system␣D:␣␣␣␣␣###########\n";
        return;
    }
    // Check if book is already borrowed
    if (booklist[bi].getborrowed()){
        cout << "\n############␣␣␣␣␣This␣book␣is␣already␣borrowed␣:o␣␣
           ␣␣␣############\n";
        return;
    }
    // Check if member already has a book
    if (memberlist[mi].returnhasbook()){
        cout << "\n############␣␣␣␣␣This␣member␣already␣have␣a␣book␣
           :o␣␣␣␣␣############\n";
        return;
    }
    // Update statuses: book borrowed and member has a book
    booklist[bi].isborrowed(true);
    memberlist[mi].borrowbook();

    cout <<
       "\n-----------------------------------------------------\n";
}
```

D Main Function

In the main function, I just created an instance of the Library class, added a few default books and a member, and built a simple menu to interact with the system. The menu allows the user to borrow and return books, register new members or books, check the current status of the library, or exit the program. A switch statement handles the user's choice, and the program keeps running until the exit option is selected.

```
int main () {
    Library lib;   // Create library instance

    // Default books and a member
    lib.booklist.push_back(Book("A␣flor␣de␣piel", "Javier␣Moro",
       false));
    lib.booklist.push_back(Book("Primer␣Plano", "Antonio␣Malpica",
       false));
    lib.memberlist.push_back(Member("Orlando", false));

    int selection;
    do {
        cout << "\n␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Tilin␣Library\n" <<
                "----------------------WELCOME----------------------\n"
                   <<
                "\nPick␣a␣number␣to␣select␣any␣of␣this␣options:\n" <<
                "1)␣Borrow␣a␣book.\n" <<
                "2)␣Return␣a␣book.\n" <<
                "3)␣Register␣a␣new␣member.\n" <<
                "4)␣Register␣a␣new␣book.\n" <<
                "5)␣See␣the␣library␣status.\n" <<
                "0)␣Exit␣the␣System.\n";
        cin >> selection;

        // Handle user selection
```

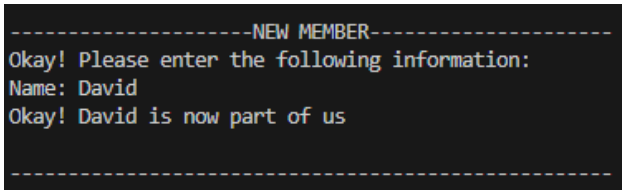
```
        switch (selection){
            case 0:{
                cout << "\nHave␣a␣good␣day!!!\n";
                break;
            }
            case 1:{
                lib.borrow();
                break;
            }
            case 2:{
                lib.returnb();
                break;
            }
            case 3:{
                lib.addmember();
                break;
            }
            case 4:{
                lib.addbook();
                break;
            }
            case 5:{
                lib.showstatus();
                break;
            }
            default:{
                cout << "\n#############␣␣␣␣␣Please␣select␣a␣number␣
                   between␣0␣and␣5␣␣␣>:(␣␣␣␣␣#############\n";
            }
        }
    } while (selection != 0);    // Repeat until user select exit
    return 0;
}
```

E Test Cases

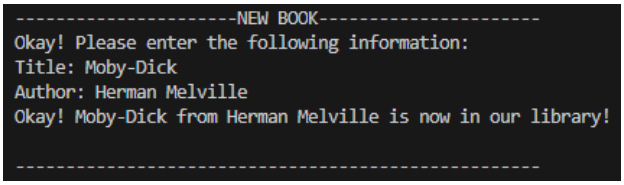
Here are some examples of how the code looks.

Test Case: Main Menu

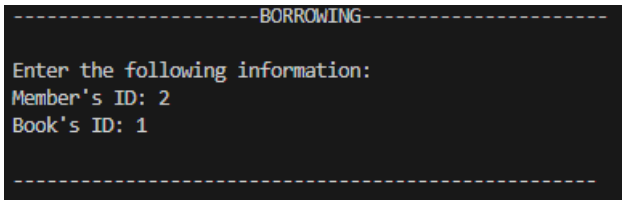
```
                    Tilin Library
----------------------WELCOME-----------------------

Pick a number to select any of this options:
1) Borrow a book.
2) Return a book.
3) Register a new member.
4) Register a new book.
5) See the library status.
0) Exit the System.
```



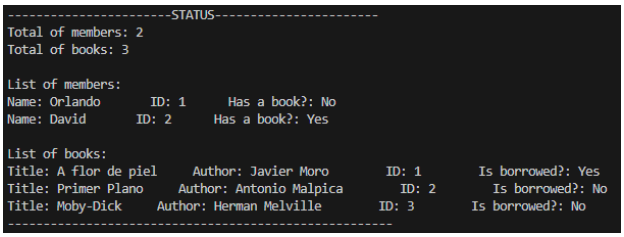
New Member



New Book



Borrowing a book



Show the status

Figure 1: Screenshots of different library operations.