Department of Computer Science and Engineering

C.V. Raman Global University

Experiential Learning Project

Library Management System using C++

SUBMITTED BY:

NAME	**REGISTRATION NO.**
Jivan Jyoti Jala	**CL2025010601910339**
Mayank kumar	**CL2025010601926091**
Riya Metha	**CL2025010601889072**
Rishi Kumar Sah	**CL2025010601937680**
Debadutta Sahoo	**CL20250106019055113**

CONTENTS

DECLARATION:

We hereby declare that this project work entitled ‘Library Management System’ has been prepared by us under the guidance of, Department of Computer Science Engineering, CGU in the regular examinations prescribed by the college.

We also declare that this project is the outcome of our own effort and has not been submitted by any other university for the award of any degree.

ACKNOWLEDGEMENT

We would like to acknowledge all those without whom this project would not have been successful. Firstly, we would wish to thank our Computer Science faculty, who guided us throughout the project and gave their immense support. They made us understand how to successfully complete this project and without them, the project would not have been complete.

This project has been a source to learn and bring our theoretical knowledge to the real-life world. So, we would really acknowledge their help and guidance for this project.

We would also like to thank our families who have always been there whenever needed.

Once again, thanks to everyone for making this project successful.

INTRODUCTION

A Library Management System (LMS) is a digital solution designed to automate and streamline library operations. It enables efficient book tracking, member management, and overall library administration. This system replaces manual record-keeping with an automated database, reducing errors and improving accessibility. Traditional library management involves maintaining registers and manual book-keeping, which is time-consuming and prone to errors. With the rapid advancement of technology, digital solutions are being implemented in various sectors, and libraries are no exception. An LMS provides a centralized platform where library staff can manage books, track borrowing and returning records, and generate reports with ease.

Libraries serve as knowledge repositories, making it essential to manage them efficiently. An LMS plays a crucial role in ensuring smooth operations and enhancing the overall user experience. The implementation of an LMS not only increases productivity but also promotes digital literacy among users.

OBJECTIVE

The primary objective of this project is to develop a robust and user-friendly Library Management System that:

- Automates book issuance, returns, and reservations, minimizing manual effort.
- Maintains an organized catalog of books for easy search and retrieval.
- Reduces human error and manual workload, improving efficiency.
- Enhances accessibility through a digital platform, allowing remote access to library data.
- Provides real-time tracking of book availability and borrower history.
- Implements a role-based access system, ensuring secure management by administrators.

This project aims to bridge the gap between traditional and modern library management by leveraging technological advancements. By doing so, it will improve the overall functionality of the library and provide an enhanced user experience for both librarians and patrons.

TOOLS & TECHS USED

The project is developed using the following tools and technologies:

- **Programming Language:** C++
- **IDE:** Visual Studio Code (VS Code)
- **Compiler:** MinGW/GCC
- **File Handling:** Text-based file storage for book and user records
- **Data Structures:** Arrays, linked lists, and structures for efficient data management
- **Object-Oriented Programming (OOP):** Utilization of classes and objects for modular design

Prerequisite

Before implementing this project, the following prerequisites are required:

- Basic Knowledge of C++: Understanding of syntax, data types, and control structures.
- Object-Oriented Programming (OOP) Concepts: Familiarity with classes, objects, inheritance, and polymorphism.
- File Handling in C++: Knowledge of reading and writing data to files for persistent storage.
- Data Structures & Algorithms: Understanding of linked lists, queues, and searching algorithms for book management.
- Command-Line Interface (CLI) Operations: Ability to work with command-line inputs and outputs efficiently.

Implementation

The development process follows a structured approach:

1. **Class Design:**
   - Creating classes for Book, User, and Library to encapsulate data and operations.
   - Implementing constructors, destructors, and member functions.
2. **File Handling:**
   - Storing book and user records in text files.
   - Using file I/O operations to read, write, append, and search records.
3. **Functionalities Implemented:**
   - **Adding Books:** Allowing the librarian to add books to the library database.
   - **Searching Books:** Implementing linear and binary search algorithms for book retrieval.
   - **Issuing & Returning Books:** Maintaining records of borrowed books and due dates.
   - **User Authentication:** Implementing login credentials for librarian and members.
4. **Error Handling & Validation:**
   - Implementing input validation to prevent incorrect data entries.
   - Handling file read/write errors to prevent data corruption.
5. **Testing & Debugging:**
   - Conducting unit tests for individual functionalities.
   - Debugging logical and runtime errors to ensure smooth execution.

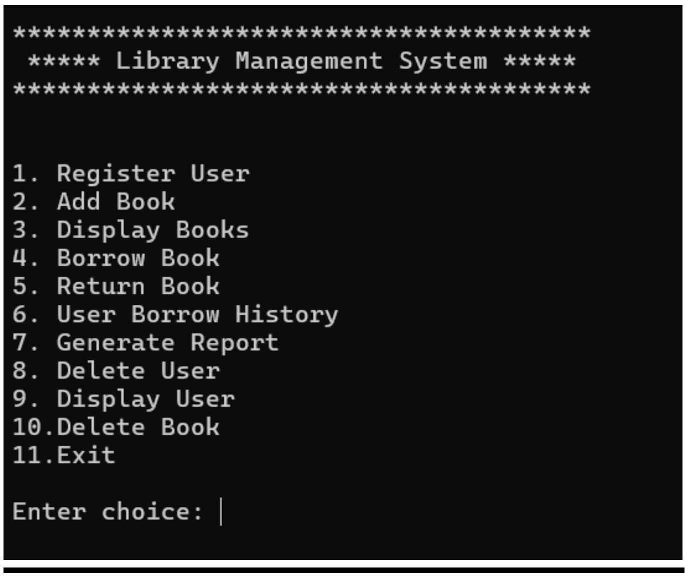
CODE

```
1#include <iostream>
#include <vector>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <map>
#include <algorithm>

using namespace std;

struct Book {
    string id;
    string title;
    string author;
    int quantity;
    int borrowed;
    void display() const {
        cout << left << setw(10) << id
             << setw(25) << title
             << setw(20) << author
             << setw(10) << quantity
             << setw(10) << borrowed << endl;
    }
};

struct User {
    string userId;
    string name;
    vector<string> borrowedBooks;

    void display() const {
        cout << left << setw(10) << userId
             << setw(20) << name
             << "Books Borrowed: ";
        for (size_t i = 0; i < borrowedBooks.size(); i++)
            cout << borrowedBooks[i] << " ";
        cout << endl;
    }
};

vector<Book> library;
map<string, User> users;

void loadFromFile();
void saveToFile();
void addBook();
void displayBooks();
void searchBook();
void deleteBook();
void borrowBook();
void returnBook();
void registerUser();
void displayUsers();
void deleteUser();
void userBorrowHistory();
void generateReport();
void menu();

int main() {
    loadFromFile();
    menu();
    return 0;
}
```

```
void loadFromFile() {
    ifstream file("library.txt");
    if (!file) return;

    Book temp;
    while (file >> temp.id >> temp.title >> temp.author >> temp.quantity >>
temp.borrowed) {
        library.push_back(temp);
    }
    file.close();

    ifstream userFile("users.txt");
    if (!userFile) return;

    User tempUser;
    string line;
    while (getline(userFile, line)) {
        stringstream ss(line);
        ss >> tempUser.userId >> tempUser.name;
        tempUser.borrowedBooks.clear();
        string bookId;
        while (ss >> bookId)
            tempUser.borrowedBooks.push_back(bookId);
        users[tempUser.userId] = tempUser;
    }
    userFile.close();
}

void saveToFile() {
    ofstream file("library.txt");
    for (size_t i = 0; i < library.size(); i++) {
        file << library[i].id << " " << library[i].title << " " << library[i].author << " "
             << library[i].quantity << " " << library[i].borrowed << endl;
    }
    file.close();

    ofstream userFile("users.txt");
    for (map<string, User>::iterator it = users.begin(); it != users.end(); ++it) {
        userFile << it->second.userId << " " << it->second.name;
        for (size_t j = 0; j < it->second.borrowedBooks.size(); j++)
            userFile << " " << it->second.borrowedBooks[j];
        userFile << endl;
    }
    userFile.close();
}
```

```
void addBook() {
    Book temp;
    cout << "Enter Book ID: ";
    cin >> temp.id;
    cout << "Enter Title: ";
    cin.ignore();
    getline(cin, temp.title);
    cout << "Enter Author: ";
    getline(cin, temp.author);
    cout << "Enter Quantity: ";
    cin >> temp.quantity;
    temp.borrowed = 0;
    library.push_back(temp);
    saveToFile();
    cout << "Book added successfully!\n";
}

void displayBooks() {
    if (library.empty()) {
        cout << "No books available.\n";
        return;
    }
    cout << left << setw(10) << "ID" << setw(25) << "Title" << setw(20) <<
"Author" << setw(10) << "Stock" << setw(10) << "Borrowed" << endl;
    cout << string(75, '-') << endl;
    for (size_t i = 0; i < library.size(); i++) {
        library[i].display();
    }
}

void borrowBook() {
    string userId, bookID;
    cout << "Enter User ID: ";
    cin >> userId;
    if (users.find(userId) == users.end()) {
        cout << "User not registered.\n";
        return;
    }
    cout << "Enter Book ID to borrow: ";
    cin >> bookID;
    for (size_t i = 0; i < library.size(); i++) {
        if (library[i].id == bookID) {
            if (library[i].quantity <= 0) {
                cout << "Book out of stock!\n";
                return;
            }
            library[i].quantity--;
            library[i].borrowed++;
            users[userId].borrowedBooks.push_back(bookID);
            saveToFile();
            cout << "Book borrowed successfully!\n";
            return;
        }
    }
    cout << "Book not found.\n";
}

void deleteBook() {
    string bookID;
    cout << "Enter Book ID to delete: ";
    cin >> bookID;
    auto it = find_if(library.begin(), library.end(), [&bookID](const Book &book) {
        return book.id == bookID;
    });
    if (it != library.end()) {
        library.erase(it, library.end());
        saveToFile();
        cout << "Book deleted successfully!\n";
    } else {
        cout << "Book not found!\n";
    }
}

void returnBook() {
    string userId, bookID;
    cout << "Enter User ID: ";
    cin >> userId;
    cout << "Enter Book ID to return: ";
    cin >> bookID;
    auto &borrowedBooks = users[userId].borrowedBooks;
    auto it = find(borrowedBooks.begin(), borrowedBooks.end(), bookID);
    if (it != borrowedBooks.end()) {
        borrowedBooks.erase(it);
        for (auto &book : library) {
            if (book.id == bookID) {
                book.quantity++;
                book.borrowed--;
                saveToFile();
                cout << "Book returned successfully!\n";
                return;
            }
        }
    }

    cout << "Book return failed.\n";
}

void displayUsers() {
    if (users.empty()) {
        cout << "No registered users.\n";
        return;
    }
    for (const auto &user : users) {
        user.second.display();
    }
}

void deleteUser() {
    string userId;
    cout << "Enter User ID to delete: ";
    cin >> userId;
    if (users.erase(userId)) {
        saveToFile();
        cout << "User deleted successfully!\n";
    } else {
        cout << "User not found.\n";
    }
}

void userBorrowHistory() {
    displayUsers();
}

void generateReport() {
    cout << "Library Report:\n";
    displayBooks();
    displayUsers();
}

void registerUser() {
    User newUser;
    cout << "Enter User ID: ";
    cin >> newUser.userId;
    cout << "Enter Name: ";
    cin.ignore();
    getline(cin, newUser.name);
    users[newUser.userId] = newUser;
    saveToFile();
    cout << "User registered successfully!\n";
}

void menu() {
    int choice;

    while (true) {
         cout<<"\n****************************************\n";
         cout<<" ***** Library Management System *****\n";
         cout<<"****************************************\n";
        cout << "\n\n1. Register User\n2. Add Book\n3. Display Books\n4. Borrow
Book\n5. Return Book\n6. User Borrow History\n7. Generate Report\n8. Delete
User\n9. Display User\n10.Delete Book\n11.Exit\n\nEnter choice: ";
        cin >> choice;
        if (choice == 11) return;
        switch (choice) {
            case 1: registerUser(); break;
            case 2: addBook(); break;
            case 3: displayBooks(); break;
            case 4: borrowBook(); break;
            case 5: returnBook(); break;
            case 6: userBorrowHistory(); break;
            case 7: generateReport(); break;
            case 8: deleteUser(); break;
            case 9: displayUsers(); break;
            case 10: deleteBook(); break;
            default: cout << "Invalid choice!\n";
        }
    }
}
```

OUTPUT



```
***************************************
 ***** Library Management System *****
***************************************


1. Register User
2. Add Book
3. Display Books
4. Borrow Book
5. Return Book
6. User Borrow History
7. Generate Report
8. Delete User
9. Display User
10.Delete Book
11.Exit

Enter choice:
```

Advantages

The Library Management System offers numerous benefits, making it an efficient and reliable solution for library operations:

- **Efficiency:** The system significantly reduces the manual workload by automating book management, user registration, and record-keeping.
- **Data Integrity:** Ensures data consistency and security by maintaining a structured file-based record system.
- **Error Reduction:** The automated process minimizes human errors in book issuance, returns, and inventory tracking.
- **Security:** Implements user authentication and role-based access control, ensuring only authorized personnel can modify library records.
- **Cost-Effective:** Provides a free and open-source alternative to expensive proprietary library management software.
- **Customization:** The system is flexible and can be expanded with additional features like GUI and database support in the future.

Future Plans

Future enhancements for the Library Management System include:

- **Graphical User Interface (GUI):** Transitioning from CLI to a user-friendly interface using Qt or GTK.
- **Database Integration:** Implementing MySQL or SQLite for structured and scalable data storage.
- **Barcode Scanner Integration:** Enabling quick book identification using barcode technology.
- **Automated Notifications:** Sending reminders for due dates and overdue books.
- **Multithreading Support:** Optimizing performance for handling multiple user operations simultaneously.

Limitations of Library Management System

Despite its advantages, the system has some limitations:

- CLI-Based Interface: Currently lacks a graphical user interface.
- Limited Scalability: File-based storage is less efficient for large-scale libraries.
- Manual Data Entry: Book and user records must be entered manually.
- Single-User Access: Does not support concurrent users accessing the system simultaneously.

Why we choose this project

We selected this project for multiple reasons, as it aligns with both educational and practical objectives:

- **Real-World Application:** Many libraries, especially small-scale ones, still rely on inefficient manual systems. This project offers a practical solution to digitize library operations.
- **Enhancing Programming Skills:** Developing this project allows us to apply and refine our understanding of C++, file handling, and object-oriented programming principles.
- **Scalability and Future Potential:** The system is designed to be modular, making it easier to expand with additional features such as GUI, barcode scanning, and online access in the future.
- **Promoting Digitalization:** As technology advances, traditional paper-based management systems are becoming obsolete. This project contributes to a more structured and efficient approach to library management.

This project serves as an excellent learning experience while providing a functional solution for library operations.

Conclusion

The Library Management System developed in C++ successfully automates key library functions such as book tracking, user management, and borrowing/return processes. By leveraging file handling and object-oriented principles, the system provides an efficient and error-free way to manage library resources.

The project demonstrates how programming can be used to solve real-world problems efficiently. Though it currently operates via a command-line interface, it lays a strong foundation for future improvements, such as GUI development and database integration, which will make it more robust and user-friendly.

Overall, this project highlights the importance of technology in optimizing traditional library management methods, making them more efficient, accessible, and scalable for future needs.