NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

MILTARY COLLEGE OF SIGNALS

Data Structures & Algorithms (CS-250)

OPEN ENDED LAB

Submitted by:	MUHAMMAD AHMAD SULTAN
CMS ID:	408709
RANK:	NC
COURSE:	BESE-28
SECTION:	C

Submitted to:	Mam Muntaha Noor

Dated: 08-01-2024

Catalog of Open Ended Lab

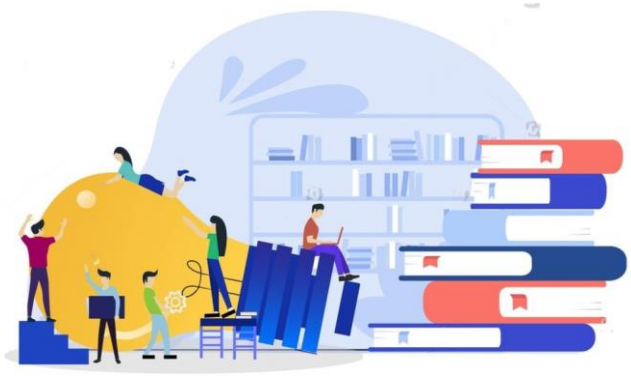
Table of Contents

DEPARTMENT OF COMPUTER SOFTWARE ENGINEERING

Military College Of Signals

National University Of Sciences and Technology

www.mcs.nust.edu.pk



TITLE OF THE OPEN-ENDED LAB

BookTrack Nexus

Library Management System

A Comprehensive Library Management System in C++, unifying Data Structures & Algorithms, Object-Oriented Programming, and File Handling.

About me

Allow me to introduce myself. I am *Muhammad Ahmad Sultan*, a highly motivated second-year student at the ***Military College of Signals***, currently pursuing a **Bachelor's degree in Software Engineering**. With unwavering determination, I am committed to achieving my goals and making significant strides in my academic journey.

Muhammad Ahmad Sultan

Developer

Abstract:

The Library Management System is a comprehensive software application designed to efficiently manage the operations of a library. It incorporates data structures, file handling, and object-oriented programming concepts in C++. The system allows administrators to add, search, and organize books and users, while users can borrow and return books. This document provides a detailed overview of the Library Management System, covering its history, objectives, scope, functionality, implementation details, input/output specifications, file handling concepts, and the use of various data structures and algorithms.

History of Library Management Systems:

Library Management Systems have evolved over the years, transitioning from manual card catalogs to digital databases. The shift to automation has significantly improved the efficiency of library operations, enabling easy cataloging, tracking, and management of library resources. The advent of computer technology has facilitated the development of sophisticated Library Management Systems, streamlining tasks for librarians and enhancing the user experience.

Objective:

The primary objective of the Library Management System is to provide a user-friendly platform for efficiently managing library resources, user interactions, and transactions. The system aims to automate tasks such as book and user management, borrowing and returning books, maintaining transaction history, and implementing search and sorting functionalities.

Scope:

The scope of the Library Management System encompasses the entire lifecycle of library resources and user interactions. It includes functionalities for administrators to add, search, and organize books and users, as well as for users to borrow and return books. The system also maintains a transaction history for accountability. The implementation of search and sorting operations enhances the overall user experience.

▪ Functionality &Implementation:

The provided C++ code implements a Library Management System with a command-line interface. The system consists of various functionalities accessible through both an Admin Dashboard and a User Dashboard. Here's a detailed breakdown of the code's functionality:

- Main Loop: run() Function

➢ **Purpose:**

The `run()` function serves as the main loop, guiding users through the Admin and User Dashboards or allowing them to exit the system.

➢ **Functionality:**

Admin Dashboard Access:

If the user chooses option 1, they enter the Admin Dashboard, where they can perform administrative tasks.

The admin tasks include adding new books or users, searching for books, and listing books or users.

There's an additional option to list users who borrowed a specific book.

User Dashboard Access:

If the user chooses option 2, they enter the User Dashboard, where they can borrow or return books, search for books, or go back to the main menu.

Exit System:

If the user chooses option 3, the system exits.

```
//Front-End
void run() {

    int choice = menu();

    if (choice == 3)
        return;

    while (choice == 1 && !cin.fail()) {
        int admin_choice = admin_dashboard();

        if (admin_choice == 1)
            add_new_book();
        else if (admin_choice == 2)
            add_new_user();
        else if (admin_choice == 3)
            search_for_book();
        else if (admin_choice == 4)
            list_books_by_id();
        else if (admin_choice == 5)
            list_books_by_name();
        else if (admin_choice == 6)
            list_users_by_id();
        else if (admin_choice == 7)
            list_users_by_name();
        else if (admin_choice == 8)
            list_users_borrowed_specific_book();
        else if (admin_choice == 9) {
            run();
            choice = 3;
        } else
            break;
    }

    }
    while (choice == 2 && !cin.fail()) {
        int user_choice = user_dashboard();

        if (user_choice == 1)
            borrow_book();
        else if (user_choice == 2)
            return_book();
        else if (user_choice == 3)
            search_for_book();
        else if (user_choice == 4) {
            run();
            choice = 3;
        } else
            break;
    }
}
```

- Admin Dashboard:

➢ **Purpose:**

The Admin Dashboard provides administrators with various options to manage the library system.

- **Functionalities:**
  - **Add New Book:**

Admins can add a new book to the system, with input validations in place to prevent adding more books than allowed (in this case, more than 5).

  - **Add New User:**

Admins can add a new user to the system, with input validations to limit the number of users (in this case, more than 5).

  - **Search Books:**

Admins can search for books using a substring of the book name. The search is performed by reading the books from the file (`books.txt`).

  - **List Books Ordered by ID or Name:**

Admins can list books ordered by ID or name. The sorting is done using the `sort()` function from the `<algorithm>` library.

  - **List Users Ordered by ID or Name:**

Admins can list users ordered by ID or name. The sorting is done using the `sort()` function.

  - **List Users Borrowed a Specific Book:**

Admins can list users who have borrowed a specific book.

  - **Back to Main Menu:**

Admins can return to the main menu by choosing option 9.

  - **Exit System:**

Admins can exit the system by choosing option 10.

- User Dashboard:

- **Purpose:**

The User Dashboard provides users with options to interact with the library system.

- **Functionalities:**
  - **Borrow Book:**

Users can borrow a book. Input validations and checks ensure that the user doesn't exceed the allowed number of copies and that the book is available.

  - **Return Book:**

Users can return a book, updating the system's records accordingly.

- **Search Books:**

Users can search for books using a substring of the book name.

- **Back to Main Menu:**

Users can return to the main menu by choosing option 4.

- **Exit System:**

Users can exit the system by choosing option 5.

- Menus: menu(), admin_dashboard(), and user_dashboard()

- **Purpose:**

The menu-related functions handle user inputs, validate choices, and provide error handling.

- **Functionalities:**

- **menu():**

Displays the main menu and validates the user's choice.

Returns the user's choice.

- **admin_dashboard():**

Displays the Admin Dashboard menu and validates the admin's choice.

Returns the admin's choice.

- **user_dashboard():**

Displays the User Dashboard menu and validates the user's choice.

Returns the user's choice.

- **Error Handling:**

Input validations are in place to handle invalid choices, ensuring that the user or admin provides a valid input corresponding to the available options.

```
int menu() {
    int choice = -1;
    while (choice == -1) {
        cout << "\n1) Admin Dashboard\n"
                "2) User Dashboard\n"
                "3) Exit\n"
                "Enter your choice: ";
        cin >> choice;

        if (cin.fail())
            break;
        if (!(1 <= choice && choice <= 3)) {
            cout << "\nInvalid choice. Try again\n\n";
            choice = -1;
        }
    }
    return choice;
}
int admin_dashboard() {
    int choice = -1;
    while (choice == -1) {
        cout << "\n1) Add new book\n"
                "2) Add new user\n"
                "3) Search books\n"
                "4) List books ordered by id\n"
                "5) List books ordered by name\n"
                "6) List users ordered by id\n"
                "7) List users ordered by name\n"
                "8) List users borrowed a specific book\n"
                "9) Back\n"
                "10)Exit\n"
                "Enter your choice:";
        cin >> choice;
        if (cin.fail())
            break;

        if (!(1 <= choice && choice <= 10)) {
            cout << "\nInvalid choice. Try again!\n";
```

```
            choice = -1;
        }
    }

    return choice;
}
int user_dashboard() {
    int choice = -1;

    while (choice == -1) {
        cout << "\n1) Borrow book\n"
                "2) Return book\n"
                "3) Search books\n"
                "4) Back\n"
                "5) Exit\n"

                "Enter your choice:";
        cin >> choice;
        if (cin.fail())
            break;
        if (!(1 <= choice && choice <= 5)) {
            cout << "\ninvalid choice. Try again!\n";
            choice = -1;
        }
    }
    return choice;
}
```

Back-End (Data Management)

- *Data Structures:*

I. ***Book Operations Structure:*** book_operations

Attributes:

name: Represents the name of the book (string name).

id: Unique identifier for each book (int id).

available_copies: Number of copies available for borrowing (int available_copies).

borrowed_copies: Number of copies currently borrowed (int borrowed_copies).

Functions:

read(): Reads input to initialize attributes.

Referenced in **add_new_book()** function.

print(): Displays information about the book.

Referenced in **list_books_by_id()** and **list_books_by_name()** functions.

is_substring(): Checks if a given substring is present in the book name.

Used in the **search_for_book()** function.

borrowing(): Handles the process of borrowing a book.

Utilized in the **borrow_book()** function.

returning(): Handles the process of returning a borrowed book.

Used in the **return_book()** function.

```
struct book_operations {
    string name;
    int id;
    int available_copies;
    int borrowed_copies;

    book_operations() {
        name = " ";
        id = -1;
        available_copies = 0;
        borrowed_copies = 0;
    }
    void read() {

        cout << "\nEnter book Name: ";
        cin >> name;
        cout << "Enter book ID: ";
        cin >> id;
        cout << "Enter no.of book Copies:";
        cin >> available_copies;
    }
    void print() {
        cout << "\n -> Book name: " << name << " -> ID: " << id
                << " -> Quantity: " << available_copies << " copies."
                << " -> Total borrowed: " << borrowed_copies << " copies.";
        if (available_copies == borrowed_copies)
            cout << "(not available!)";
        cout << "\n";
    }
    bool is_substring(string &substring) {
    if (substring.size() > name.size())
        return false;

    for (int i = 0; i <= name.size() - substring.size(); ++i) {  // Fix loop condition
        bool is_match = true;



        for (int j = 0; j < substring.size() && is_match; ++j) {
            if (substring[j] != name[i + j])
                is_match = false;
        }

        if (is_match)
            return true;
    }

    return false;
}


    bool borrowing() {
        if (available_copies - borrowed_copies == 0)
            return false;
        ++borrowed_copies;
        return true;
    }
    void returning() {
        assert(borrowed_copies > 0);
        --borrowed_copies;
    }
};
```

2. *User Operations Structure:* user_operations

Attributes:

name: Represents the name of the user (string name).

id: Unique identifier for each user (int id).

borrowed_books_ids: Set containing unique identifiers of books borrowed by the user (set<int> borrowed_books_ids).

Functions:

read(): Reads input to initialize attributes.

Referenced in **add_new_user()** function.

print(): Displays information about the user.

Used in **list_users_by_id()** and **list_users_by_name()** functions.

borrow_copy(): Adds a borrowed book ID to the set.

Utilized in the **borrow_book()** function.

return_copy(): Removes a returned book ID from the set.

Referenced in the **return_book()** function.

is_borrowed(): Checks if a specific book is borrowed by the user.

```
struct user_operations {
    string name;
    int id;
    set<int> borrowed_books_ids;

    user_operations() {
        name = " ";
        id = -1;
    }

    void read() {
        cout << "\nEnter user Name:";
        cin >> name;
        cout << "Enter user ID:";
        cin >> id;
    }
    void print() {
        cout << "\n-> user name : " << name << " -> ID: " << id
                << " Borrowed books IDs: ";
        for (int book_id : borrowed_books_ids)
            cout << book_id << " ";
        if (borrowed_books_ids.empty())
            cout << "(No borrowed copies)";
        cout << "\n";
    }
```

```
    void borrow_copy(int &book_id) {
        borrowed_books_ids.insert(book_id);
    }
    void return_copy(int book_id) {
        //find the borrowed book id in the borrowed books set and returning it by removing from the set
        auto it = borrowed_books_ids.find(book_id);

        if (it != borrowed_books_ids.end())
            borrowed_books_ids.erase(it);
        else
            cout << "\nUser " << name << " never borrowed a book with id" << book_id
                << "\n";
    }
    bool is_borrowed(int book_id) const {
    auto it = borrowed_books_ids.find(book_id);
    return it != borrowed_books_ids.end();
}

};
```

3. *Library System Structure:* library_system

Attributes:

books: Vector storing instances of book_operations (vector<book_operations> books).

users: Vector storing instances of user_operations (vector<user_operations> users).

Functions:

add_new_book(): Adds a new book to the system.

Calls **book_operations::read()** to input book details.

list_books_by_id(): Lists books ordered by ID.

Calls **book_operations::print()** for each book.

list_books_by_name(): Lists books ordered by name.

Calls **book_operations::print()** for each book.

search_for_book(): Searches for a book by name.

Uses **book_operations::is_substring()** for searching.

add_new_user(): Adds a new user to the system.

Calls **user_operations::read()** to input user details.

list_users_by_id(): Lists users ordered by ID.

Calls **user_operations::print()** for each user.

list_users_by_name(): Lists users ordered by name.

Calls **user_operations::print()** for each user.

list_users_borrowed_specific_book(): Lists users who borrowed a specific book.

Utilizes **user_operations::is_borrowed()** for book tracking.

borrow_book(): Handles the process of borrowing a book.

Calls **book_operations::borrowing()** and **user_operations::borrow_copy().**

return_book(): Handles the process of returning a borrowed book.

Calls **book_operations::returning()** and **user_operations::return_copy().**

find_user_id_by_user_name(): Finds the user ID based on the user's name.

find_book_id_by_book_name(): Finds the book ID based on the book's name.

```
struct library_system {

    vector<book_operations> books;
    vector<user_operations> users;

    library_system() {

    }
```

Additional Data Structures:

Vector (<vector>):

Used to store dynamic arrays of book and user data (books and users vectors).

Enables the dynamic management of book and user information.

Set (<set>):

Used in user_operations to store unique identifiers of books borrowed by a user (borrowed_books_ids set).

Ensures that each book ID is unique within the set.

```
#include <vector>
#include <set>
```

Additional Data Structures:

Sorting Functions: `compare_books_by_id`, `compare_books_by_name`, `compare_users_by_id`, `compare_users_by_name`

Purpose:

Custom sorting functions to facilitate sorting of books and users

- *Searching and Sorting Details in the Library Management System*

Searching Functionality:

➢ Book Search:

The book search functionality allows users to find books by entering a substring of the book name.

This functionality is implemented in the search_for_book() function.

The user provides a substring, and the program searches through the existing book records in the file (books.txt).

If a match is found, the book name and ID are displayed.

```
void search_for_book() {
    string substring;
    cout << "\nEnter book name: ";
    cin >> substring;

    int cnt = 0;
    ifstream file("books.txt");
    if (file.is_open()) {
        book_operations book_item;
        while (file >> book_item.name >> book_item.id >> book_item.available_copies >> book_item.borrowed_copies) {
            if (book_item.is_substring(substring)) {
                cout << "\n" << book_item.name << " " << book_item.id << "\n";
                cnt++;
            }
        }
        file.close();
    } else {
        cerr << "Error: Unable to open file for reading (books)." << endl;
        return;
    }

    if (!cnt)
        cout << "\nNo book with such name\n";
}
```

➢ User Search:

The find_user_id_by_user_name() function is used internally to locate a user by their name.

```
int find_user_id_by_user_name(string user_name) {
    for (int i = 0; i < (int) users.size(); ++i)
        if (user_name == users[i].name)
            return i;

    return -1;
}
```

Sorting Functionality:

I. Sorting Books:

Sorting by ID:

The list_books_by_id() function sorts books by their IDs using the compare_books_by_id() comparison function.

The sort() function from the <algorithm> library is utilized.

```
void library_system::list_books_by_id() {
   if (books.empty()) {
      cout << "\nNo books added in the system yet\n";
      return;
   }

   sort(books.begin(), books.end(), compare_books_by_id);

   for (book_operations book_item : books)
      book_item.print();
   cout << "\n";
}
```

Sorting by Name:

The list_books_by_name() function sorts books alphabetically by name using the compare_books_by_name() comparison function.

Again, the sort() function is employed.

```
void library_system::list_books_by_name() {
   if (books.empty()) {
      cout << "\nNo books added in the system yet\n";
      return;
   }
   sort(books.begin(), books.end(), compare_books_by_name);

   for (book_operations book_item : books)
      book_item.print(); cout << "\n"; }
```

II. Sorting Users:

Sorting by ID:

The list_users_by_id() function sorts users by their IDs using the compare_users_by_id() comparison function.

```
void library_system::list_users_by_id() {
    if (users.empty()) {
        cout << "\nNo users added in the system yet\n";
        return;
    }
    sort(users.begin(), users.end(), compare_users_by_id);

    for (user_operations &user_item : users)
        user_item.print();
}
```

Sorting by Name:

The list_users_by_name() function sorts users alphabetically by name using the compare_users_by_name() comparison function.

```
void library_system::list_users_by_name() {
    if (users.empty()) {
        cout << "\nNo users added in the system yet\n";
        return;
    }
    sort(users.begin(), users.end(), compare_users_by_name);

    for (user_operations &user_item : users)
        user_item.print(); }
```

III. Sorting Users Borrowing a Specific Book:

The list_users_borrowed_specific_book() function lists users who have borrowed a specific book, sorted by their names.

```
void list_users_borrowed_specific_book() {
    string book_name;
    cout << "\nEnter book name: ";
    cin >> book_name;

    int book_id = find_book_id_by_book_name(book_name);

    if (book_id == -1) {
        cout << "\nInvalid book name\n";
        return;
    }

    int book_id_container = books[book_id].id;

    vector<user_operations> users_borrowed_book;

    for (const user_operations &user_item : users) {
        if (user_item.is_borrowed(book_id_container)) {
            users_borrowed_book.push_back(user_item);
        }
    }

    if (users_borrowed_book.empty()) {
        cout << "\nNo users have borrowed copies of this book\n";
        return;
    }

    cout << "\nUsers who borrowed copies of " << book_name << " (Book ID: " << book_id_container << "):\n";

    for (const user_operations &user_item : users_borrowed_book) {
        cout << "\n" << user_item.name << " " << user_item.id << "\n";
    }
}
```

- *C++ Libraries Used:*

1. `<iostream>`: Input/Output Operations

Purpose:

Facilitates basic input and output operations.

Used for displaying information to the user and obtaining input from the user.

Functions in Use:

`cout`: Standard output stream used for displaying information.

`cin`: Standard input stream used for obtaining user input.

2. `<fstream>`: File Handling Operations

Purpose:

Provides facilities for file input and output operations.

Functions in Use:

`ifstream`: Input file stream, used for reading data from files.

`ofstream`: Output file stream, used for writing data to files.

3. `<algorithm>`**: Sorting Algorithms**

Purpose:

Offers a collection of functions especially designed to be used with ranges of elements.

Functions in Use:

`sort()`: Sorts the elements in the given range, here used for sorting books and users.

4. `<cassert>`**: Assertions for Error Checking**

Purpose:

Provides assert macro for debugging purposes.

Used to check conditions that the program assumes to be true.

Functions in Use:

`assert()`: Evaluates the specified expression and, if it is false, calls **`abort()`** to terminate the program.

5. `<vector>`**: Dynamic Array for Storing Book and User Data**

Purpose:

Implements a dynamic array that can grow or shrink in size.

Used for storing instances of **`book_operations`** and **`user_operations`**.

Functions in Use:

`push_back()`: Adds an element to the end of the vector.

6. `<set>`**: Data Structure for Storing Unique User-Borrowed Book IDs**

Purpose:

Represents a set of unique elements.

Used for storing book IDs that a user has borrowed to ensure uniqueness.

Functions in Use:

`insert()`: Inserts an element into the set.

7. `<sstream>`**: String Stream for Parsing File Data**

Purpose:

Provides a way to manipulate strings as if they were input/output streams.

Used for parsing data from files, especially in the **`load_users_from_file`** function.

Classes in Use:

`istringstream`: Input string stream, used for extracting formatted data from strings.

- *Object-Oriented Programming (OOP) Concepts:*

1. Encapsulation:

Definition:

Encapsulation is a fundamental OOP concept that involves bundling data (attributes) and the methods (functions) that operate on the data into a single unit, known as a class.

Application in Code:

The `book_operations` and `user_operations` structures encapsulate related data (name, ID, etc.) and operations (borrowing, returning, etc.) related to books and users, respectively.

The `library_system` structure encapsulates both book and user vectors, providing a higher-level abstraction of the library system.

2. Abstraction:

Definition:

Abstraction involves simplifying complex systems by modeling classes based on essential features and ignoring unnecessary details.

Application in Code:

Functions like `borrow_book()`, `return_book()`, etc., provide a level of abstraction by hiding the underlying complexity of book and user operations.

Users interact with these abstracted functions without needing to understand the internal details of the borrowing or returning processes.

3. Modularity:

Definition:

Modularity is the concept of breaking down a program into separate, independently replaceable, and upgradeable modules or functions.

Application in Code:

The code is organized into modular functions like `add_new_book()`, `list_books_by_id()`, etc., which enhances readability and maintainability.

Each function handles a specific aspect of the system's functionality, making it easier to understand, modify, or extend.

4. Inheritance:

Definition:

Inheritance allows a class (subclass or derived class) to inherit properties and behaviors from another class (superclass or base class).

Application in Code:

Although not explicitly demonstrated in the current code, the structure allows for future extensions.

Additional features or functionalities could be added by creating new classes that inherit properties from existing ones, promoting code reuse and extensibility.

- *File Handling Concepts:*

File handling in C++ involves using the <fstream> library, which provides classes for reading from and writing to files. The two main classes are ifstream (for reading) and ofstream (for writing), both derived from the fstream class. To read from a file, you open it using ifstream, and to write to a file, you use ofstream. Operations like reading and writing are performed through overloaded << and >> operators. The basic steps include opening a file, performing operations, and closing the file. Error handling, such as checking if a file is open, is essential for robust file handling in C++.

1. Loading Books Data from File (load_books_from_file()):

```
void load_books_from_file(const string &filename) {
    ifstream file(filename);
    if (file.is_open()) {
        books.clear(); // Clear existing data

        while (file >> ws && !file.eof()) {  // Check for end-of-file
            book_operations book;
            file >> book.name >> book.id >> book.available_copies >> book.borrowed_copies;

            if (!file.fail()) {
                books.push_back(book);
            }
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for reading (books)." << endl;
    }
}
```

Explanation:

The function load_books_from_file() reads book data from a specified file into the books vector.

It uses an ifstream object (file) to open and read from the file.

The loop continues reading until the end of the file is reached.

For each book entry in the file, a book_operations instance is created and populated with data.

The book is then added to the books vector.

2. Saving Books Data to File (save_books_to_file()):

```
// Function to save books data to file
void save_books_to_file(const string &filename) {
    ofstream file(filename);
    if (file.is_open()) {
        for (const book_operations &book : books) {
            file << book.name << " " << book.id << " " << book.available_copies << " " << book.borrowed_copies << endl;
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for writing (books)." << endl;
    }
}
```

Explanation:

The function save_books_to_file() writes book data from the books vector to a specified file.

It uses an ofstream object (file) to open and write to the file.

For each book in the books vector, its attributes are written to the file in a formatted manner.

The file is then closed.

3. Loading Users Data from File (load_users_from_file()):

```
void load_users_from_file(const string &filename) {
    ifstream file(filename);
    if (file.is_open()) {
        users.clear(); // Clear existing data

        string line;
        while (getline(file, line)) {
            istringstream iss(line);
            user_operations user;
            iss >> user.name >> user.id;

            int book_id;
            while (iss >> book_id) {
                user.borrowed_books_ids.insert(book_id);
            }

            users.push_back(user);
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for reading (users)." << endl;
    }
}
```

Explanation:

The function load_users_from_file() reads user data from a specified file into the users vector.

It uses an ifstream object (file) to open and read from the file.

The function reads each line of the file, where each line represents a user's data.

For each user entry, a user_operations instance is created and populated with data.

The set borrowed_books_ids is also populated with book IDs.

The user is then added to the users vector.

4. Saving Users Data to File (save_users_to_file()):

```
// Function to save users data to file
void save_users_to_file(const string &filename) {
    ofstream file(filename, ios::trunc); // Use ios::trunc to overwrite the file
    if (file.is_open()) {
        for (const user_operations &user : users) {
            file << user.name << " " << user.id << " ";

            for (int book_id : user.borrowed_books_ids) {
                file << book_id << " ";
            }

            file << endl;
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for writing (users)." << endl;
    }
}
```

Explanation:

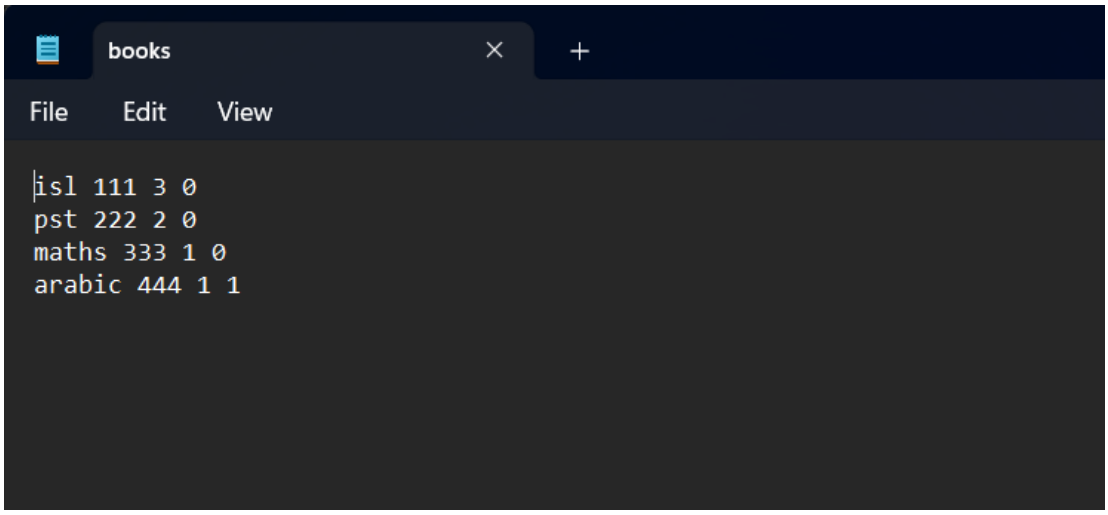
The function save_users_to_file() writes user data from the users vector to a specified file.

It uses an ofstream object (file) to open and write to the file, using ios::trunc to overwrite the existing file.

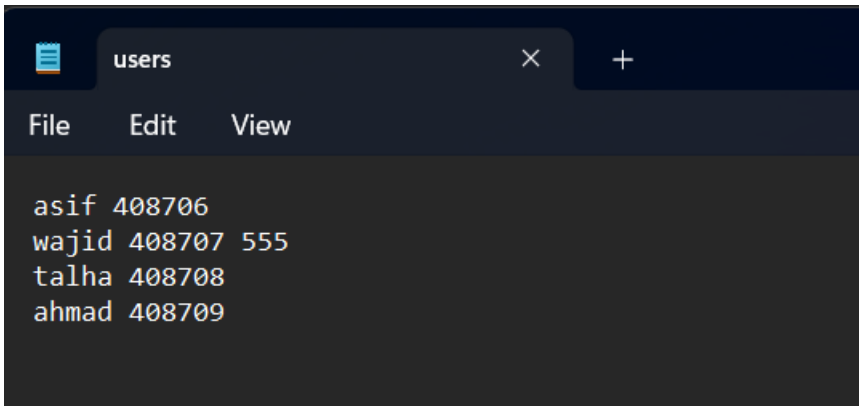
For each user in the users vector, their attributes and borrowed book IDs are written to the file in a formatted manner.

The file is then closed.

Books.txt :



Users.txt :



- ## Execution & Output:

- `Initialization:`

The program begins by creating an instance of the `library_system` class.

The data is loaded from the existing files, namely "books.txt" and "users.txt," using the `load_books_from_file` and `load_users_from_file` functions.

- `Main Loop (run function):`

The program enters the main loop, where users are presented with options to access the Admin Dashboard, User Dashboard, or exit the system.

- `Admin Dashboard:`

If the Admin Dashboard is selected, the program enters the admin loop (`admin_dashboard` function).

Admins can perform various operations:

Add new books or users (`add_new_book` and `add_new_user` functions).

Search for books (`search_for_book` function).

List books or users ordered by ID or name (`list_books_by_id`, `list_books_by_name`, `list_users_by_id`, and `list_users_by_name` functions).

List users who borrowed a specific book (`list_users_borrowed_specific_book` function).

Admins can navigate back or exit the program.

- `User Dashboard:`

If the User Dashboard is selected, the program enters the user loop (`user_dashboard` function).

Users can:

Borrow books (`borrow_book` function).

Return books (`return_book` function).

Search for books.

Users can navigate back or exit the program.

- `Input Validation:`

Throughout the program, input validations are in place to handle invalid choices and provide appropriate messages.

- `Menu Functions (menu, admin_dashboard, user_dashboard):`

These functions handle user inputs, ensuring valid choices and providing error messages for invalid entries.

- `File Handling:`

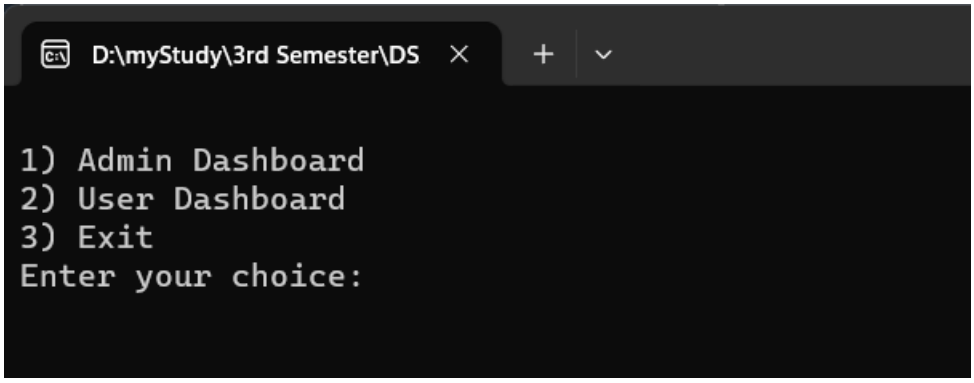
The `load_books_from_file`, `save_books_to_file`, `load_users_from_file`, and `save_users_to_file` functions manage the reading and writing of book and user data to and from files.

- `Console Output:`

Various print statements (`cout`) are used to display information about books, users, and system status on the console.

The user is informed about the success or failure of operations, such as adding a new book, borrowing a book, etc.

Display of Output:



```
Enter your choice: 1

1) Add new book
2) Add new user
3) Search books
4) List books ordered by id
5) List books ordered by name
6) List users ordered by id
7) List users ordered by name
8) List users borrowed a specific book
9) Back
10)Exit
Enter your choice:
```

```
Enter your choice:1

Enter book Name: dsa
Enter book ID: 666
Enter no.of book Copies:7
```

```
Enter your choice:2

Enter user Name:rashid
Enter user CMS ID:408703
```

```
Enter your choice:3

Enter book name: dsa

dsa 666
```

```
Enter your choice:4

-> Book name: isl -> ID: 111 -> Quantity: 3 copies. -> Total borrowed: 0 copies.

-> Book name: pst -> ID: 222 -> Quantity: 2 copies. -> Total borrowed: 0 copies.

-> Book name: maths -> ID: 333 -> Quantity: 1 copies. -> Total borrowed: 0 copies.

-> Book name: arabic -> ID: 444 -> Quantity: 1 copies. -> Total borrowed: 1 copies.(not available!)

-> Book name: dsa -> ID: 666 -> Quantity: 7 copies. -> Total borrowed: 0 copies.
```

```
Enter your choice:5

 -> Book name: arabic -> ID: 444 -> Quantity: 1 copies. -> Total borrowed: 1 copies.(not available!)

 -> Book name: dsa -> ID: 666 -> Quantity: 7 copies. -> Total borrowed: 0 copies.

 -> Book name: isl -> ID: 111 -> Quantity: 3 copies. -> Total borrowed: 0 copies.

 -> Book name: maths -> ID: 333 -> Quantity: 1 copies. -> Total borrowed: 0 copies.

 -> Book name: pst -> ID: 222 -> Quantity: 2 copies. -> Total borrowed: 0 copies.
```

```
Enter your choice:6

-> user name : rashid -> ID: 408703 Borrowed books IDs: (No borrowed copies)

-> user name : asif -> ID: 408706 Borrowed books IDs: (No borrowed copies)

-> user name : wajid -> ID: 408707 Borrowed books IDs: 555

-> user name : talha -> ID: 408708 Borrowed books IDs: (No borrowed copies)

-> user name : ahmad -> ID: 408709 Borrowed books IDs: (No borrowed copies)
```

```
Enter your choice:7

-> user name : ahmad -> ID: 408709 Borrowed books IDs: (No borrowed copies)

-> user name : asif -> ID: 408706 Borrowed books IDs: (No borrowed copies)

-> user name : rashid -> ID: 408703 Borrowed books IDs: (No borrowed copies)

-> user name : talha -> ID: 408708 Borrowed books IDs: (No borrowed copies)

-> user name : wajid -> ID: 408707 Borrowed books IDs: 555
```

```
Enter your choice:8

Enter book name: arabic

Users who borrowed copies of arabic (Book ID: 444):

wajid 408707

1) Add new book
2) Add new user
3) Search books
4) List books ordered by id
5) List books ordered by name
6) List users ordered by id
7) List users ordered by name
8) List users borrowed a specific book
9) Back
10)Exit
Enter your choice:9

1) Admin Dashboard
2) User Dashboard
3) Exit
Enter your choice:
```

```
Enter your choice: 2

1) Borrow book
2) Return book
3) Search books
4) Back
5) Exit
Enter your choice:
```

```
1) Borrow book
2) Return book
3) Search books
4) Back
5) Exit
Enter your choice:1
Enter user name: ahmad
Enter book name dsa
```

```
-> Book name: dsa -> ID: 666 -> Quantity: 7 copies. -> Total borrowed: 1 copies.
```

```
-> user name : ahmad -> ID: 408709 Borrowed books IDs: 666
```

```
Enter your choice: 2

1) Borrow book
2) Return book
3) Search books
4) Back
5) Exit
Enter your choice:2
Enter user name: ahmad
Enter book name dsa
```

```
-> Book name: dsa -> ID: 666 -> Quantity: 7 copies. -> Total borrowed: 0 copies.
```

```
-> user name : ahmad -> ID: 408709 Borrowed books IDs: (No borrowed copies)
```

```
1) Admin Dashboard
2) User Dashboard
3) Exit
Enter your choice: 3

--------------------------------
Process exited after 264.8 seconds with return value 0
Press any key to continue . . .
```

books

File Edit View

```
isl 111 3 0
pst 222 2 0
maths 333 1 0
arabic 444 1 1
dsa 666 7 0
```

users

File Edit View

```
asif 408706
wajid 408707 444
talha 408708
ahmad 408709
rashid 408703
```

- ## Potential Questions:

I. How does your Library Management System handle concurrent access to shared data, ensuring data integrity and consistency in a multi-user environment?

II. Can you explain the mechanisms in place for extending the functionality of your system, accommodating new features or modifications to existing ones without disrupting the current codebase?

III. How does your system handle and recover from potential errors during file operations, ensuring the reliability and robustness of the data storage and retrieval processes?

IV. Describe the security measures implemented in your Library Management System to protect user and book information, especially in the context of user authentication and authorization.

▪ Source Code :

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <cassert>
#include <vector>
#include <set>
#include <sstream>

using namespace std;

struct book_operations {
        string name;
        int id;
        int available_copies;
        int borrowed_copies;

        book_operations() {
                name = " ";
                id = -1;
                available_copies = 0;
                borrowed_copies = 0;
        }
        void read() {

                cout << "\nEnter book Name: ";
                cin >> name;
                cout << "Enter book ID: ";
                cin >> id;
                cout << "Enter no.of book Copies:";
                cin >> available_copies;
        }
        void print() {
                cout << "\n -> Book name: " << name << "
-> ID: " << id
                                << " -> Quantity: "
<< available_copies << " copies."
                                << " -> Total
borrowed: " << borrowed_copies << " copies.";
                if (available_copies == borrowed_copies)
                        cout << "(not available!)";
                cout << "\n";
        }
        bool is_substring(string &substring) {
  if (substring.size() > name.size())
    return false;

  for (int i = 0; i <= name.size() - substring.size(); ++i) {  // Fix
loop condition
    bool is_match = true;

    for (int j = 0; j < substring.size() && is_match; ++j) {
      if (substring[j] != name[i + j])
        is_match = false;
    }

    if (is_match)
      return true;
  }

  return false;
}

        bool borrowing() {
                if (available_copies - borrowed_copies ==
0)
                        return false;
                ++borrowed_copies;
                return true;
        }
        void returning() {
                assert(borrowed_copies > 0);
```

```
                --borrowed_copies;

        }

};


bool compare_books_by_id(book_operations &item1,
book_operations &item2) {

        return item1.id < item2.id;

}

bool compare_books_by_name(book_operations &item1,
book_operations &item2) {

        return item1.name < item2.name;

}


struct user_operations {

        string name;

        int id;

        set<int> borrowed_books_ids;


        user_operations() {

                name = " ";

                id = -1;

        }


        void read() {

                cout << "\nEnter user Name:";

                cin >> name;

                cout << "Enter user CMS ID:";

                cin >> id;

        }

        void print() {

                cout << "\n-> user name : " << name << "
-> ID: " << id

                                << " Borrowed
books IDs: ";

                for (int book_id : borrowed_books_ids)

                        cout << book_id << " ";

                if (borrowed_books_ids.empty())

                        cout << "(No borrowed
copies)";

                cout << "\n";

        }

        void borrow_copy(int &book_id) {

                borrowed_books_ids.insert(book_id);

        }

        void return_copy(int book_id) {

                //find the borrowed book id in the
borrowed books set and returning it by removing from the set

                auto it =
borrowed_books_ids.find(book_id);


                if (it != borrowed_books_ids.end())

                        borrowed_books_ids.erase(it);

                else

                        cout << "\nUser " << name <<
" never borrowed a book with id" << book_id

                                << "\n";

        }

        bool is_borrowed(int book_id) const {

   auto it = borrowed_books_ids.find(book_id);

   return it != borrowed_books_ids.end();

}


};


bool compare_users_by_id(user_operations &item1,
user_operations &item2) {

        return item1.id < item2.id;

}

bool compare_users_by_name(user_operations &item1,
user_operations &item2) {

        return item1.name < item2.name;

}


struct library_system {


        vector<book_operations> books;

        vector<user_operations> users;


        library_system() {
```

```
	}

//Front-End
	void run() {

		int choice = menu();

		if (choice == 3)
			return;

		while (choice == 1 && !cin.fail()) {
			int admin_choice =
admin_dashboard();

			if (admin_choice == 1)
				add_new_book();
			else if (admin_choice == 2)
				add_new_user();
			else if (admin_choice == 3)
				search_for_book();
			else if (admin_choice == 4)
				list_books_by_id();
			else if (admin_choice == 5)

	list_books_by_name();
			else if (admin_choice == 6)
				list_users_by_id();
			else if (admin_choice == 7)

	list_users_by_name();
			else if (admin_choice == 8)

	list_users_borrowed_specific_book();
			else if (admin_choice == 9) {
				run();
				choice = 3;
			} else
				break;
		}
		while (choice == 2 && !cin.fail()) {
			int user_choice =
user_dashboard();

			if (user_choice == 1)
				borrow_book();
			else if (user_choice == 2)
				return_book();
			else if (user_choice == 3)
				search_for_book();
			else if (user_choice == 4) {
				run();
				choice = 3;
			} else
				break;
		}
	}
	int menu() {
		int choice = -1;
		while (choice == -1) {
			cout << "\n1) Admin
Dashboard\n"
				"2) User
Dashboard\n"
				"3)
Exit\n"
				"Enter
your choice: ";
			cin >> choice;

			if (cin.fail())
				break;
			if (!(1 <= choice && choice <=
3)) {
				cout << "\nInvalid
choice. Try again\n\n";
				choice = -1;
			}
		}
		return choice;
```

```
}
int admin_dashboard() {
	int choice = -1;
	while (choice == -1) {
		cout << "\n1) Add new book\n"
			"2) Add new user\n"
			"3) Search books\n"
			"4) List books ordered by id\n"
			"5) List books ordered by name\n"
			"6) List users ordered by id\n"
			"7) List users ordered by name\n"
			"8) List users borrowed a specific book\n"
			"9) Back\n"

		"10)Exit\n"
			"Enter your choice:";
		cin >> choice;
		if (cin.fail())
			break;

		if (!(1 <= choice && choice <= 10)) {
			cout << "\nInvalid choice. Try again!\n";
			choice = -1;
		}
	}

	return choice;
}
int user_dashboard() {
	int choice = -1;
	while (choice == -1) {
		cout << "\n1) Borrow book\n"
			"2) Return book\n"
			"3) Search books\n"
			"4) Back\n"
			"5) Exit\n"

		"Enter your choice:";
		cin >> choice;
		if (cin.fail())
			break;
		if (!(1 <= choice && choice <= 5)) {
			cout << "\ninvalid choice. Try again!\n";
			choice = -1;
		}
	}
	return choice;
}

//Back-End
//Admin options: Book operations
bool add_new_book() {
	if (books.size() >= 5) {
		cout << "\nYou are Not allowed to add more than " << books.size()
		<< " books in the system. Remove some books\n";
		return false;
	}
	book_operations book_item;
	book_item.read();
	books.push_back(book_item);
	return true;
}
```

```
void list_books_by_id() {
        if (books.empty()) {
                cout << "\nNo books added in the system yet\n";
                return;
        }

        sort(books.begin(), books.end(), compare_books_by_id);

        for (book_operations book_item : books)
                book_item.print();
        cout << "\n";

}
void list_books_by_name() {
        if (books.empty()) {
                cout << "\nNo books added in the system yet\n";
                return;
        }
        sort(books.begin(), books.end(), compare_books_by_name);

        for (book_operations book_item : books)
                book_item.print();
        cout << "\n";
}
void search_for_book() {
  string substring;
  cout << "\nEnter book name: ";
  cin >> substring;

  int cnt = 0;
  ifstream file("books.txt");
  if (file.is_open()) {
    book_operations book_item;
    while (file >> book_item.name >> book_item.id >> book_item.available_copies >> book_item.borrowed_copies) {
      if (book_item.is_substring(substring)) {
        cout << "\n" << book_item.name << " " << book_item.id << "\n";
        cnt++;
      }
    }
    file.close();
  } else {
    cerr << "Error: Unable to open file for reading (books)." << endl;
    return;
  }

  if (!cnt)
    cout << "\nNo book with such name\n";
}

//Admin options: Users operations
bool add_new_user() {
        if (users.size() >= 5) {
                cout << "\nYou are Not allowed to add more than " << users.size()
                << " users in the system. Remove some users\n";
                return false;
        }
        user_operations user_item;
        user_item.read();
        users.push_back(user_item);
        return true;
}
void list_users_by_id() {
        if (users.empty()) {
                cout << "\nNo users added in the system yet\n";
                return;
        }
        sort(users.begin(), users.end(), compare_users_by_id);
```

```
                for (user_operations &user_item : users)
                        user_item.print();
        }
        void list_users_by_name() {
                if (users.empty()) {
                        cout << "\nNo users added in
the system yet\n";
                        return;
                }
                sort(users.begin(), users.end(),
compare_users_by_name);

                for (user_operations &user_item : users)
                        user_item.print();
        }
        void list_users_borrowed_specific_book() {
  string book_name;
  cout << "\nEnter book name: ";
  cin >> book_name;

  int book_id = find_book_id_by_book_name(book_name);

  if (book_id == -1) {
    cout << "\nInvalid book name\n";
    return;
  }

  int book_id_container = books[book_id].id;

  vector<user_operations> users_borrowed_book;

  for (const user_operations &user_item : users) {
    if (user_item.is_borrowed(book_id_container)) {
      users_borrowed_book.push_back(user_item);
    }
  }

  if (users_borrowed_book.empty()) {
    cout << "\nNo users have borrowed copies of this book\n";
    return;
  }

  cout << "\nUsers who borrowed copies of " << book_name <<
" (Book ID: " << book_id_container << "):\n";

  for (const user_operations &user_item :
users_borrowed_book) {
    cout << "\n" << user_item.name << " " << user_item.id <<
"\n";
  }
}

//Users options
        int find_user_id_by_user_name(string user_name) {
                for (int i = 0; i < (int) users.size(); ++i)
                        if (user_name ==
users[i].name)
                                return i;

                return -1;
        }
        int find_book_id_by_book_name(string book_name)
{
                for (int i = 0; i < (int) books.size(); ++i)
                        if (book_name ==
books[i].name)
                                return i;
                return -1;
        }
        bool read_user_name_and_book_name(int
&user_id, int &book_id,
                        int trails = 3) {
                //this function take user name and book
name and generate user id and book id
                string user_name;
                string book_name;
```

```
                while (trails--) {

                        //reading user name and
generating user id by its name

                        cout << "Enter user name: ";

                        cin >> user_name;


                        user_id =
find_user_id_by_user_name(user_name);


                        if (user_id == -1) {
                                cout << "\nInvalid
user name. try again\n";

                                continue;

                        }

                        //reading book name and
generating book id by its name

                        cout << "Enter book name ";

                        cin >> book_name;


                        book_id =
find_book_id_by_book_name(book_name);


                        if (book_id == -1) {

                                cout << "\nInvalid
book name. Try again\n";

                                continue;
                        }

                        return true;
                }

                cout << "\nYou have tried several times.
Try again later\n";

                return false;

        }

        void borrow_book() {

                int user_id, book_id;



        if(!read_user_name_and_book_name(user_id,
book_id))

                                return;
```

```
                if (!books[book_id].borrowing()) {

                        cout<< "\nNo more copies
available right now. borrow another book\n";

                        return;

                }


                int book_id_container =
books[book_id].id;


        users[user_id].borrow_copy(book_id_container);

        }

        void return_book() {

                int user_id, book_id;



        if(!read_user_name_and_book_name(user_id,
book_id))

                        return;


                books[book_id].returning();


                int book_id_container =
books[book_id].id;


        users[user_id].return_copy(book_id_container);

        }

        // Function to load books data from file

void load_books_from_file(const string &filename) {

  ifstream file(filename);

  if (file.is_open()) {

    books.clear(); // Clear existing data


    while (file >> ws && !file.eof()) {  // Check for end-of-file

      book_operations book;

      file >> book.name >> book.id >> book.available_copies
>> book.borrowed_copies;


      if (!file.fail()) {

        books.push_back(book);

      }

    }
```

```
        file.close();
    } else {
        cerr << "Error: Unable to open file for reading (books)." << endl;
    }
}

// Function to save books data to file
void save_books_to_file(const string &filename) {
    ofstream file(filename);
    if (file.is_open()) {
        for (const book_operations &book : books) {
            file << book.name << " " << book.id << " " << book.available_copies << " " << book.borrowed_copies << endl;
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for writing (books)." << endl;
    }
}



/// Function to load users data from file

void load_users_from_file(const string &filename) {
    ifstream file(filename);
    if (file.is_open()) {
        users.clear(); // Clear existing data

        string line;
        while (getline(file, line)) {
            istringstream iss(line);
            user_operations user;
            iss >> user.name >> user.id;
            int book_id;
            while (iss >> book_id) {
                user.borrowed_books_ids.insert(book_id);
            }

            users.push_back(user);
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for reading (users)." << endl;
    }
}

// Function to save users data to file
void save_users_to_file(const string &filename) {
    ofstream file(filename, ios::trunc); // Use ios::trunc to overwrite the file
    if (file.is_open()) {
        for (const user_operations &user : users) {
            file << user.name << " " << user.id << " ";

            for (int book_id : user.borrowed_books_ids) {
                file << book_id << " ";
            }

            file << endl;
        }

        file.close();
    } else {
        cerr << "Error: Unable to open file for writing (users)." << endl;
    }
}



};
```

```
int main() {
    library_system library;

    // Load data from files at the beginning
    library.load_books_from_file("books.txt");
    library.load_users_from_file("users.txt");

    // Run the library system
    library.run();

    // Save data to files before exiting
    library.save_books_to_file("books.txt");
    library.save_users_to_file("users.txt");

    return 0;
}
```

Conclusion:

In a nutshell, the Library Management System implemented in C++ with Data Structures and Object-Oriented Programming principles showcases a robust and organized approach to data management. Through adept use of structures like book_operations and user_operations, and the overarching library_system structure, the project encapsulates, abstracts, and modularizes functionalities. Leveraging C++ libraries like <iostream>, <fstream>, <algorithm>, <cassert>, <vector>, <set>, and <sstream>, it seamlessly integrates input/output operations, file handling, sorting algorithms, and dynamic data structures. The code's commitment to OOP concepts of encapsulation, abstraction, and modularity ensures readability, maintainability, and potential for future extensions. The implementation of sorting and searching functionalities, along with detailed file handling concepts, adds a layer of sophistication. In conclusion, this project not only addresses the functional requirements of a Library Management System but also exemplifies industry-standard practices, making it a noteworthy exemplar of software design and implementation.

Reference Citations:

https://www.youtube.com/

https://www.w3schools.com/

https://www.codecademy.com/

https://www.learncpp.com/

https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial/

THE END