Worksheet 1

Student Name: Komal Gupta **UID: 22BCA10262**
Branch: BCA **Section/Group: 3 ‘A’**
Semester: 5th **Submitted to: Ruchika Rana**
Subject Name: Computing Aptitude **Subject Code: 22CAP-306**

1. Aim/Overview of the practical:

The objective of this project is to design and implement a Library Management System using C++. This system helps in efficiently managing the operations of a library, including book inventory, user records, and transaction history for book issues and returns. By automating these tasks, the system reduces manual effort and potential errors, making it a streamlined solution for library management.

1.1 Introduction

The Library Management System (LMS) is a software application developed in C++ to simplify and automate various processes involved in a library's daily operations, such as book inventory management, user management, and transaction tracking. This project provides an efficient solution for managing books, handling user details, and maintaining accurate records of book issues and returns. This report details the requirements, design, implementation, and functionality of the Library Management System.

1.2 Overview

The Library Management System (LMS) project is developed using C++ to streamline the management of library resources. This system provides an organized approach for libraries to maintain records of books, track user information, and handle transactions for book issuance and returns. The LMS aims to eliminate the need for manual record-keeping, minimizing errors and enhancing the efficiency of library operations.

Through an interactive console interface, librarians can easily add new books, update book details, manage user accounts, and monitor the status of issued and returned books. The program’s structure is modular, allowing for future scalability and additional features. Overall, this system enhances library workflows by providing a reliable, fast, and user-friendly solution for handling daily library tasks.

2. Task to be done:

The main tasks for developing the Library Management System in C++ are as follows:

1. **Design the System Architecture**:
   - Define the core components and functionalities of the Library Management System.
   - Identify necessary modules, such as *Book Management*, *User Management*, *Transaction Management*, and *Report Generation*.
   - Establish a modular structure to allow future enhancements and ensure smooth operation.
2. **Implement Book Management**:
   - Develop functions to add new books with details such as title, author, publication year, and availability status.
   - Implement features to search for books by title, author, or book ID.
   - Create functions to update book information or delete books from the system.
3. **Implement User Management**:
   - Design functions for adding and managing user details (e.g., name, user ID, contact information).
   - Implement search and retrieval features to view user records.
   - Develop functionality to delete user records if required.
4. **Develop Book Issue and Return System**:
   - Create functions for issuing books, which updates the book's availability and records the transaction details (e.g., user ID, issue date).
   - Implement the return book functionality, which updates the book's status and records the return date.
   - Set rules for book issuance, such as restricting users from issuing books that are currently unavailable.
5. **Generate Reports and Views**:
   - Provide options to display lists of all books, users, and transaction histories.
   - Implement a summary view for librarians to see the status of all books (available vs. issued) and user activities.
   - Design user-friendly, clear, and organized output for each report.
6. **Testing and Debugging**:
   - Test individual functions and modules to ensure they work as expected.
   - Check for any bugs or errors in the implementation of tasks, such as adding books, managing users, and handling transactions.
   - Conduct integration testing to ensure that all components work seamlessly together.

3. Algorithm/Flowchart:

Algorithm for Library Management System

This algorithm describes the step-by-step logic for implementing the core functionalities of the Library Management System.

1. **Start the Program**:
    - Initialize the system with any necessary setup, such as defining data structures for storing books, users, and transactions.
    - Display the main menu with options to perform various tasks, such as managing books, managing users, issuing books, returning books, and viewing reports.
2. **Display Main Menu and Accept User Choice**:
    - Show the following options:
        1. **Manage Books**: Add, view, update, or delete book records.
        2. **Manage Users**: Add, view, or delete user records.
        3. **Issue Book**: Issue a book to a user.
        4. **Return Book**: Mark a book as returned by a user.
        5. **View Reports**: Show summaries of books, users, and transactions.
        6. **Exit**: End the program.
3. **Process User Selection**:
    - Based on the choice made by the user, perform the following tasks:
        - **Manage Books**:
            - *Add Book*: Prompt the user for book details (ID, title, author, publication date, etc.) and add it to the book list.
            - *View Books*: Display a list of all books, showing each book's details and availability status.
            - *Update Book*: Allow the user to modify information for a specific book based on its ID.
            - *Delete Book*: Remove a specific book from the records using its ID.
        - **Manage Users**:
            - *Add User*: Prompt for user information (user ID, name, contact details) and store it.
            - *View Users*: Display all user records.
            - *Delete User*: Remove a user's record from the database using their ID.
        - **Issue Book**:
            - Check if the book is available.

        - If available, prompt for user details and update the book's status to "Issued."
        - Record the transaction, including details like book ID, user ID, and issue date.
    - **Return Book**:
        - Prompt the user to enter the book ID for return.
        - Update the book's status to "Available" and record the return date.
        - Update the transaction history to mark the book as returned.
    - **View Reports**:
        - Display a summary of the books, showing the count of available and issued books.
        - Show a list of all users and their transaction history.
        - Display details of all issued books, including user details and due dates (if applicable).
4. **Return to Main Menu**:
    - After completing each task, return to the main menu.
    - Allow the user to choose another operation or exit.
5. **Exit Program**:
    - End the program when the user selects the "Exit" option.

4. Dataset:

This dataset records each transaction involving book issuance and return.

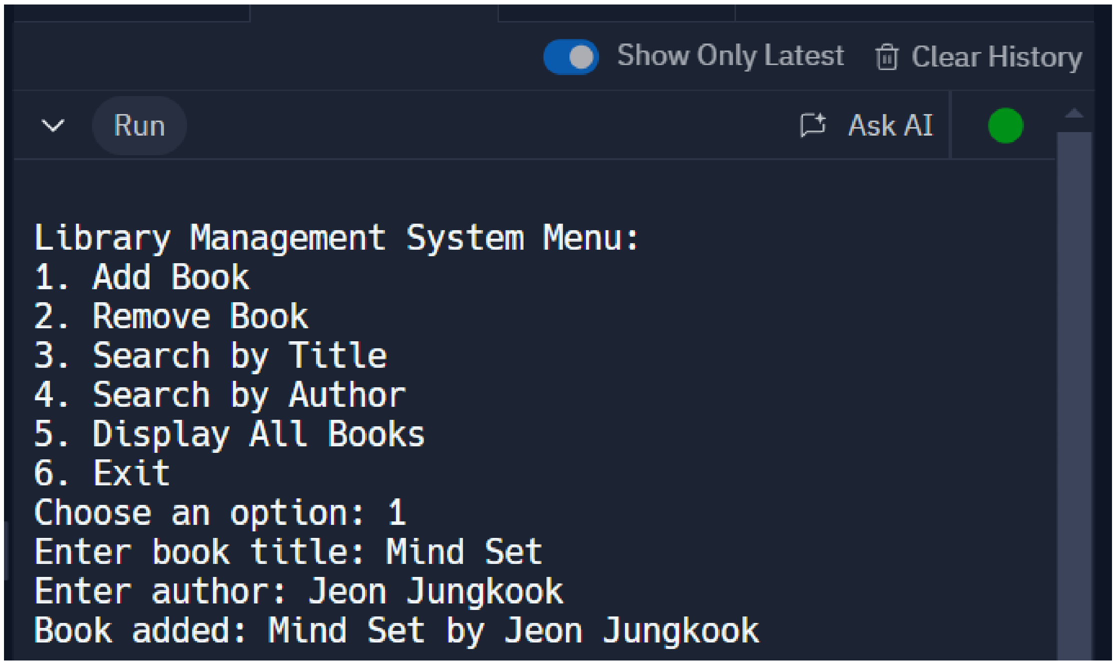
- **Attributes:**
    - **Transaction ID**: Unique identifier for each transaction.
    - **User ID**: ID of the user involved in the transaction.
    - **Book ID**: ID of the book being issued or returned.
    - **Transaction Type**: Type of transaction (Issue/Return).
    - **Transaction Date**: Date of the transaction.
    - **Due Date**: Date by which the book must be returned (for issuance).

- **Return Date**: Date the book was returned (for returns).

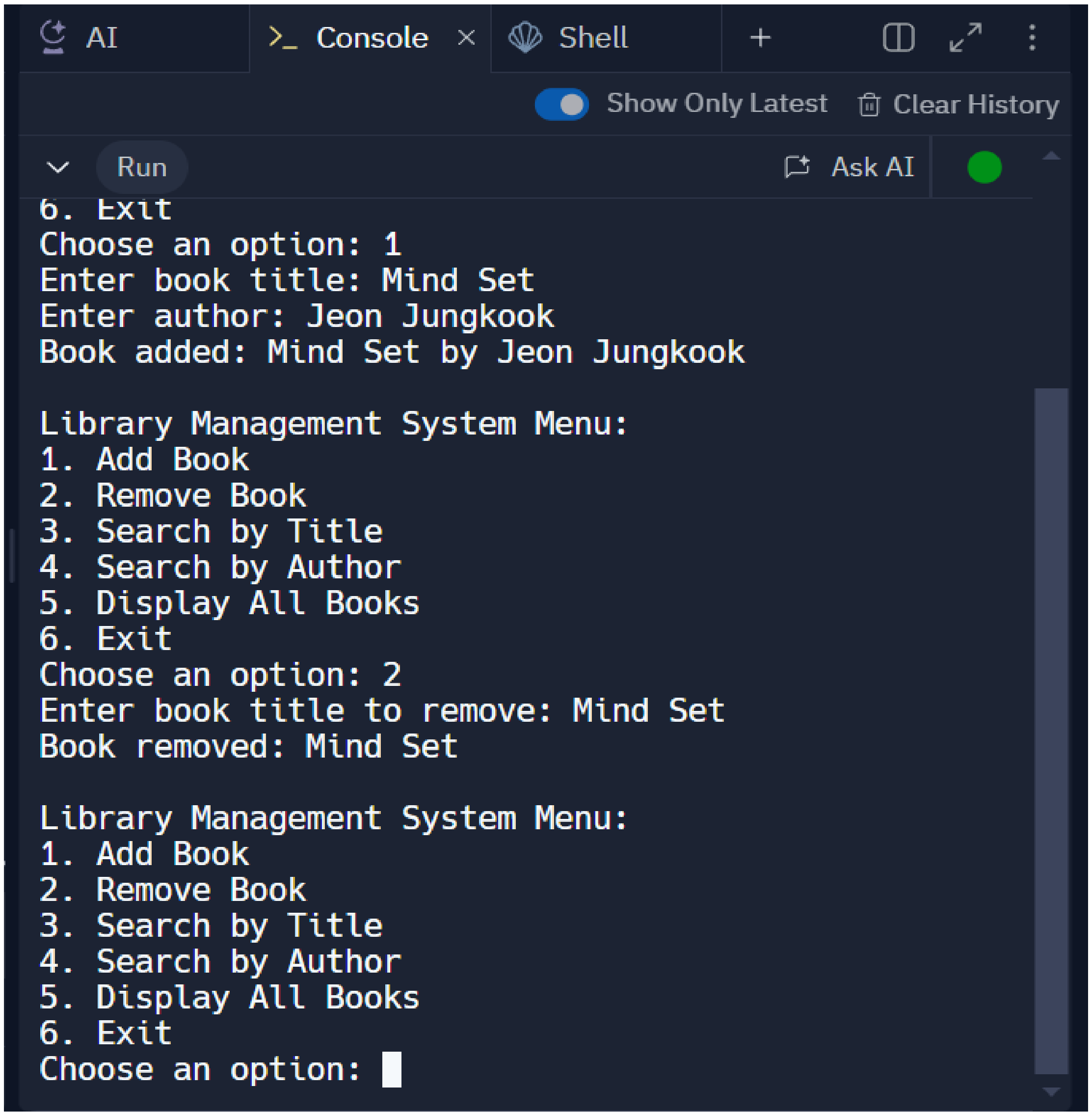
5. Code for experiment/practical:

```
#include<iostream>
#include<string>
#include<unordered_map>

struct Book {
    std::string title;
    std::string author;
    Book* next; // Pointer to the next book in the linked list

    Book(std::string t, std::string a) : title(t), author(a), next(nullptr) {}
};

class Library {
private:
    Book* head; // Head of the linked list
    std::unordered_map<std::string, Book*> titleMap; // Hash table for quick search by
title
    std::unordered_map<std::string, Book*> authorMap; // Hash table for quick search by
author

public:
    Library() : head(nullptr) {}

    void addBook(const std::string& title, const std::string& author) {
        Book* newBook = new Book(title, author);
        newBook->next = head; // Add to the front of the linked list
        head = newBook;
 // Update hash tables
        titleMap[title] = newBook;
        authorMap[author] = newBook;

        std::cout << "Book added: " << title << " by " << author << std::endl;
    }

    void removeBook(const std::string& title) {
        Book* current = head;
        Book* previous = nullptr;

        while (current != nullptr) {
            if (current->title == title) {
                if (previous == nullptr) { // Removing the head
                    head = current->next;
                } else {
                    previous->next = current->next;
                }

                // Remove from hash tables
                titleMap.erase(current->title);
                authorMap.erase(current->author); // This is a simplification, as the
author may have other books

                delete current; // Free memory
                std::cout << "Book removed: " << title << std::endl;
                return;
 }
            previous = current;
            current = current->next;
        }

        std::cout << "Book not found: " << title << std::endl;
    }

    Book* searchByTitle(const std::string& title) {
        if (titleMap.find(title) != titleMap.end()) {
            return titleMap[title];
        }
        return nullptr;
    }

    void searchByAuthor(const std::string& author) {
        if (authorMap.find(author) != authorMap.end()) {
            std::cout << "Books by " << author << ": " << std::endl;
            Book* current = authorMap[author];
            while (current != nullptr) {
                if (current->author == author) {
                    std::cout << "- " << current->title << std::endl;
                }
                current = current->next; // This is a simplification; ideally, we would
store all titles
            }
        } else {
 std::cout << "No books found by " << author << std::endl;
        }
    }

    void displayBooks() {
        if (head == nullptr) {
            std::cout << "No books available." << std::endl;
            return;
        }
        std::cout << "Available books:" << std::endl;
        Book* current = head;
        while (current != nullptr) {
            std::cout << "- " << current->title << " by " << current->author << std::endl;
            current = current->next;
        }
    }

    ~Library() {
        // Free all allocated memory
        Book* current = head;
        while (current != nullptr) {
            Book* next = current->next;
            delete current;
            current = next;
        }
    }
};

int main() {
    Library library;
    int choice;
    std::string title, author;

    while (true) {
        std::cout << "\nLibrary Management System Menu:\n";
        std::cout << "1. Add Book\n";
        std::cout << "2. Remove Book\n";
        std::cout << "3. Search by Title\n";
        std::cout << "4. Search by Author\n";
        std::cout << "5. Display All Books\n";
        std::cout << "6. Exit\n";
        std::cout << "Choose an option: ";
        std::cin >> choice;
        std::cin.ignore(); // Clear newline from input buffer

        switch (choice) {
            case 1:
                  std::cout << "Enter book title: ";
                std::getline(std::cin, title);
                std::cout << "Enter author: ";
                std::getline(std::cin, author);
                library.addBook(title, author);
                break;
           case 2:
                std::cout << "Enter book title to remove: ";
                std::getline(std::cin, title);
                library.removeBook(title);
                break;
            case 3:
                std::cout << "Enter book title to search: ";
                std::getline(std::cin, title);
                if (Book* foundBook = library.searchByTitle(title)) {
                    std::cout << "Found: " << foundBook->title << " by " << foundBook-
>author << std::endl;
                } else {
                    std::cout << "Book not found: " << title << std::endl;
                }
                break;
            case 4:
                std::cout << "Enter author name to search: ";
                std::getline(std::cin, author);
                library.searchByAuthor(author);
                break;
            case 5:
                library.displayBooks();
                break;
            case 6:
                std::cout << "Exiting the system." << std::endl;
                return 0;
            default:
 std::cout << "Invalid choice. Please try again." << std::endl;
        }
    }

    return 0;
}
```

6. Result/Output/Writing Summary:



```
Library Management System Menu:
1. Add Book
2. Remove Book
3. Search by Title
4. Search by Author
5. Display All Books
6. Exit
Choose an option: 1
Enter book title: Mind Set
Enter author: Jeon Jungkook
Book added: Mind Set by Jeon Jungkook
```



```
6. Exit
Choose an option: 1
Enter book title: Mind Set
Enter author: Jeon Jungkook
Book added: Mind Set by Jeon Jungkook

Library Management System Menu:
1. Add Book
2. Remove Book
3. Search by Title
4. Search by Author
5. Display All Books
6. Exit
Choose an option: 2
Enter book title to remove: Mind Set
Book removed: Mind Set

Library Management System Menu:
1. Add Book
2. Remove Book
3. Search by Title
4. Search by Author
5. Display All Books
6. Exit
Choose an option: 
```

Learning outcomes (What I have learnt):

1. Understanding the Basics of System Design
2. Improved Programming Skills in C++
3. Working with Object-Oriented Programming (OOP)
4. Implementing Basic CRUD Operations
5. Understanding Data Flow and Logic Building
6. Experience with Error Handling
7. Basic User Interface Design in Console Applications
8. Enhancing Problem-Solving Skills
9. Learning Project Planning and Time Management

Conclusion

The development of the Library Management System in C++ has been an insightful project, allowing me to gain hands-on experience with designing, coding, and implementing a functional, real-world application. By breaking down the system into essential components such as books, users, and transactions, I was able to apply key programming concepts, including Object-Oriented Programming (OOP) principles, data handling, and file management.

This project not only enhanced my technical skills in C++ but also helped me understand the fundamentals of system design, user interface development, and data flow management. Implementing core features such as book addition, issuance, and return highlighted the importance of a logical approach to program flow and error handling.

In conclusion, creating this Library Management System has strengthened my problem-solving abilities and given me a solid foundation in building applications that manage and organize data efficiently. It also underscored the importance of planning and structuring code to ensure that the system is maintainable and easy to use. This project has been a rewarding experience, and it has prepared me well for more complex programming and system design challenges in the future.

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance (Pre Lab Quiz)		5
2.	Worksheet		10
3.	Post Lab Quiz		5