THE UNIVERSITY OF AZAD JAMMU AND KASHMIR, MUZAFFARABAD

Department of Software Engineering

Submitted to:	Engr. Sidra Rafique
Course Title:	Data Structure and Algorithm
Course Code:	CS-2101
Session:	2024-2028
Lab No:	03
Roll No:	2024-SE-15
Submitted By:	Shahzad Ahmed Awan
Submission date:	October 30, 2025

Table of Contents

Table of Figures

Lab 03 – Abstract Data Type (ADT): Library Management System

---

1. Objective

The objective of this **LAB** is to design and implement an **Abstract Data Type (ADT)** for a real-world system.
The chosen system is a **Library Management System (LMS)** that manages books using data abstraction and encapsulation principles.
This program demonstrates how an ADT can be used to perform operations such as adding, removing, searching, and displaying books while keeping implementation details hidden from the user.

---

2. Background

In Data Structures, an **Abstract Data Type (ADT)** is a user-defined data model that specifies the behavior of data rather than its implementation.
It provides a clear separation between the *what* (operations performed) and the *how* (internal details).

This **LAB** applies the concept of ADT to a **Library Management System**, where a class named Library encapsulates all book-related data and operations.
The data members (like books and totalBooks) are kept private to ensure **encapsulation**, and member functions provide controlled access to manipulate this data.

Through this, the principles of **data abstraction** and **encapsulation** are demonstrated, two fundamental concepts in object-oriented and data structure design.

---

3. Algorithm

1. Start the program.
2. Define a class named Library representing the Library ADT.
3. Inside the class, define a structure named Book with three fields: ID, Title, and Author.
4. Create an array books[50] to store up to 50 book records.
5. Initialize totalBooks to zero in the constructor.
6. Implement the following class member functions:
    - addBook(): Adds a new book record with unique ID, Title, and Author.
    - removeBook(): Removes a book record by its ID.
    - searchBook(): Searches for a specific book using its ID.
    - displayBooks(): Displays all stored books.

- countBooks(): Displays the total number of books in the library.

7. Implement input validation for numeric fields.
8. In the main function, create a Library object and use a menu-driven interface to perform operations repeatedly until the user exits.
9. End the program.

---

4. Source Code

```cpp
#include <iostream>
#include <string>
#include <limits>   // for input validation
#include <cstdlib>  // for system("cls")
using namespace std;

// Library ADT class
class Library {
private:
    struct Book {
        int id;
        string title;
        string author;
    };

    Book books[50];   // Maximum 50 books
    int totalBooks;

public:
    // Constructor
    Library() {
        totalBooks = 0;
    }

    // Clear screen utility
    void clearScreen() {
        system("cls");
    }

    // Add a new book
    void addBook() {
        clearScreen();
        if (totalBooks >= 50) {
            cout << "Library is full! Cannot add more books.\n";
            return;
        }

        int id;
        string title, author;

        cout << "Enter Book ID: ";
        while (!(cin >> id)) {
            cout << "Invalid input! Please enter a numeric Book ID: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        cin.ignore();

        cout << "Enter Book Title: ";
        getline(cin, title);

        cout << "Enter Author Name: ";
        getline(cin, author);

        // Check duplicate ID
        for (int i = 0; i < totalBooks; i++) {
            if (books[i].id == id) {
                cout << "\nA book with this ID already exists!\n";
                return;
            }
        }

        books[totalBooks].id = id;
        books[totalBooks].title = title;
        books[totalBooks].author = author;
        totalBooks++;

        cout << "\nBook added successfully!\n";
```

Figure 1: Source Code Screenshot-1

Source Code 2nd Screenshot:

```
        cout << "\nBook added successfully!\n";
    }

    // Remove a book by ID
    void removeBook() {
        clearScreen();
        if (totalBooks == 0) {
            cout << "Library is empty! No books to remove.\n";
            return;
        }

        int id;
        cout << "Enter Book ID to remove: ";
        while (!(cin >> id)) {
            cout << "Invalid input! Enter a valid numeric ID: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }

        bool found = false;
        for (int i = 0; i < totalBooks; i++) {
            if (books[i].id == id) {
                found = true;
                for (int j = i; j < totalBooks - 1; j++) {
                    books[j] = books[j + 1];
                }
                totalBooks--;
                cout << "\nBook with ID " << id << " removed successfully.\n";
                break;
            }
        }
        if (!found)
            cout << "\nBook not found!\n";
    }

    // Search a book by ID
    void searchBook() {
        clearScreen();
        if (totalBooks == 0) {
            cout << "Library is empty! No books to search.\n";
            return;
        }

        int id;
        cout << "Enter Book ID to search: ";
        while (!(cin >> id)) {
            cout << "Invalid input! Enter a valid numeric ID: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }

        bool found = false;
        for (int i = 0; i < totalBooks; i++) {
            if (books[i].id == id) {
                cout << "\nBook Found:\n";
                cout << "--------------------------------\n";
                cout << "ID: " << books[i].id << endl;
                cout << "Title: " << books[i].title << endl;
                cout << "Author: " << books[i].author << endl;
                cout << "--------------------------------\n";
                found = true;
                break;
            }
        }
        if (!found)
            cout << "\nBook not found in the library.\n";
    }
```

Figure 2:: Source Code Screenshot-2

Source Code 3rd Screenshot:

```
        if (!found)
            cout << "\nBook not found in the library.\n";
    }

    // Display all books
    void displayBooks() {
        clearScreen();
        cout << "========== Library Book List ==========\n";
        if (totalBooks == 0) {
            cout << "No books currently in the library.\n";
        } else {
            for (int i = 0; i < totalBooks; i++) {
                cout << i + 1 << ". ID: " << books[i].id
                     << " | Title: " << books[i].title
                     << " | Author: " << books[i].author << endl;
            }
        }
        cout << "=======================================\n";
    }

    // Display total book count
    void countBooks() {
        cout << "\nTotal books in library: " << totalBooks << endl;
    }
};

// ================ MAIN PROGRAM ==================
int main() {
    Library myLibrary;
    int choice;

    do {
        myLibrary.clearScreen();
        cout << "========== LIBRARY MANAGEMENT SYSTEM ==========\n";
        cout << "1. Add a new book\n";
        cout << "2. Remove a book\n";
        cout << "3. Search for a book\n";
        cout << "4. Display all books\n";
        cout << "5. Show total book count\n";
        cout << "6. Exit\n";
        cout << "==============================================\n";
        cout << "Enter your choice: ";

        // Input validation for menu choice
        while (!(cin >> choice)) {
            cout << "Invalid choice! Please enter a number between 1 and 6: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }

        switch (choice) {
        case 1:
            myLibrary.addBook();
            break;
        case 2:
            myLibrary.removeBook();
            break;
        case 3:
            myLibrary.searchBook();
            break;
        case 4:
            myLibrary.displayBooks();
            break;
        case 5:
            myLibrary.countBooks();
            break;
        case 6:
            cout << "\nExiting... Thank you for using the Library ADT!\n";
            break;
        default:
            cout << "\nInvalid option! Please try again.\n";
        }

        if (choice != 6) {
            cout << "\nPress Enter to return to menu...";
            cin.ignore();
            cin.get();
        }
```

Figure 3: Source Code Screenshot-3

5. Explanation of Code

5.1 Class Design and Abstraction

The class **Library** defines the ADT.
All data members (books and totalBooks) are declared as **private**, ensuring that direct access is restricted.
The user can only interact with the data through public functions such as addBook(), removeBook(), and searchBook().
This separation between implementation and access is the core idea of **abstraction**.

5.2 Encapsulation

Encapsulation is achieved by grouping related data and operations within a single class.
The class hides details like how books are stored, checked, or removed.
The user interacts through well-defined functions without knowing internal data structures.

5.3 Add Book Operation

The function addBook() allows the user to add new books by entering ID, title, and author.
It performs input validation and checks for duplicate IDs before saving the data.
If the library is full or an invalid ID is entered, it displays appropriate messages.

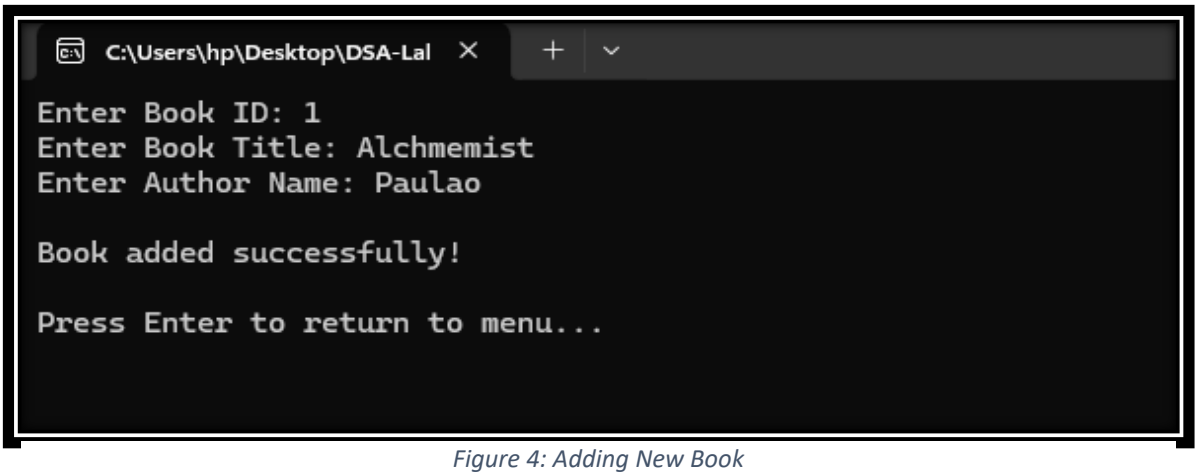

Figure 4: Adding New Book

5.4 Remove Book Operation

The function removeBook() deletes a book based on its ID.
It searches the array and shifts remaining records to maintain proper order.

This demonstrates how data manipulation can be safely handled inside the class without exposing internal storage.

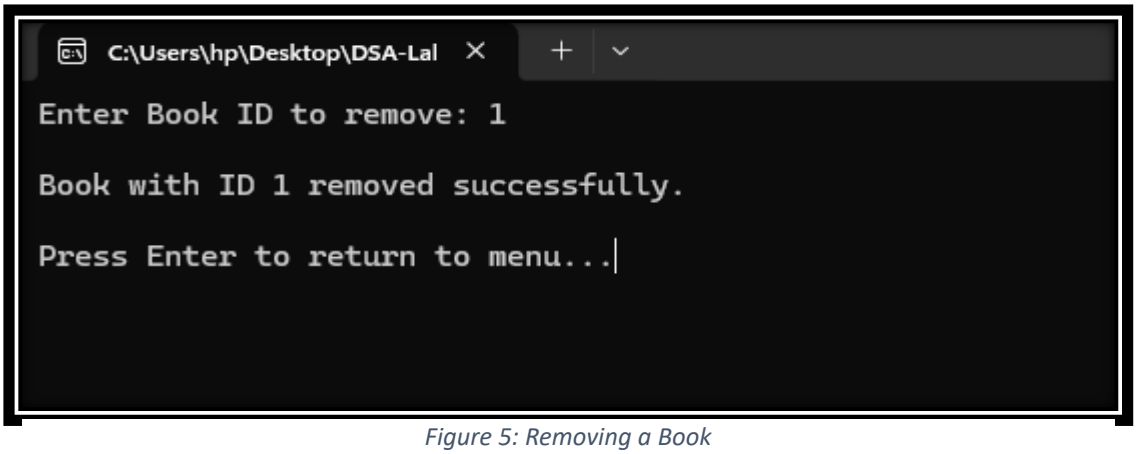

Figure 5: Removing a Book

5.5 Search and Display Functions

The searchBook() function locates a specific book by ID and prints its details if found.
The displayBooks() function lists all books currently stored in the library.
Both functions utilize encapsulation by accessing private data only through controlled methods.

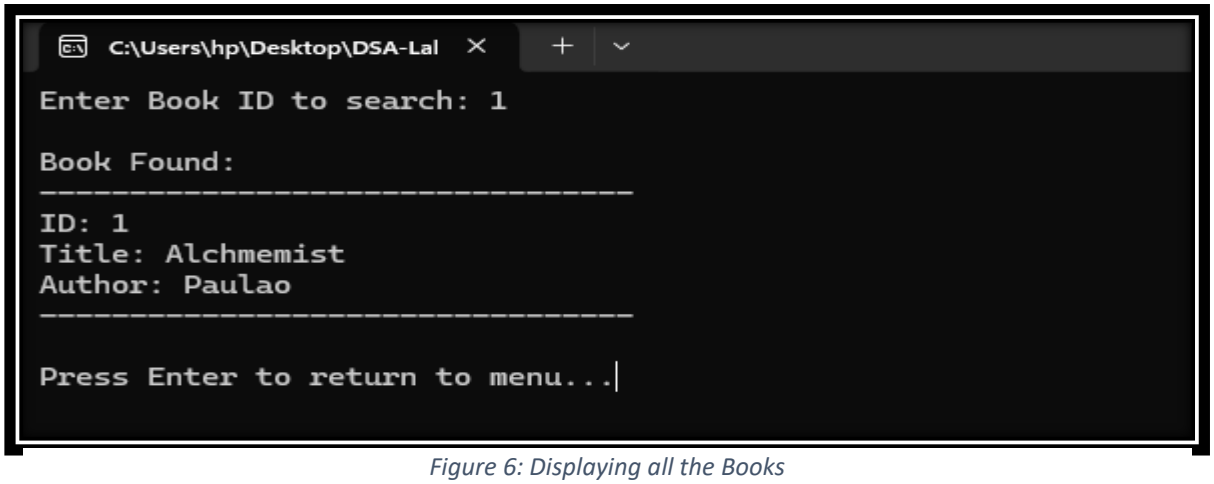

Figure 6: Displaying all the Books

5.6 Counting Books

The countBooks() function simply displays the number of books currently stored.
This gives users a quick summary of library contents without accessing internal variables.

---

6. Output

When executed, the program displays a user-friendly menu-driven interface.
The user can perform operations such as:

- Adding new books.
- Removing existing books.
- Searching for books by ID.
- Displaying all books.
- Viewing the total book count.

Each operation executes successfully while maintaining data abstraction and encapsulation.

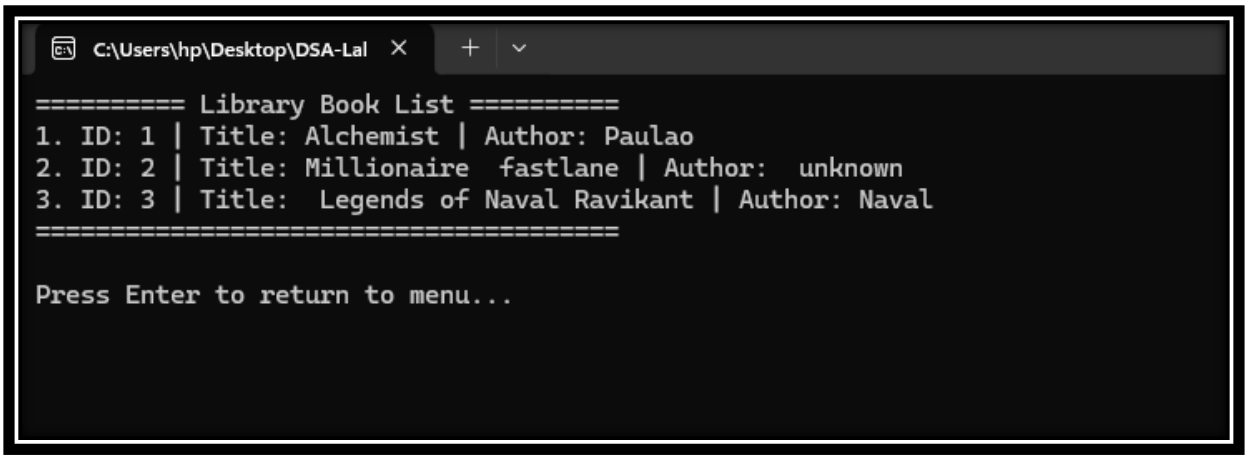

Figure 7: An output sample

---

7. Conclusion

This LAB successfully implemented an Abstract Data Type (ADT) for a Library Management System using C++.
The system demonstrates how abstraction and encapsulation work together to protect data integrity and provide a simplified interface for users.

By defining a class with private data members and public operations, the program hides implementation details while exposing only necessary functionality.
This LAB effectively reinforces key Data Structure principles and shows how ADTs are used in real-world systems for data management.

---

8. Reflection

This LAB provided a practical understanding of how ADTs are designed and implemented in programming.
Through this task, I learned and practiced the following:

- How to design an Abstract Data Type (ADT) using a class.
- The importance of abstraction in simplifying user interaction.

- How encapsulation protects data from direct external modification.
- Implementing data operations such as insertion, deletion, and search inside an ADT.
- Creating a menu-driven interface to interact with an ADT.
- Writing modular, maintainable, and readable code for real-world systems.

This LAB strengthened my knowledge of how ADTs are applied in software systems to manage data securely and efficiently while maintaining a clear interface for users.