Library Management System – Project Report

Project Name: Library Management System Using Dynamic Array

Course: Data Structures & Algorithms (DSA)
Instructor: Sir Ghulam Jillani
Date of Submission: 15 December 2025

Group Members

- **Abdul Mateen** (Roll No. 15)
- **Husnain Khalid** (Roll No. 08)
- **Muhammad Talha** (Roll No. 18)

Table of Contents

1. Summary of the Project
2. Features and Requirements
3. Flowcharts and Module Diagrams
4. Explanation of Each Module
5. Explanation of Data Structures Used
6. File Handling Explanation
7. How to Run the Project
8. Test Cases
9. Limitations and Future Improvements

1. Summary of the Project

The **Library Management System** is a desktop-based application engineered to streamline and automate the core operations of a library. Developed in **C++** using the **FLTK (Fast Light Toolkit)** graphical library, the system provides an intuitive and user-friendly interface for managing a book collection.

The project addresses the need for an efficient, reliable, and low-overhead solution for small to medium-sized libraries by eliminating manual record-keeping. Data persistence is ensured through a file-based storage mechanism, allowing all book records to be saved and restored between application sessions. This project demonstrates practical implementation of fundamental computer science concepts such as **object-oriented programming, dynamic data structures, and file handling**.

2. Features and Requirements

2.1 Functional Requirements (Features)

The system supports the following functionalities:

- **Add New Book:** Allows the librarian to add a new book by entering its title, author, category, and number of copies. Each book is automatically assigned a unique incremental ID.
- **Borrow Book:** Enables borrowing of a book using its ID after checking availability. The available copy count is reduced upon successful borrowing.
- **Return Book:** Allows returning a borrowed book, incrementing the available copy count.
- **Delete Book:** Permanently removes a book record using its unique ID.
- **View All Books:** Displays a complete list of all books including ID, title, author, category, and available copies.

2.2 Non-Functional Requirements (System Needs)

- **Usability:** The system provides a simple and intuitive GUI requiring minimal user training.
- **Performance:** Operations remain responsive even with a large dataset (1000+ books).
- **Reliability:** The system ensures stability and saves data after every modification to prevent data loss.
- **Portability:** The application can be compiled and executed on Windows, Linux, and macOS platforms.

3. Flowcharts and Module Diagrams

To provide a clear understanding of system logic and architecture, the following diagrams are included in the `docs/` directory:

- **Use Case Diagram:** Illustrates interactions between the librarian and system functionalities.
- **Class Diagram:** Shows the structure of `Book` and `Library` classes, their attributes, methods, and relationships.
- **Main Application Flowchart:** Describes the overall control flow from application start to exit.

4. Explanation of Each Module

4.1 Presentation Module (ui/ Directory)

Purpose: Handles all user interactions and visual elements using FLTK.

Files:

- `gui_main.cpp`

- `add_book_window.cpp`
- `borrow_book_window.cpp`
- `return_book_window.cpp`
- `delete_book_window.cpp`
- `view_books_window.cpp`

Functionality: Captures user input, displays output, and communicates with the business logic module.

4.2 Business Logic Module (code/ Directory)

Purpose: Contains core application logic and data management.

Files:

- `library.h`, `library.cpp`
- `book.h`, `book.cpp`

Functionality: Manages book records, enforces business rules, and ensures data integrity.

4.3 Data Persistence Module (server/data/ Directory)

Purpose: Maintains long-term storage of library records.

File: `books.txt`

Functionality: Stores book data in a structured pipe-delimited format to preserve data between sessions.

5. Explanation of Data Structures Used

5.1 Data Structure Used

A **dynamic array** is used to store book objects within the `Library` class. It is implemented using a pointer (`Book* books`) with manual memory allocation.

5.2 Reason for Selection

- **Fast Access:** O(1) random access time
- **Memory Efficiency:** Contiguous memory allocation
- **Simplicity:** Ideal for academic learning and implementation
- **Efficient Resizing:** Capacity doubles when full, ensuring amortized O(1) insertion

5.3 Time and Space Complexity

Operation	Time Complexity	Explanation
addBook()	O(n) (worst)	Reallocation and copying when array is full
borrowBook(id)	O(n)	Linear search by ID
returnBook(id)	O(n)	Linear search by ID
deleteBook(id)	O(n)	Search + shifting elements
viewAllBooks()	O(n)	Iteration through array

Space Complexity: O(m), where m is the current capacity of the array.

6. File Handling Explanation

6.1 Data Storage Location

All records are stored in `server/data/books.txt`. The directory is created automatically if it does not exist.

6.2 File Format

ID|Title|Author|Category|Copies

Sample Data:

1|The C++ Programming Language|Bjarne Stroustrup|Programming|3

2|Clean Code|Robert C. Martin|Programming|5

3|The Pragmatic Programmer|Andy Hunt|Programming|2

4|Design Patterns|Erich Gamma|Software Engineering|0

6.3 File Operations

- **Loading:** Data is read at startup using `loadFromFile()`.
- **Saving:** Data is written after every modification using `saveToFile()`.

7. How to Run the Project

7.1 Prerequisites

- C++ Compiler (G++ / Clang)
- FLTK Development Libraries
- Visual Studio Code (recommended)

7.2 Build & Run Steps

1. Open the project folder in VS Code.
2. Run the build task using **Ctrl + Shift + P → Run Build Task**.
3. Execute the generated file:
    - Linux/macOS: `./library_app`
    - Windows: `LibraryGUI.exe`

8. Test Cases

A complete set of manual test cases was designed to validate system functionality.

- **Positive Testing:** Valid inputs for all features
- **Negative Testing:** Invalid inputs and error handling
- **Edge Cases:** Dynamic array resizing and zero-copy borrowing

Detailed test cases are documented in `Testcases/testcases.md`.

9. Limitations and Future Improvements

9.1 Current Limitations

- Single-user system
- No authentication or roles
- No advanced search functionality
- Text-file-based storage only

9.2 Future Improvements

- Database integration (SQLite)
- User login and role management
- Advanced search and filtering
- Report generation and export
- Data import/export support