

**“PROGRAMMING FUNDAMENTALS**

**SEMESTER PROJECT “**

**“LIBRARY MANAGEMENT SYSTEM”**

**REPORT BY :**

**WALEED KHALID**

**F24CSC002**

# Library Management System - Comprehensive Documentation

This document provides an in-depth explanation of the Library Management System (LMS), developed in C++ to facilitate the management of students, books, and their interactions. The system is equipped with features for account management, book transactions, and data persistence. This documentation includes code snippets, detailed descriptions, and use-case examples.

## Table of Contents

1. System Overview

2. Constants and Global Variables

3. Core Functionalities

3.1 Admin Features

3.2 Student Features

4. Utility Functions

5. File Handling

6. System Architecture and Workflow

7. Future Enhancements

## 1. System Overview

The Library Management System is designed to simplify the management of library resources. The system supports two types of users: Admin and Student. Each user type has specific privileges and features.  
Key Features:  
- Admin: Add, edit, and view books, and manage student accounts.  
- Student: Create accounts, issue books, return books, and manage their account balance.  
The system is implemented using modular programming, with each feature encapsulated in its respective function.

## 2. Constants and Global Variables

The constants and global variables ensure the system's scalability and centralize data management. Constants define the maximum limits for students, books, and name lengths, while global variables store the system's data.

Defined Constants:

const int MAX\_STUDENTS = 20;  
const int MAX\_BOOKS = 15;  
const int MAX\_NAME\_LENGTH = 50;

Global Variables for Data Management:

int student\_count = 0; // Tracks the number of students  
int book\_count = 0; // Tracks the number of books  
int student\_roll[MAX\_STUDENTS]; // Stores student roll numbers  
char student\_name[MAX\_STUDENTS][MAX\_NAME\_LENGTH]; // Stores student names  
double student\_balance[MAX\_STUDENTS]; // Tracks student account balances  
char book\_title[MAX\_BOOKS][MAX\_NAME\_LENGTH]; // Stores book titles  
char book\_author[MAX\_BOOKS][MAX\_NAME\_LENGTH]; // Stores book authors  
int book\_isbn[MAX\_BOOKS]; // Stores book ISBN numbers  
bool book\_available[MAX\_BOOKS]; // Tracks book availability

## 3. Core Functionalities

### 3.1 Admin Features

Admins have privileged access to manage books and view student details. Below are examples of key admin functionalities.

Add a New Book:

void add\_book() {  
 if (book\_count >= MAX\_BOOKS) {  
 cout << "Maximum book limit reached.\n";  
 return;  
 }  
 cout << "Enter book title: ";  
 cin.ignore();  
 cin.getline(book\_title[book\_count], MAX\_NAME\_LENGTH);  
 cout << "Enter book author: ";  
 cin.getline(book\_author[book\_count], MAX\_NAME\_LENGTH);  
 cout << "Enter ISBN number: ";  
 cin >> book\_isbn[book\_count];  
 book\_available[book\_count] = true;  
 book\_count++;  
 cout << "Book added successfully.\n";  
}

This function prompts the admin to enter the book's title, author, and ISBN. If the maximum book limit has not been reached, the details are stored in the system.

Edit Book Details:

void edit\_book() {  
 view\_books();  
 int isbn;  
 cout << "Enter ISBN of the book to edit: ";  
 cin >> isbn;  
 int book\_index = find\_book(isbn);  
 if (book\_index == -1) {  
 cout << "Book not found.\n";  
 return;  
 }  
 cout << "Enter new title: ";  
 cin.ignore();  
 cin.getline(book\_title[book\_index], MAX\_NAME\_LENGTH);  
 cout << "Enter new author: ";  
 cin.getline(book\_author[book\_index], MAX\_NAME\_LENGTH);  
 cout << "Book updated successfully.\n";  
}

The `edit\_book` function allows the admin to modify details of an existing book by providing its ISBN.

### 3.2 Student Features

Students can create accounts, deposit funds, issue books, and return books. Below are detailed examples of these functionalities.

Create a New Account:

void create\_account() {  
 if (student\_count >= MAX\_STUDENTS) {  
 cout << "Maximum student limit reached.\n";  
 return;  
 }  
 int roll;  
 char name[MAX\_NAME\_LENGTH];  
 cout << "Enter roll number: ";  
 cin >> roll;  
 cout << "Enter student name: ";  
 cin.ignore();  
 cin.getline(name, MAX\_NAME\_LENGTH);  
 student\_roll[student\_count] = roll;  
 strcpy(student\_name[student\_count], name);  
 student\_balance[student\_count] = 0.0;  
 student\_count++;  
 cout << "Account created successfully for " << name << " (Roll: " << roll << ").\n";  
}

## Detailed Use Cases and Workflows

Example Workflow for Admin  
1. Admin logs in using a secure password.  
2. Admin adds a new book to the system by providing its title, author, and ISBN.  
3. Admin edits book details if required, e.g., correcting typos in the title.  
4. Admin views the list of available books to ensure the inventory is up-to-date.

Example:  
  
Admin logs in -> Selects 'Add Book' -> Enters Title: 'C++ Programming', Author: 'Bjarne Stroustrup', ISBN: 12345.  
  
System Response:  
  
Book added successfully: 'C++ Programming' by 'Bjarne Stroustrup' (ISBN: 12345).

## Expanded Student Use Cases

Example Workflow for Students  
1. A student creates an account by providing their roll number and name.  
2. They deposit money into their account for fines or book purchases.  
3. The student views the list of available books and issues one.  
4. Once done, the student returns the book, ensuring its availability for others.

## Utility Functions Explained

Utility functions play a critical role in maintaining smooth operations. Below is a detailed explanation of some utility functions.

Clear Input Buffer:

void clear\_input() {  
 cin.clear();  
 cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
}

This function clears the input buffer, preventing unexpected behavior caused by invalid inputs or leftover characters.

## System Architecture and Workflow

The system architecture is modular, ensuring scalability and maintainability. Each feature is encapsulated within a function, and global variables manage the state. Data persistence is achieved using file handling.

## Future Enhancements

The system can be improved with the following enhancements:  
- Implement a graphical user interface (GUI).  
- Add fine management for overdue books.  
- Include book categorization and search functionality.  
- Integrate with external databases for large-scale operations.

The Library Management System provides a robust platform for managing resources effectively. Its modular design ensures that functionalities are isolated and easy to maintain. Additional features such as fine management, advanced search options, and categorization can further enhance its usability. By incorporating modern techniques, the system can be scaled to meet the demands of larger institutions.

The Library Management System provides a robust platform for managing resources effectively. Its modular design ensures that functionalities are isolated and easy to maintain. Additional features such as fine management, advanced search options, and categorization can further enhance its usability. By incorporating modern techniques, the system can be scaled to meet the demands of larger institutions.

The Library Management System provides a robust platform for managing resources effectively. Its modular design ensures that functionalities are isolated and easy to maintain. Additional features such as fine management, advanced search options, and categorization can further enhance its usability. By incorporating modern techniques, the system can be scaled to meet the demands of larger institutions.

The Library Management System provides a robust platform for managing resources effectively. Its modular design ensures that functionalities are isolated and easy to maintain. Additional features such as fine management, advanced search options, and categorization can further enhance its usability. By incorporating modern techniques, the system can be scaled to meet the demands of larger institutions.

The Library Management System provides a robust platform for managing resources effectively. Its modular design ensures that functionalities are isolated and easy to maintain. Additional features such as fine management, advanced search options, and categorization can further enhance its usability. By incorporating modern techniques, the system can be scaled to meet the demands of larger institutions.