LIBRARY MANAGEMENT SYSTEM

A Console-Based Project in C Language

Submitted in partial fulfillment of the requirements for the subject
Logic Building With Computer Programming

Submitted by:
Name : Raman
Roll No. : GF202570164
Class : B.Tech (CSE core)
Semester : 1st Semester
University : Shoolini University, Solan (H.P.)

Submitted to:
Mr. Anurag Rana
Professor
Department of Computer Science & Engineering

Academic Year: 2025–2026

DECLARATION

I hereby declare that the project entitled **"Library Management System"** submitted by me is an original work carried out under the guidance of Mr. Anurag Rana for the subject Logic Building With Computer Programming. This project has not been submitted elsewhere for the award of any degree or diploma. I have not plagiarized any content, and all references have been duly acknowledged.

Place: Shoolini University
Date: December 2025

Signature: ____________________
Raman (GF202570164)

CERTIFICATE

This is to certify that Mr. Raman, Roll No. GF202570164, student of B.Tech (CSE/AI/ML), has successfully completed the project titled **"Library Management System"** in C language as part of the curriculum of the subject **Logic Building With Computer Programming**. The project work is original, demonstrates practical application of programming concepts, and has been carried out under my guidance. The student has shown diligence in coding, testing, and documentation.

Signature of Guide: ____________________
Mr. Anurag Rana
Assistant Professor

Date: _________________

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Mr. Anurag Rana**, my subject teacher and project guide, for his constant guidance, encouragement, and valuable suggestions throughout the development of this project. His expertise in C programming and system design helped me overcome technical challenges and refine my approach.

I am also thankful to the **Department of Computer Science & Engineering**, Shoolini University, for providing the necessary infrastructure, lab facilities, and access to compilers like GCC and Dev-C++, which were instrumental in completing this work.

Additionally, I extend my thanks to the university library staff for sharing insights into real-world library operations, which inspired the functional requirements of this system.

Finally, I thank my classmates for their collaborative discussions, peer reviews, and moral support, as well as my family members for their unwavering encouragement during late-night debugging sessions.

Raman
GF202570164

ABSTRACT

The Library Management System is a console-based application developed in C programming language to automate basic library operations such as adding books, registering students, borrowing and returning books, and searching records. This project leverages fundamental C concepts including structures for data modeling, arrays for storage, functions for modularity, and loops for user interactions. It handles up to 50 books and 30 students in memory, with features like borrowing limits and real-time availability updates. While data is not persisted to files in this version, the system provides a robust foundation for expansion. The project emphasizes error handling, user-friendly interfaces, and efficient data management, making it suitable for small-scale educational libraries.

Keywords: C Programming, Library Management, Structures, Console Application, Data Management

---

TABLE OF CONTENTS

1. Introduction 1
2. Objectives of the Project 2
3. Scope of the Project 3
4. System Requirements 4
5. System Design 5
   5.1 Data Flow Diagram (DFD) – Level 0 5
   5.2 ER Diagram (Conceptual) 6
   5.3 Modules Description 7
   5.4 Data Structures Used 8
6. Features Implemented 9
7. Code Explanation (Key Functions) 10
8. Sample Screenshots / Output 12
9. Limitations 13
10. Future Enhancements 14
11. Conclusion 15
12. References 16
13. Source Code (Appendix) 17

---

1. INTRODUCTION

In the digital age, efficient management of resources is crucial for educational institutions. Libraries, as knowledge hubs, require systematic tools to handle inventory, user registrations, and transactions. Manual processes often lead to errors, delays, and inefficiencies, such as misplaced books or untracked borrowings.

This project presents a **console-based Library Management System** developed entirely in the **C programming language**. It utilizes core programming constructs like structures to model entities (books and students), arrays for in-memory data storage, functions for reusable code blocks, and control structures for decision-making and iterations. File handling is conceptualized but not implemented in this version to focus on foundational logic building.

The system targets librarians as primary users, enabling them to:

- Maintain a catalog of books with details like ID, title, author, quantity, and availability.
- Manage student records including roll number, name, class, and borrowed books.
- Facilitate borrowing and returning processes with validation checks.
- Perform searches and view operations for quick access.

By simulating real-world library workflows, this project bridges theoretical programming knowledge with practical application. It was developed using Dev-C++ on Windows, ensuring portability across platforms with minimal modifications.

The report is structured to cover objectives, design, implementation, outputs, limitations, and future scope, providing a comprehensive overview.

---

2. OBJECTIVES OF THE PROJECT

The primary goal of this project is to apply and reinforce concepts learned in the "Logic Building With Computer Programming" course. Specific objectives include:

- **Mastering C Fundamentals:** To effectively use structures for data encapsulation, arrays and pointers for dynamic memory management, and functions to promote code reusability and modularity.

- **Implementing Real-World Logic:** To model a practical system that handles entity relationships, such as one-to-many (a student can borrow multiple books), and enforce business rules like borrowing limits.

- **Enhancing Problem-Solving Skills:** To incorporate input validation, error handling (e.g., for invalid IDs or unavailable books), and efficient search algorithms to ensure robustness.

- **Promoting User-Centric Design:** To create a menu-driven interface that is intuitive, with clear prompts and feedback, reducing user errors.

- **Demonstrating Scalability Concepts:** To design the system with extensibility in mind, such as preparing for file-based persistence or database integration.

- **Documentation and Testing:** To practice thorough documentation of code and design, along with manual testing of various scenarios to verify correctness.

Through these objectives, the project aims to build a functional prototype that can evolve into a production-ready application.

---

3. SCOPE OF THE PROJECT

The scope defines the boundaries and capabilities of the system in its current iteration:

In-Scope:

- Core operations: Adding, viewing, searching books; registering, viewing students; borrowing and returning books.
- In-memory data management for up to 50 books and 30 students.
- Validation rules: Students limited to 3 books; real-time availability tracking.

- Basic search functionality using string matching.
- Console interface with menu navigation.

Out-of-Scope:

- Persistent storage (data resets on program exit).
- User authentication or role-based access (e.g., admin vs. student views).
- Advanced features like fine calculations, due dates, or reporting.
- Graphical interfaces or web-based deployment.
- Integration with external hardware (e.g., barcode scanners) or databases.

This version is ideal for small libraries with low traffic, such as school or department-level setups. Future iterations can expand the scope to address out-of-scope items, making it suitable for larger institutions.

---

4. SYSTEM REQUIREMENTS

To ensure smooth development, execution, and testing, the following requirements were considered:

Hardware Requirements:

- Processor: Intel Core i3 or equivalent (minimum).
- RAM: 512 MB (recommended 2 GB for better performance during compilation).
- Storage: 50 MB free disk space for source code and compiler.
- Input/Output: Standard keyboard, monitor, and mouse.

Software Requirements:

- Operating System: Windows 10/11, Linux (Ubuntu), or macOS (with GCC installed).
- Compiler: GCC (GNU Compiler Collection) version 8.0+, Turbo C++, Dev-C++ 5.11, or Code::Blocks.
- Programming Language: ANSI C (C89/C99 standards).
- Development Tools: Text editor like Notepad++ or integrated IDE for debugging.
- Additional Libraries: Standard C libraries (stdio.h, string.h); no external dependencies.

The system was tested on Windows 11 with Dev-C++, ensuring no platform-specific issues. For Linux/macOS, minor adjustments to file paths or console clearing commands may be needed.

---

5. SYSTEM DESIGN

System design outlines the architecture, data flows, and components. This section includes diagrams (described in text with ASCII art for visualization) and detailed descriptions.

5.1 Data Flow Diagram (DFD) – Level 0

A Level 0 DFD provides a high-level overview of the system, showing interactions between external entities and the core process.

ASCII Representation of Level 0 DFD:

```
+--------------------+     +--------------------+     +--------------------+
|     Librarian      |     |  Library System    |     |      Database      |
| (External Entity)  | --> | (Central Process)  | <-> | (In-Memory Array)  |
+--------------------+     +--------------------+     +--------------------+
          ^                           |
          |                           v
          |                 +--------------------+
          +-----------------|   Students/Books   |
                            |    (Data Stores)   |
                            +--------------------+
```

- **Processes:** The central "Library System" handles inputs from the librarian (e.g., add book, borrow) and updates data stores.
- **Data Flows:** Inputs include book/student details; outputs are views, search results, and status messages.
- **External Entities:** Librarian as user; potential future expansion to students.
- **Data Stores:** Books and Students arrays.

This DFD illustrates the system's simplicity, focusing on single-user interactions.

5.2 ER Diagram (Conceptual)

The Entity-Relationship (ER) Diagram models entities, attributes, and relationships.

ASCII Representation of ER Diagram:

```
+-------------+          +-------------+
|   Book      |          |  Student    |
+-------------+          +-------------+
| - id (PK)   | 1..*     | - roll_no   |
| - title     | BORROWS  |   (PK)      |
| - author    | <------> | - name      |
| - quantity  |          | - class     |
| - available|          | - borrowed  |
+-------------+          +-------------+
                     (0..3)
```

- **Entities:** Book (with attributes like ID as primary key) and Student (roll_no as PK).

- **Relationship:** "BORROWS" – A student can borrow 0 to 3 books (cardinality 1:* with max 3); a book can be borrowed by multiple students over time.
- **Attributes:** Borrowed is an array in Student to store book IDs.

This conceptual ER highlights the relational aspect without database implementation.

5.3 Modules Description

The system is divided into modular functions for maintainability:

1. **Main Menu Module:** Displays options and handles user choices using switch-case.
2. **Book Management Modules:** addBook(), viewBooks(), searchBook() – Handle CRUD-like operations on books.
3. **Student Management Modules:** addStudent(), viewStudents() – Manage student records.
4. **Transaction Modules:** borrowBook(), returnBook() – Update availability and borrowed lists.
5. **Utility Modules:** clearScreen(), inputValidation() – For UI and error checks.

Each module is self-contained, with global arrays shared for data access.

5.4 Data Structures Used

Custom structures define entities:

```
struct Book {
    int id;                 // Unique auto-generated ID
    char title[50];         // Book title
    char author[30];        // Author name
    int quantity;           // Total copies
    int available;          // Available copies
};

struct Student {
    int roll_no;            // Unique auto-generated roll number
    char name[30];          // Student name
    int std_class;          // Class/Year
    int borrowed[3];        // Array of borrowed book IDs (-1 for empty)
    int count;              // Current borrowed count
};
```

- Global Arrays: `struct Book books[50]; int bookCount = 0;`
- `struct Student students[30]; int studentCount = 0;`

These structures encapsulate data, with arrays providing simple storage. Pointers could be used in future for dynamic allocation.

6. FEATURES IMPLEMENTED

The system incorporates several key features to ensure functionality and usability:

- **Auto-ID Generation:** Books start from ID 1001, students from 100, incrementing automatically to avoid duplicates.
- **Borrowing Limits:** Students restricted to 3 books; checks prevent over-borrowing.
- **Availability Tracking:** Real-time updates to 'available' field during borrow/return.
- **Search Functionality:** Case-insensitive partial matching using strstr() and tolower() for titles/authors.
- **Validation and Error Handling:** Checks for existence (e.g., student/book not found), availability, and input types (e.g., no negative quantities).
- **User Interface:** Menu-driven with numbered options, clear prompts, and success/error messages.
- **Modular Code:** Functions separated by responsibility, improving readability and debugging.

These features make the system practical for basic use, with emphasis on clean code practices.

7. CODE EXPLANATION (Key Functions)

This section details key functions with pseudocode and explanations. Full code is in the Appendix.

addBook()

- Purpose: Adds a new book to the array.
- Logic: Checks if array is full; auto-assigns ID; reads title, author, quantity; sets available = quantity.
- Pseudocode:

```
if (bookCount >= 50) { error: full; return; }
books[bookCount].id = 1001 + bookCount;
scanf title, author, quantity;
books[bookCount].available = quantity;
bookCount++;
```

borrowBook()

- Purpose: Allows a student to borrow a book.

- Logic: Validate student existence and count < 3; check book existence and available > 0; add book ID to student's borrowed array; decrement available.
- Pseudocode:

```
input student_roll, book_id;
find student index; if not found { error; return; }
if (student.count >= 3) { error: limit reached; return; }
find book index; if not found or available <= 0 { error; return; }
student.borrowed[student.count] = book_id;
student.count++;
book.available--;
```

returnBook()

- Purpose: Returns a borrowed book.
- Logic: Validate student and book; check if book is in borrowed list; remove from array (shift elements); increment available.
- Pseudocode:

```
input student_roll, book_id;
find student; if not { error; }
find position in borrowed; if not { error: not borrowed; }
shift array elements to remove;
student.count--;
find book; book.available++;
```

searchBook()

- Purpose: Searches books by title or author.
- Logic: Reads search string; iterates array using strstr() for partial match (convert to lower case for insensitivity).
- Pseudocode:

```
input search_str; tolower(search_str);
for each book {
  tolower(title_copy); tolower(author_copy);
  if strstr(title_copy, search_str) or strstr(author_copy, search_str)
{
    print book details;
  }
}
```

Other functions like viewBooks() iterate and print arrays; main() handles loop for menu.

---

8. SAMPLE OUTPUT / SCREENSHOTS

(In the Word document, insert actual screenshots from running the program. Here, describe sample console outputs.)

Main Menu Output:

```
Library Management System
1. Add New Book
2. View All Books
3. Search Book
4. Add New Student
5. View All Students
6. Borrow Book
7. Return Book
8. Exit
Enter choice:
```

Add Book Sample:

```
Enter Title: C Programming
Enter Author: Dennis Ritchie
Enter Quantity: 5
Book added successfully! ID: 1001
```

Borrow Book Sample (Success):

```
Enter Student Roll No: 100
Enter Book ID: 1001
Book borrowed successfully!
```

Error Sample:

```
Enter Student Roll No: 999
Student not found!
```

Include 6-8 such samples with descriptions. For visuals, run in console and capture.

---

9. LIMITATIONS

Despite its functionality, the system has constraints:

- **Data Volatility:** In-memory storage means data loss on exit; no file persistence.
- **Security Absence:** No authentication; anyone can access all features.

- **Scalability Issues:** Fixed array sizes limit to 50 books/30 students; no dynamic allocation.
- **No Advanced Tracking:** Lacks due dates, fines, or history logs.
- **Editing/Deletion Missing:** Cannot update or remove records once added.
- **Search Limitations:** Basic string match; no fuzzy or advanced queries.
- **Platform Dependency:** Console clearing uses system("cls"); may vary on OS.

These are due to focus on basic C concepts; addressed in future enhancements.

---

10. FUTURE ENHANCEMENTS

To evolve the system:

1. **Persistence:** Use fopen(), fwrite(), fread() for file-based storage.
2. **Security:** Implement password login for admins.
3. **Time-Based Features:** Add issue/return dates using time.h; calculate fines.
4. **CRUD Operations:** Add edit/delete for books/students.
5. **Reporting:** Generate stats like top books or overdue lists.
6. **UI Upgrade:** Use graphics.h for GUI or ncurses for enhanced console.
7. **Integration:** Connect to MySQL via C API; add barcode support.
8. **Scalability:** Switch to linked lists or dynamic arrays with malloc().
9. **Multi-User:** Support student self-service views.

These would make it enterprise-ready.

---

11. CONCLUSION

This Library Management System project effectively applies C programming principles to create a functional application. It demonstrates data modeling with structures, modular design with functions, and logical flow with control statements. Through development, I encountered challenges like array shifting for returns and string handling for searches, which enhanced my debugging skills.

The system meets basic requirements for small libraries, providing efficiency over manual methods. It serves as a learning milestone, highlighting the importance of planning, coding, and iteration. Overall, this project has strengthened my logic-building abilities and prepared me for complex software development.

---

12. REFERENCES

1. Kanetkar, Yashavant. *Let Us C*. BPB Publications, 17th Edition, 2020.
2. Balagurusamy, E. *Programming in ANSI C*. Tata McGraw-Hill, 8th Edition, 2019.

3. Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd Edition, 1988.
4. GeeksforGeeks. "C Programming Language Tutorials." Accessed November 2025. https://www.geeksforgeeks.org/c-programming-langu