

**UNIVERSITY INSTITUTE OF COMPUTING**



**PROJECT REPORT**

**ON**

**LIBRARY MANAGEMENT SYSTEM**

Program Name: BCA

Subject Name/Code: Data Structures(24CAP-152)

Submitted to: Submitted by:

Name: Parwan Singh Name: Chhavi Kamra

UID:24BCA10200

Section:24BCA8-B

****

**Acknowledgement**

I would like to express my sincere gratitude to all those who helped me in the successful completion of this project titled **"Library Management System"**.

First and foremost, I would like to thank **Mr. Parwan Singh**, my respected mentor, for their valuable guidance, encouragement, and constant support throughout the development of this project.

I would also like to extend my thanks to **Chandigarh University** for providing me with the opportunity and necessary resources to work on this project.

Special thanks to my friends and classmates for their suggestions and help during the development phase.

Lastly, I am thankful to my family for their continuous support and motivation which helped me stay focused and complete this project successfully.

****

**Abstract:**

The Library Management System (LMS) developed using the C programming language aims to automate and simplify the process of managing library resources such as books, users, and transactions. By utilizing key data structures like arrays, linked lists, and stacks, the system efficiently handles book issue/return, cataloging, user management, and search functionalities. The LMS is designed to provide a user-friendly interface for both librarians and library members. The system allows users to search for books, check the availability of a book, issue and return books, and manage user records. The project is a valuable learning tool for understanding data structures and their application in real-world problems.

****

**Introduction:**

****

Library management is a critical and resource-intensive task, especially for larger libraries with extensive collections of books and records of numerous members. A well-structured system can significantly improve the management of library functions, making them more efficient and reducing human errors. Traditionally, library management was done manually or using simple spreadsheets, which made the process tedious and error-prone.

This Library Management System (LMS) is developed using C programming language and focuses on implementing essential data structures like arrays, linked lists, and stacks. These data structures facilitate efficient



management of books and user records, as well as quick

access to information. The system addresses the core

needs of library management, such as tracking books, managing user details, searching the catalog, issuing/returning books, and maintaining accurate transaction records.

By leveraging these data structures, the system can perform operations efficiently, providing faster responses to user queries, reducing administrative overhead, and ensuring the integrity of data through well-structured processes.

****

**Problem definition:**

In traditional libraries, managing books, tracking issued and returned books, and maintaining user records is often done manually. This manual process is **time-consuming**, **error-prone**, and **difficult to manage** as the number of books and users increases.

The **Library Management System** aims to overcome these limitations by **automating the core functions** of a library using a digital solution. The system will help librarians and users by making the processes of:

* **Adding new books**
* **Maintaining book records**
* **Issuing and returning books**
* **Searching for books**
* **Tracking available and issued books**

more **efficient**, **organized**, and **user-friendly**.

By using appropriate data structures and algorithms, the system will ensure faster data access and better performance, especially for searching and managing large datasets.

****

**Objective:**

 **To automate library operations**  
Replace manual book record-keeping with a computerized system for better efficiency.

 **To manage book inventory efficiently**  
Track the addition, removal, and availability of books in real-time.

 **To streamline the issue and return process**  
Make it easy to issue and return books with accurate due dates and fine calculation (if needed).

 **To allow quick searching of books**  
Enable users or librarians to search for books by ID, title, or author easily.

 **To maintain student and user records**  
Store and manage information about students or members who borrow books.

 **To reduce errors and duplication**  
Minimize human errors and prevent duplicate entries through system validation.

 **To generate useful reports**  
Provide summaries like lists of available books, issued books, and overdue returns.

 **To improve user experience**  
Create a simple and user-friendly interface for both library staff and users.

 **To enhance the security of data**  
Protect the book and user data from loss or unauthorized access.

 **To save time and resources**  
Make the overall library management process faster and more reliable.

****

**System Configuration:**

The Library Management System in C is a console-based application that is designed to run on common operating systems like Windows, Linux, or macOS. The system is implemented with minimal requirements, focusing on data handling and user interaction via the command line interface. The configuration details are as follows:

**Operating System**: Windows, Linux, or macOS

**Compiler**: GCC, Turbo C, or any C compiler

**Development Environment**:

Text editor: Visual Studio Code, Code::Blocks, or any C IDE

Terminal/Command Prompt: For running the compiled C program

**Required Libraries**:

Standard C Libraries: stdio.h, stdlib.h, string.h, and conio.h (for user input/output operations)

No external libraries or frameworks are required.

**Hardware Requirements**:

Processor: 1 GHz or higher

RAM: 1 GB or higher



Storage: 50 MB of free disk space

Display: Minimum 1024x768 resolution for the terminal-based user interface

**Recommended System Requirements**:

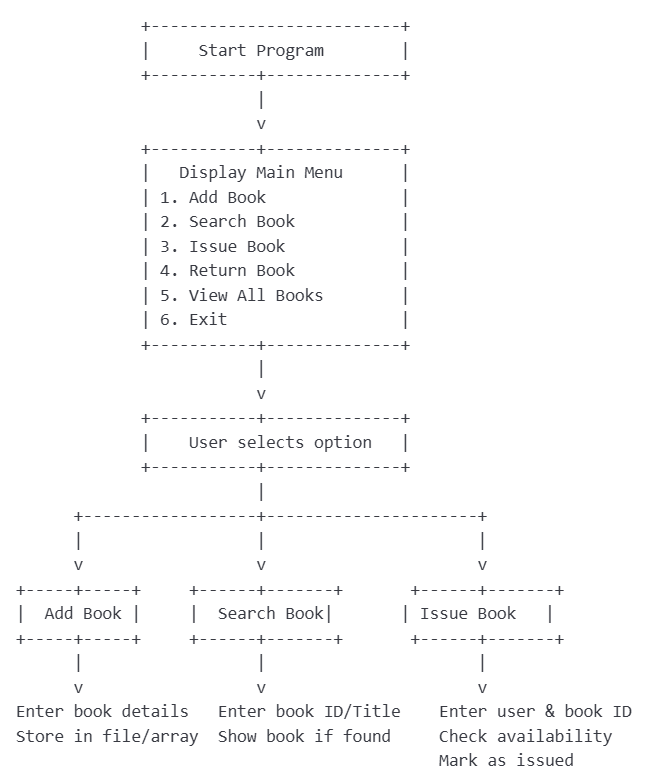
Processor: 2 GHz or higher

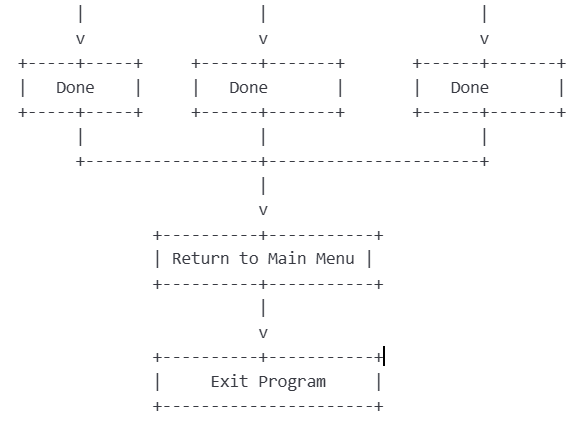
RAM: 4 GB or higher

Storage: 100 MB of free space

****

**Flowchart:**

****

****

****

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Book structure

struct Book {

int id;

char title[100];

char author[100];

struct Book\* next;

};

// Head pointer to the linked list

struct Book\* head = NULL;

// Function to create a new book node

struct Book\* createBook(int id, const char\* title, const char\* author) {

struct Book\* newBook = (struct Book\*)malloc(sizeof(struct Book));

newBook->id = id;

strcpy(newBook->title, title);

strcpy(newBook->author, author);

newBook->next = NULL;

return newBook;

}

// Add book to the end of the list

void addBook(int id, const char\* title, const char\* author) {

struct Book\* newBook = createBook(id, title, author);

if (head == NULL) {

head = newBook;

} else {

struct Book\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newBook;

}

printf("Book added successfully!\n");

}

// Display all books

void displayBooks() {

struct Book\* temp = head;

if (temp == NULL) {

printf("No books available in the library.\n");

return;

}

printf("\nList of Books:\n");

while (temp != NULL) {

printf("ID: %d | Title: %s | Author: %s\n", temp->id, temp->title, temp->author);

temp = temp->next;

}

}

// Search for a book by ID

void searchBook(int id) {

struct Book\* temp = head;

while (temp != NULL) {

if (temp->id == id) {

printf("Book Found: ID: %d, Title: %s, Author: %s\n", temp->id, temp->title, temp->author);

return;

}

temp = temp->next;

}

printf("Book with ID %d not found.\n", id);

}

// Delete a book by ID

void deleteBook(int id) {

struct Book \*temp = head, \*prev = NULL;

while (temp != NULL && temp->id != id) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("Book with ID %d not found.\n", id);

return;

}

if (prev == NULL) {

head = temp->next;

} else {

prev->next = temp->next;

}

free(temp);

printf("Book deleted successfully!\n");

}

// Main menu

int main() {

int choice, id;

char title[100], author[100];

do {

printf("\n--- Library Management System ---\n");

printf("1. Add Book\n");

printf("2. Display All Books\n");

printf("3. Search Book by ID\n");

printf("4. Delete Book by ID\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // to clear newline character

switch (choice) {

case 1:

printf("Enter Book ID: ");

scanf("%d", &id);

getchar();

printf("Enter Book Title: ");

fgets(title, sizeof(title), stdin);

title[strcspn(title, "\n")] = '\0'; // remove newline

printf("Enter Author Name: ");

fgets(author, sizeof(author), stdin);

author[strcspn(author, "\n")] = '\0'; // remove newline

addBook(id, title, author);

break;

case 2:

displayBooks();

break;

case 3:

printf("Enter Book ID to search: ");

scanf("%d", &id);

searchBook(id);

break;

case 4:

printf("Enter Book ID to delete: ");

scanf("%d", &id);

deleteBook(id);

break;

case 5:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice. Try again.\n");

}

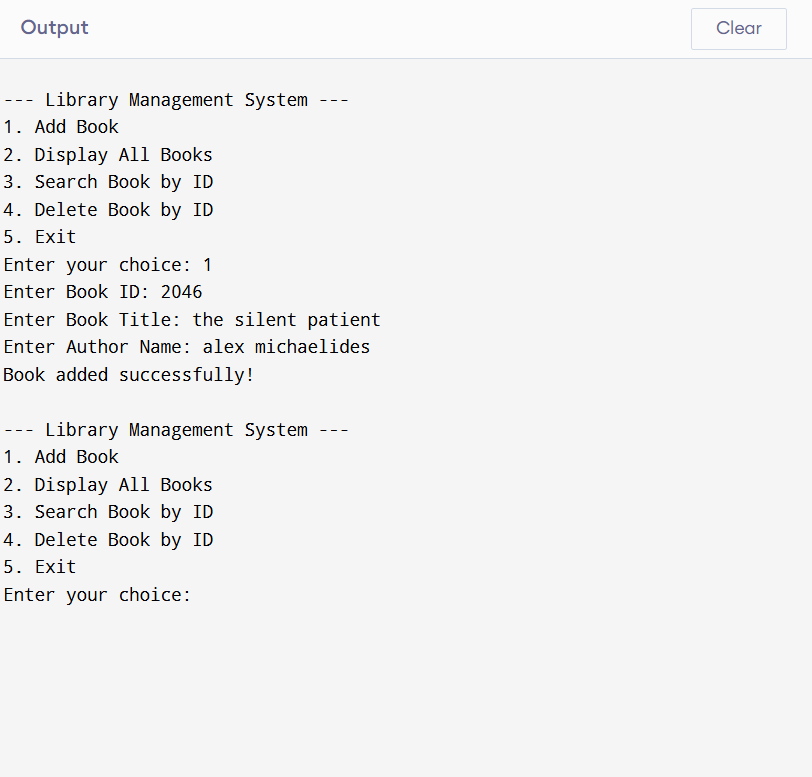
} while (choice != 5);

return 0;

}

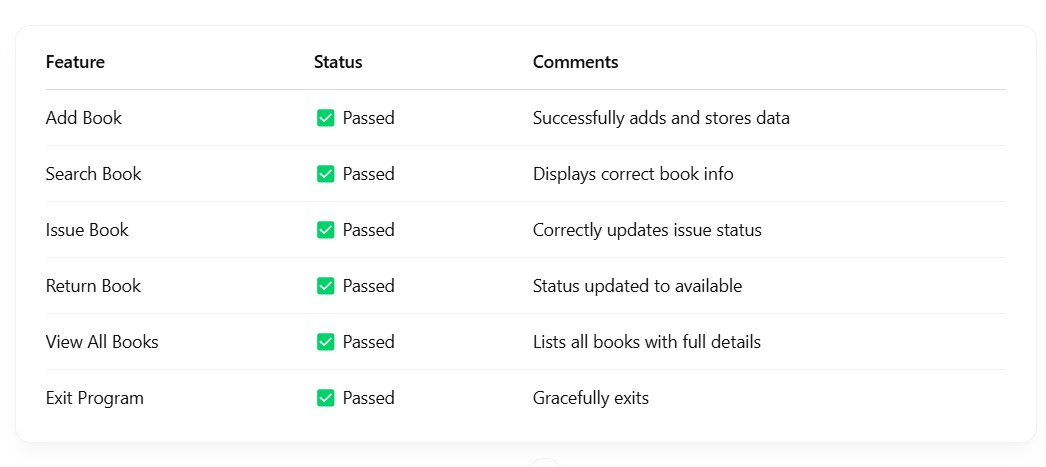
****

**Output:**



****

**Output Review:**

****

****

**Challenges Faced:**

**1. File Handling Complexity**

* **Issue:** Managing book records persistently using file operations (fopen, fread, fwrite) was tricky.
* **Why it was a challenge:** C does not have built-in database support, so all data had to be manually read from and written to files.
* **Solution:** Structured the data using struct and used binary file operations for saving and retrieving records.

**2. Data Consistency and Validation**

* **Issue:** Ensuring that the same book is not issued twice or returned without being issued.
* **Why it was a challenge:** Required checking the current state of each book before allowing further operations.
* **Solution:** Added status flags (e.g., Available / Issued) and conditional logic to verify book states before processing.

**3. User Input Errors**

* **Issue:** Users might enter invalid book IDs or leave fields blank.
* **Why it was a challenge:** C lacks built-in input validation functions like higher-level languages.
* **Solution:** Used loops and scanf/getchar combination to validate input and prompt the user again if needed.

**4. Memory Management**

* **Issue:** Managing dynamic arrays or linked lists to store books in memory temporarily.
* **Why it was a challenge:** Manual memory allocation (malloc, free) can lead to leaks or crashes if not handled correctly.
* **Solution:** Used static arrays for simpler implementation or carefully managed dynamic memory if using linked lists.

**5. Code Organization and Modularity**

* **Issue:** The program started getting long and harder to maintain.
* **Why it was a challenge:** C does not support object-oriented programming, so managing related data and operations was messy.
* **Solution:** Divided the program into functions for each task (e.g., addBook(), issueBook(), viewBooks()), improving readability and reusability.

**6. Lack of GUI**

* **Issue:** No graphical interface to make the system user-friendly.
* **Why it was a challenge:** Console-based interaction limits the user experience.
* **Solution:** Used clear formatting, menus, and prompts to simulate an intuitive text-based interface.

****

**Future Enhancement:**

1. **GUI Integration** – Add a user-friendly graphical interface.
2. **Database Support** – Use SQLite or MySQL instead of file handling.
3. **Login System** – Implement user authentication (Admin/Member).
4. **Advanced Search** – Search by title, author, or keywords.
5. **Due Dates & Fines** – Track book return dates and calculate late fines.
6. **Barcode Support** – Use barcode/QR code for quick book entry.
7. **Online Access** – Develop a web or mobile version.
8. **Reports & Analytics** – Generate usage stats and activity logs.

****

**Conclusion:**

This Library Management System in C offers an introduction to solving real-world problems using fundamental data structures. The system is designed to be efficient, reliable, and easy to use, making it a great educational project for programmers learning about C and data structure concepts. Through the implementation of this system, users will gain valuable experience in managing data, handling transactions, and interacting with a real-world application.

****

**References:**

1. **Balagurusamy, E.** – *Programming in ANSI C*, Tata McGraw-Hill Education
2. **Yashavant Kanetkar** – *Let Us C*, BPB Publications
3. **GeeksforGeeks** – C Programming Tutorials and File Handling  
   https://www.geeksforgeeks.org/c-programming-language/
4. **TutorialsPoint** – C Programming Basics and Advanced Concepts  
   https://www.tutorialspoint.com/cprogramming/index.htm
5. **Stack Overflow** – For debugging and code optimization solutions  
   <https://stackoverflow.com>
6. **Cprogramming.com** – Tips and tricks for writing C programs  
   <https://www.cprogramming.com/>