**LIBRARY MANAGEMENT SYSTEM**

**Project report submitted in partial fulfillment of the Requirements for the Award of the Degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**C.DEEKSHITHA REDDY-24KB1A05B1**

**B.SRAVYA-24KB1A0564**

**C.SUJANA SREE-24KB1A0578**

**B.SAI SHIVANI-24KB1A0551**

**Under the Guidance of**

**Smt.B.SRUTHI**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**NBKR INSTITUTE OF SCIENCE &TECHNOLOGY**

**VIDYANAGAR-524413,TIRUPATHI DISTRICT**

**ANDHRA PRADESH**

**LIBRARY MANAGEMENT SYSTEM**

**Project report submitted in partial fulfillment of the Requirements for the Award of the Degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**C.DEEKSHITHA REDDY-24KB1A05B1**

**B.SRAVYA-24KB1A0564**

**B.SAI SHIVANI-24KB1A0551**

**C.SUJANA SREE-24KB1A0578**

Under the Guidance of

**SHIVAN RAJ**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**NBKR INSTITUTE OF SCIENCE &TECHNOLOGY**

**VIDYANAGAR-524413**

**TIRUPATHI.**

**NBKR INSTITUTE OF SCIENCE &TECHNOLOGY**

**(AUTONOMOUS)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

This is to certify that the project report entitled LIBRARY MANAGEMENT SYSTEM being submitted by

C.Deekshitha Reddy 24KB1AO5B1

B.Sravya 24KB1A0564

B.Sai Shivani 24KB1A0551

C.Sujana Sree 24KB1A0578

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the **NBKR INSTITUTE OF SCIENCE & TECHNOLOGY ,**is a record of bonafied work carried out under my guidance and supervision.

|  |  |
| --- | --- |
| **Smt.B.SRUTHI** | **Dr.A.Rajasekhar Reddy**  **HOD**  **Dept.of CSE** |

**DECLARATION**

I hereby declare that the dissertation entitled **Library Management System** submitted for the B.Tech Degree is my original work and the dissertation has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

C.Sujana Sree-24KB1A0578

Place:Vidyanagar B.Sravya -24KB1A0564

Date: 05-05-2025 B.Sai Shivani -24KB1A0551

C.Deekshitha -24KB1A05B1

**ABSTRACT**

The Library Management System is a console-based application developed in the C programming language to automate the fundamental operations of a library. It enables efficient handling of book records and issuing transactions using structured programming concepts. The system allows users to add, delete, search, and display books, as well as issue and track issued books.

Books are stored in a fixed-size array, while issued books are managed using a dynamic linked list, demonstrating the practical use of both static and dynamic data structures in C. The user interacts with the system through a menu-driven interface, allowing for easy navigation and use.

This project provides a foundational understanding of data structures, memory management, and user input handling in C. While the system effectively handles basic functionalities, it can be enhanced further with features like file-based data persistence, user authentication, and a graphical user interface for broader usability.

**ACKNOWLEDGEMENT**

I would like to express my heartfelt gratitude to everyone who contributed to the successful completion of my project titled **"Library Management System".**

First and foremost, I am deeply thankful to my project guide**, Shivan raj sir** and my faculty guide ,**Smt.B.Sruthi mam**, for their constant guidance, insightful feedback, and continuous encouragement throughout the development of this project. Their mentorship helped me understand the practical aspects of programming and improved the overall quality of my work.

I would also like to thank the faculty and staff of the Computer Science &Engineering, NBKR Institute of science and technology, for providing the necessary academic environment and support that facilitated my learning and growth.

I am sincerely grateful to **my team mates** who provided helpful suggestions, shared ideas, and supported me during the various stages of the project.

**TABLE OF CONTENT**

**CHAPTER NO CHAPTER NAME PAGE.NO**

**CHAPTER 1: INTRODUCTION 1-2**

* 1. Introduction 1
  2. Problem Statement 1
  3. Scope 2
  4. Objective 2

**CHAPTER 2: LITERATURE SURVEY 3**

**CHAPTER 3: SOFTWARE REQUIREMENT ANALYSIS 4-5**

3.1 Software requirements 4

3.2 Functional requirement definitions 4

3.3 Nonfunctional requirement definitions 5

**CHAPTER 4: SOFTWARE DESIGN 6-9**

4.1 System Architecture 6

4.2Data structure used 7

4.3 Core functions 7

4.4 Control Flow 8

4.5 Design Considerations 9

**CHAPTER 5: PROPOSED SYSTEM 10-11**

5.1 System Overview 10

**CHAPTER 6: CODING 12-23**

6.1 Code 12-20

6.2 Function Explainations 20-23

**CHAPTER 7: 0UTPUT SCREENS 24-26**

**CHAPTER 8: CONCLUSION 27-28**

8.1 Conclusion 27

8.2 Future Work27-28

**TABLE OF FIGURES**

**FIGURE NO FIGURE NAME PAGE NO**

4.1.1 System architecture diagram 6

4.4.1 Control flow diagram 9

7.1 Main menu Screen 24

7.2 Add book interface 24

7.3 Search book output 25

7.4 Display all books 25

7.5 Issue book 25

7.6 Display issued books 26

7.7 Delete book interface 26

7.8 Exit 26

**1.INTRODUCTION**

**1.1.Introduction**

The **Library Management System** is a simple C-based console application designed to manage a small library's core operations. It provides functionalities to **add**, **delete**, **search**, **issue**, and **return** books using both an array and a linked list data structure.

* **Book details** are stored in a fixed-size array (library[]) using a Book structure that includes attributes like ID, title, author, and issue status.
* **Issued books** are managed separately using a singly linked list (IssuedBook), tracking which books are currently borrowed.

This project demonstrates fundamental concepts of **structured programming in C**, including:

* Structs
* Arrays
* Linked lists
* Dynamic memory allocation
* String handling
* Menu-driven user interface

**1.2 Problem Statement**

In educational institutions and small libraries, managing book records manually is a tedious and inefficient process. It often leads to challenges such as misplaced records, difficulty in tracking issued books, errors in availability status, and lack of a centralized system for book management. This not only affects productivity but also results in poor user experience for both librarians and readers.

To address these issues, there is a need for a simple, reliable, and user-friendly system that can automate the core functionalities of a library. These include adding new books, deleting outdated or damaged books (if not issued), searching for books by ID, issuing and returning books, and displaying current inventory and issued books.

This project aims to develop a **Library Management System using the C programming language**, which provides a menu-driven interface and utilizes fundamental data structures (arrays and linked lists) to handle book records effectively. The system will ensure data integrity, prevent invalid operations (e.g., issuing already borrowed books or deleting issued books), and streamline the overall management process in a small library environment.

**1.3 Scope**

The **Library Management System** project is designed to provide a basic yet functional solution for managing book records in a small library or academic setting. The scope of this project is limited to command-line (console-based) operations implemented in the C programming language and focuses on the following features:

**In-Scope Features:**

* Adding new books with unique IDs, titles, and authors.
* Deleting books from the library, provided they are not currently issued.
* Searching for books by their unique ID.
* Displaying all books in the library with their availability status.
* Issuing books and maintaining a record of all issued books using a linked list.
* Returning books and updating their status to available.
* Displaying a list of all currently issued books.
* Providing a menu-driven user interface for interaction.

**1.4 0bjective**

The main objective of the **Library Management System** is to simplify and automate the basic operations of a small library using the C programming language. This project aims to provide an efficient, user-friendly system that minimizes manual effort and errors in managing book records.

**Specific Objectives:**

* To design and implement a structured system for managing book information using arrays and linked lists.
* To allow librarians or users to add, delete, and search for books by their unique ID.
* To track the availability of books through a status indicator (Available / Issued).
* To maintain a dynamic record of issued books using a linked list data structure.
* To ensure that only available books can be issued, and issued books cannot be deleted.
* To provide a clear and intuitive menu-driven interface for user interaction.
* To demonstrate the use of fundamental C programming concepts such as structures, pointers, dynamic memory allocation, and user input handling.

This project serves as a foundational step toward building more advanced library systems and helps reinforce the understanding of core programming and data structure concepts.

**2.LITERATURE SURVEY**

The management of library resources has traditionally been carried out through manual record-keeping methods, involving registers, catalogs, and paper-based inventory systems. While these methods served their purpose in earlier times, they are increasingly becoming inefficient due to the growing volume of library resources and the demand for quick access and real-time updates.

With the advancement of technology, computerized library management systems have emerged, offering speed, accuracy, and better control over library operations. Numerous studies and existing systems demonstrate the need for automation in handling basic functions such as book addition, deletion, issuing, returning, and searching.

**Key Findings from Existing Systems and Studies:**

* **Manual Systems:** Often lead to data redundancy, difficulty in tracking issued books, and errors due to human intervention. They are not scalable for even moderately sized libraries.
* **Commercial Library Software (e.g., Koha, LibSys):** These are robust, feature-rich systems that include cataloging, member management, barcode scanning, and data persistence through databases. However, they are often complex and overkill for small libraries or educational projects.
* **Academic Projects in C/C++:** Numerous student-level projects use procedural languages like C for implementing small-scale management systems. These projects typically focus on fundamental operations using data structures such as arrays and linked lists. However, many of them lack persistent storage and are limited to console interfaces.

This project builds upon those simpler models by focusing on clarity, usability, and demonstrating the use of **data structures in C**, specifically arrays for book storage and linked lists for tracking issued books. It is tailored for academic purposes, offering a hands-on learning opportunity in programming, memory management, and basic system design.

By reviewing similar academic projects and understanding the limitations of both manual systems and high-end library software, this project aims to strike a balance by providing a minimal yet effective solution for small libraries or learning environments.

**3.SOFTWARE REQUIREMENT ANALYSIS**

This section outlines the software requirements necessary for the development and functioning of the **Library Management System**. The system is designed as a console-based application in the C programming language, focusing on core library operations for small-scale usage or academic learning.

| **Category** | **Requirement** |
| --- | --- |
| **Hardware** |  |
| Processor | Intel Pentium IV or higher / AMD equivalent |
| RAM | Minimum 512 MB (1 GB or more recommended) |
| Storage | Minimum 100 MB of free disk space |
| Display | Standard VGA or higher resolution monitor |
| Input Devices | Keyboard and Mouse |
| **3.1 Software** |  |
| Operating System | Windows XP/7/10, Linux (Ubuntu, Fedora), or macOS |
| Compiler | GCC (GNU Compiler Collection), MinGW (for Windows) |
| IDE (Optional) | Code::Blocks, Dev-C++, Turbo C++, or Visual Studio Code |
| Libraries Used | Standard C Library (stdio.h, stdlib.h, string.h) |

**3.2 Functional Requirements**

Functional requirements define the core operations that the system must perform to meet its intended purpose.

* **FR1:** The system shall allow the user to add new books with a unique ID, title, and author.
* **FR2:** The system shall allow the user to delete books, only if they are not currently issued.
* **FR3:** The system shall allow the user to search for a book by its unique ID and display its details.
* **FR4:** The system shall display a list of all books along with their current availability status.
* **FR5:** The system shall allow the user to issue a book only if it is currently available.
* **FR6:** The system shall track issued books using a linked list data structure.
* **FR7:** The system shall allow the user to return a book and update its status to "Available."
* **FR8:** The system shall display a list of all currently issued books.
* **FR9:** The system shall provide a menu-driven interface for navigation and operation.

**3.3 Non-Functional Requirements**

Non-functional requirements define the quality attributes and constraints of the system.

* **NFR1 (Performance):** The system shall respond to user inputs promptly, ensuring minimal delay during operations.
* **NFR2 (Usability):** The system shall have a simple, text-based interface that is easy to navigate and operate by users with basic computer knowledge.
* **NFR3 (Portability):** The system shall be portable and executable on any system with a C compiler (e.g., GCC).
* **NFR4 (Maintainability):** The code shall be modular and well-commented to facilitate easy understanding and maintenance.
* **NFR5 (Scalability):** The system shall support up to 100 books, which is sufficient for a small library or learning environment.
* **NFR6 (Data Volatility):** The system does not provide data persistence. All data is stored in memory and will be lost upon program termination.

**4.SOFTWARE DESIGN**

The Library Management System is designed using a modular and structured programming approach in the C language. The design emphasizes clarity, maintainability, and the practical use of fundamental data structures such as arrays and linked lists to manage library operations effectively.

**4.1 System Architecture**

The system is divided into several functional modules, each responsible for a specific task:

* Book Management Module: Handles operations like adding, deleting, searching, and displaying books using an array of Book structures.
* Issue Management Module: Manages book issuance and returns using a singly linked list (IssuedBook structure) to store IDs of currently issued books.
* User Interface Module: Provides a menu-driven interface that accepts user input and calls appropriate functions based on the selected operation.

**4.1.1 SYSTEM ARCHITECTURE DIAGRAM**

+------------------------+

| User Interface | <==> Console (Menu-based Input/Output)

+------------------------+

|

+------------------------+

| Application Logic | <-- Core Operations: Add, Delete, Search, Issue, Return

| (C Functions & Menu) |

+------------------------+

|

+------------------------+ +--------------------------+

| Data Storage | <------> | Issued Books List |

| (Book Array - Static) | | (Linked List - Dynamic)|

+------------------------+ +--------------------------+

**4.2 Data Structures Used**

* Book Structure (struct Book):

struct Book {

int id;

char title[100];

char author[100];

int isIssued; // 0 = Available, 1 = Issued

};

Stored in a fixed-size array (library[MAX\_BOOKS]).

Tracks all available and issued books.

* Issued Book Structure (struct IssuedBook):

struct IssuedBook {

int id;

struct IssuedBook\* next;

};

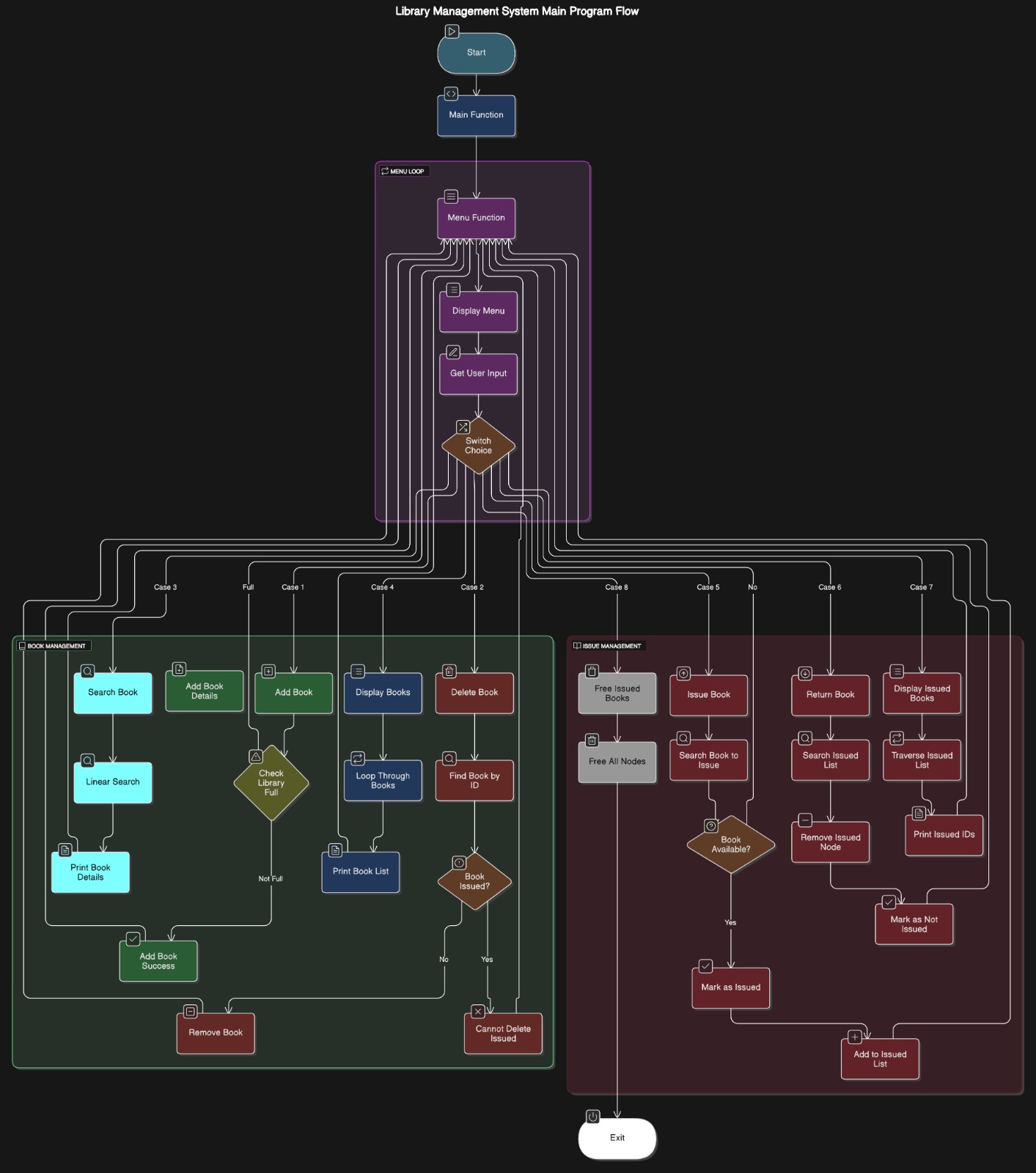
* + Implements a singly linked list to store issued book IDs dynamically.
  + Helps track and manage returned books without modifying the main array.

**4.3 Core Functions**

| Function Name | Purpose |
| --- | --- |
| addBook() | Adds a new book to the library. |
| deleteBook() | Deletes a book if it is not issued. |
| searchBook() | Searches for a book by ID and displays its details. |
| displayBooks() | Displays all books with ID, title, author, and status. |
| issueBook() | Issues a book and adds it to the issued list. |
| returnBook() | Returns a book and removes it from the issued list. |
| displayIssuedBooks() | Displays all currently issued books. |
| freeIssuedBooks() | Frees dynamically allocated memory before exiting. |
| menu() | Provides the main interface for user interaction. |

**4.4 Control Flow**

1. Program starts and displays a menu.
2. User selects an option (e.g., add, delete, issue).
3. The corresponding function is called.
4. Control returns to the menu until the user chooses to exit.
5. Upon exit, any dynamically allocated memory for issued books is freed.

****

**4.4.1 CONTROL FLOW DIAGRAM**

**4.5 Design Considerations**

* Modularity: Each function handles a specific task, improving readability and maintainability.
* Data Integrity: Issued books are protected from deletion; a book can’t be issued twice.
* Efficiency: Book lookup is linear (array traversal); sufficient for small libraries.
* Scalability Limit: Maximum of 100 books; adjustable via #define MAX\_BOOKS.

**5.PROPOSED SYSTEM**

The proposed Library Management System is a command-line based application developed in C to manage a small-scale library's inventory and issued book records. The system is modular in nature and encapsulates each core functionality into discrete, manageable components, thereby ensuring scalability, readability, and ease of maintenance.

**5.1 System Overview**

The system uses:

* A **static array** to store book records.
* A **singly linked list** to manage issued books.
* A **menu-driven interface** for user interaction.

**5.1.1 Book Inventory Module**

**Objective:** Manage library book records including creation, deletion, viewing, and search operations.

**Key Functionalities:**

* **Add Book:** Inserts a new book into the array with unique ID, title, and author. Prevents addition if the inventory is full.
* **Delete Book:** Removes a book by ID only if it is not currently issued.
* **Display All Books:** Lists all book entries with ID, title, author, and availability status.
* **Search Book:** Locates and displays book details using its ID.

**5.1.2 Issuance Management Module**

**Objective:** Handle issuing and returning of books using a linked list to track issued book IDs.

**Key Functionalities:**

* **Issue Book:** Checks if the book is available and not already issued. If valid, marks it as issued and adds the ID to the linked list.
* **Return Book:** Searches the linked list for the book ID, removes it, and updates its status to available.
* **Display Issued Books:** Traverses the linked list to show all currently issued book IDs.

**5.1.3 User Interface (Menu) Module**

**Objective:** Provide an interactive interface for end-users to navigate system features.

**Key Functionalities:**

* Presents a clear menu with all available operations.
* Handles user inputs, validates them, and dispatches control to corresponding modules.
* Includes input validation to handle invalid or unexpected user actions.
* Repeats the menu after each operation until the user chooses to exit.

**5.1.4 Memory Management Module**

**Objective:** Ensure efficient memory usage and cleanup for dynamically allocated data.

**Key Functionalities:**

* Frees all dynamically allocated memory used for the issued books linked list upon exit.
* Prevents memory leaks and ensures system stability during shutdown.

**Advantages of the Proposed System**

* **Modular Architecture:** Clearly separated functionalities enhance maintainability and allow independent module testing.
* **Efficient Data Handling:** Combines fixed-size and dynamic data structures for optimal performance in both book lookup and issued tracking.
* **Scalability:** Can be extended with features like user accounts, categories, or persistent file storage.
* **Robust Error Handling:** Checks for edge cases like issuing already issued books or deleting non-existent entries.
* **User-Friendly Interface:** Simple and intuitive text-based interaction with clear prompts and messages.

**6.CODING**

**6.1 CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_BOOKS 100

// Book structure for the array

struct Book {

int id;

char title[100];

char author[100];

int isIssued;

};

// Linked list node for issued books

struct IssuedBook {

int id;

struct IssuedBook\* next;

};

// Global array of books

struct Book library[MAX\_BOOKS];

int bookCount = 0;

// Head of issued books linked list

struct IssuedBook\* issuedHead = NULL;

// Add a book to the array

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

if (bookCount >= MAX\_BOOKS) {

printf("Library is full. Cannot add more books.\n");

return;

}

library[bookCount].id = id;

strcpy(library[bookCount].title, title);

strcpy(library[bookCount].author, author);

library[bookCount].isIssued = 0;

bookCount++;

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

}

// Delete a book from the array

void deleteBook(int id) {

int found = 0;

for (int i = 0; i < bookCount; i++) {

if (library[i].id == id) {

if (library[i].isIssued) {

printf("Cannot delete book with ID %d as it is currently issued.\n", id);

return;

}

found = 1;

for (int j = i; j < bookCount - 1; j++) {

library[j] = library[j + 1];

}

bookCount--;

printf("Book with ID %d deleted successfully.\n", id);

break;

}

}

if (!found) {

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

}

}

// Search for a book by ID

void searchBook(int id) {

for (int i = 0; i < bookCount; i++) {

if (library[i].id == id) {

printf("\nBook Found:\n");

printf("ID: %d\n", library[i].id);

printf("Title: %s\n", library[i].title);

printf("Author: %s\n", library[i].author);

printf("Status: %s\n", library[i].isIssued ? "Issued" : "Available");

return;

}

}

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

}

// Display all books

void displayBooks() {

if (bookCount == 0) {

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

return;

}

printf("\nLibrary Book List:\n");

printf("-------------------------------------\n");

printf("ID | Title | Author | Status\n");

printf("-------------------------------------\n");

for (int i = 0; i < bookCount; i++) {

printf("%-4d | %-15s | %-10s | %-8s\n",

library[i].id,

library[i].title,

library[i].author,

library[i].isIssued ? "Issued" : "Available");

}

printf("-------------------------------------\n");

}

// Issue a book

void issueBook(int id) {

for (int i = 0; i < bookCount; i++) {

if (library[i].id == id && !library[i].isIssued) {

struct IssuedBook\* newIssued = (struct IssuedBook\*)malloc(sizeof(struct IssuedBook));

newIssued->id = id;

newIssued->next = issuedHead;

issuedHead = newIssued;

library[i].isIssued = 1;

printf("Book with ID %d issued successfully.\n", id);

return;

}

}

printf("Book not available or already issued.\n");

}

// Return/Unissue a book

void returnBook(int id) {

struct IssuedBook\* temp = issuedHead;

struct IssuedBook\* prev = NULL;

while (temp != NULL) {

if (temp->id == id) {

if (prev == NULL) {

issuedHead = temp->next;

} else {

prev->next = temp->next;

}

free(temp);

for (int i = 0; i < bookCount; i++) {

if (library[i].id == id) {

library[i].isIssued = 0;

break;

}

}

printf("Book with ID %d returned successfully.\n", id);

return;

}

prev = temp;

temp = temp->next;

}

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

}

// Display issued books

void displayIssuedBooks() {

struct IssuedBook\* temp = issuedHead;

if (!temp) {

printf("No books have been issued.\n");

return;

}

printf("\nIssued Book List:\n");

while (temp != NULL) {

printf("ID: %d\n", temp->id);

temp = temp->next;

}

}

// Free allocated memory before exiting

void freeIssuedBooks() {

struct IssuedBook\* temp;

while (issuedHead != NULL) {

temp = issuedHead;

issuedHead = issuedHead->next;

free(temp);

}

}

// Menu

void menu() {

int choice, id;

char title[100], author[100];

while (1) {

printf("\nLibrary Management System\n");

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

printf("2. Delete Book\n");

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

printf("4. Display All Books\n");

printf("5. Issue Book\n");

printf("6. Return Book\n");

printf("7. Display Issued Books\n");

printf("8. Exit\n");

printf("Enter choice: ");

if (scanf("%d", &choice) != 1) {

printf("Invalid input! Please enter a number.\n");

while (getchar() != '\n'); // Clear input buffer

continue;

}

switch (choice) {

case 1:

printf("Enter Book ID: ");

scanf("%d", &id);

getchar();

printf("Enter Title: ");

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

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

printf("Enter Author: ");

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

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

addBook(id, title, author);

break;

case 2:

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

scanf("%d", &id);

deleteBook(id);

break;

case 3:

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

scanf("%d", &id);

searchBook(id);

break;

case 4:

displayBooks();

break;

case 5:

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

scanf("%d", &id);

issueBook(id);

break;

case 6:

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

scanf("%d", &id);

returnBook(id);

break;

case 7:

displayIssuedBooks();

break;

case 8:

freeIssuedBooks();

printf("Exiting...\n");

exit(0);

default:

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

}

}

}

int main() {

menu();

return 0;

}

**6.2 EXPLAINATION**

**Book Management (Array-based)**

**Structure: struct Book**

Represents a book in the library.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | int | Unique identifier for a book |
| title | char[100] | Title of the book |
| author | char[100] | Author of the book |
| isIssued | int | 1 if issued, 0 if available |

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

* **Purpose**: Adds a new book to the library[] array.
* **Inputs**:
  + id: Unique identifier
  + title: Title string
  + author: Author string
* **Output**: None
* **Effect**: Updates library[] and increments bookCount.

**Function: void deleteBook(int id)**

* **Purpose**: Deletes a book if it exists and is not issued.
* **Input**:
  + id: ID of the book to delete
* **Output**: None
* **Effect**: Shifts array elements and decrements bookCount.

**Function: void searchBook(int id)**

* **Purpose**: Searches and displays book details by ID.
* **Input**:
  + id: ID of the book to find
* **Output**: None
* **Effect**: Displays book info or "not found" message.

**Function: void displayBooks()**

* **Purpose**: Displays all books in the library.
* **Input**: None
* **Output**: None
* **Effect**: Prints a formatted list of all books.

**Issued Book Management (Linked List)**

**Structure: struct IssuedBook**

Represents a node in the linked list of issued books.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | int | Book ID that is issued |
| next | struct IssuedBook\* | Pointer to next node |

**Function: void issueBook(int id)**

* **Purpose**: Marks a book as issued and adds its ID to the issued book list.
* **Input**:
  + id: Book ID to issue
* **Output**: None
* **Effect**: Sets isIssued = 1 and appends to issuedHead list.

**Function: void returnBook(int id)**

* **Purpose**: Returns a previously issued book.
* **Input**:
  + id: Book ID to return
* **Output**: None
* **Effect**: Removes node from issuedHead and updates isIssued = 0.

**Function: void displayIssuedBooks()**

* **Purpose**: Displays a list of all issued book IDs.
* **Input**: None
* **Output**: None
* **Effect**: Traverses and prints the linked list issuedHead.

**Function: void freeIssuedBooks()**

* **Purpose**: Frees memory used by the issued book list.
* **Input**: None
* **Output**: None
* **Effect**: Deallocates all IssuedBook nodes.

**User Interface**

**Function: void menu()**

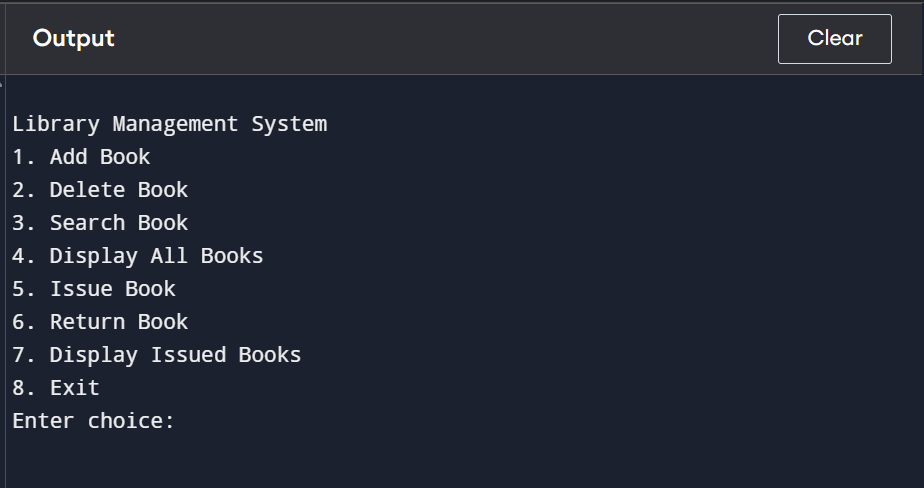
* **Purpose**: Provides a text-based interactive menu.
* **Input**: User choice from 1–8
* **Output**: None
* **Effect**: Calls relevant functions based on user selection:
  + addBook()
  + deleteBook()
  + searchBook()
  + displayBooks()
  + issueBook()
  + returnBook()
  + displayIssuedBooks()
  + exit() (also calls freeIssuedBooks())

**Function: int main()**

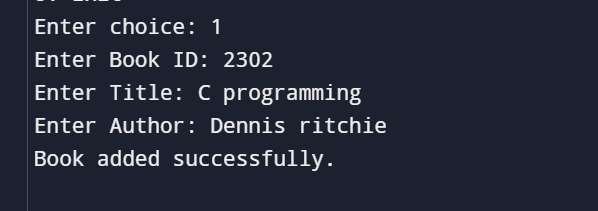
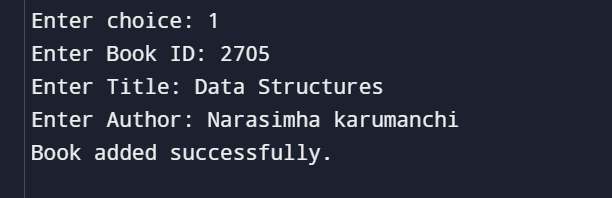
* **Purpose**: Program entry point.
* **Input**: None
* **Output**: 0 (standard successful termination)
* **Effect**: Calls menu().

**7.OUTPUT SCREENS**

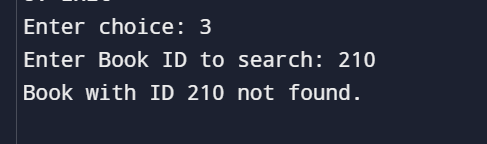
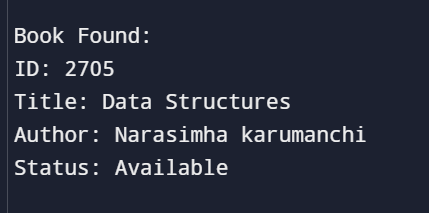
**7.1 Main Menu Screen**

****

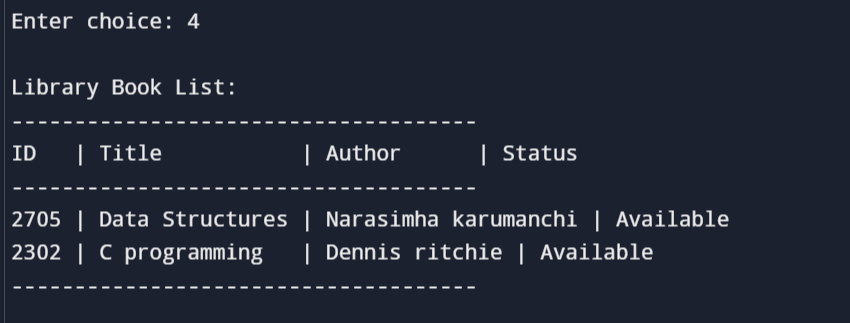
**7.2 Add Book Interface**

****

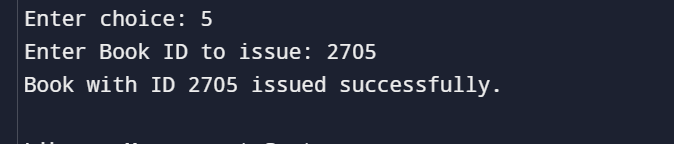
**7.3 Search Book output**

****

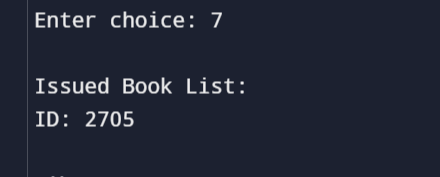
**7.4 Display all books**

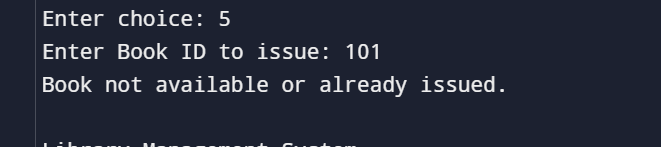
****

**7.5 Issue book**

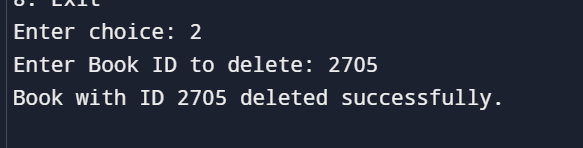
****

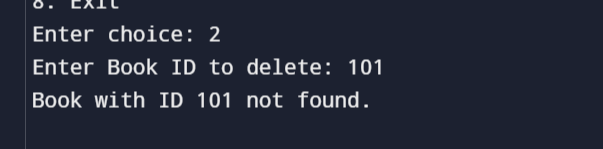
**7.6 Display issued books**

****

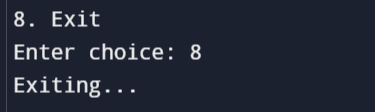
****

**7.7 Delete book interface**

****

****

**7.8 Exit**

****

**8.CONCLUSION**

**8.1 CONCLUSION**

The Library Management System implemented in C successfully demonstrates the core functionalities required for managing a small-scale digital library. By leveraging a structured array for storing book data and a linked list for handling issued books, the system maintains efficiency while ensuring data consistency. It provides features for adding, deleting, searching, displaying, issuing, and returning books—all through a user-friendly, menu-driven interface.

The project reinforces core programming concepts including structures, dynamic memory allocation, file-less data persistence (in-memory), and modular code organization. This system is especially well-suited for academic environments or small community libraries where basic book tracking is essential and resource constraints exist.

**8.2 Future Work**

While the current system meets basic requirements, there are several areas for enhancement:

1. **Persistent Storage**
   * Implement file I/O to save and load the library and issued book records across sessions.
2. **User Authentication**
   * Add user roles such as Admin and Member with login credentials for secure access control.
3. **Search Improvements**
   * Extend search functionality to include searching by title or author, not just by ID.
4. **Enhanced Interface**
   * Develop a GUI using libraries like GTK+ or port the system to a web-based interface for better usability.
5. **Data Validation & Error Handling**
   * Improve input validation (e.g., check for duplicate IDs, special characters in titles).
6. **Sorting & Filtering**
   * Add features to sort books by title, author, or status, and filter issued/available books.
7. **Barcode or QR Integration**
   * Integrate barcode/QR code scanning for quicker book issue/return operations.
8. **Database Integration**
   * Migrate the backend to use SQLite or MySQL for scalability and better data management.