**N.B.K.R INSTITUTE OF SCIENCE AND TECHNOLOGY**

VIDYANAGAR- TIRUPATHI

**LIBRARY MANAGEMENT USING**

**LINKED LIST**

**Course:** Data Structures

**Department:** Computer Science

**Section:** F

**Year:** Ⅰ

**Semester:** Ⅱ

**Date:** 05-05-2025

**Submitted To:** Ashok Selva Kumar

**Submitted by:**

**E.Madhuri-(24KB1A05F1)**

**G.Ananya-(24KB1A05G5)**

**Acknowledgement:**

This program demonstrates the implementation of a basic Library Management System using fundamental data structures in C. It utilizes a singly linked list to manage a collection of books, allowing users to add, delete, search, and display books based on their unique ID. Each book is represented as a node in the list, and the system ensures efficient memory management and user interaction through a menu-driven interface**.**

Special acknowledgment goes to the effective use of:

* Dynamic Book Management:  
  Each book record (with ID, title, and author) is dynamically allocated using malloc, ensuring efficient use of memory.
* Singly Linked List Implementation:  
  All book records are maintained in a linked list, providing flexibility for insertions and deletions without reallocation.
* CRUD Operations:  
  The system allows:
* Create – Add a new book to the library.
* Read – Search for a book by ID or view all books.
* Update – (Not implemented in current version, but can be added).
* Delete – Remove a book using its ID.
* User Interaction:  
  A menu-driven interface guides users with clear prompts and feedback, enhancing usability.
* Input Validation and String Handling:  
  Care is taken to handle newline characters and input buffer issues using functions like fgets() and strcspn().

**Abstract:**

The Library Management System implemented in the C programming language is a console-based application designed to manage basic operations in a library environment. This system allows users to add, delete, search, and display books stored in memory, using a singly linked list data structure. Each book is uniquely identified by an ID and includes a title and author name as attributes.

The program is structured using modular functions to promote clarity and maintainability. Dynamic memory allocation is used for efficient management of book records at runtime. The user interacts with the system through a menu-driven interface, making the application straightforward and intuitive to operate.

This project demonstrates core programming concepts in C such as structures, pointers, dynamic memory management, and linked list manipulation. It is intended for educational purposes and serves as a foundational exercise for learners exploring data structures and system-level programming.

Although the current implementation stores data in memory only, it can be enhanced further by integrating file I/O for persistent storage, sorting algorithms, and even a graphical interface for better user experience.

**Introduction:**

Libraries play a vital role in organizing and maintaining collections of books, making them easily accessible to readers and administrators. Managing these collections efficiently, especially in digital or computerized systems, requires structured and logical programming approaches. This project presents a Library Management System developed in the C programming language, designed to simulate basic library operations such as adding, deleting, searching, and displaying book records.

The program uses a singly linked list to store and manage book data dynamically in memory. Each book is represented as a node in the linked list and contains three key attributes: Book ID, Title, and Author. By leveraging linked lists, the system ensures efficient handling of insertions and deletions without the need for memory reallocation or shifting elements, as would be required in an array-based approach.

This application provides a menu-driven interface, making it user-friendly and interactive. Users can continuously perform operations until they choose to exit. The program also demonstrates key programming concepts such as:

* Structures for organizing related data,
* Pointers for dynamic memory management,
* String handling, and
* Modular function-based design for code clarity and reuse.

**Objectives:**

The primary goal of this project is to design and implement a Library Management System using the C programming language that can perform essential operations on a collection of books. The key objectives of the project are:

1. To understand and apply data structures in C
   * Implement a singly linked list to manage a dynamic collection of book records efficiently.
2. To design a functional and user-friendly interface
   * Create a menu-driven program that allows users to perform actions like adding, deleting, searching, and viewing books with ease.
3. To develop a modular and maintainable codebase
   * Use functions to separate logic into reusable and manageable parts, improving readability and scalability of the program.
4. To practice dynamic memory allocation and deallocation
   * Utilize malloc() and free() to manage memory effectively during runtime, preventing memory leaks and optimizing resource use.
5. To enhance understanding of pointers and structures
   * Use structures to represent complex data (Book) and pointers to manipulate nodes in a linked list.
6. To simulate real-world library operations
   * Model a simplified version of a library system that mimics core tasks such as catalog maintenance and book lookup.

**System requirements:**

Hardware Requirements

* Processor: Pentium Iv Or Higher
* Ram: Minimum 512 Mb
* Storage: At Least 10 Mb Free Space

Software Requirements

* Operating System: Windows / Linux / Macos
* Compiler: Gcc / Turbo C++ / Clang
* Editor/Ide: Vs Code / Code::Blocks / Dev-C++

**Methodology:**

The development of the Library Management System followed a structured and step-by-step approach to ensure proper planning, implementation, and testing. The methodology used is outlined below:

1. Requirement Analysis
   * Identified core functionalities needed: Add, Delete, Search, and Display books.
   * Determined suitable data structure (singly linked list) for dynamic book management.
2. Design
   * Defined a Book structure with id, title, author, and a pointer to the next node.
   * Designed a menu-driven user interface for interaction.
3. Implementation
   * Implemented the core features:
     + addBook() to insert a book node at the end.
     + deleteBook() to remove a book by ID.
     + searchBook() to find and display book details.
     + displayBooks() to show all books in the list.
   * Used functions and modular coding to maintain clarity.
4. Testing and Debugging
   * Performed manual testing for each function using various inputs.
   * Checked edge cases such as:
     + Deleting/searching from an empty list.
     + Adding multiple books.
     + Searching for non-existent IDs.
5. Optimization and Refinement
   * Cleaned up code for readability and maintainability.
   * Handled string inputs properly to avoid buffer overflow and newline issues.
6. Documentation
   * Created comments and supporting documentation to explain each part of the code.

**Project Description:**

**Problem Statement**

In many traditional libraries, managing a collection of books, keeping track of book details, and performing operations like adding, removing, or searching for books can become complex and time-consuming, especially when handled manually. Maintaining accurate records of a large number of books without an efficient system can result in errors, inefficiency, and difficulty in retrieving or updating data. A computerized library management system is needed to streamline and simplify these tasks.

**Proposed Solution**

The Library Management System proposed in this project is designed to solve the problem of inefficient book management by utilizing a singly linked list to dynamically store and manage book records. This system allows users to easily perform operations such as adding, deleting, searching, and displaying books through a simple, interactive, menu-driven interface. The use of a linked list ensures that the system can efficiently handle dynamic memory allocation and deallocation, making it adaptable for libraries of varying sizes.

**Key Features**

* Dynamic Book Management:  
  Books are stored dynamically in memory using a singly linked list, allowing efficient insertion and deletion without the need for reallocation or shifting elements.
* Add Book:  
  Users can add a book by providing a unique book ID, title, and author, which is then appended to the linked list.
* Delete Book:  
  A book can be deleted from the system using its unique ID. The system handles edge cases such as empty lists or non-existent IDs.
* Search Book:  
  Users can search for a book by its ID, and if found, the book’s details (ID, title, and author) are displayed.
* Display All Books:  
  The system can display a list of all the books currently stored in the library, showing their IDs, titles, and authors.
* User-Friendly Interface:  
  A menu-driven interface allows users to interact with the system intuitively, providing easy navigation between operations.
* Modular and Maintainable Code:  
  The system is implemented with modular functions, making the code clean, maintainable, and easy to extend with additional features (e.g., file handling, sorting).

**Flow chart:**

**Algorithm:**

Step 1:Start program.  
Step 2: Initialize empty linked list.  
Step 3: Display menu options.  
Step 4: Get user choice.  
Step 5: Perform action based on choice:

* Add Book: Input details and add to list
* Delete Book: Input ID and delete if found
* Search Book: Input ID and display if found
* Display Books: Show all books in list
* Exit: End program

Step 6: Repeat from Step 3 until Exit.  
Step 7: Stop

**Program code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure for a Book

struct Book {

int id;

char title[100];

char author[100];

struct Book\* next;

};

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 at end

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");

}

// Delete book by ID

void deleteBook(int id) {

struct Book\* temp = head;

struct Book\* 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 with ID %d deleted successfully.\n", id);

}

// Search for a book by ID

void searchBook(int id) {

struct Book\* temp = head;

while (temp != NULL) {

if (temp->id == id) {

printf("Book Found:\n");

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

printf("Title: %s\n", temp->title);

printf("Author: %s\n", temp->author);

return;

}

temp = temp->next;

}

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

}

// Display all books

void displayBooks() {

struct Book\* temp = head;

if (temp == NULL) {

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

return;

}

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

while (temp != NULL) {

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

temp = temp->next;

}

}

// 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. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter Book ID: ");

scanf("%d", &id);

getchar(); // Consume newline

printf("Enter Title: ");

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

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

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("Exiting...\n");

exit(0);

default:

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

}

}

}

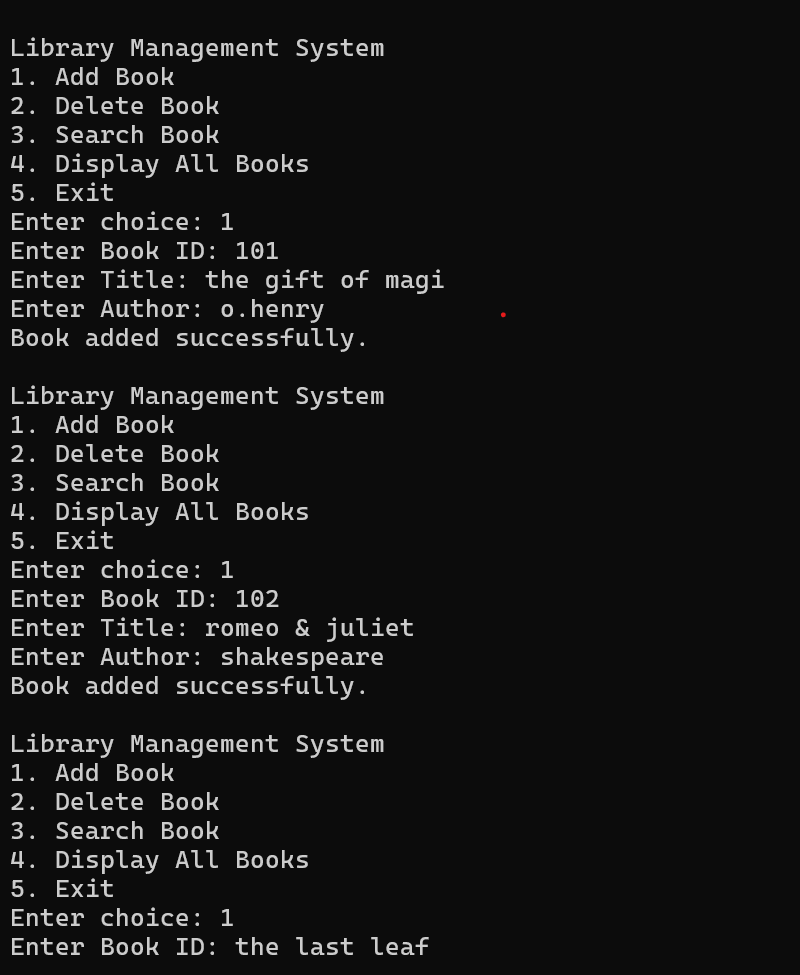
int main() {

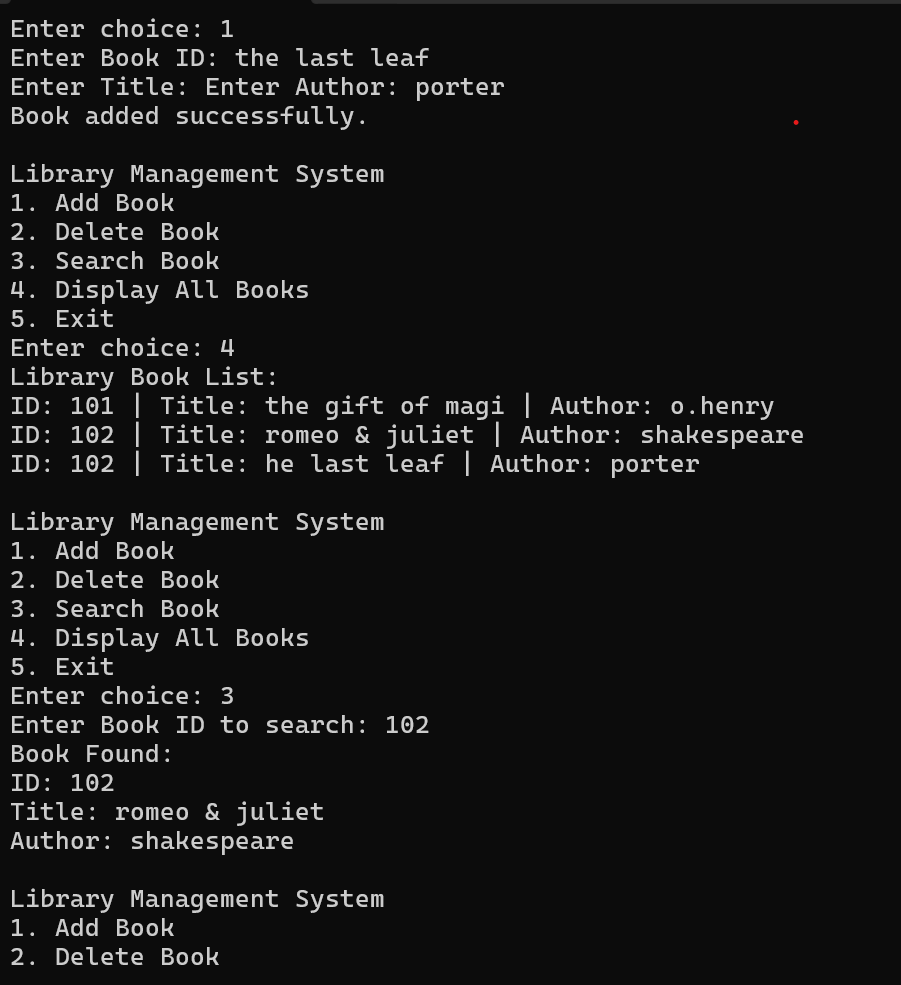
menu();

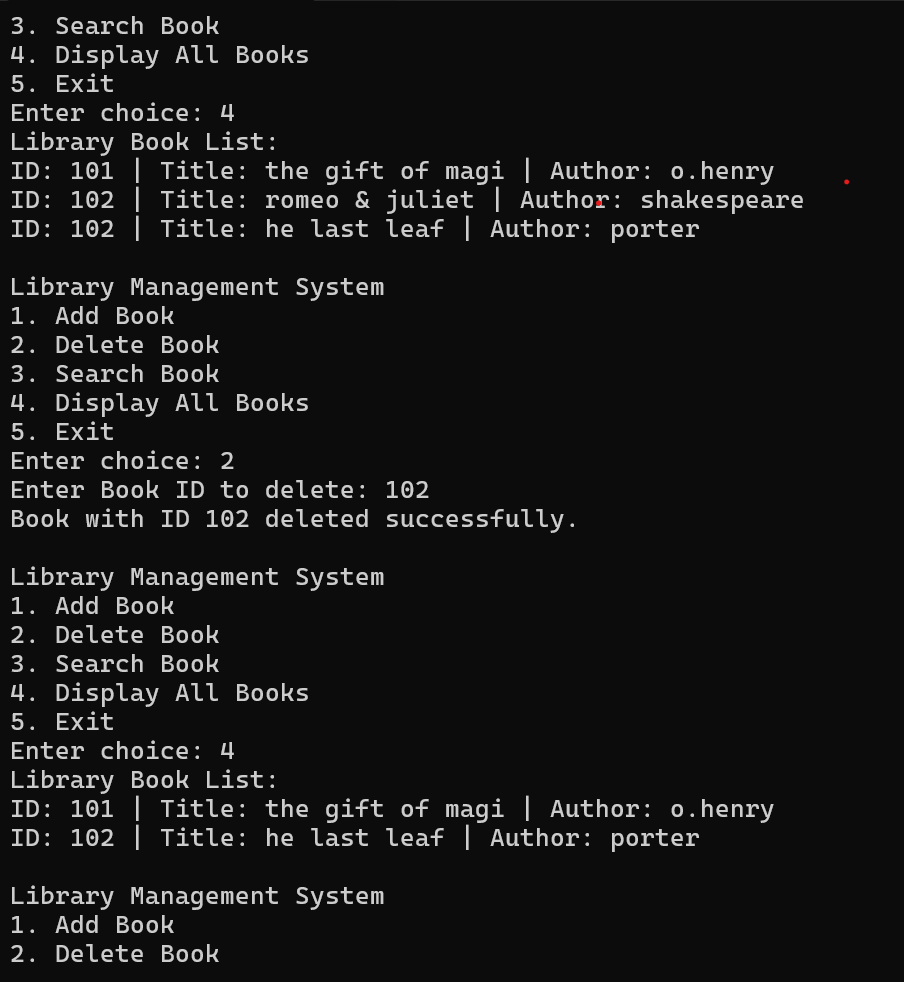
return 0;

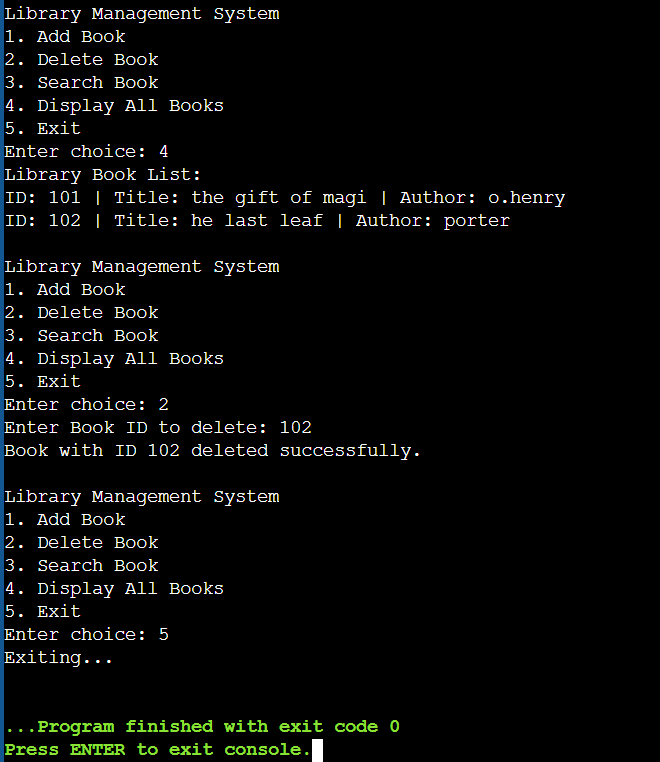
}

**Output:**

****







**Testing &validation:**

Library Management System

1. Add Book

2. Delete Book

3. Search Book

4. Display All Books

5. Exit

Enter choice: 1

Enter Book ID: 101

Enter Title: C Programming

Enter Author: Dennis Ritchie

Book added successfully.

Library Management System

1. Add Book

2. Delete Book

3. Search Book

4. Display All Books

5. Exit

Enter choice: 4

Library Book List:

ID: 101 | Title: C Programming | Author: Dennis Ritchie

Library Management System

1. Add Book

2. Delete Book

3. Search Book

4. Display All Books

5. Exit

Enter choice: 3

Enter Book ID to search: 101

Book Found:

ID: 101

Title: C Programming

Author: Dennis Ritchie

Library Management System

1. Add Book

2. Delete Book

3. Search Book

4. Display All Books

5. Exit

Enter choice: 2

Enter Book ID to delete: 101

Book with ID 101 deleted successfully.

Library Management System

1. Add Book

2. Delete Book

3. Search Book

4. Display All Books

5. Exit

Enter choice: 4

No books in the library.

Library Management System

1. Add Book

2. Delete Book

3. Search Book

4. Display All Books

5. Exit

Enter choice: 5

Exiting...

**Limitations:**

* Data is lost when the program exits, as there is no mechanism to save book data to a file or database.
* The system allows books with duplicate IDs, which can create confusion.
* The system can only search for books by ID, with no option to search by Title or Author.
* User input is not validated, allowing invalid or empty data for book title, author, or ID.
* The system may face performance issues when handling a large number of books in memory.
* The program doesn't handle memory allocation failures when malloc can't allocate memory.
* There is no user login system to restrict access to book management operations.
* Once a book is added, its details cannot be modified.
* Books are displayed in the order they are added, with no sorting options by ID, Title, or Author.

**Future enhancements:**

Top of Form

* Add file handling to save and load book data between sessions for data persistence.
* Implement user authentication to restrict access to sensitive operations like adding or deleting books.
* Add input validation to ensure valid book IDs and non-empty title/author fields.
* Prevent duplicate book IDs by checking existing records before adding a new book.
* Introduce a feature to update book details (title, author, or ID).
* Add the ability to search books by Title or Author in addition to ID.
* Implement sorting options to display books in order by ID, Title, or Author.
* Add confirmation prompts before deleting a book to avoid accidental deletions.
* Improve the interface with a GUI or web-based frontend for better usability.

**Conclusion:**

The Library Management System is a command-line based application developed in C that effectively showcases the fundamental principles of data structures and dynamic memory allocation using linked lists. It provides users with essential operations such as adding new books, deleting existing ones, searching for specific records by ID, and displaying all the stored books in an organized manner. The system is designed to be simple, interactive, and easy to use.

Throughout the development of this project, key programming concepts such as structures, pointers, dynamic memory management, and user input handling were practically applied. The modular design of the program ensures clarity and ease of future upgrades or maintenance.

Although the system meets its basic goals, it also highlights areas for growth. Enhancements like file storage for data persistence, more comprehensive search and update functionalities, validation of user inputs, and user authentication could greatly increase its usefulness in real-world scenarios. With such additions, this project can evolve into a more complete and reliable software solution for managing a real library.

In conclusion, the project not only achieves its intended functionality but also serves as a strong learning tool for understanding structured programming and software development practices in C.

Bottom of Form

**Reference:**

1. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2011). *Data Structures and Algorithms in C*. Wiley.
2. Sedgewick, R. (1998). *Algorithms in C* (Parts 1–5). Addison-Wesley.
3. Miller, B. N., & Ranum, D. L. (2011). *Problem Solving with Algorithms and Data Structures*. Franklin, Beedle & Associates Inc.
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
5. Levitin, A. (2012). *Introduction to the Design and Analysis of Algorithms*. Pearson Education**.**