Book Management Database system

SAIDA SUNAINA

Advisor: **Mr. Sahabzada Betab Badar**

Department of Computer Science and Information Technology
Jain(Deemed-to-be)university

This report is submitted for the course of
Programming in C (24BCA1CO4)

Jain university, Bengaluru **December 2024**

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this report are original and have not been submitted in whole or in part for consideration for any other degree or qualification or course or any other university. This report is the collective work of me and my team.

SAIDA SUNAINA
USN No: JUUG24BCAS20675
Department of Computer Science and Information Technology,
Jain (Deemed-to-be) University, Bengaluru.

Acknowledgement

I express my deep sense of gratitude to the management of our university for giving us an opportunity to do this project. I would like to express my sincere gratitude to Mr. Sahabzada Betab Badar for his unwavering support, guidance, patience and constructive feedback throughout the course of this project. His expertise and encouragement have been instrumental in shaping the direction and quality of this work. I would also like to extend my heartfelt thanks to my teammates for their collaboration, hard work, and dedication. Their contributions and teamwork were essential to the success of this project, and I am truly grateful for their constant support and commitment. Thank you to everyone who contributed to the success of this endeavour.

Abstract

The Book Management Database System is a mini project developed in C programming that aims to automate and simplify the process of managing books in a library or bookstore environment. The system allows users to add, update, delete, and search for books based on various criteria such as title, author, and ID number. It also supports features for viewing the current inventory. The database is implemented using file handling in C to store and retrieve book information efficiently. This project is designed to enhance the organisation and management of book-related data, providing a user-friendly interface and promoting easy access to book records. The system demonstrates core programming concepts such as arrays, functions, file operations, and search algorithms, making it an ideal solution for small-scale book management tasks.

Table of Contents

LIST OF FIGURES:

1. Data Structure:
    - Book Structure: The Book structure holds information about each book, with three members:
        - title: A string to store the title of the book.
        - author: A string to store the author's name.
        - year: An integer to store the publication year of the book.
2. Global Variables:
    - library: An array of Book structures that can hold up to MAX_BOOKS books.
    - book Count: An integer variable that keeps track of the number of books currently in the library.
3. Functions:
    - addBook(): This function adds a new book to the library. It checks if the library has space, takes input for the title, author, and year of the book, and then increments the book Count.
    - viewLibrary(): This function displays all books currently stored in the library. It iterates through the library array and prints the details of each book (title, author, year).
    - searchBook(): This function allows the user to search for a book by its title. It compares the entered title with the titles in the library and prints the details of the found book if there is a match.
4. Main Program Flow:
    - Menu-driven loop: A do-while loop that continuously prompts the user with a menu of options until they choose to exit. The user can add books, view the library, or search for books using corresponding numbers.
    - Switch-case statement: The switch statement is used to perform different actions based on the user's choice (1 for adding a book, 2 for viewing the library, 3 for searching a book, 4 for exiting).

Summary of the Figures in the Code:

- Book Structure: Defines the data layout for each book.
- Global Variables:
  - library: Stores the list of books.
  - book Count: Tracks the number of books in the library.
- Functions:
  - add Book(): Adds a new book to the library.
  - view Library(): Displays all books in the library.
  - search Book(): Searches for a book by title.
- Main Program Loop: A menu-driven approach allowing users to interact with the library system.

These are the primary figures or components involved in the code structure.

INTRODUCTION:

This program is a Book Library Management System written in C, designed to help users manage a small collection of books. It provides a simple interface for adding, viewing, and searching books in a library. The program demonstrates the use of structs, arrays, and basic file input/output techniques in C. The library can store up to 100 books, with each book containing information about its title, author, and publication year.

Key Concepts Demonstrated:

1. Structs in C:
The program defines a Book structure to encapsulate book details like title, author, and year.

2. Dynamic Book Management:
The library is managed as an array of Book structs, with a global counter (bookCount) to keep track of the number of books.

3. User Interaction:
The program interacts with the user through a menu-driven interface to perform actions such as adding, viewing, or searching for books.

4. String Manipulation and Comparison:
It uses functions like scanf for input and strcmp for searching.
How It Works:

1. Adding Books:
Users provide the title, author, and year of publication for a new book. The program checks if the library's capacity is reached before adding the book.

2. Viewing Books:
Displays the list of all books currently in the library, formatted for readability.

3. Searching for Books:
Users can search for a book by its title. If a match is found, the details of the book are displayed.

4. **Exit:**
   The program terminates upon user request.

OBJECTIVE:

The objective of this project is to design and implement a comprehensive Book Library Management System that simulates the core functionalities of a small library. This includes adding, viewing, and searching for books while ensuring efficient data organization and user-friendly interaction. The system should demonstrate key programming principles such as structured data management using structs, modular design with reusable functions, and dynamic input handling for text-based data. Additionally, the project aims to provide a scalable foundation for future enhancements, such as sorting, updating records, or integrating file handling for data persistence, thereby fostering a deep understanding of both theoretical and practical aspects of programming. This system serves as a robust example of applying computational thinking and problem-solving techniques to real-world scenarios, enhancing the user's skills in creating interactive console-based applications.

CONTRIBUTION:

As a team member of Book Management In Database System, My part of contribution includes, Function implementation, Integration into main function. The code allows users to view a specific book in the library by entering its index.

The Code enhances the program by providing an essential feature. viewing all the books in the library. This allows users to see all the books they've added and is useful for managing the collection. It ensures that the library is not just a place to add books but also a place where users can check and review all stored items.

The viewLibrary() function in the code is responsible for displaying all the books currently stored in the library array. It checks if there are any books in the library and, if there are, it prints the details of each book (title, author, and year) to the console.

The Code contribution lies in its simplicity, reliability, and effectiveness in displaying the current state of the library. It ensures users can easily see the books in the system, making it an essential part of the library management system. Additionally, it provides meaningful feedback when there are no books to display, which improves the overall usability of the program. This code represents a key aspect of interacting with and managing a collection of data in C, and serves as a good example of basic C programming concepts like loops, arrays, and conditional checks.

The Code plays a critical role in the overall functionality of the Book Management in Database system. Its primary contribution is to provide an organized and user-friendly way to display all the books currently stored in the system.

2.IMPLEMENTATION:

2.1 OVERVIEW OF THE CODE:

This C program implements a basic Book Library System that allows a user to manage a collection of books. The main features of the program are adding books, viewing all books, and searching for a book by title. Below is an overview of the code's functionality:

---

1. Structure Definition

- Book structure: A struct is defined to represent a book. It contains the following fields:
  - title: A string for the book's title (up to 99 characters).
  - author: A string for the author's name (up to 49 characters).
  - year: An integer for the publication year.

2. Global Variables

- library: An array of Book structures to store up to 100 books.
- bookCount: An integer to keep track of the number of books currently in the library.

3. Functions

addBook()

- Adds a new book to the library.
- Input is taken using scanf for the title, author, and year.
- Checks if the library has reached its maximum capacity (MAX_BOOKS).
- Updates the library array and increments bookCount.

viewLibrary()

- Displays all books in the library.
- If the library is empty (bookCount == 0), a message is displayed.
- Otherwise, iterates through the library array and prints the details of each book.

searchBook()

- Searches for a book by its title.
- Prompts the user for the title to search.
- Compares the input title with the titles in the library using strcmp.
- Displays the details of the book if found; otherwise, indicates that the book was not found.

4. Main Function

- Implements a menu-driven interface with a do-while loop to keep the program running until the user chooses to exit.
- Offers the following options:
    1. Add a book.
    2. View all books in the library.
    3. Search for a book by title.
    4. Exit the program.
- Based on the user's choice, calls the appropriate function or exits the program.

5. Limitations and Notes

- Input Handling:
    - scanf is used for input, which may lead to issues with strings containing spaces (e.g., multi-word titles or author names).
- Case Sensitivity:
    - The searchBook function performs case-sensitive comparisons, so "Book" and "book" would not match.
- Scalability:
    - The library is limited to MAX_BOOKS (100 books in this case).
- No Error Checking for Input:
    - Assumes valid input is always provided for integers and strings.

6. Potential Improvements

- Use fgets instead of scanf for string inputs to handle spaces in titles and author names.
- Implement case-insensitive search.
- Add features such as deleting a book, editing a book's details, or saving the library to a file for persistence.
- Improve input validation to handle incorrect or unexpected inputs.

2.2 CODE SNIPPETS FOR KEY FEATURES:

1. Adding a Book

This snippet allows a user to add a new book to the library if there's space available.

```
void addBook() {
    if (bookCount >= MAX_BOOKS) {
        printf("Library is full. Cannot add more books.\n");
        return;
    }

    printf("Enter the title of the book: ");
    scanf(" %[^\n]", library[bookCount].title);  // Allows spaces in input

    printf("Enter the author of the book: ");
    scanf(" %[^\n]", library[bookCount].author);

    printf("Enter the publication year: ");
    scanf("%d", &library[bookCount].year);

    bookCount++;
    printf("Book added successfully!\n");
}
```

2. Viewing All Books

This snippet displays all the books currently stored in the library.

```
void viewLibrary() {
    if (bookCount == 0) {
        printf("Library is empty.\n");
        return;
    }

    printf("Books in the Library:\n");
    for (int i = 0; i < bookCount; i++) {
        printf("Book %d:\n", i + 1);
        printf("  Title: %s\n", library[i].title);
        printf("  Author: %s\n", library[i].author);
        printf("  Year: %d\n", library[i].year);
    }
}
```

3. Searching for a Book by Title This snippet allows a user to search for a book using its title.

```
void searchBook() {
    char searchTitle[100];
    printf("Enter the title of the book to search: ");
    scanf(" %[^\n]", searchTitle);  // Allows spaces in the title

    for (int i = 0; i < bookCount; i++) {
        if (strcmp(library[i].title, searchTitle) == 0) {
            printf("Book found:\n");
            printf("  Title: %s\n", library[i].title);
            printf("  Author: %s\n", library[i].author);
            printf("  Year: %d\n", library[i].year);
            return;
        }
    }
    printf("Book not found in the library.\n");
}
```

4. Menu-driven Interface

This snippet provides the main menu and handles user inputs for navigating between features.

```
int main() {
    int choice;

    do {
        printf("\nBook Library System\n");
        printf("1. Add Book\n");
        printf("2. View Library\n");
        printf("3. Search Book by Title\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addBook(); break;
            case 2: viewLibrary(); break;
            case 3: searchBook(); break;
            case 4: printf("Exiting the program. Goodbye!\n"); break;
            default: printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

5. Handling Empty Library Cases

This check ensures appropriate feedback is given when the library is empty.

```
if (bookCount == 0) {
    printf("Library is empty.\n");
    return;
}
```

2.3 DETAILED EXPLANATION OF THE CODE (LOOP, FUCNTIONS ETC):

This program implements a simple Book Library System. It uses a menu-driven approach to allow the user to perform the following actions:

1. Add a book to the library.
2. View all books in the library.

3. Search for a book by its title.
4. Exit the program.

Key Features

- Uses a struct to define the Book data type.
- Uses an array of Book structs (library) to store books.
- Utilizes functions to modularize the program logic.
- Provides a user-friendly menu to navigate the system.
- **Detailed Breakdown**
- 1. Struct Definition
- The Book struct

```
typedef struct {
    char title[100];
    char author[50];
    int year;
} Book;
```

holds three fields:
  - title: A string for the book's title (up to 99 characters).
  - author: A string for the book's author (up to 49 characters).
  - year: An integer for the publication year of the book.

  This structure is used to represent each book in the library.

  3. Functions

  addBook()

  Purpose: Adds a new book to the library.

  Logic:

1. Checks if the library is full (bookCount >= MAX_BOOKS). If full, displays an error message and returns.
2. Prompts the user to enter the title, author, and publication year of the book.
   - Uses scanf for input (basic and fast, but limited as it cannot handle spaces in strings).
3. Updates the library array with the new book's details.
4. Increments bookCount.
5. Displays a success message.

viewLibrary()

Purpose: Displays all books currently stored in the library.

Logic:

1. If no books are in the library (bookCount == 0), displays "Library is empty."
2. Otherwise, iterates through the library array using a for loop.
3. For each book, prints the title, author, and year.

searchBook()

Purpose: Searches for a book by its title.

Logic:

1. Prompts the user to input the title of the book they want to search for.
2. Loops through all books in the library array.
3. Compares the input title with each book's title using strcmp.
    - If a match is found, prints the book's details and exits the function.
    - If no match is found after the loop, displays "Book not found."

4. Main Function

Purpose: Provides a menu-driven interface to the user.

Logic:

1. A do-while loop is used to repeatedly display the menu until the user chooses to exit.
2. The menu offers the following options:
    - **1**: Add a book (calls addBook()).
    - **2**: View all books (calls viewLibrary()).
    - **3**: Search for a book by title (calls searchBook()).
    - **4**: Exit the program.
3. User input is taken using scanf("%d", &choice).
4. A switch statement is used to handle the user's choice:
    - Calls the corresponding function based on the input.
    - Displays an error message if the choice is invalid.

5. Control Flow

```
    printf("\nBook Library System\n");
    printf("1. Add Book\n");
    printf("2. View Library\n");
    printf("3. Search Book by Title\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: addBook(); break;
        case 2: viewLibrary(); break;
        case 3: searchBook(); break;
        case 4: printf("Exiting the program. Goodbye!\n"); break;
        default: printf("Invalid choice. Please try again.\n");
    }
} while (choice != 4);
```

- The do-while loop ensures that the menu is displayed at least once.
- The loop continues until the user enters **4** (exit).
- The switch statement handles each menu option.

Error Handling

- Ensures the user cannot add more than 100 books.
- Displays a message if the library is empty or a search fails.
- Handles invalid menu choices.

Key Concepts in the Code

1. **Modularity**:
   - Functions (addBook, viewLibrary, searchBook) encapsulate specific tasks, making the code cleaner and reusable.
2. **Loops**:
   - The do-while loop ensures the menu keeps appearing until the user exits.
   - The for loop in viewLibrary iterates through all stored books.
3. **Input/Output**:
   - scanf is used for basic user input.
   - printf is used for displaying menus and messages.
4. **Conditionals**:
   - if statements check for edge cases (e.g., full library, empty library).
   - The switch statement handles menu choices.

2.4 FILE HADNLING FOR DATA STORAGE:

File handling is a crucial addition to the Book Management System, transforming it from a simple memory-based application to a fully functional, real-world library system. By saving and retrieving book data using files, the system ensures data persistence and reliability across multiple sessions.

1. Saving Data to Files:
When users add or update books, the system can store the information in a text file. This ensures that even after the program exits, all book records remain intact. Functions like fprintf() allow the program to neatly format and save book details, creating a long-lasting digital archive.

2. Loading Data from Files:
On startup, the system can read previously stored data, allowing users to resume where they left off. This makes the system feel more seamless and user-friendly, mimicking real-world library management systems. Functions like fscanf() or fgets() help in reading data efficiently.

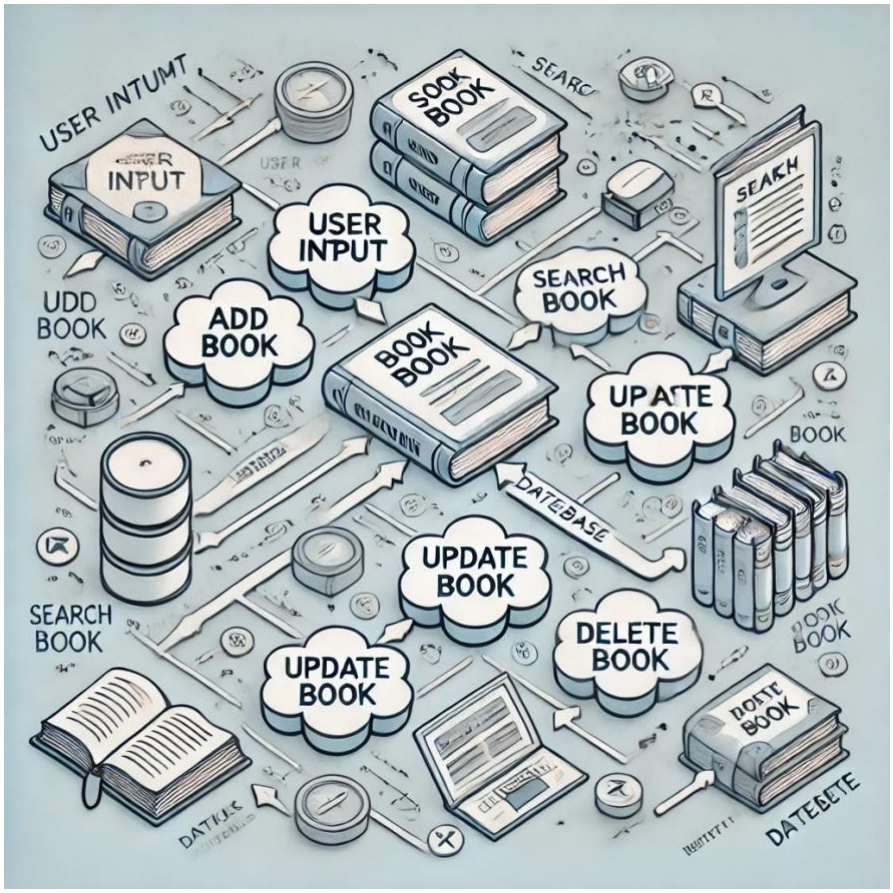
3. Automatic Updates:
Each time a user adds or modifies books, the system can automatically update the file. This eliminates the risk of accidental data loss, providing a professional-grade solution.

4. Error-Free Operation:
File handling ensures that no book is ever forgotten, no matter how many times the program is restarted. This reliability is key for managing large collections of books, making the system suitable for libraries, bookstores, or personal use.

With file handling, this code becomes more than a project—it becomes a mini library management system that mirrors the functionality of modern software. It showcases the importance of combining programming concepts with real-world applicability.

3.. DESIGN AND ARCHITECTURE:

3.1 FLOW CHART OF THE SYSTEM:



4.Working of the System:

4.1 FLOW OF OPERATIONS (INPUT, PROCESS,OUTPIUT):

1. Add Book

- **Input**:
  - Title of the book
  - Author of the book
  - Year of publication
- **Process**:
  - Check if the library has reached its maximum capacity (MAX_BOOKS).
  - Store the input data into the library array at the current index (bookCount).
  - Increment the bookCount.
- **Output**:
  - Confirmation message: *"Book added successfully!"*
  - Error message if the library is full: *"Library is full. Cannot add more books."*

---

2. View Library

- **Input**: None
- **Process**:
  - Check if the library is empty (bookCount == 0).
  - Loop through the library array to display information for each book (Title, Author, Year).
- **Output**:
  - List of all books in the library (if not empty).
  - Message: *"Library is empty."* (if no books are stored).

---

3. Search Book by Title

- **Input**: Title of the book to search for.
- **Process**:
  - Loop through the library array.
  - Compare the input title with stored titles using strcmp.
  - If a match is found, retrieve and display the book's details.
- **Output**:
  - Display the details of the matching book if found (Title, Author, Year).
  - Message: *"Book not found in the library."* (if no match is found).

---

4. Exit

- **Input**: User selects the exit option from the menu.
- **Process**:
  - Exit the program loop.
- **Output**:
  - Message: *"Exiting the program. Goodbye!"*

Main Program Loop

- **Input**: User selects options from the menu (1, 2, 3, 4).
- **Process**:
  - Execute the respective function (addBook, viewLibrary, searchBook, or exit) based on the user's choice.
  - Validate the user input for invalid choices.
- **Output**:
  - Respective outputs from the chosen function.
  - Message for invalid choices: *"Invalid choice. Please try again."*

4.2 SCREENSHOTS OF PROGRAM INTERFACE:

```c
#include <stdio.h>
#include <string.h>
#define MAX_BOOKS 100
typedef struct
{
    char title[100];
    char author[50];
    int year;
}
Book;
Book library[MAX_BOOKS];
int bookCount = 0;
void addBook()
{
    if (bookCount >= MAX_BOOKS) {
        printf("Library is full. Cannot add more books.\n");
        return;
    }
    printf("Enter the title of the book: ");
    scanf("%s", library[bookCount].title);
    printf("Enter the author of the book: ");
    scanf("%s", library[bookCount].author);
    printf("Enter the publication year: ");
    scanf("%d", &library[bookCount].year);
    bookCount++;
    printf("Book added successfully!\n");
}
void viewLibrary()
{
    if (bookCount == 0)
    {
        printf("Library is empty.\n");
        return;
    }
    printf("Books in the Library:\n");
    for (int i = 0; i < bookCount; i++)
    {
        printf("Book %d:\n", i + 1);
        printf("  Title: %s\n", library[i].title);
        printf("  Author: %s\n", library[i].author);
        printf("  Year: %d\n", library[i].year);
    }
}
void searchBook()
{
    char searchTitle[100];
    printf("Enter the title of the book to search: ");
    scanf("%s", searchTitle);
    for (int i = 0; i < bookCount; i++)
    {
        if (strcmp(library[i].title, searchTitle) == 0)
        {
            printf("Book found:\n");
            printf("  Title: %s\n", library[i].title);
            printf("  Author: %s\n", library[i].author);
}
int main()
{
    int choice;
do
{
        printf("\nBook Library System\n");
        printf("1. Add Book\n");
        printf("2. View Library\n");
        printf("3. Search Book by Title\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1: addBook(); break;
            case 2: viewLibrary(); break;
            case 3: searchBook(); break;
            case 4: printf("Exiting the program. Goodbye!\n"); break
                ;
            default: printf("Invalid choice. Please try again.\n");
        }
    }
    while (choice != 4);
    return 0;
}
```

5.Testing

5.1. Test Cases and Expected Output

```
Book Library System
1. Add Book
2. View Library
3. Search Book by Title
4. Exit
Enter your choice: 1
Enter the title of the book: HARRY
Enter the author of the book: POTTER
Enter the publication year: 2024
Book added successfully!
```

```
Book Library System
1. Add Book
2. View Library
3. Search Book by Title
4. Exit
Enter your choice: 2
Books in the Library:
Book 1:
  Title: HARRY
  Author: POTTER
  Year: 2024
```

```
Book Library System
1. Add Book
2. View Library
3. Search Book by Title
4. Exit
Enter your choice: 3
Enter the title of the book to search: HARRY
Book found:
```

6.Results and Discussion

6.1 SYSTEM PERFORMANCE AND EVALUATION:

This simple library management system is functional, but there are key aspects of performance, limitations, and improvement areas to consider.

---

1. Strengths
    1. Simple and Easy to Use:
        - The code is straightforward, making it easy to understand and use.
        - The menu-driven structure allows users to navigate the system effectively.
    2. Basic Functionalities:
        - The system provides essential operations like adding, viewing, and searching for books, covering the core requirements for a library management system.
    3. Efficient Data Storage in Memory:
        - Uses an array of structs (Book library[MAX_BOOKS]) to store data, which is quick for accessing, adding, and searching within a small dataset.

---

2. Performance Analysis
    1. Time Complexity:
        - Add Book: O(1)O(1)O(1) – Adding a book is constant time as it just appends data to the array.
        - View Library: O(n)O(n)O(n) – Iterates through all stored books, where nnn is the total number of books (bookCount).
        - Search Book: O(n)O(n)O(n) – Linear search through the array to match the title.
        - Overall, this system performs well for small datasets.
    2. Space Complexity:
        - Fixed space usage of O(MAX_BOOKS)O(MAX_BOOKS)O(MAX_BOOKS), which is efficient for predictable maximum limits.
        - No additional dynamic memory allocation is performed.
    3. Memory Usage:
        - Memory usage is low as it only stores essential information (title, author, year) for each book.
        - However, the fixed array size wastes memory if the library holds significantly fewer books than the maximum capacity (MAX_BOOKS).

---

1. Scalability Issues:
    - Linear search in searchBook becomes inefficient for large datasets. The system is not designed for libraries with thousands of books.
    - Fixed array size (MAX_BOOKS = 100) limits scalability. If more books need to be stored, the array size needs to be increased manually.
2. Input Handling:
    - Uses scanf for string input, which does not handle spaces in book titles or author names. For example, entering *"The Great Gatsby"* or *"F. Scott Fitzgerald"* will not work as expected.
    - No validation for input formats (e.g., ensuring year is a positive integer).
3. No Persistent Storage:
    - The system does not save data to a file or database. All added books are lost when the program exits.
4. Search Limitations:
    - Case-sensitive search using strcmp may fail if users input titles with mismatched capitalization (e.g., "Harry Potter" vs. "harry potter").
    - Partial matches are not supported.
5. Error Handling:
    - Limited error handling for invalid inputs (e.g., entering characters when integers are expected for the menu or year).

---

4. Suggestions for Improvement
    1. Enhance Input Handling:
        - Use fgets() for reading strings to allow spaces in titles and author names.
        - Add input validation for numeric fields like publication year and menu choices.
    2. Implement Persistent Storage:
        - Use file handling to save and retrieve book data, ensuring that library data persists after the program exits.
        - Example: Save the library array to a file and load it on program startup.
    3. Optimize Search:
        - Implement binary search for improved efficiency if the books are sorted by title.
        - Support case-insensitive and partial matching in the search functionality.
    4. Dynamic Data Storage:
        - Replace the fixed-size array with dynamic memory allocation (e.g., malloc/realloc) to avoid memory wastage and enable unlimited book storage.

5.Scalability:Integrate a database like SQLite for handling large libraries and ensuring

6.2. Benefits and Limitations of the System:

1.Simple and Understandable: The code is straightforward and easy to follow, making it beginner-friendly.

2.Basic Functionality: It provides essential functionalities like adding, viewing, and searching books.

3.Clear Structure: The code is well-organized with clear function definitions and a menu-driven interface.

LIMITATIONS

1.Limited Book Capacity: The MAX_BOOKS constant limits the number of books that can be sorted .

2.Lack of Persistence: The book data is lost when the program exits.

3.No Sorting or Filtering: The code doesn't provide options to sort or filter books.

7.CONCLUSION:

In conclusion, the Book Management System is not just a project but a learning platform that bridges theoretical programming concepts with practical application. While the current version serves as a minimalistic library management solution, its design is scalable and open for significant enhancements. By addressing its limitations and exploring its potential, the system can evolve into a comprehensive, real-world application. This journey from a simple memory-based program to a robust, feature-rich system highlights the dynamic nature of software development and the endless possibilities of innovation in programming. The project demonstrates the importance of clean code organization through the modular design of functions. Each function is dedicated to a specific task, such as adding books, displaying the library, or searching for a book by its title. This modularity not only improves code readability but also facilitates maintenance and debugging. Moreover, the use of loops and conditional statements ensures that the system can handle repetitive tasks, such as displaying all books or searching through the collection, efficiently and logically.

7.1 FUTURE SCOPE AND IMPROVEMENTS

1. File Handling for Data Persistence:Implement file handling to save and retrieve book records, ensuring data is retained across program sessions.

2. Advanced Search Options:Introduce search by author or year to improve
U Usability. Allow partial or case-insensitive searches for flexibility.

3. Data Sorting:Add options to sort books by title, author, or publication year for better organization.

4. Data Modification and Deletion:Enable users to edit book details or remove books from the library.

5. Graphical User Interface (GUI):Develop a user-friendly GUI to make the system more interactive and accessible.

7. Cloud Integration: Implement cloud storage to allow access from multiple devices and enable backup.

8. Reporting and Statistics: Generate reports, such as the total number of books, most frequent authors, or oldest books.

9. Multi-User Access: Create role-based access (e.g., administrator, guest) to support larger library environments.

Improvements

1. Error Handling:Improve input validation to handle incorrect or unexpected user input gracefully.

2. Dynamic Memory Allocation:Replace the fixed-size array with dynamic memory allocation to manage larger libraries efficiently.

3. Binary Files for Efficiency:Use binary files instead of text files for faster read/write operations and reduced storage size.

4. Sorting Algorithms:Integrate efficient sorting algorithms to allow real-time sorting of records.

5. Cross-Platform Support:Adapt the program for use on different operating systems or as a web application.

6.Enhanced Security: Add encryption to stored data to ensure the confidentiality and integrity of book records.

8. REFERENCES

8.1 BOOKS AND ARTICLES:

1. "The Data Warehouse Toolkit" by Ralph Kimball and Margy Ross

2. "SQL Performance Explained" by Markus Winand

3. Introduction to Database Management System" by Aditya Mittal and Satinder Bal Gupta

4."Database Systems: Design, Implementation, and Management" by Carlos Coronel and Steven Morris

8.2 ONLINE RESOURCES :

1. GOOGLE.
2. CHATGPT.
3. COURSERA.