Library Management System

By

Hossain Ahammed

221-15-4694

Fabiha Chowdhury Momo

221-15-5092

Somiya Akter

221-15-5696

Supervised by

Ms. Amatul Bushra Akhi

Assistant Professor

Department of CSE

Daffodil International University

Daffodil International University.

1.Introduction

1.Introduction:

The Library Management System is a data structure project designed to efficiently organize and manage a library's resources. Using linked lists, the system tracks books with their details such as title, author, pages, and cost. It supports operations like adding, removing, and displaying books, as well as implementing a stack for quick access and a queue for systematic processing. This project showcases fundamental data structure concepts to enhance the management of a library's collection.

2.problem Statement:

The Library Management System project aims to address the inefficiencies in traditional library processes by implementing a data structure-based solution. The primary challenges include organizing and optimizing book storage, facilitating user-friendly book transactions, managing book availability, and providing quick retrieval of information. The project seeks to enhance overall library operations through the effective application of data structures, promoting streamlined book management and user services.

3.Objectives:

The Library Management System project utilizing data structures aims to achieve the following objectives:

1.Efficiently organize and store library resources.
2.Enable seamless addition and removal of books.
3.Implement a stack for quick book access and retrieval.
4.Utilize a queue for systematic book processing.
5.Provide a user-friendly interface for book transactions.
6.Ensure optimal utilization of memory through data structures.
7.Track and manage book details such as title, author, and availability.
8.Enhance search and retrieval mechanisms for library users.
9.Minimize manual efforts in library management through automation.
10.Demonstrate the practical application of data structures in real-world scenarios.

4. Data Structure Used:

Linked List:
Purpose: Efficiently stores and manages the details of each book in a dynamic structure.
Usage: Additions and removals of books, displaying the list of books.
Stack:
Purpose: Facilitates quick access to recently added books.
Usage: Adding books to the stack and removing the top book for immediate retrieval.
Queue:
Purpose: Enables systematic processing of book requests in a first-come, first-served manner.
Usage: Enqueuing and dequeuing books for orderly user transactions.

5.Implementation details:

Source code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


struct book {
    int book_id;
    char title[100];
    char author[100];
    int copies;
    struct book *next;
};


#define MAX_QUEUE_SIZE 100
int queue[MAX_QUEUE_SIZE];
int front = -1;
int rear = -1;


#define MAX_STACK_SIZE 100
int stack[MAX_STACK_SIZE];
int top = -1;


typedef struct book Book;
Book *head = NULL;
```

```c
int dequeue();
void displayQueue();
void push(int item);
int pop();
void displayStack();


void add_book();
void add_book_at_beginning();
void add_book_at_end();
void add_book_at_middle();
void add_book_at_position(int position);
void update_book(int book_id);
void delete_book(int book_id);
void search_book(const char *title);
void display_books();



int isQueueFull() {
    return rear == MAX_QUEUE_SIZE - 1;
}

int isQueueEmpty() {
    return front == -1 || front > rear;
}

void enqueue(int item) {
    if (isQueueFull()) {
        printf("Queue is full. Cannot enqueue more ite
```

```
        printf("Queue is full. Cannot enque
        return;
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = item;
    printf("Enqueued %d\n", item);
}

int dequeue() {
    if (isQueueEmpty()) {
        printf("Queue is empty. Nothing to
        return -1;
    }
    int item = queue[front];
    front++;
    return item;
}

void displayQueue() {
    if (isQueueEmpty()) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
```

```
}

void push(int item) {
    if (isStackFull()) {
        printf("Stack is full. Cannot push more items.\n");
        return;
    }
    top++;
    stack[top] = item;
    printf("Pushed %d\n", item);
}

int pop() {
    if (isStackEmpty()) {
        printf("Stack is empty. Nothing to pop.\n");
        return -1;
    }
    int item = stack[top];
    top--;
    return item;
}

void displayStack() {
    if (isStackEmpty()) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack: ");
    for (int i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}

void add_book() {
    int sub_choice;
    printf("Add Book Sub-Options:");
```

```
    switch (sub_choice) {
        case 1:
            add_book_at_beginning();
            break;
        case 2:
            add_book_at_end();
            break;
        case 3:
            add_book_at_middle();
            break;
        case 4:
            {
                int position;
                printf("Enter the position to add the book: ")
                scanf("%d", &position);
                add_book_at_position(position);
            }
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
}

void add_book_at_beginning() {
    Book *new_book = (Book *)malloc(sizeof(Book));
    if (new_book == NULL) {
        printf("Memory allocation failed.");
        return;
    }

    new_book->next = NULL;
    new_book->book_id = rand();
    printf("Enter the title of the book: ");
    scanf("%s", new_book->title);
    printf("Enter the author of the book: ");
    scanf("%s", new_book->author);
    printf("Enter the number of copies: ");
```

```c
void add_book_at_middle() {
    int count = 0;
    Book *current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }

    if (count < 2) {
        printf("Not enough books to add in the middle.\n");
        return;
    }

    int middle = count / 2;
    add_book_at_position(middle);
}

void add_book_at_position(int position) {
    if (position < 1) {
        printf("Invalid position. Position must be at least 1.\n");
        return;
    }

    if (position == 1) {
        add_book_at_beginning();
        return;
    }

    int count = 0;
    Book *current = head;
    while (current != NULL) {
        count++;
        if (count == position - 1) {
            Book *new_book = (Book *)malloc(sizeof(Book));
            if (new_book == NULL) {
                printf("Memory allocation failed.");
                return;
```

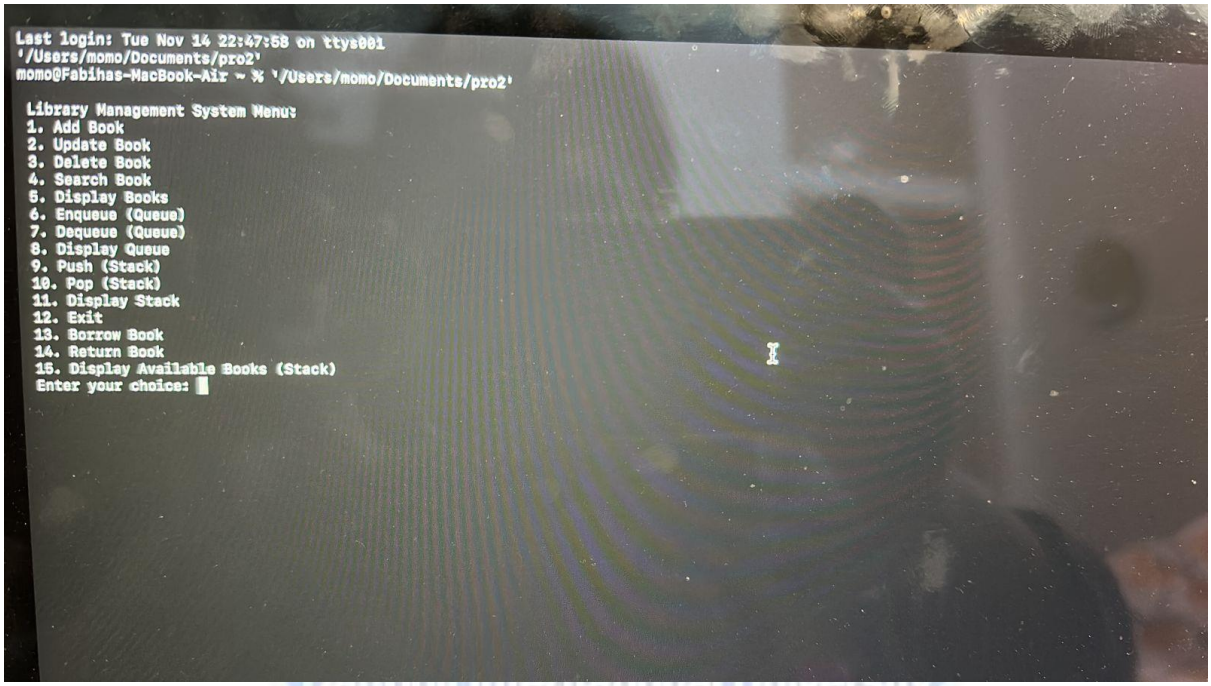
```c
void update_book(int book_id) {
    Book *current = head;
    while (current != NULL) {
        if (current->book_id == book_id) {
            printf("Enter the new title: ");
            scanf("%s", current->title);
            printf("Enter the new author: ");
            scanf("%s", current->author);
            printf("Enter the new number of copies: ");
            scanf("%d", &(current->copies));
            printf("Book with ID %d updated successfully.\n", book_id);
            return;
        }
        current = current->next;
    }

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

void delete_book(int book_id) {
    Book *current = head;
    Book *prev = NULL;

    while (current != NULL) {
        if (current->book_id == book_id) {
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            printf("Book with ID %d deleted successfully.\n", book_id);
            return;
        }
        prev = current;
        current = current->next;
```

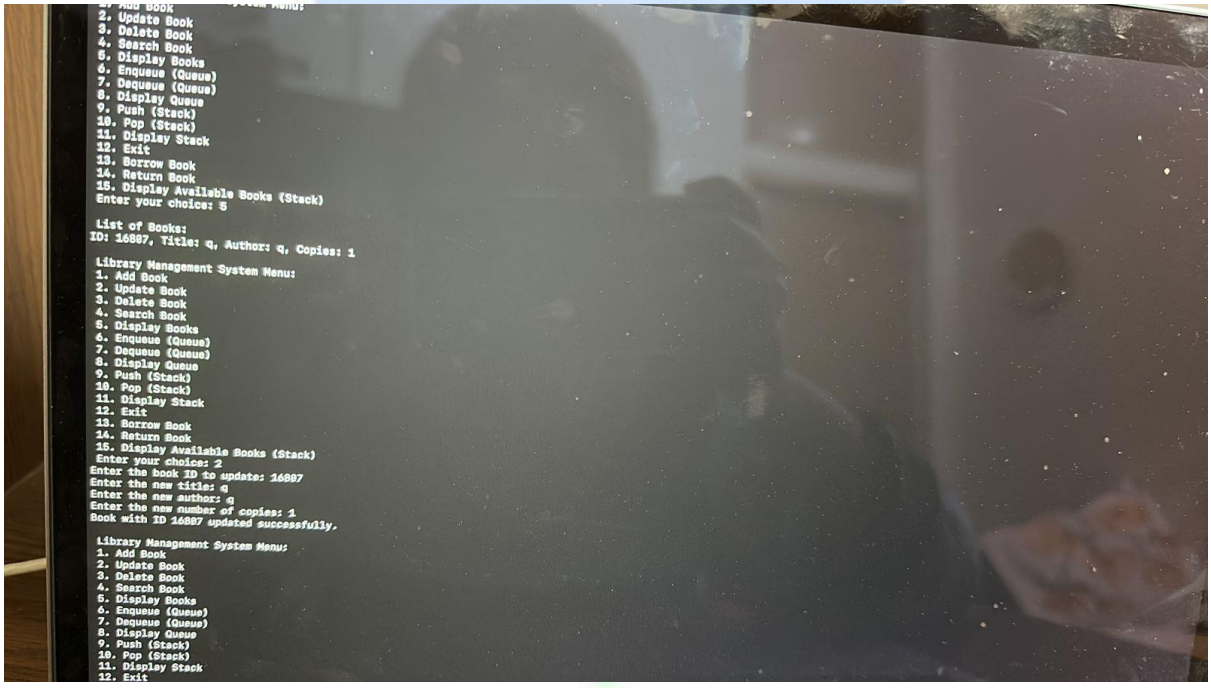
```c
void borrow_book(int book_id) {

    Book *current = head;
    while (current != NULL) {
        if (current->book_id == book_id && current->copies > 0) {
            push(book_id);
            enqueue(book_id);
            current->copies--;
            printf("Book with ID %d has been borrowed successfully.\n", book_id);
            return;
        }
        current = current->next;
    }
    enqueue(book_id);
    printf("Book with ID %d is either not available or doesn't exist.\n", book_id);
}

int return_book() {
    int book_id = pop();
    if (book_id == -1) {
        printf("No books borrowed.\n");
        return -1;
    }


    int dequeued = dequeue();
    if (dequeued != book_id) {
        printf("Error: Book ID mismatch between borrowed and returned books.\n");
        return -1;
    }


    Book *current = head;
    while (current != NULL) {
        if (current->book_id == book_id) {
            current->copies++;
            printf("Book with ID %d has been returned successfully.\n", book_id);
```

```
        printf("\n 8. Display Queue");
        printf("\n 9. Push (Stack)");
        printf("\n 10. Pop (Stack)");
        printf("\n 11. Display Stack");
        printf("\n 12. Exit");
        printf("\n 13. Borrow Book");
        printf("\n 14. Return Book");

rintf("\n 15. Display Available Books (Stack)");

        printf("\n Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {

            case 1:
                add_book();
                break;
            case 2:
                {
                    int book_id;
                    printf("Enter the book ID to update: ");
                    scanf("%d", &book_id);
                    update_book(book_id);
                }
                break;
            case 3:
                {
                    int book_id;
                    printf("Enter the book ID to delete: ");
                    scanf("%d", &book_id);
                    delete_book(book_id);
                }
                break;
            case 4:
                {
                    char title[100];
```

This C program implements a Library Management System using a linked list to efficiently organize book records. The system allows users to add books at the beginning, end, middle, or a specific position in the library. It supports operations such as updating book details, deleting books, searching by title, and displaying the entire book collection. The code also integrates a stack to manage borrowed books, enabling users to borrow and return books seamlessly. Additionally, a queue is employed to handle the return order of books. The program maintains memory allocation using dynamic memory allocation for each new book entry. The interactive menu system ensures user-friendly interaction, covering a range of functionalities, from basic book management to advanced stack and queue operations. Overall, this program provides a robust and versatile solution for a library management system, combining data structure principles with practical features.

6.Code Implementation:





7.Conclution:

In conclusion, the Library Management System, designed with a foundation of linked lists, stacks, and queues, proves to be a robust and efficient solution for modernizing library operations. The integration of these data structures offers several advantages:

Optimized Book Storage:
Linked lists provide a flexible and dynamic structure for storing and managing detailed information about each book, accommodating the library's evolving collection.

Quick Access with Stacks:
The stack facilitates immediate access to recently added books, streamlining the retrieval process and enhancing user experience.
Systematic Transaction Processing:
The queue ensures a fair and systematic approach to processing user requests, promoting orderly access to library resources in a first-come, first-served manner. Efficient User Interactions:
The system's architecture enables seamless addition and removal of books, enhancing overall efficiency and reducing manual efforts in library management.
Dynamic and Responsive Interface:
The use of linked lists allows for a dynamic and responsive interface, ensuring adaptability to changes in the library's book inventory.
Real-world Application of Data Structures:
The project serves as a practical example of how data structures like linked lists, stacks, and queues can be effectively employed in a real-world scenario, demonstrating their versatility and utility.
Enhanced User Services:
Through the systematic organization and quick retrieval of books, the Library Management System enhances user services, providing library staff and users with a more streamlined and user-friendly experience.