A Project Report On

<u>“LIBRARY MANAGEMENT SYSTEM”</u>

MASTER OF SCIENCE INFORMATION TECHNOLOGY

By

SUJAL MANSURI /2426040

JENISH Kotadiya / 2426017

Under the guidance of

Prof. SOURABH DATTALKAR

SKIPS UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

(Session: 2024-26)

ABSTRACT

The Library Management System is a console-based application developed in C that facilitates the efficient management of a library's operations. The system implements role-based access control, allowing administrators to manage books (add, edit, delete) and users to search, borrow, and return books. Data persistence is achieved through file operations, ensuring that book records remain intact across sessions. The program incorporates input validation to prevent errors and unauthorized access. The project aims to streamline library processes by providing an organized, user-friendly system for managing book stock, lending, and returning transactions. This system can serve as a foundation for more advanced library management solutions..

TABLE OF CONTENTS

1.Title Page

Project Title

“Library Management System in C”

Participent:

Sujal Mansuri / 2426040
Jenish Kotadiya / 2426017

Program : Msc in Computer Science

Submission Date: December 26,2024

Institution Name:

Skips University , Department of Computer Science

2.Introduction

PROJECT AIMS AND OBJECTIVE

The **Library Management System** is a C-based project designed to automate and simplify library operations. It allows administrators to manage books efficiently by adding, editing, and deleting book records, while users can search, borrow, and return books. The system incorporates role-based access with a secure login, ensuring admins and users have appropriate permissions. Book data is stored persistently in a file, ensuring records are maintained across sessions. With a user-friendly interface, robust error handling, and support for up to 100 books, the system aims to streamline library management, reduce manual efforts, and enhance accuracy..

Importance of Library Management Automation

1. **Efficiency and Speed**
   - Automation eliminates manual processes, significantly reducing the time required to manage books and records. Tasks such as searching for a book or updating stock can be done instantly.
2. **Accuracy and Consistency**
   - By replacing manual entries with automated systems, errors in book records, such as duplicate IDs or incorrect stock levels, are minimized.
3. **Data Security**
   - Automation ensures better control over sensitive library data, with access restricted to authorized personnel through login credentials.
4. **Cost-Effectiveness**
   - Automated systems reduce the need for extensive manpower, saving costs for institutions in the long run.

Scope of the Project

1. **Book Management**
   - Add, edit, and delete book records with details like book ID, title, author, and stock levels.
2. **User Roles**
   - Two distinct roles:
     - **Admin**: Full access to manage the library.
     - **User**: Restricted to searching, borrowing, and returning books.
3. **Search Functionality**
   - Search books by title or author for ease of access.
4. **Borrowing and Returning**
   - Users can borrow available books, reducing stock automatically, and return them to update the stock count.
5. **Data Persistence**
   - Book data is stored in a file, ensuring it remains intact even after the program exits.

3.SOFTWARE REQUIREMENT

1. **Software Requirements**:

- C Compiler (e.g., GCC, Turbo C).
- Text Editor/IDE (e.g., Visual Studio Code, Code::Blocks, or Dev-C++).

2. **Hardware Requirements**:

- Minimum 1 GB RAM.
- Any standard PC or laptop.

3.Functional Requirements

1.User Roles and Permissions

- Admins can:
  - Add, edit, and delete book records.
  - View all books in the library.
- Users can:
  - Search for books by title or author.
  - Borrow and return books.

2. **Book Management**
   - Add new books with details such as ID, title, author, and stock count.
   - Edit existing book information.
   - Delete books from the library.
3. **Login System**
   - Role-based login functionality for admin and user.
   - Validate credentials for access:
     - Admin: Username = "admin", Password = "admin123"
     - User: Username = "user", Password = "user123"
4. **Search Functionality**
   - Allow users to search books using keywords from titles or authors.
5. **Borrowing and Returning Books**
   - Borrow a book:

        - Reduce stock by 1 if the book is available.
    - Return a book:
        - Increase stock by 1 upon return.
6. **File Handling**
    - Store book records persistently in a text file (library.txt).
    - Load data from the file at startup and save data at exit.
7. **Error Handling**
    - Validate input for book IDs, stock counts, and other fields.
    - Display error messages for invalid actions (e.g., borrowing unavailable books, duplicate IDs).

5. System Requirements

Functional Requirements

These requirements define the core functionality and operations of the Library Management System.

1. **User Roles and Permissions**
    - Admins can:
        - Add, edit, and delete book records.
        - View all books in the library.
    - Users can:
        - Search for books by title or author.
        - Borrow and return books.
2. **Book Management**
    - Add new books with details such as ID, title, author, and stock count.
    - Edit existing book information.
    - Delete books from the library.
3. **Login System**
    - Role-based login functionality for admin and user.
    - Validate credentials for access:
        - Admin: Username = "admin", Password = "admin123"
        - User: Username = "user", Password = "user123"
4. **Search Functionality**
    - Allow users to search books using keywords from titles or authors.
5. **Borrowing and Returning Books**
    - Borrow a book:
        - Reduce stock by 1 if the book is available.

- Return a book:
  - Increase stock by 1 upon return.

6. **File Handling**
   - Store book records persistently in a text file (library.txt).
   - Load data from the file at startup and save data at exit.
7. **Error Handling**
   - Validate input for book IDs, stock counts, and other fields.
   - Display error messages for invalid actions (e.g., borrowing unavailable books, duplicate IDs).

4.Non-Functional Requirements

These requirements describe the overall quality and performance of the system.

1. **Usability**
   - Provide a simple, menu-driven console interface for ease of use.
2. **Performance**
   - The system should handle up to 100 books efficiently.
   - Operations such as adding, searching, borrowing, and returning books should execute quickly.
3. **Reliability**
   - Ensure data persistence through file handling, with no data loss after exiting the program.
4. **Scalability**
   - The system can be extended to accommodate more books or additional features in the future.
5. **Security**
   - Restrict access through role-based login, ensuring only authorized users can modify or access data.
6. **Maintainability**
   - The code structure allows for easy updates and the addition of new features.
7. **Portability**
   - The system should run on any platform supporting a C compiler (e.g., Windows, Linux).
8. **Error Recovery**
   - Handle invalid input gracefully and prompt the user to correct errors.

4.Features of the System

1. **User Roles and Role-Based Access Control**

   - **Admin Features:**
     - Add new books with complete details (ID, title, author, stock count).
     - Edit existing book information, including title, author, and stock.
     - Delete books from the library inventory.
   - **User Features:**
     - Search for books by title or author.
     - Borrow books (decrease stock count).
     - Return books (increase stock count).

---

2. **Book Management**

   - **Add Books:**
     - Input book details such as ID, title, author, and number of copies.
     - Ensure unique book IDs to avoid duplicates.
   - **Edit Books:**
     - Modify existing book details while maintaining data integrity.
   - **Delete Books:**
     - Remove books from the inventory.

---

3. **Persistent Data Storage**

   - Uses a text file (library.txt) to store book records.
   - Automatically loads existing data during program startup.
   - Saves updated data when the program exits to ensure no information is lost.

---

4. **Search Functionality**

   - Allows users to search for books by entering keywords from the title or author.
   - Displays matching books, including their ID, title, author, and availability.

---

5. **Borrowing and Returning Books**

   - **Borrowing:**
     - Reduces the stock count of a book by 1 if it is available.
     - Displays an error message if the book is out of stock.
   - **Returning:**
     - Increases the stock count of a book by 1.
     - Keeps accurate track of available copies.

---

6. **Login System**

   - Role-based login for two types of users:
     - **Admin**: Username = "admin", Password = "admin123".
     - **User**: Username = "user", Password = "user123".
   - Ensures secure access to specific system functionalities based on the user role.

7.Input Validation

- Validates numerical inputs to ensure they are within acceptable ranges (e.g., book IDs, stock levels).
- Ensures string inputs are not empty and meet the required format.

---

8. User-Friendly Interface

- Menu-driven console interface for easy navigation.
- Clear prompts and messages for user actions.
- Error messages guide users when invalid inputs are provided.

9.Scalability

- Capable of managing up to 100 books, with room for future expansion.
- Provides a flexible framework for adding advanced features.

---

10. Error Handling

- Handles invalid inputs gracefully and prompts users to re-enter correct values.
- Prevents operations on non-existent or unavailable books.

PROGRAM CODE & Explation

Library.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



#define MAX_TITLE_LENGTH 100

#define MAX_AUTHOR_LENGTH 100

#define MAX_BOOKS 100

#define FILENAME "library.txt"



// Structure for storing book details

typedef struct {

int book_id;

char title[MAX_TITLE_LENGTH];

char author[MAX_AUTHOR_LENGTH];

int stock;  // Number of available copies

} Book;



Book books[MAX_BOOKS];
```

```
int bookCount = 0;


// Function declarations

void loadBooksFromFile();

void saveBooksToFile();

void addBook();

void editBook();

void deleteBook();

void searchBook();

void displayBooks();

void borrowBook();

void returnBook();

void login();

int isAdmin = 0;  // Role-based access: 1 = Admin, 0 =
User


// Main function

int main() {

int choice;


// Load existing books from file
```

```
loadBooksFromFile();


// Login

login();


// Menu loop

while (1) {

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

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

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

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

printf("4. Search Book\n");

printf("5. Display All Books\n");

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

printf("7. Return Book\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar();  // Consume newline left in the buffer


switch (choice) {
        case 1:

        if (isAdmin) addBook();

        else printf("Access denied! Only admins can add

books.\n");

        break;

        case 2:

        if (isAdmin) editBook();

        else printf("Access denied! Only admins can edit

books.\n");

        break;

        case 3:

        if (isAdmin) deleteBook();

        else printf("Access denied! Only admins can delete

books.\n");

        break;

        case 4:

        searchBook();

        break;

        case 5:

        displayBooks();

        break;
```

```
case 6:

borrowBook();

break;

case 7:

returnBook();

break;

case 8:

saveBooksToFile();

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

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}



return 0;

}



// Function to load books from file

void loadBooksFromFile() {

FILE *file = fopen(FILENAME, "r");
```

```
    if (file == NULL) {
    printf("No previous data found. Starting with an
empty library.\n");
    return;
    }

    while (fscanf(file, "%d\n",
&books[bookCount].book_id) != EOF) {
    fgets(books[bookCount].title,
MAX_TITLE_LENGTH, file);
    books[bookCount].title[strcspn(books[bookCount].tit
le, "\n")] = '\0';
    fgets(books[bookCount].author,
MAX_AUTHOR_LENGTH, file);
    books[bookCount].author[strcspn(books[bookCount]
.author, "\n")] = '\0';
    fscanf(file, "%d\n", &books[bookCount].stock);
    bookCount++;
    }

    fclose(file);
```

```
}


// Function to save books to file

void saveBooksToFile() {

FILE *file = fopen(FILENAME, "w");

if (file == NULL) {

printf("Error saving data.\n");

return;

}


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

fprintf(file, "%d\n", books[i].book_id);

fprintf(file, "%s\n", books[i].title);

fprintf(file, "%s\n", books[i].author);

fprintf(file, "%d\n", books[i].stock);

}


fclose(file);

}


// Function to add a book
```

```
void addBook() {

if (bookCount >= MAX_BOOKS) {

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

return;

}



Book newBook;

printf("Enter Book ID: ");

scanf("%d", &newBook.book_id);

getchar();



// Check for duplicate ID

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

if (books[i].book_id == newBook.book_id) {

printf("Book ID already exists. Try again.\n");

return;

}

}



printf("Enter Book Title: ");

fgets(newBook.title, MAX_TITLE_LENGTH, stdin);
```

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


        printf("Enter Book Author: ");

        fgets(newBook.author, MAX_AUTHOR_LENGTH,

stdin);

        newBook.author[strcspn(newBook.author, "\n")] =

'\0';


        printf("Enter Stock (Number of Copies): ");

        scanf("%d", &newBook.stock);


        books[bookCount++] = newBook;

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

        }


        // Function to edit a book

        void editBook() {

        int book_id, found = 0;

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

        scanf("%d", &book_id);

        getchar();
```

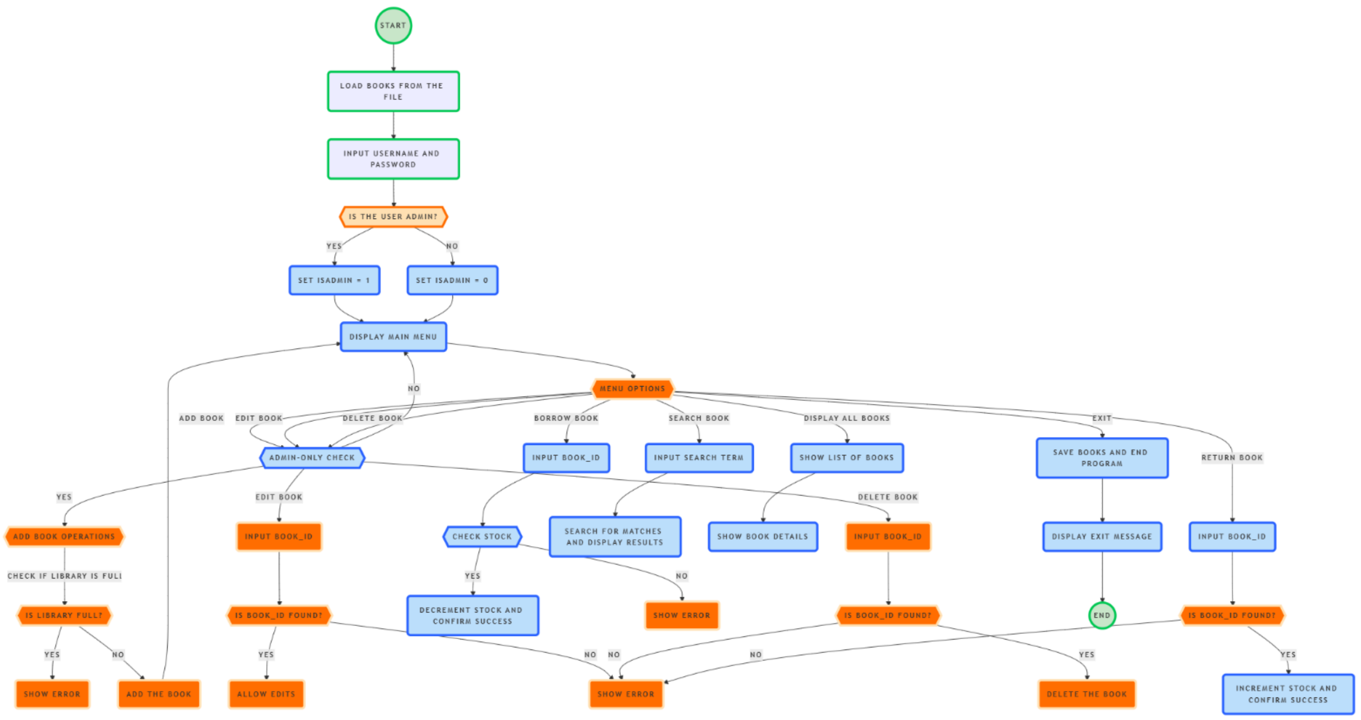
```
}

    for (int i = 0; i < bookCount; i++) {
    if (books[i].book_id == book_id) {
    printf("Editing Book ID %d\n", book_id);
    printf("Enter New Title: ");
    fgets(books[i].title, MAX_TITLE_LENGTH, stdin);
    books[i].title[strcspn(books[i].title, "\n")] = '\0';

    printf("Enter New Author: ");
    fgets(books[i].author, MAX_AUTHOR_LENGTH,
stdin);
    books[i].author[strcspn(books[i].author, "\n")] = '\0';

    printf("Enter New Stock: ");
    scanf("%d", &books[i].stock);

    printf("Book updated successfully!\n");
    found = 1;
    break;
    }
    }
```

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

        }


        // Function to delete a book

        void deleteBook() {

        int book_id, found = 0;

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

        scanf("%d", &book_id);


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

        if (books[i].book_id == book_id) {

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

        books[j] = books[j + 1];

        }

        bookCount--;

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

        found = 1;

        break;
        }

        }


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

        }


        // Function to search for a book

        void searchBook() {

        char searchTerm[MAX_TITLE_LENGTH];

        int found = 0;


        printf("Enter Book Title or Author to search: ");

        fgets(searchTerm, MAX_TITLE_LENGTH, stdin);

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


        printf("\nSearch Results:\n");

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

        if (strstr(books[i].title, searchTerm) != NULL ||
strstr(books[i].author, searchTerm) != NULL) {

        printf("Book ID: %d\n", books[i].book_id);
        printf("Title: %s\n", books[i].title);

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

        printf("Stock: %d\n\n", books[i].stock);

        found = 1;

        }

        }


        if (!found) printf("No books found matching the
search term.\n");

        }


        // Function to display all books

        void displayBooks() {

        if (bookCount == 0) {

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

        return;

        }


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

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

        printf("Book ID: %d\n", books[i].book_id);
        printf("Title: %s\n", books[i].title);

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

        printf("Stock: %d\n\n", books[i].stock);

        }

        }



        // Function to borrow a book

        void borrowBook() {

        int book_id, found = 0;

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

        scanf("%d", &book_id);



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

        if (books[i].book_id == book_id) {

        if (books[i].stock > 0) {

        books[i].stock--;

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

        } else {

        printf("Book is out of stock.\n");

        }

        found = 1;
```

```
        break;

        }

        }



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

        }



        // Function to return a book

        void returnBook() {

        int book_id, found = 0;

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

        scanf("%d", &book_id);



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

        if (books[i].book_id == book_id) {

        books[i].stock++;

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

        found = 1;

        break;

        }
        }


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


        // Function to handle login
        void login() {
        char username[20], password[20];


        printf("Login:\n");
        printf("Username: ");
        scanf("%s", username);
        printf("Password: ");
        scanf("%s", password);


        if (strcmp(username, "admin") == 0 &&
strcmp(password, "pass123") == 0) {
        printf("Admin access granted.\n");
        isAdmin = 1;
        } else {
```

```
        printf("User access granted. Limited
permissions.\n");
        isAdmin = 0;
        }
        }
```

6.Flow Execution



1. System Initialization (Starting the Program)

- **Load Data from File: When the program starts, it**

loads the existing book data from a file (library.txt) into the memory. If the file does not exist, the system will notify the user that there is no existing data, and it will start with an empty library.

- **Login: The system p rompts the user to log in. It checks for credentials to differentiate between admin and user roles:**
  - **Admin: Username = "admin", Password = "admin123"**
  - **User: Username = "user", Password = "user123"**
- **Based on the credentials entered, the system assigns either admin privileges or user privileges. If invalid credentials are entered, the system will terminate.**

---

2. Main Menu Loop (After Successful Login)

- **Menu Display: Once logged in, the user is presented with a main menu that offers various actions, such as:**
  - **Add Book (Admin only)**
  - **Edit Book (Admin only)**
  - **Delete Book (Admin only)**
  - **Search Book**
  - **Display All Books**
  - **Borrow Book**
  - **Return Book**

    - **Exit**

- **User Selection: The user enters a choice from the menu, and the program validates the choice to ensure it's within the given range (1-8).**

---

3. Executing User Choices

Depending on the user's choice from the menu, the system performs the following actions:

- **Option 1: Add Book (Admin Only):**
    - **The admin enters the book's ID, title, author, and stock (number of copies).**
    - **The system checks for duplicate book IDs. If the ID already exists, the admin is asked to provide a new one.**
    - **After adding the book, the system updates the book list and increases the bookCount.**
- **Option 2: Edit Book (Admin Only):**
    - **Admin enters the book ID to edit.**
    - **If the book is found, the admin can modify the book's title, author, and stock.**
    - **The updated book details are saved in the book list.**
- **Option 3: Delete Book (Admin Only):**
    - **Admin enters the book ID to delete.**
    - **If the book is found, it is removed from the list, and all books are shifted to close the gap.**

        - **The bookCount is reduced.**

    - **Option 4: Search Book:**
        - **The user enters a search keyword (title or author).**
        - **The system searches for matching books and displays their details (ID, title, author, stock).**
        - **If no books are found, it notifies the user.**
- **Option 5: Display All Books:**
    - **The system displays a list of all available books, showing each book's ID, title, author, and available stock.**
- **Option 6: Borrow Book:**
    - **The user enters the book ID of the book they want to borrow.**
    - **If the book is available (i.e., stock > 0), the system decreases the stock by 1.**
    - **If the book is unavailable (i.e., stock = 0), it informs the user that the book is out of stock.**
- **Option 7: Return Book:**
    - **The user enters the book ID of the book they want to return.**
    - **The system increases the stock by 1 to reflect the returned book.**
- **Option 8: Exit:**
    - **When the user selects "Exit," the program saves the current book data (book details and stock) into the**

library.txt file.

- The system displays a message indicating that the data has been saved and then exits.

---

4. Input Validation

- **Integer Validation: For fields like book ID,**
- **stock, and menu options, the system ensures the input is a valid integer within the expected range.**
- **String Validation: For book titles, authors, and login credentials, the system ensures no empty or invalid strings are entered. It also strips unwanted newline characters from user inputs.**

---

5. Data Persistence

- **File Handling: At the start of the program, existing book data is loaded from the library.txt file. During the session, any changes made (adding/editing/deleting books, borrowing/returning books) are stored in memory. At the end of the session, the updated data is saved back to the file, ensuring that the book data persists between program runs.**

---

6. Program Termination

- **Data Saving: When the user exits the program (via option 8), the updated book data (book ID, title, author, stock) is** saved to the library.txt file. This ensures the next time the system is run, the data will be preserved.

- **Exit Message:** The system prints a message saying "Exiting... Data saved" and then terminates.

Conclusion

The Library Management System project successfully demonstrates how a computer program can be used to automate the process of managing books in a library. It allows both admin and user roles to interact with the system based on their privileges. The admin has full control over the library's catalog, including adding, editing, and deleting books, while the user can perform basic operations such as borrowing, returning, and searching for books.