**UNIVERSITY INSTITUTE OF COMPUTING**

**PROJECT REPORT**

**ON**

**BANKING MANAGEMENT SYSTEM**

Program Name: BCA

Subject Name/Code: Data Structures(24CAP-152)

**Submitted by: Submitted to:**

**Name:** Anisha Kumari **Name: Monika Choudhary**

**UID:** 24BCA10498 **Designation: Professor**

**Section:** BCA – 7 “B”

ABSTRACT

Introduction:

The **Library Management System (LMS)** is a software-based solution designed to automate and streamline the process of managing a library's operations. The system efficiently handles the addition, deletion, and modification of books and member records. It also supports the issuing and returning of books, keeping track of due dates and calculating fines for overdue returns.

Built using core data structures such as **arrays**, **linked lists**, and **hash tables**, the system ensures quick data access and efficient memory usage. The search and sort functionalities allow users to locate books or members by various attributes like name, ID, or category.

The LMS improves the traditional manual process, reducing human errors and enhancing productivity. It is user-friendly, scalable, and can be extended to support digital libraries, barcode scanning, and user authentication. Overall, this project demonstrates the practical application of **data structures and algorithms (DSA)** in solving real-world problems in the domain of information management.

Technique ­­­­­­­­­­­­­­­­­­­--­­---­­

#### 1. ****Data Structures****

* **Arrays**:  
  Used for storing fixed-size lists such as book categories or predefined user roles. Arrays enable quick access to elements using indexing.
* **Linked Lists**:  
  Employed to manage dynamic collections such as the list of currently issued books or members. Linked lists allow easy insertion and deletion of records without reallocation or reorganization of the data structure.
* **Hash Tables (Hash Maps)**:  
  Used for efficient storage and retrieval of book and member records based on unique identifiers like ISBN or student ID. Hashing allows for average-case constant time complexity (O(1)) for search, insert, and delete operations.
* **Stacks**:  
  Utilized in implementing undo/redo features, such as undoing a book issue or return operation.
* **Queues**:  
  Helpful in managing waitlists for books that are currently issued. The first-come, first-served approach ensures fairness.
* **Trees (Optional / Advanced)**:  
  Binary Search Trees (BST) or AVL Trees can be used for maintaining sorted records of books or members, enabling fast in-order traversals and range queries.

#### 2. ****Algorithms****

* **Searching Algorithms**:
  + **Linear Search**: Used when searching in unsorted lists.
  + **Binary Search**: Applied on sorted lists for fast lookup of book titles or IDs.
* **Sorting Algorithms**:
  + **Merge Sort / Quick Sort**: Used to sort books or members based on different fields like title, author, or date of issue.
  + Sorting improves the efficiency of binary search and enhances the user experience in search and browse operations.
* **Hashing**:
  + Hash functions are used to uniquely index books and users for fast access.
  + Collision handling techniques like **chaining** or **open addressing** may be implemented.
* **Graph Algorithms (Optional)**:
  + If the system is extended to manage inter-library networks or digital catalog relationships, graph traversal algorithms (like BFS/DFS) can be used.

#### 3. ****File Handling / Persistence****

* Although not a DSA technique, file handling is used to store data persistently.
* Data structures are serialized and saved to files (e.g., .txt or .csv) to maintain state between session

SYSYEM CONFIGURATION:

The following system configuration was used for the development, testing, and execution of the Library Management System:

#### 💻 ****Hardware Requirements****

* **Processor**: Intel Core i3 or higher
* **RAM**: 4 GB minimum (8 GB recommended)
* **Hard Disk**: Minimum 250 MB free space
* **Display**: 1024×768 resolution or higher
* **Input Devices**: Keyboard, Mouse

#### 🖥️ ****Software Requirements****

* **Operating System**: Windows 10/11, Linux (Ubuntu), or macOS
* **Programming Language**:
  + C++ / Java / Python (choose based on what you used)
* **Compiler / Interpreter**:
  + For C++: GCC (MinGW for Windows)
  + For Java: JDK 8 or above
  + For Python: Python 3.8 or above
* **IDE / Code Editor** (any of the following):
  + Visual Studio Code
  + PyCharm / IntelliJ IDEA
  + Eclipse
  + Code::Blocks (for C++)
* **Version Control (Optional but recommended)**:
  + Git and GitHub for source code management

#### 🗃️ ****Optional Technologies (for extended features)****

* **Database** (if integrated): SQLite / MySQL
* **GUI Framework** (for UI-based systems):
  + Tkinter (Python)
  + JavaFX or Swing (Java)
  + Qt (C++)
* **Libraries/Packages**:
  + Python: pickle, csv, tkinter
  + Java: java.util, java.io, javax.swing
  + C++: Standard Template Library (STL)

**SUMMARY :**

The **Library Management System** is a software application designed to simplify and automate the daily operations of a library. This project focuses on the efficient organization, storage, and retrieval of information related to books, members, and transactions using core **Data Structures and Algorithms (DSA)**.

Through the use of **arrays**, **linked lists**, **hash tables**, and **search/sort algorithms**, the system provides fast and reliable features such as book issue and return, member management, due date tracking, and fine calculation. The integration of DSA concepts ensures optimal performance and scalability, making it suitable for both small and large library environments.

This project not only demonstrates a practical application of theoretical DSA knowledge but also promotes structured coding practices, logical thinking, and problem-solving skills. It can be further enhanced with database integration, a graphical user interface (GUI), or network capabilities to support digital librariIn conclusion, the Library Management System is a robust, efficient, and scalable solution for managing library operations, making it a valuable tool for educational institutions and a strong DSA project.

Code:-

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_BOOKS 100

#define MAX\_TITLE\_LENGTH 100

#define MAX\_AUTHOR\_LENGTH 50

typedef struct {

    int id;

    char title[MAX\_TITLE\_LENGTH];

    char author[MAX\_AUTHOR\_LENGTH];

    int year;

    int is\_available; // 1 for available, 0 for borrowed

} Book;

Book library[MAX\_BOOKS];

int book\_count = 0;

void display\_menu();

void add\_book();

void display\_books();

void search\_book();

void update\_book();

void delete\_book();

void save\_to\_file();

void load\_from\_file();

int get\_valid\_int\_input(const char \*prompt);

void clear\_input\_buffer();

int main() {

    load\_from\_file();

    int choice;

    do {

        display\_menu();

        printf("Enter your choice: ");

        scanf("%d", &choice);

        clear\_input\_buffer();

        switch(choice) {

            case 1:

                add\_book();

                break;

            case 2:

                display\_books();

                break;

            case 3:

                search\_book();

                break;

            case 4:

                update\_book();

                break;

            case 5:

                delete\_book();

                break;

            case 6:

                save\_to\_file();

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

                break;

            default:

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

        }

    } while(choice != 6);

    return 0;

}

void display\_menu() {

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

    printf("1. Add a new book\n");

    printf("2. Display all books\n");

    printf("3. Search for a book\n");

    printf("4. Update book details\n");

    printf("5. Delete a book\n");

    printf("6. Exit\n");

}

void add\_book() {

    if (book\_count >= MAX\_BOOKS) {

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

        return;

    }

    Book new\_book;

    printf("\nEnter book details:\n");

    new\_book.id = book\_count + 1;

    printf("Title: ");

    fgets(new\_book.title, MAX\_TITLE\_LENGTH, stdin);

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

    printf("Author: ");

    fgets(new\_book.author, MAX\_AUTHOR\_LENGTH, stdin);

    new\_book.author[strcspn(new\_book.author, "\n")] = '\0'; // Remove newline

    new\_book.year = get\_valid\_int\_input("Publication Year: ");

    new\_book.is\_available = 1;

    library[book\_count++] = new\_book;

    printf("Book added successfully with ID: %d\n", new\_book.id);

}

void display\_books() {

    if (book\_count == 0) {

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

        return;

    }

    printf("\n%-5s %-30s %-20s %-10s %s\n", "ID", "Title", "Author", "Year", "Status");

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

    for (int i = 0; i < book\_count; i++) {

        printf("%-5d %-30s %-20s %-10d %s\n",

               library[i].id,

               library[i].title,

               library[i].author,

               library[i].year,

               library[i].is\_available ? "Available" : "Borrowed");

    }

}

void search\_book() {

    if (book\_count == 0) {

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

        return;

    }

    int choice;

    printf("\nSearch by:\n");

    printf("1. ID\n");

    printf("2. Title\n");

    printf("3. Author\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    clear\_input\_buffer();

    switch(choice) {

        case 1: {

            int id = get\_valid\_int\_input("Enter book ID: ");

            int found = 0;

            for (int i = 0; i < book\_count; 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("Year: %d\n", library[i].year);

                    printf("Status: %s\n", library[i].is\_available ? "Available" : "Borrowed");

                    found = 1;

                    break;

                }

            }

            if (!found) {

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

            }

            break;

        }

        case 2: {

            char title[MAX\_TITLE\_LENGTH];

            printf("Enter book title: ");

            fgets(title, MAX\_TITLE\_LENGTH, stdin);

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

            int found = 0;

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

            for (int i = 0; i < book\_count; i++) {

                if (strstr(library[i].title, title) != NULL) {

                    printf("ID: %d, Title: %s, Author: %s, Year: %d, Status: %s\n",

                           library[i].id,

                           library[i].title,

                           library[i].author,

                           library[i].year,

                           library[i].is\_available ? "Available" : "Borrowed");

                    found = 1;

                }

            }

            if (!found) {

                printf("No books found with title containing '%s'.\n", title);

            }

            break;

        }

        case 3: {

            char author[MAX\_AUTHOR\_LENGTH];

            printf("Enter author name: ");

            fgets(author, MAX\_AUTHOR\_LENGTH, stdin);

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

            int found = 0;

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

            for (int i = 0; i < book\_count; i++) {

                if (strstr(library[i].author, author) != NULL) {

                    printf("ID: %d, Title: %s, Author: %s, Year: %d, Status: %s\n",

                           library[i].id,

                           library[i].title,

                           library[i].author,

                           library[i].year,

                           library[i].is\_available ? "Available" : "Borrowed");

                    found = 1;

                }

            }

            if (!found) {

                printf("No books found by author '%s'.\n", author);

            }

            break;

        }

        default:

            printf("Invalid choice.\n");

    }

}

void update\_book() {

    if (book\_count == 0) {

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

        return;

    }

    int id = get\_valid\_int\_input("Enter the ID of the book to update: ");

    int found = -1;

    for (int i = 0; i < book\_count; i++) {

        if (library[i].id == id) {

            found = i;

            break;

        }

    }

    if (found == -1) {

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

        return;

    }

    printf("\nCurrent book details:\n");

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

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

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

    printf("Year: %d\n", library[found].year);

    printf("Status: %s\n", library[found].is\_available ? "Available" : "Borrowed");

    printf("\nEnter new details (leave blank to keep current value):\n");

    char input[MAX\_TITLE\_LENGTH];

    printf("Title [%s]: ", library[found].title);

    fgets(input, MAX\_TITLE\_LENGTH, stdin);

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

    if (strlen(input) > 0) {

        strcpy(library[found].title, input);

    }

    printf("Author [%s]: ", library[found].author);

    fgets(input, MAX\_AUTHOR\_LENGTH, stdin);

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

    if (strlen(input) > 0) {

        strcpy(library[found].author, input);

    }

    printf("Year [%d]: ", library[found].year);

    fgets(input, MAX\_TITLE\_LENGTH, stdin);

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

    if (strlen(input) > 0) {

        library[found].year = atoi(input);

    }

    printf("Status (1 for Available, 0 for Borrowed) [%d]: ", library[found].is\_available);

    fgets(input, MAX\_TITLE\_LENGTH, stdin);

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

    if (strlen(input) > 0) {

        library[found].is\_available = atoi(input) ? 1 : 0;

    }

    printf("Book details updated successfully.\n");

}

void delete\_book() {

    if (book\_count == 0) {

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

        return;

    }

    int id = get\_valid\_int\_input("Enter the ID of the book to delete: ");

    int found = -1;

    for (int i = 0; i < book\_count; i++) {

        if (library[i].id == id) {

            found = i;

            break;

        }

    }

    if (found == -1) {

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

        return;

    }

    printf("You are about to delete this book:\n");

    printf("ID: %d, Title: %s, Author: %s\n",

           library[found].id, library[found].title, library[found].author);

    printf("Are you sure? (1 for Yes, 0 for No): ");

    int confirm;

    scanf("%d", &confirm);

    clear\_input\_buffer();

    if (confirm == 1) {

        // Shift all books after the found index one position left

        for (int i = found; i < book\_count - 1; i++) {

            library[i] = library[i + 1];

        }

        book\_count--;

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

    } else {

        printf("Deletion cancelled.\n");

    }

}

void save\_to\_file() {

    FILE \*file = fopen("library.dat", "wb");

    if (file == NULL) {

        printf("Error opening file for writing.\n");

        return;

    }

    fwrite(&book\_count, sizeof(int), 1, file);

    fwrite(library, sizeof(Book), book\_count, file);

    fclose(file);

}

void load\_from\_file() {

    FILE \*file = fopen("library.dat", "rb");

    if (file == NULL) {

        printf("No existing data file found. Starting with empty library.\n");

        return;

    }

    fread(&book\_count, sizeof(int), 1, file);

    fread(library, sizeof(Book), book\_count, file);

    fclose(file);

}

int get\_valid\_int\_input(const char \*prompt) {

    int value;

    char input[20];

    while (1) {

        printf("%s", prompt);

        fgets(input, sizeof(input), stdin);

        if (sscanf(input, "%d", &value) == 1) {

            return value;

        }

        printf("Invalid input. Please enter a valid integer.\n");

    }

}

void clear\_input\_buffer() {

    int c;

    while ((c = getchar()) != '\n' && c != EOF);

}

**OUTPUT:**

pgsql

CopyEdit

===== Library Management System =====

1. Add New Book

2. Display All Books

3. Search Book by Title

4. Register New Member

5. Issue Book

6. Return Book

7. Exit

Enter your choice: \_

#### 📚 2. ****Adding a New Book****

mathematica

CopyEdit

Enter Book Title: Data Structures in C++

Enter Author Name: N. Narasimha Karumanchi

Enter ISBN: 9781234567890

Enter Quantity: 5

Book added successfully!

#### 🔍 3. ****Searching for a Book****

yaml

CopyEdit

Enter Book Title to Search: Data Structures in C++

Book Found!

Title: Data Structures in C++

Author: N. Narasimha Karumanchi

ISBN: 9781234567890

Available Copies: 5

**PROCESS :**

The **Library Management System** is developed using fundamental **Data Structures and Algorithms (DSA)** to manage library operations such as book management, member registration, issuing and returning books, and tracking availability. The development process involves several stages as outlined below:

#### 🔹 ****1. Requirement Analysis****

* Identify the core functionalities required:
  + Book addition, deletion, and search
  + Member registration and management
  + Issuing and returning books
  + Tracking availability and due dates
* Determine suitable data structures for each function (e.g., arrays, linked lists, hash tables).

#### 🔹 ****2. System Design****

* Design modules based on functionalities:
  + **Book Management Module**
  + **Member Management Module**
  + **Issue/Return Module**
* Choose appropriate DSA techniques:
  + **Hash Tables** for quick lookup of books/members
  + **Linked Lists** for dynamic storage
  + **Stacks/Queues** for managing undo actions or waitlists

#### 🔹 ****3. Data Structure Implementation****

* Implement core data structures:
  + Custom **Book** and **Member** classes/structures
  + Maintain lists of books and members using **linked lists** or **arrays**
  + Use **hashing** for fast retrieval using keys like ISBN or Member ID

#### 🔹 ****4. Functionality Development****

* Code the core functionalities:
  + **Add/Delete Book**: Insert/remove nodes from a list
  + **Search Book/Member**: Apply linear or binary search
  + **Issue/Return Book**: Check availability and update records
  + **Fine Calculation**: Use date difference algorithms if needed

#### 🔹 ****5. File Handling / Persistence (Optional)****

* Add basic file I/O for saving data:
  + Store book and member records in .txt or .csv files
  + Load data on program start, save on exit