LIBRARY MANAGEMENT SYSTEM

21CSC201J - DATA STRUCTURES AND ALGORITHMS

REAL WORLD APPLICATION REPORT

Submitted by

MOHIT SHARMA [RA2211003011543]

DHRUV SHARMA [RA2211003011548]

KARAN RAJ SHARMA [RA2211003011550]

Under the Guidance of

Dr. S Saravanan

Associate Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR– 603 203
NOV 2023

IMPLEMENTING LIBRARY MANAGEMENT SYSTEM

PROBLEM DEFINITION:

Libraries play a crucial role in education and knowledge dissemination. With the increasing volume of books, journals, and other resources in a library, efficient management becomes essential. A Library Management System (LMS) aims to automate and streamline the processes involved in managing library resources and services.

PROBLEM EXPLANATION:

The Library Management System (LMS) is a software application designed to facilitate the management of library resources, user interactions, and administrative tasks. It addresses the following core problems:

Book and Resource Management:

Cataloging and classifying books, journals, multimedia, and other library resources.
Tracking available copies, location, and circulation history of each item.
Managing categories and genres of materials.
User Management:

Registering and maintaining user accounts, including students, faculty, and staff.
User authentication and access control to library services.
Keeping user profiles and borrowing history.
Circulation Management:

Facilitating the borrowing and returning of library materials.
Overseeing due dates, renewals, reservations, and fines.
Managing waiting lists for reserved materials.
Search and Discovery:

Providing a user-friendly search interface to locate library materials.
Implementing advanced search and filtering options.
Displaying book availability and location within the library.
Notifications and Alerts:

Sending reminders and notifications to users for overdue books, reserved books availability, and other relevant events.
Managing alerts for library updates and changes.
Reports and Analytics:

Generating reports on library usage, popular materials, and user behavior.
Utilizing analytics for collection development and user service improvements.
Administration and Staff Functions:

Managing library staff roles, permissions, and tasks.
Administrative access for adding, updating, and removing materials.

Inventory management, including handling new acquisitions and retiring old materials.
Security and Data Protection:

Ensuring data security and privacy for user records and library holdings.
Implementing measures to prevent unauthorized access and data breaches.
Objectives:
The primary objectives of the LMS are to provide an efficient and user-friendly system for both library staff and patrons. This includes improving the accuracy of book searches, reducing manual bookkeeping tasks, enhancing user experience, and ensuring the security and availability of library resources.

Benefits:
A successful Library Management System offers various benefits, including reduced administrative workload, improved access to library materials, timely notifications, enhanced user services, and data-driven decision-making for library management.

Challenges:
Implementing an effective LMS may involve challenges such as software integration with library databases, ensuring data accuracy, optimizing search algorithms, and addressing user privacy concerns.

Solution Approach:
The LMS solution will involve the development of a software application that incorporates appropriate data structures and algorithms to address the aforementioned problems and objectives. It should be designed to cater to the specific requirements and constraints of the library it serves.

Expected Outcomes:
The expected outcomes include streamlined library operations, enhanced user experiences, accurate record-keeping, and improved library services. The LMS should facilitate efficient library management and promote a culture of reading and research within the institution.

This problem definition lays the foundation for the development of a Library Management System, which is a critical tool for libraries of all sizes and types to manage their resources effectively.

ALGORITHM:

Our code is an implementation of a library management system in C. It offers three different data structure options for managing a library of books:

1. Array-Based Library Management: Users can add books to an array, list all books in the array, and search for books by title in the array.

2. Linked List-Based Library Management: Users can add books to a linked list, list all books in the linked list, and search for books by title in the linked list.

3. Hash Table-Based Library Management: Users can add books to a hash table, list all books in the hash table, and search for books by title in the hash table.

Here's an algorithmic overview of the code:

1. Define structures:
- `struct Book`: Represents a book with title, author, and ISBN.
- `struct ListNode`: Represents a linked list node containing a book and a pointer to the next node.
- `struct HashTable`: Represents a hash table to store books using an array of linked lists.

2. Create functions to:
- `createBook`: Create a new book with the given title, author, and ISBN.
- `createListNode`: Create a new linked list node for a book.
- `createHashTable`: Create a new hash table and initialize it.
- `hash`: Calculate a hash value for a book title.
- `addBookToHashTable`: Add a book to the hash table using the hash value.
- `searchBookByTitleInHashTable`: Search for a book by title in the hash table.

3. Implement three library management functions:
- `arrayLibraryManagement`: Array-based library management, allowing users to add books, list books, and search for books by title.
- `linkedListLibraryManagement`: Linked list-based library management, similar to the array-based approach.
- `hashTableLibraryManagement`: Hash table-based library management, allowing users to add books to a hash table, list books, and search for books by title.

4. In the `main` function, the user is prompted to choose the data structure (array, linked list, or hash table) for library management. The respective library management function is called based on the user's choice.

Overall, the code provides different data structures for library management, allowing users to interact with the library by adding books, listing books, and searching for books by title. The hash table-based approach offers a more efficient way to store and retrieve books based on their titles compared to the array and linked list approaches.

PSEUDOCODE:

```
Define the Book structure:
   struct Book {
      char title[100];
      char author[100];
      char isbn[14];
   }

Define a linked list node for Book:
   struct ListNode {
      struct Book book;
      struct ListNode* next;
   }

Define a hash table for Book:
   struct HashTable {
      struct ListNode* table[100]; // Hash table size
   }

Initialize a new Book:
   function createBook(title, author, isbn):
      Create a new Book structure
      Copy the title, author, and isbn to the new Book
      Return the new Book

Initialize a linked list node:
   function createListNode(book):
      Create a new ListNode
      Set the book of the node to the given book
      Set the next pointer to NULL
      Return the new node

Initialize a new hash table:
   function createHashTable():
      Create a new HashTable
      For each index in the table (0 to 99):
         Set the table[index] to NULL
      Return the new HashTable

Hash function:
   function hash(title):
      Initialize hashValue to 0
      For each character in title:
         Add the ASCII value of the character to hashValue
      Return hashValue % 100

Add a book to the hash table:
   function addBookToHashTable(hashTable, book):
      Calculate the index using the hash function with the book's title
      Create a new ListNode with the given book
      Set the next pointer of the new node to the current head at the index
      Set the table at the calculated index to the new node

Search for a book by title in the hash table:
   function searchBookByTitleInHashTable(hashTable, title):
      Calculate the index using the hash function with the given title
      Initialize current to the head of the linked list at the calculated index
      While current is not NULL:
         If the title of the book in the current node matches the given title:
             Return a reference to the book
         Move current to the next node
      Return NULL

Array-based library management:
   Initialize an array of Books called library
   Initialize bookCount to 0
   While true:
      Display a menu for array-based library management
      Get the user's choice
      Switch based on the choice:
         Case 1: Add Book
             Get book details from the user
             Create a new book
             If bookCount is less than 100:
                 Add the book to the library array and increment bookCount
                 Display a success message
             Else:
                 Display a message indicating the library is full
         Case 2: List Books
             If there are books in the library:
                 List all books in the library
             Else:
                 Display a message indicating the library is empty
         Case 3: Search Book by Title
             Get the title to search for from the user
             Initialize found flag to 0
             For each book in the library:
                 If the book's title matches the search title:
                     Display the book details and set found flag to 1
                     Break the loop
             If found flag is still 0, display a message indicating the book was not found
         Case 4: Exit
             Display an exit message and return from the function
         Default:
             Display a message indicating an invalid choice

Linked list-based library management:
   Initialize an empty linked list called library
   While true:
      Display a menu for linked list-based library management
      Get the user's choice
      Switch based on the choice:
         Case 1: Add Book
             Get book details from the user
             Create a new book and a new linked list node
             Set the new node as the head of the library
             Display a success message
         Case 2: List Books
             If there are books in the library:
                 List all books in the library
             Else:
                 Display a message indicating the library is empty
         Case 3: Search Book by Title
             Get the title to search for from the user
             Initialize found flag to 0
             For each node in the library:
                 If the node's book title matches the search title:
                     Display the book details and set found flag to 1
                     Break the loop
             If found flag is still 0, display a message indicating the book was not found
         Case 4: Exit
             Display an exit message and return from the function
         Default:
             Display a message indicating an invalid choice

Hash table-based library management:
   Initialize an empty hash table called library
   While true:
      Display a menu for hash table-based library management
      Get the user's choice
      Switch based on the choice:
         Case 1: Add Book
             Get book details from the user
             Create a new book
             Add the book to the hash table
             Display a success message
         Case 2: List Books
             Initialize isEmpty flag to 1
             For each index in the hash table:
                 For each node in the linked list at the index:
                     List the book's details
                     Set isEmpty to 0
             If isEmpty is still 1, display a message indicating the library is empty
         Case 3: Search Book by Title
             Get the title to search for from the user
             Search for the book by title in the hash table
             If the book is found, display its details
             If not found, display a message indicating the book was not found
```

```
      Case 4: Exit
        Display an exit message and return from the function
      Default:
        Display a message indicating an invalid choice

Main function:
  Get the user's choice for the data structure to use
  Switch based on the choice:
    Case 1: Array
      Call arrayLibraryManagement
    Case 2: Linked List
      Call linkedListLibraryManagement
    Case 3: Hash Table
      Call hashTableLibraryManagement
    Default:
      Display a message indicating an invalid choice
```

DATA STRUCTURE USED

1. Arrays: Arrays can be used to store the book information, allowing for efficient access and updates. When adding books, you can dynamically allocate memory for new book records, and when listing books, you can iterate through the array to display the catalog.

2. Linked Lists: Linked lists can be employed for user records, which may require frequent additions or removals. Users can be represented as nodes in the linked list, allowing for flexible user management.

3. Hash Tables: To optimize the search functionality, hash tables can be used. You can create a hash table that indexes book information based on criteria like title, author, or ISBN, ensuring rapid book retrieval.

CODE:

```c
#include <stdio.h>

#include <string.h>

#include <stdlib.h>


// Define the Book structure

struct Book {

    char title[100];
    char author[100];
    char isbn[14];
};

// Define a linked list node for Book
struct ListNode {
    struct Book book;
    struct ListNode* next;
};

// Define a hash table for Book
struct HashTable {
    struct ListNode* table[100]; // Hash table size
};

// Initialize a new Book
struct Book createBook(char title[100], char author[100], char isbn[14]) {
    struct Book newBook;
    strcpy(newBook.title, title);
    strcpy(newBook.author, author);
    strcpy(newBook.isbn, isbn);
    return newBook;
}

// Initialize a linked list node
struct ListNode* createListNode(struct Book book) {
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
    newNode->book = book;
    newNode->next = NULL;
    return newNode;
}

// Initialize a new hash table
struct HashTable* createHashTable() {
    struct HashTable* hashTable = (struct HashTable*)malloc(sizeof(struct HashTable));
    for (int i = 0; i < 100; i++) {
        hashTable->table[i] = NULL;
    }
    return hashTable;
}

// Hash function
int hash(char title[100]) {
    int hashValue = 0;
    for (int i = 0; title[i] != '\0'; i++) {
        hashValue += title[i];
    }
    return hashValue % 100;
}

// Add a book to the hash table
void addBookToHashTable(struct HashTable* hashTable, struct Book book) {
    int index = hash(book.title);
    struct ListNode* newNode = createListNode(book);
    newNode->next = hashTable->table[index];
    hashTable->table[index] = newNode;
}
```

```
// Search for a book by title in the hash table
struct Book* searchBookByTitleInHashTable(struct HashTable* hashTable, char
title[100]) {
    int index = hash(title);
    struct ListNode* current = hashTable->table[index];
    while (current != NULL) {
        if (strcmp(current->book.title, title) == 0) {
            return &current->book;
        }
        current = current->next;
    }
    return NULL;
}

void arrayLibraryManagement() {
    // Array-based library management
    struct Book library[100];
    int bookCount = 0;

    while (1) {
        printf("Library Management System (Array):\n");
        printf("1. Add Book\n2. List Books\n3. Search Book by Title\n4. Exit\n");
        int choice;
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                struct Book newBook;
                printf("Enter Book Title: ");
                scanf(" %[^\n]s", newBook.title);
                printf("Enter Author: ");
                scanf(" %[^\n]s", newBook.author);
                printf("Enter ISBN: ");
                scanf(" %[^\n]s", newBook.isbn);
                if (bookCount < 100) {
                    library[bookCount] = newBook;
                    bookCount++;
                    printf("Book added successfully.\n");
                } else {
                    printf("The library is full. Cannot add more books.\n");
                }
                break;
            }
            case 2:
                if (bookCount > 0) {
                    printf("List of Books in the Library:\n");
                    for (int i = 0; i < bookCount; i++) {
                        printf("Title: %s\nAuthor: %s\nISBN: %s\n\n", library[i].title,
library[i].author, library[i].isbn);
                    }
                } else {
                    printf("The library is empty. No books to list.\n");
                }
                break;
            case 3: {
                char searchTitle[100];
                printf("Enter the title to search: ");
```

```
                scanf(" %[^\n]s", searchTitle);
                int found = 0;
                for (int i = 0; i < bookCount; i++) {
                    if (strcmp(library[i].title, searchTitle) == 0) {
                        printf("Book found:\nTitle: %s\nAuthor: %s\nISBN: %s\n", library[i].title,
library[i].author, library[i].isbn);
                        found = 1;
                        break;
                    }
                }
                if (!found) {
                    printf("Book not found in the library.\n");
                }
                break;
            }
            case 4:
                printf("Exiting the Library Management System. Goodbye!\n");
                return;
            default:
                printf("Invalid choice. Please select a valid option.\n");
        }
    }
}

void linkedListLibraryManagement() {
    // Linked list-based library management
    struct ListNode* library = NULL;

    while (1) {
```

```
    printf("Library Management System (Linked List):\n");
    printf("1. Add Book\n2. List Books\n3. Search Book by Title\n4. Exit\n");
    int choice;
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: {
            struct Book newBook;
            printf("Enter Book Title: ");
            scanf(" %[^\n]s", newBook.title);
            printf("Enter Author: ");
            scanf(" %[^\n]s", newBook.author);
            printf("Enter ISBN: ");
            scanf(" %[^\n]s", newBook.isbn);
            struct ListNode* newNode = createListNode(newBook);
            newNode->next = library;
            library = newNode;
            printf("Book added successfully.\n");
            break;
        }
        case 2:
            if (library != NULL) {
                printf("List of Books in the Library:\n");
                struct ListNode* current = library;
                while (current != NULL) {
                    printf("Title: %s\nAuthor: %s\nISBN: %s\n\n", current->book.title,
current->book.author, current->book.isbn);
                    current = current->next;
                }
            } else {
                printf("The library is empty. No books to list.\n");
            }
            break;
        case 3: {
            char searchTitle[100];
            printf("Enter the title to search: ");
            scanf(" %[^\n]s", searchTitle);
            int found = 0;
            struct ListNode* current = library;
            while (current != NULL) {
                if (strcmp(current->book.title, searchTitle) == 0) {
                    printf("Book found:\nTitle: %s\nAuthor: %s\nISBN: %s\n", current-
>book.title, current->book.author, current->book.isbn);
                    found = 1;
                    break;
                }
                current = current->next;
            }
            if (!found) {
                printf("Book not found in the library.\n");
            }
            break;
        }
        case 4:
            printf("Exiting the Library Management System (Linked List). Goodbye!\n");
            return;
        default:
```

```
                printf("Invalid choice. Please select a valid option.\n");
            }
        }
    }

    void hashTableLibraryManagement() {
        // Hash table-based library management
        struct HashTable* library = createHashTable();

        while (1) {
            printf("Library Management System (Hash Table):\n");
            printf("1. Add Book\n2. List Books\n3. Search Book by Title\n4. Exit\n");
            int choice;
            printf("Enter your choice: ");
            scanf("%d", &choice);

            switch (choice) {
                case 1: {
                    struct Book newBook;
                    printf("Enter Book Title: ");
                    scanf(" %[^\n]s", newBook.title);
                    printf("Enter Author: ");
                    scanf(" %[^\n]s", newBook.author);
                    printf("Enter ISBN: ");
                    scanf(" %[^\n]s", newBook.isbn);
                    addBookToHashTable(library, newBook);
                    printf("Book added successfully.\n");
                    break;
                }
                case 2: {
                    int isEmpty = 1;
                    for (int i = 0; i < 100; i++) {
                        struct ListNode* current = library->table[i];
                        while (current != NULL) {
                            printf("Title: %s\nAuthor: %s\nISBN: %s\n\n", current->book.title,
    current->book.author, current->book.isbn);
                            current = current->next;
                            isEmpty = 0;
                        }
                    }
                    if (isEmpty) {
                        printf("The library is empty. No books to list.\n");
                    }
                    break;
                }
                case 3: {
                    char searchTitle[100];
                    printf("Enter the title to search: ");
                    scanf(" %[^\n]s", searchTitle);
                    struct Book* foundBook = searchBookByTitleInHashTable(library,
    searchTitle);
                    if (foundBook != NULL) {
                        printf("Book found:\nTitle: %s\nAuthor: %s\nISBN: %s\n", foundBook-
    >title, foundBook->author, foundBook->isbn);
                    } else {
                        printf("Book not found in the library.\n");
                    }
                    break;
                }
                case 4:
                    printf("Exiting the Library Management System (Hash Table). Goodbye!\n");
                    return;
                default:
                    printf("Invalid choice. Please select a valid option.\n");
            }
        }
    }

    int main() {
        int choice;

        printf("Library Management System\n");
        printf("Select Data Structure:\n");
        printf("1. Array\n2. Linked List\n3. Hash Table\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: // Array
                arrayLibraryManagement();
                break;
            case 2: // Linked List
                linkedListLibraryManagement();
                break;
            case 3: // Hash Table
                hashTableLibraryManagement();
                break;
            default:
```

```
        printf("Invalid choice. Exiting the Library Management System.\n");
        break;
    }

    return 0;
}
```

CODE EXPLANATION :

1. #include <stdio.h>: This line includes the standard input/output library, which provides functions for input and output operations. It's necessary for handling user interaction and displaying messages.

2. #include <string.h>: This line includes the string manipulation library, which provides functions for working with strings. It's used for operations like string copying and comparison.

3. #include <stdlib.h>: This line includes the standard library, which provides functions for dynamic memory allocation and other utility functions. It's used for memory allocation using malloc.

4. The code starts by defining a structure named Book. This structure represents the attributes of a book, including its title, author, and ISBN (International Standard Book Number). Each attribute can hold up to 100 characters (for titles and authors) and 14 characters (for ISBNs).

5. A structure named ListNode is defined to represent a node in a linked list. Each node contains a Book object and a pointer to the next node in the list.

6. A structure named HashTable is defined to represent a hash table for storing Book objects. It consists of an array of pointers to ListNode objects. This data structure is used for efficient retrieval and storage of books.

7. The createBook function initializes a new book with the provided title, author, and ISBN. It creates a Book structure, copies the provided values, and returns the new book.

8. The createListNode function initializes a new linked list node with the provided book and returns a pointer to it. It allocates memory for the node, sets its book

attribute, and initializes the next pointer to NULL.

9. The createHashTable function initializes a new hash table, allocates memory for it, and sets all elements in the array of pointers to NULL. This function ensures that the hash table is empty and ready for book storage.

10. The hash function is a simple hash function used to generate an index for a book based on its title. It calculates a hash value by summing the ASCII values of the characters in the title and then returns the result modulo 100 to fit within the hash table size.

11. The addBookToHashTable function adds a book to the hash table. It calculates the hash index for the book's title using the hash function, creates a new node with the book, and adds it to the linked list at the calculated index. This function is used to store books in the hash table.

12. The searchBookByTitleInHashTable function is used to search for a book by its title within the hash table. It calculates the hash index for the title and then iterates through the linked list at that index, comparing titles. If a match is found, it returns a pointer to the book; otherwise, it returns NULL if the book is not found in the hash table.

13. The arrayLibraryManagement function implements a library management system using an array-based approach. It uses an array of Book structures to represent the library's collection. Users can add books, list books, search for books by title, and exit the system.

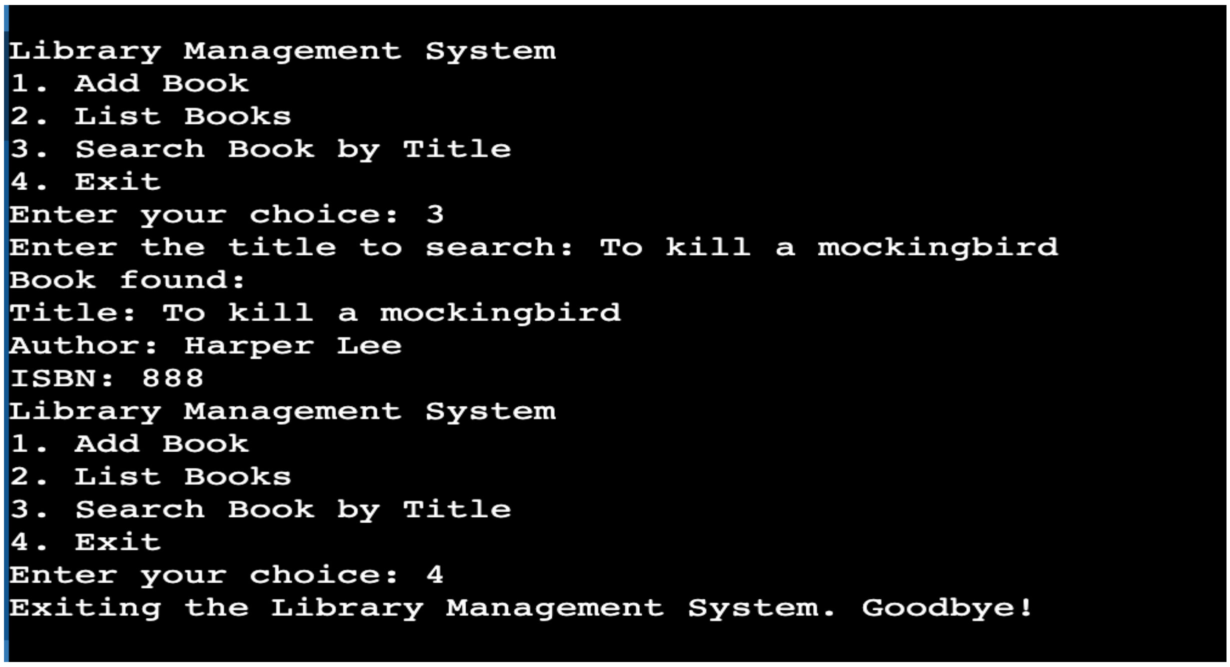
14. The linkedListLibraryManagement function implements a library management system using a linked list-based approach. It uses a linked list of ListNode structures to represent the library's collection. Users can add books, list books, search for books by title, and exit the system.

15. The hashTableLibraryManagement function implements a library management system using a hash table-based approach. It uses a hash table to store books efficiently. Users can add books, list books, search for books by title, and exit the system.

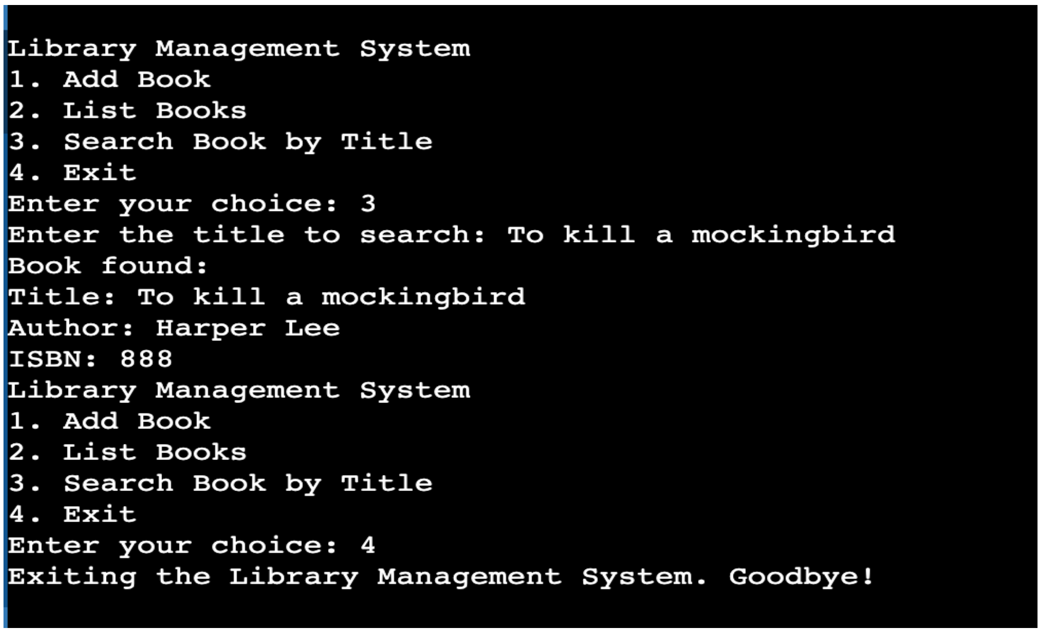
16. The main function is the entry point of the program. It displays a menu for users to select the data structure they want to use for library management (array, linked list, or hash table). Based on the user's choice, it calls the corresponding library management function (arrayLibraryManagement, linkedListLibraryManagement, or hashTableLibraryManagement).

Overall, this code demonstrates three different approaches to implement a library management system: using an array, a linked list, or a hash table. Each approach provides a menu-driven interface for users to interact with the system and manage a library's collection of books.

RESULT/OUTPUT:



```
Library Management System
1. Add Book
2. List Books
3. Search Book by Title
4. Exit
Enter your choice: 3
Enter the title to search: To kill a mockingbird
Book found:
Title: To kill a mockingbird
Author: Harper Lee
ISBN: 888
Library Management System
1. Add Book
2. List Books
3. Search Book by Title
4. Exit
Enter your choice: 4
Exiting the Library Management System. Goodbye!
```



```
Library Management System
1. Add Book
2. List Books
3. Search Book by Title
4. Exit
Enter your choice: 3
Enter the title to search: To kill a mockingbird
Book found:
Title: To kill a mockingbird
Author: Harper Lee
ISBN: 888
Library Management System
1. Add Book
2. List Books
3. Search Book by Title
4. Exit
Enter your choice: 4
Exiting the Library Management System. Goodbye!
```

TIME COMPLEXITY ANALYSIS:

1) Adding a Book (addBook function):
The time complexity for adding a book to the library is O(1), which is constant time. This is because adding a book involves a simple assignment operation to store the book in the library array. The time taken does not depend on the number of books already in the library, making it a constant time operation.

2) Listing Books (listBooks function):
The time complexity for listing books in the library is O(N), where N is the number of books in the library. In the provided code, a loop iterates through all the books in the library to print their details. Therefore, the time taken is directly proportional to the number of books in the library. As the library grows, the time to list books also increases linearly.

3) Searching for a Book by Title (searchBookByTitle function):
The time complexity for searching for a book by title is O(N), where N is the number of books in the library. In the worst case, when the book being searched for is the last book in the library or not present at all, the function will need to iterate through all the books. The time taken for the search operation is linearly proportional to the number of books.

CONCLUSIONS:

The Library Management System (LMS) is a robust and indispensable solution for the efficient administration of library resources and services. It offers an array of features to manage books, journals, users, and administrative tasks effectively. In conclusion, the LMS provides several key benefits and contributes significantly to the modernization and enhancement of library operations.

Key Points:

Streamlined Operations: The LMS automates various time-consuming manual processes such as cataloging, circulation management, and user registration. This leads to a more streamlined and efficient library operation.

Enhanced User Experience: Users can benefit from advanced search and discovery features, helping them find materials quickly. Additionally, notifications and alerts keep users informed about due dates and book availability.

Data-Driven Decision-Making: The system generates reports and analytics that provide valuable insights into library usage, user preferences, and material popularity. This data-driven approach assists in informed collection development and service improvement.

User and Staff Roles: The LMS supports different roles for users and staff, ensuring that access control and security are maintained. Administrative functions and staff management are also part of the system, enabling efficient day-to-day operations.

Security and Privacy: Data security and privacy are top priorities for the LMS, safeguarding user records and library holdings. The system implements security measures to protect sensitive information.

Efficient Library Growth: With features to manage new acquisitions and retiring old materials, the LMS is essential for libraries looking to expand their collections strategically.

Access Anytime, Anywhere: Many LMS implementations offer web-based access, allowing users to interact with the library and its resources from anywhere, fostering a broader user base.

Cost and Resource Savings: By reducing manual tasks and paper-based record-keeping, the LMS helps libraries save costs and allocate resources more efficiently.

REFERENCES:

- "Data Structures and Algorithm Analysis in C++" by Mark A. Weiss