Department of Computing

BS Computer Science

NASTP Institute of Information Technology

Submitted By: Maisum Abbas (023)

M.Qasim Shakeel (027)

Submitted To: Sir Ibaad

Semester: II

Session: Fall - 2024

Subject: Data Structures

Date of submission: 7 Jan 2026

Library Management System - Lab Report

1. Project Overview

This project implements a Library Management System using C++ programming language. The system demonstrates the practical application of fundamental data structures including dynamic arrays, linked lists, queues, stacks, and hash tables. The system provides a menu-driven interface for managing books, members, and library operations.

Objectives

The primary objectives of this project are:

- To implement a library management system using object-oriented programming principles
- To demonstrate the practical use of various data structures
- To provide a user-friendly interface for library operations
- To enable fast book search using hashing technique
- To manage member information using linked list data structure

Key Features

The library management system includes the following features:

1. **Book Management:** Add, store, and sort books by their identification numbers
2. **Member Management:** Register and manage library members using linked list
3. **Book Reservation:** Handle book reservations using queue data structure
4. **Book Returns:** Track recently returned books using stack data structure
5. **Fast Search:** Search books efficiently using hash table implementation

2. System Design and Architecture

2.1 Data Structures Used

The system implements five fundamental data structures:

Dynamic Array for Book Storage

A dynamic array is used to store book records. The array automatically resizes when it reaches capacity, demonstrating dynamic memory allocation and management. The initial capacity is set to 10 books, and when the array becomes full, it is resized to twice its current capacity.

Linked List for Member Management

A singly linked list is implemented to manage library members. Each member node contains the member's identification number, name, and a pointer to the next member in the list. This structure allows efficient insertion of new members at the end of the list.

Queue for Book Reservations

A circular queue data structure is used to manage book reservations. The queue follows the First-In-First-Out (FIFO) principle, ensuring that reservations are processed in the order they were received. The queue has a maximum capacity of 20 reservations.

Stack for Returned Books

A stack data structure is implemented to track recently returned books. The stack follows the Last-In-First-Out (LIFO) principle, allowing the library to easily track the most recently returned books. The stack can hold up to 20 returned books.

Hash Table for Book Search

A hash table is implemented to enable fast book searching by identification number. The hash function uses the modulo operation to calculate the hash index. Linear probing is used to handle collisions when two books hash to the same index.

2.2 Class Structure

The system is organized using object-oriented programming principles with a single Library class that encapsulates all data structures and operations.

Structure: Book

```
struct Book {
    int bookID;           // Book identification number
    string title;         // Book title
    string author;        // Book author name
};
```

Structure: Member

```
struct Member {
    int memberID;         // Member identification number
    string name;          // Member name
    Member* next;         // Pointer to next member node
};
```

Class: Library

The Library class contains all private member variables for the data structures and public member functions for the operations.

3. Implementation Details

3.1 Book Management (Dynamic Array)

The book management system uses a dynamic array that can grow as needed. The implementation includes:

Key Functions:

- `addBook()`: Adds a new book to the array and automatically resizes if necessary
- `resizeBookArray()`: Doubles the array capacity when the current array is full
- `sortBooks()`: Sorts books by identification number using bubble sort algorithm

Dynamic Array Implementation:

The dynamic array starts with an initial capacity of 10. When the book count reaches the capacity, a new larger array is allocated (twice the current size), all books are copied to the new array, and the old array is deallocated.

3.2 Member Management (Linked List)

Library members are managed using a singly linked list. The implementation includes:

Key Functions:

- `addMember()`: Adds a new member to the end of the linked list

Linked List Operations:

The linked list maintains a head pointer that points to the first member in the list. When adding a new member, the code traverses to the end of the list and appends the new member node.

3.3 Book Reservation (Queue)

Book reservations are managed using a circular queue implementation. The implementation includes:

Key Functions:

- `reserveBook()`: Adds a book reservation to the queue
- `processReservation()`: Processes (removes) the next reservation from the queue

Queue Implementation:

The circular queue uses an array with front and rear pointers. When adding to the queue, the rear pointer moves forward. When removing from the queue, the front pointer moves forward. When either pointer reaches the end of the array, it wraps around to the beginning.

3.4 Book Returns (Stack)

Recently returned books are tracked using a stack data structure. The implementation includes:

Key Functions:

- `returnBook()`: Pushes a returned book onto the stack
- `viewRecentReturns()`: Displays recently returned books from the stack

Stack Implementation:

The stack uses an array with a top pointer. When pushing a book, the top pointer is incremented and the book ID is stored. When viewing recent returns, books are displayed from top to bottom.

3.5 Fast Book Search (Hash Table)

Books can be searched quickly using a hash table implementation. The implementation includes:

Key Functions:

- `addToHashTable()`: Adds a book to the hash table when a new book is added
- `searchBook()`: Searches for a book by identification number

Hash Function:

The hash function uses the modulo operation: `index = bookID % hashTableSize`

Collision Handling:

Linear probing is used to handle collisions. If a slot is already occupied, the algorithm checks the next slot in sequence until an empty slot is found.

4. Menu Options and Operations

The system provides a menu-driven interface with the following options:

Option 1: Add New Book

Allows the librarian to add a new book to the library. The user must provide:

- Book identification number
- Book title
- Book author name

Option 2: Add New Member

Allows the librarian to register a new library member. The user must provide:

- Member identification number
- Member name

Option 3: Search Book by ID

Allows the librarian to search for a book using its identification number. The hash table provides O(1) average-case search time.

Option 4: Sort Books by ID

Sorts all books in the library by their identification numbers using the bubble sort algorithm.

Option 5: Reserve a Book

Adds a book reservation to the reservation queue. The user provides the book identification number.

Option 6: Process Next Reservation

Processes the next reservation in the queue (FIFO order).

Option 7: Return a Book

Records a book return by pushing the book ID onto the return stack.

Option 8: View Recent Returns

Displays the recently returned books from the stack (LIFO order).

Option 9: Display All Information

Displays complete information about the library including:

- All books in the library
- All registered members
- Current reservation queue
- Recent book returns

Option 10: Exit

Exits the library management system.

5. Code Structure and Organization

5.1 File Structure

The complete program is contained in a single C++ source file:

- `library_management_system.cpp`: Contains all class definitions, function implementations, and the main function

5.2 Function Organization

The code is organized into the following sections:

1. **Header Includes:** Standard library headers for input/output and string operations
2. **Structure Definitions:** Book and Member structure definitions
3. **Class Definition:** Library class with all member variables and functions
4. **Helper Functions:** Menu display and input functions
5. **Main Function:** Program entry point and menu loop

5.3 Object-Oriented Principles

The implementation follows object-oriented programming principles:

- **Encapsulation:** All data structures are private members of the Library class
- **Abstraction:** Users interact with the system through well-defined public methods
- **Modularity:** Each operation is implemented in a separate function

6. Sample Execution and Output

6.1 Adding Books

When adding a book, the system prompts for the book ID, title, and author. The book is added to both the dynamic array and the hash table.

```
--- ADD NEW BOOK ---
Enter Book ID: 101
Enter Book Title: Introduction to C++
Enter Book Author: John Smith
Book added successfully!
```

6.2 Adding Members

When adding a member, the system prompts for the member ID and name. The member is added to the linked list.

```
--- ADD NEW MEMBER ---
Enter Member ID: 1
```

```
Enter Member Name: Alice Brown
Member added successfully!
```

6.3 Searching for a Book

The hash table enables fast book searching:

```
--- SEARCH BOOK ---
Enter Book ID to search: 101

--- BOOK FOUND ---
Book ID  : 101
Title    : Introduction to C++
Author   : John Smith
```

6.4 Displaying All Information

The display function shows complete library status:

```
========================================
         LIBRARY INFORMATION
========================================

--- ALL BOOKS (3 books) ---
ID: 101 | Title: Introduction to C++ | Author: John Smith
ID: 102 | Title: Data Structures | Author: Jane Doe
ID: 103 | Title: Programming Basics | Author: Bob Johnson

--- ALL MEMBERS ---
Member 1: ID=1, Name=Alice Brown
Member 2: ID=2, Name=Bob Davis

--- RESERVATION QUEUE (2 reservations) ---
Reservation 1: Book ID 101
Reservation 2: Book ID 102

--- RECENT RETURNS ---
Total books returned: 2
```

7. Testing and Validation

7.1 Compilation

The program compiles successfully with the following command:

```
g++ -o library_system library_management_system.cpp -std=c++11
```

7.2 Test Scenarios

The following test scenarios were validated:

1. **Adding Books:** Multiple books can be added successfully
2. **Dynamic Resizing:** When more than 10 books are added, the array automatically resizes
3. **Member Management:** Members are correctly added to the linked list
4. **Reservation Queue:** Reservations are processed in FIFO order
5. **Return Stack:** Returned books are tracked in LIFO order
6. **Book Search:** Books are found quickly using the hash table

7.3 Edge Cases Handled

The system handles various edge cases:

- Empty queues and stacks
- Full queues and stacks
- Searching for non-existent books
- Sorting when few or no books are present

8. Conclusion

This library management system successfully demonstrates the practical application of fundamental data structures in C++. The implementation provides a comprehensive solution for managing library operations while maintaining code clarity and educational value.

Key Learnings

Through this project, the following concepts were reinforced:

1. **Dynamic Memory Management:** Understanding how to allocate and deallocate memory dynamically
2. **Data Structure Implementation:** Implementing data structures from scratch rather than using STL containers
3. **Object-Oriented Design:** Organizing code using classes and objects
4. **Algorithm Implementation:** Understanding sorting algorithms (bubble sort)
5. **Hashing Concepts:** Implementing hash functions and collision handling

Potential Enhancements

The system can be extended with additional features:

1. **Book Borrowing:** Implement actual book borrowing with due dates
2. **Persistent Storage:** Save and load data from files
3. **GUI Interface:** Create a graphical user interface
4. **Book Categories:** Add category management
5. **Search Enhancements:** Add search by title or author

Appendix: Source Code

The complete source code is provided in the file `library_management_system.cpp`. The code includes extensive comments explaining each section for educational purposes.