**International Islamic University Chattogram**

**Course Title:** Data Structures Lab

**Course Code:** CSE-2322

**Lab Report Title:** Personal Library Management System

**Submitted To:**

Mysha Sarin Kabisha

**Submitted By:**

**Name:** Syeda Afaf Anbar Faiza

**ID:** C241425

**Section:** 3AF

**Semester:** 3rd

**Submission Date:** 10/7/25

**1. Abstract**

The Personal Library Management System is a console-based application developed in C++ to organize and manage book records efficiently. The system demonstrates the practical application of core data structures, including arrays, linked lists, stacks, and queues. Each structure handles specific operations like insertion, deletion, searching, and displaying data. The array and linked list store the main book database, while stacks and queues are used to simulate reading progress and reading history. All operations are accessible via a text-based menu system, ensuring clarity and ease of use. The project highlights how choosing the appropriate data structure can improve functionality, execution speed, and memory usage. The system offers a foundational example of real-world use of data structures in personal organization systems.

**2. Introduction**

**Project Overview:**  
Library management systems are commonly used in schools, universities, and personal collections to keep track of books. In this project, a simplified version of such a system is built using C++ and several fundamental data structures. This project implements a personal library management system that helps users organize and track their reading activity.

**Purpose:**  
The system allows users to insert, delete, search, and display books using both static and dynamic data structures. It allows the user to store book records, maintain a reading queue, and view a recently read history. It helps demonstrate the efficiency and applicability of data structures in organizing real-world data.

**Tools Used:**

* Programming Language: C++
* Compiler: Code blocks
* User Interface: Text-based console interface

**3. Functional Requirements**

**Insert**

* Sorted Array: Adds a book to the main bookshelf — maintained in sorted order.
* Linked List: Inserts a book into a dynamic shelf or “new arrivals” list, also kept sorted.
* Stack: Records a book in the "Recently Read" history — Last-In-First-Out.
* Queue: Adds a book to the personal "To-Read" list — First-In-First-Out.

**Delete**

* Sorted Array: Removes a book from the main shelf.
* Linked List: Deletes a book from the dynamic/new arrivals shelf.
* Stack: Removes the most recently read book from the history.
* Queue: Removes the next book in line from the reading list after it's read.

**Search**

* Sorted Array: Searches for a book in the main shelf.
* Linked List: Searches for a book in the dynamic shelf.
* Stack: Checks if a book exists in recent reads.
* Queue: Searches the reading queue for an upcoming book.

**Display**

* Sorted Array: Displays all books in the main collection.
* Linked List: Displays dynamically added books (recent additions).
* Stack: Shows recently read books in reverse reading order.
* Queue: Displays your "To-Read" list in order.

**Exit**

* Safely exits the program.

**4. System Design**

**Architecture Overview:** The system is a text-based, menu-driven program that simulates a personal library manager. The main menu provides options to insert, delete, search, or display books. A nested menu lets the user choose which data structure they want to use — array, linked list, queue, or stack — each representing a different functionality. The program uses simple console I/O to guide the user through interactions. Internally, each option triggers specific functions tied to that data structure, which are defined modularly. This structure showcases how different data structures can be applied to manage and organize personal reading data effectively.

**Modules and Their Roles:**

* Array Module: The main storage for books in sorted order, simulating a permanent bookshelf.
* Linked List Module: A flexible “new arrivals” shelf for books added recently.
* Queue Module: Maintains a "To-Read" list — books to be read next, in order of addition.
* Stack Module: Stores a "Recently Read" history — the last books the user finished.

This division makes the system a practical tool to organize reading habits as well as demonstrate the behavior of different data structures.

**5. Program Implementation**

**Language Used:** C++

**User Interface:** Text-based console

**Sample code snippet:**

1. **Array insertion:**

void array\_insert(string item)

{

if (asize >= 100)

{

cout << "Array is full\n";

return;

}

int i = asize - 1;

while (i >= 0 && name[i] > item)

{

name[i + 1] = name[i];

i--;

}

name[i + 1] = item;

asize++;

cout << "Book is inserted\n";

}

1. **Linked List insertion:**

void insert\_llist(string item)

{

node\* newnode = new node();

newnode->bname = item;

newnode->next = nullptr;

if (head == nullptr || item < head->bname)

{

newnode->next = head;

head = newnode;

return;

}

node\* temp = head;

while (temp->next != nullptr && temp->next->bname < item)

{

temp = temp->next;

}

newnode->next = temp->next;

temp->next = newnode;

}

1. **Stack insertion**:

void push(string item)

{

if(top>=n-1)

{

cout<<"overflow\n";

return ;

}

if(bottom==-1)

bottom++;

top++;

s\_book[top]=item;

cout<<"item stored sucessfully"<<endl;

}

1. **Queue insertion:**

void enqueue (string item)

{

if(rear>=n-1)

{

cout<<"overflow\n";

return ;

}

if(front==-1)

front++;

rear++;

q\_book[rear]=item;

cout<<"item stored sucessfully"<<endl;

}

1. **Array Deletion:**

void array\_dlt(string item)

{

if(asize==0)

{

cout<<"no books in array\n";

return;

}

int pos=-1;

for(int i=0; i<asize; i++)

{

if(name[i]==item)

{

pos=i;

break;

}

}

if(pos==-1)

{

cout<<"Book not found\n";

return;

}

for(int i=pos; i<asize-1; i++)

{

name[i]=name[i+1];

}

asize--;

}

1. **Linked List Deletion:**

void dlt\_llist(string item)

{

if (head==nullptr)

{

cout<<"no books in the list\n";

return;

}

if(head->bname==item)

{

node\*temp =head;

head=head->next;

return;

}

node\* temp=head;

node\* prev;

while(temp!= nullptr && temp->bname!=item )

{

prev=temp;

temp=temp->next;

}

if(temp == nullptr)

{

cout<<"Book not found in the list\n";

return;

}

prev->next=temp->next;

delete temp;

cout<<"Book deleted\n";

}

1. **Stack Deletion**:

void pop()

{

if(bottom==-1 || bottom > top)

{

cout<<"no items"<<endl;

return ;

}

top--;

cout<<"item deleted sucessfully\n";

}

1. **Queue Deletion:**

void dequeue()

{

if(front==-1 || front > rear)

{

cout<<"no items"<<endl;

return ;

}

front++;

cout<<"item deleted sucessfully\n";

}

1. **Array Search:**

void array\_search(string item)

{

if(asize==0)

{

cout<<"no books in array\n";

return;

}

int pos=-1;

for(int i=0; i<asize; i++)

{

if(name[i]==item)

{

cout<<"Book found in position: "<<i+1;

cout<<"\n";

return;

}

}

if(pos==-1)

cout<<"Book not found\n";

}

1. **Linked List Search:**

void search\_llist(string item)

{

if(head==nullptr)

{cout<<"no books in the list\n";

return;

}

node\* temp=head;

int pos=1;

while(temp!= nullptr && temp->bname!=item)

{

temp=temp->next;

pos++;

}

if(temp==nullptr)

cout<<"Book not found in the list\n";

else

cout<<"book found at position: "<<pos<<"\n";

}

1. **Stack Search:**

void s\_search(string item)

{

if(bottom==-1 || bottom>top)

{

cout<<"no items"<<endl;

return;

}

for(int i=bottom; i <= top;i++)

{

if(s\_book[i]== item)

{

cout<<"item found in position: "<<i+1<<endl;

return;

}

}

cout<<"item not found :(\n";

}

1. **Queue Search:**

void q\_search(string item)

{

if(front==-1 || front>rear)

{

cout<<"no items"<<endl;

return;

}

for(int i=front; i <= rear;i++)

{

if(q\_book[i]== item)

{

cout<<"item found in position: "<<i+1<<endl;

return;

}

}

cout<<"item not found :(\n";

}

1. **Array Display:**

void array\_display()

{

if(asize==0)

{

cout<<"no books in array\n";

return;

}

for(int i=0; i<asize; i++)

{

cout<<i+1<<"."<<name[i]<<" ";

}

cout<<"\n";

}

1. **Linked List Display:**

void display\_llist()

{

if(head==nullptr)

{

cout<<"no books in list\n";

return;

}

node\* temp=head;

while(temp!=nullptr)

{

cout<<temp->bname;

if(temp->next != nullptr)

cout<<" -> ";

temp=temp->next;

}

cout<<"\n";

}

1. **Stack Display:**

void s\_display()

{

if(bottom==-1 || bottom>top)

{

cout<<"no items"<<endl;

return;

}

for(int i=top; i >=bottom ;i--)

{

cout<<i+1<<"."<<s\_book[i]<<" ";

}

cout<<endl;

}

1. **Queue Display**:

void q\_display()

{

if(front==-1 || front>rear)

{

cout<<"no items"<<endl;

return;

}

int j=1;

for(int i=front; i <= rear;i++)

{

cout<<j<<"."<<q\_book[i]<<" ";

j++;

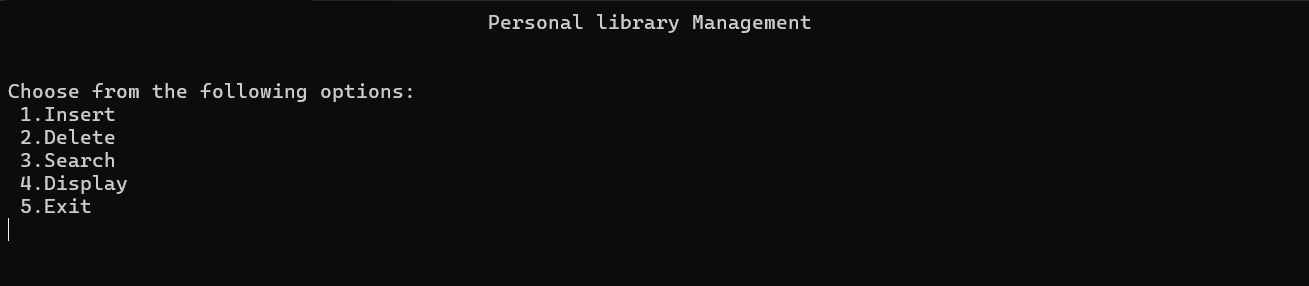
}

cout<<endl;

}

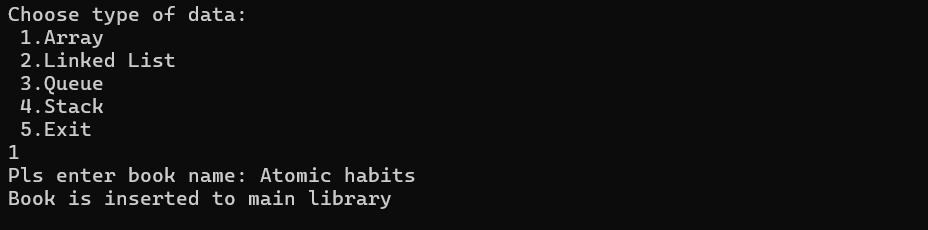
**6. Sample Input/Output**

**Menu**

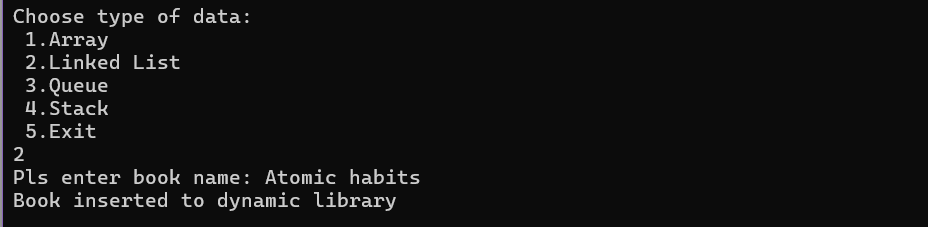
****

**Insert**

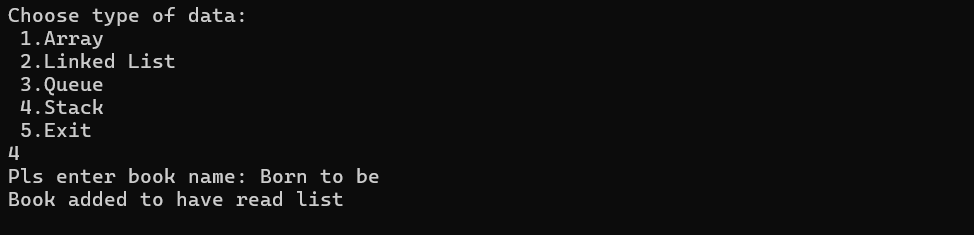
1. **Array:**

****

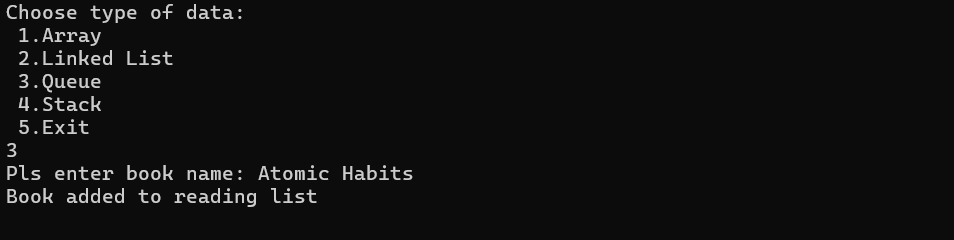
1. **Linked list:**

****

1. **Stack:**

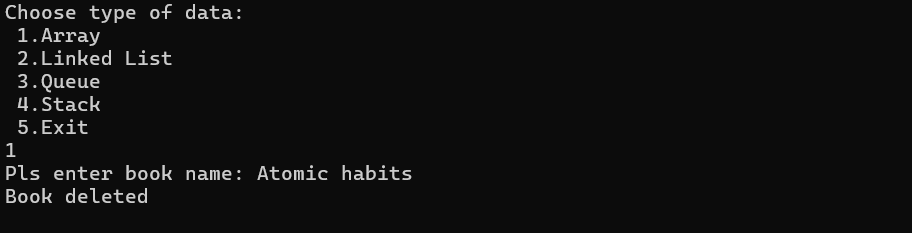
****

1. **Queue:**

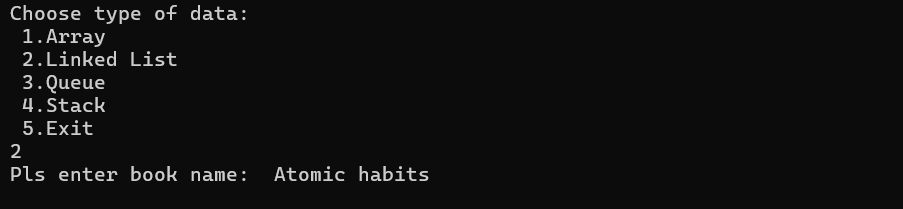
****

**Delete**

1. **Array:**

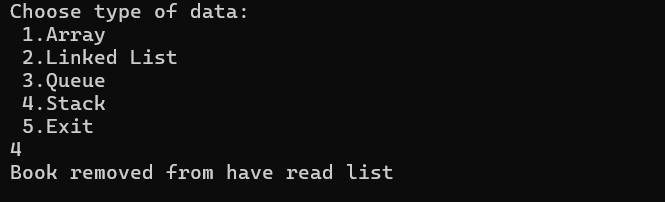
****

1. **Linked list:**

****

1. **Stack:**

**5**

****

1. **Queue:**

****

**Search**

1. **Array:**

****

1. **Linked list:**

****

1. **Stack:**

****

1. **Queue:**

****

**Display**

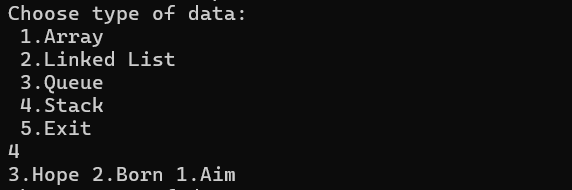
* **Array:**

****

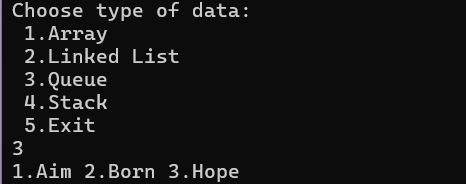
* **Linked list:**

****

* **Stack:**

****

* **Queue:**

****

**7. Testing**

| **Test Case** |  | **Operation** | **Input** | **Expected Output** | **Actual Output** | **Pass/Fail** |
| --- | --- | --- | --- | --- | --- | --- |
| TC01 |  | Insert in Sorted Array | Hope | Inserted in position 3 | Inserted in position 3 | Pass |
| TC02 |  | Pop Stack | - | Top item removed | Item popped | Pass |
| TC03 |  | Search in Linked List | Title="Born" | Book found at position 2 | Book found at position 2 | Pass |

**8. Challenges Faced**

* Ensuring correct sorted insertions in array and linked list
* Handling edge cases like overflow and underflow in stack and queue
* Managing memory manually in the linked list
* Keeping the menu responsive and avoiding infinite loops
* Debugging pointer issues in the linked list implementation

**9. Conclusion**

This project allowed for hands-on practice with important data structures and their real-world applications. Implementing a mini-library system helped reinforce concepts like sorted insertion, linear search, and stack/queue behaviors. The use of different data structures for different operations made the system more modular and efficient. In the future, the project could be expanded with features like file handling, GUI interface, and persistent storage for book records.

**10. References**

* Date structures with C- Shaums series
* https://www.geeksforgeeks.org/

**11. Appendix**

<https://github.com/SmileMaven/Personal-library-management>