Assessment Task 4: Library Management System

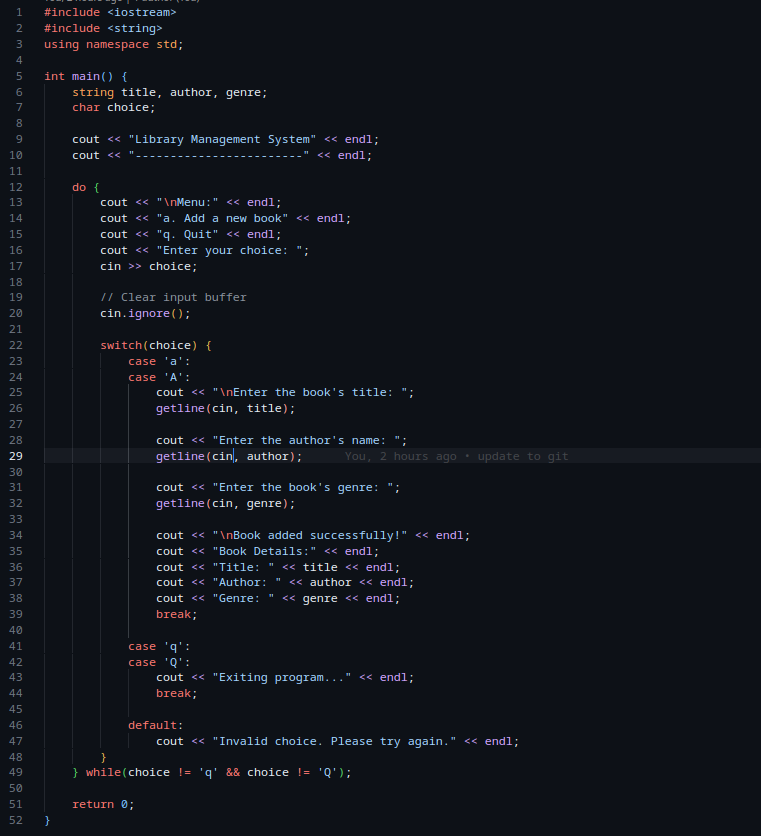
This document presents the development journey of a menu-driven Library Management System created using C++. The project was implemented in incremental stages, with each phase introducing new features and complexity. The system allows users to add, view, and search for books, serving as both a functional tool and an educational project to demonstrate a wide range of programming concepts in C++.

# Basic Implementation:

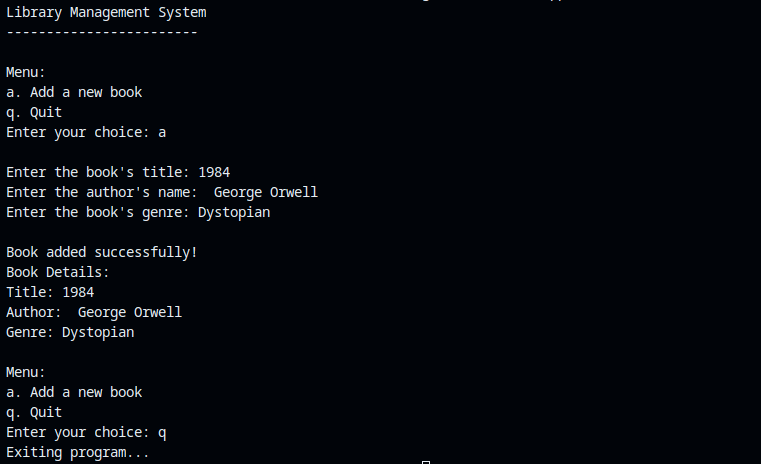
The initial version of the system focused on the core functionality of adding and displaying a single book. This stage used simple string variables to store the title, author, and genre of the book. The user interface was minimal, involving prompts for data entry followed by immediate display of the stored book details.

However, this version was limited to storing only one book at a time. It lacked a menu system, looping capabilities, and any form of persistent storage.

Implementation Code:



The code used a simple do while loop to display the context menu of the system and a switch statement to choose whether to add a book or quit the program

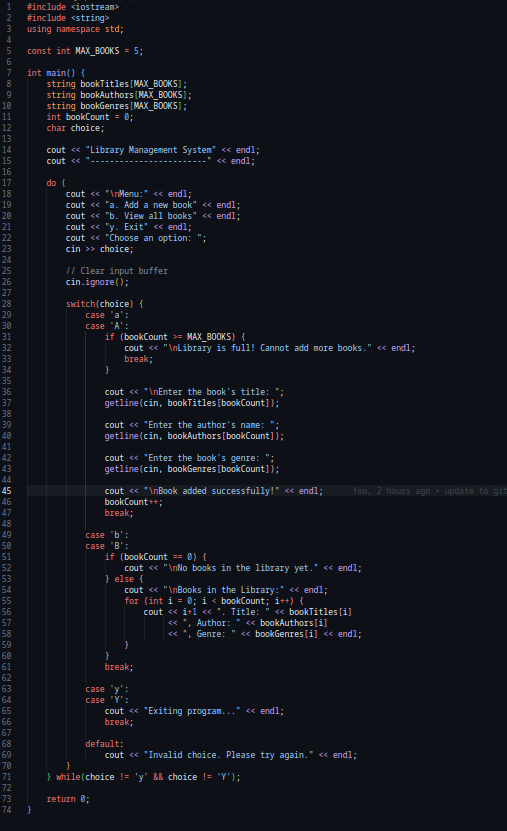
Example Output:  


# Enhanced Implementation

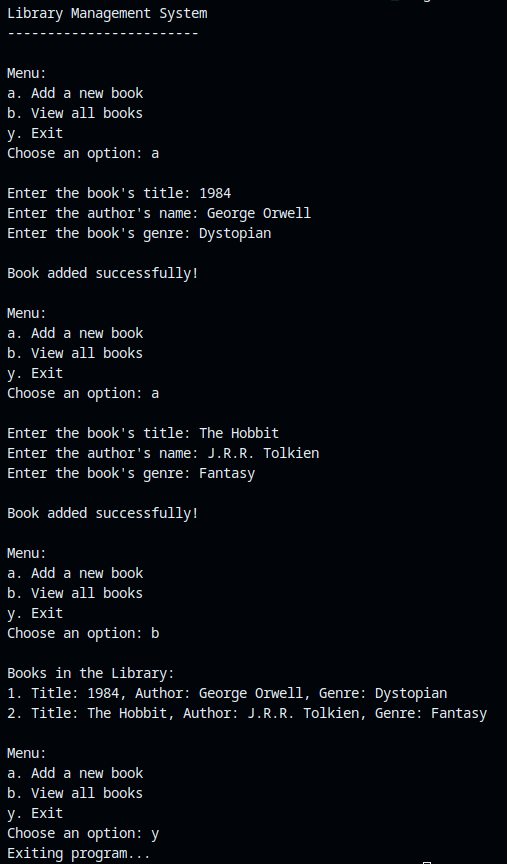
### Building on the basic structure, the second phase introduced the ability to store multiple books, with a maximum capacity of five. This was achieved by implementing parallel arrays for book titles, authors, and genres.

A key advancement in this stage was the introduction of a simple, menu-driven interface. Users could choose from options to add a new book, view all books, or exit the program. The program looped until the user opted to exit, improving usability and functionality significantly over the previous version.

**Implementation Code:**



Example Output:



The program also introduced a new feature that allows retrieval of saved book from the books array

# Search Implementation

The third implementation phase added a search feature, allowing users to search for a book by its title. If the book was found in the system, its details were displayed; otherwise, the user was notified that the book was not found.

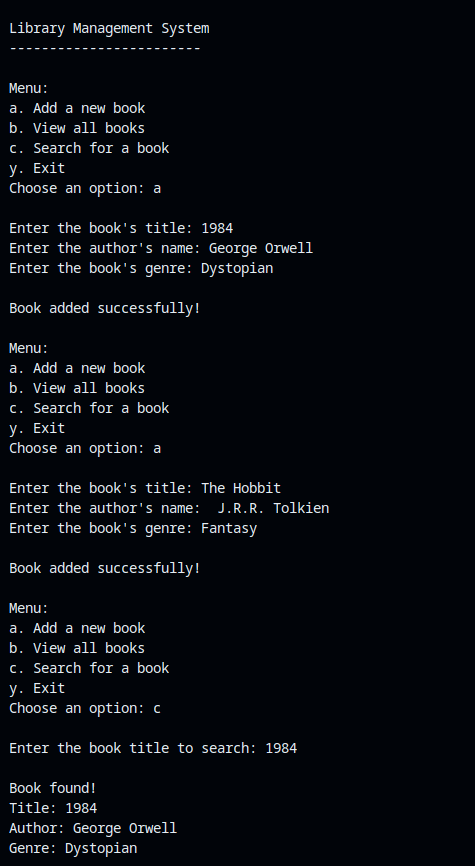
Additionally, basic error handling was introduced. The system could now detect and respond to invalid menu selections, guiding the user to make valid choices. These features enhanced the user experience and made the program more robust and user-friendly.

Implementation Code:



This code introduced the search feature in the form of a case statement in which the search functionality is part of the switch statement (display menu) it also features error handling.

Example Output:

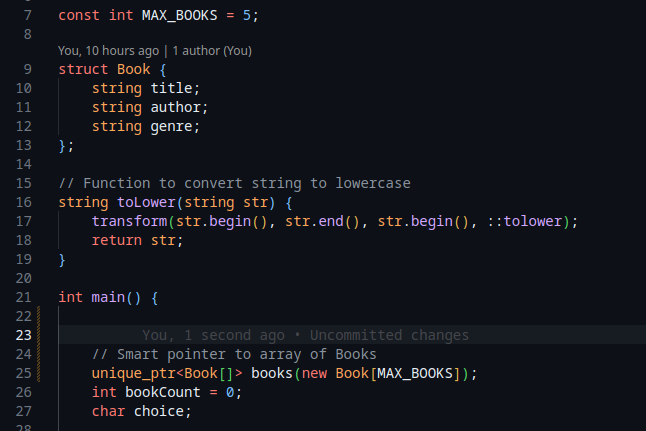


# Pointer Implementation

The final and most sophisticated phase of development focused on optimizing performance and improving code structure. The use of pointers replaced static arrays, enabling dynamic memory management. This allowed for more efficient memory usage and opened the door to scalability.

A struct was introduced to encapsulate the book's details, improving code readability and data organization. For even greater efficiency and safety, smart pointers such as unique\_ptr were optionally introduced to handle automatic memory deallocation.

These enhancements not only refined the technical design but also introduced advanced C++ concepts, making the program more efficient, scalable, and maintainable.

**Pointer and struct implementation:**  


The first implementation was defining a Book data structure with struct . It groups together related information: the **title**, **author**, and **genre** of a book where each book in the library will be represented as a Book object.  
The second implementation (unique\_ptr<Book[]> books(new Book[MAX\_BOOKS]);) introduces a smart pointer that is used to dynamically manage an array of Book objects.

unique\_ptr automatically deletes the memory when it goes out of scope, preventing memory leaks.

The [] tells the compiler that this is managing an **array**, not just a single object.

new Book[MAX\_BOOKS]:  
 Dynamically allocates an array of Book objects of size MAX\_BOOKS.

books(...):  
 Initializes the smart pointer with the address of the dynamically allocated array.

#### Why Use unique\_ptr Here?

* No need to manually call delete[].
* Exception-safe: memory is automatically cleaned up.
* Ensures **exclusive ownership**—only one pointer can own this memory.

This implementation makes the code cleaner, well optimized and dynamic to give room for more improvements and scalability.