[LIBRARY MANAGEMENT SYSTEM]

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

MALINENI MITHESH

(AP22110010195)

Under the Guidance of

(Mrs. KAVITHA RANI)

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[DEC 2023]

Certificate

Date: 16-Nov-22

This is to certify that the work present in this Project entitled "**LIBRARY MANAGEMENT SYSTEM**" has been carried out by MALINENI MITHESH under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. [K. KAVITHA RANI]

Designation- CODING TRAINER

Affiliation -CSC

Acknowledgements:-

- ✧ This implementation of a library management system in C++ draws inspiration from various programming paradigms and incorporates fundamental concepts such as abstraction, inheritance, encapsulation, polymorphism, and the use of standard library functions .

- ✧ I would like to express gratitude to the broader C++ programming community and the contributors whose insights and tutorials have contributed to shaping my understanding of these programming concepts. Additionally, I acknowledge the foundational knowledge gained from various educational resources, textbooks, and online tutorials that have helped in conceptualizing the structure and functionalities of this code.

Table of Contents

Abstract

This program implements a simple library management system that allows users to perform various operations such as adding books, searching for books, updating book quantities, deleting books, and searching for books based on author, genre, and price range. It utilizes object-oriented programming concepts such as encapsulation, inheritance, polymorphism, and abstraction.

Key Features:

Abstraction and Inheritance:

The program uses an abstract base class Book with a pure virtual function displayInfo() to provide a common interface for displaying book information.

The ConcreteBook class inherits from the abstract Book class, demonstrating inheritance and polymorphism.

Encapsulation:

- ❖ The ConcreteBook class encapsulates book details such as ISBN, title, author, quantity, price, and genre using private member variables.
- ❖ Getter and setter methods are used to access and modify private member variables, ensuring controlled access to the internal state of the ConcreteBook objects.

Polymorphism:

- ❖ The displayInfo() function is declared in the base class Book and overridden in the derived class ConcreteBook. This showcases polymorphism, allowing the program to display book information differently based on the specific type of book.

Object-Oriented Design:

- ❖ The program follows object-oriented design principles by creating classes (Book, ConcreteBook, and BookCollection) to represent real-world entities and their interactions.
- ❖ The BookCollection class manages a collection of books and provides functions for adding, searching, updating, and deleting books.

User Interaction:

- ❖ The program interacts with the user through a menu-driven system, allowing them to choose operations to perform on the book collection.

❖ User input is validated, and appropriate actions are taken based on the selected menu option.

Algorithms and Standard Library:

❖ The program uses standard algorithms from the C++ Standard Library, such as find_if and remove_if, to search for and manipulate books within the collection.

Exception Handling:

❖ The program incorporates basic error handling to inform the user if a book is not found during search or if an invalid choice is made in the menu.

Flexibility and Extensibility:

❖ The program is designed to be flexible and extensible. Additional functionality or features can be added by extending existing classes or introducing new classes.

Memory Management:

❖ The program dynamically allocates memory for objects using new and appropriately releases memory using delete to avoid memory leaks.

Readability and Code Organization:

❖ The code is organized into well-defined classes and functions, promoting readability and maintainability. Proper comments are included to explain the purpose of each section.

Abbreviations

ISBN: International Standard Book Number.

Cin and Cout: Input and output streams in C++.

STL: Standard Template Library.

Vector: A dynamic array in C++ from the STL.

Abstraction: The concept of hiding complex implementation details and showing only the essential features of an object.

Destructor: A special member function that is called when an object goes out of scope or is explicitly deleted.

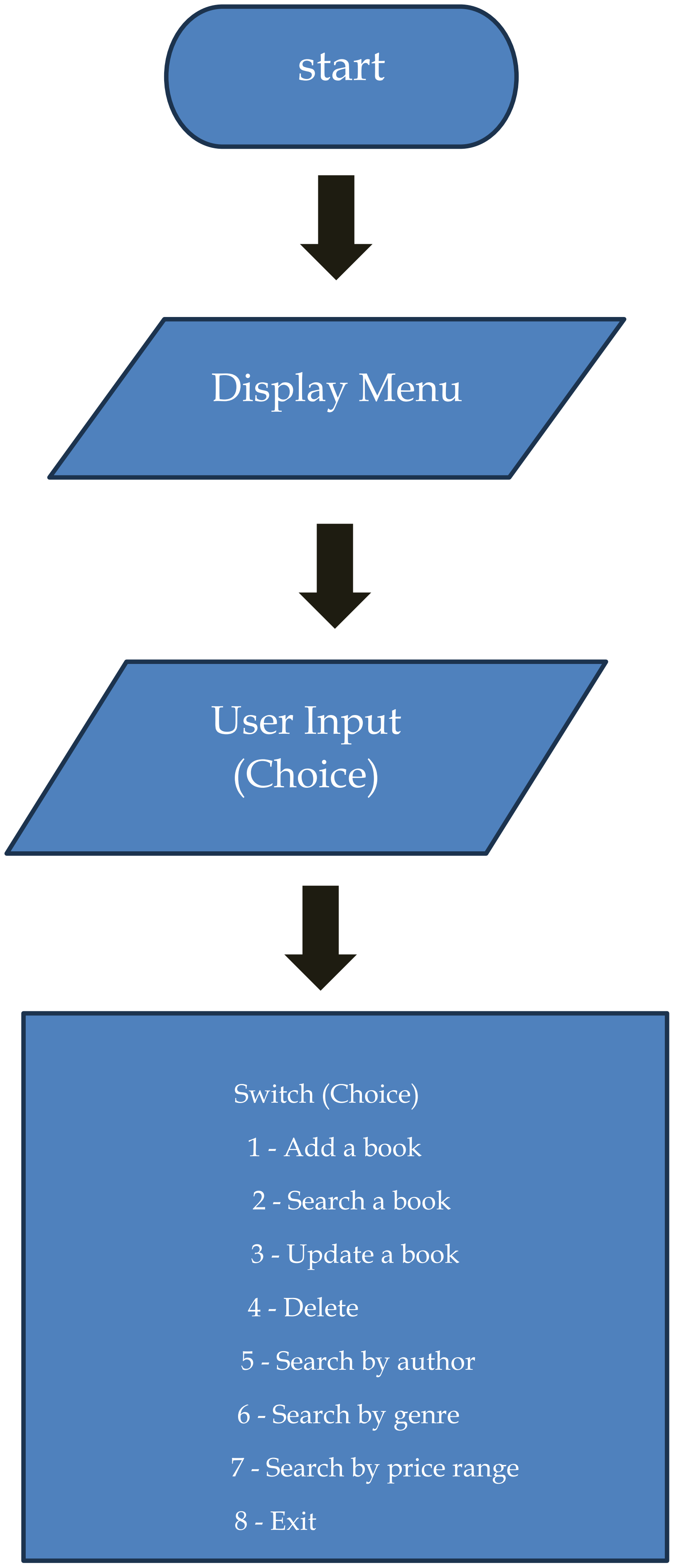
Switch Statement: A control statement used for multi-way branching.

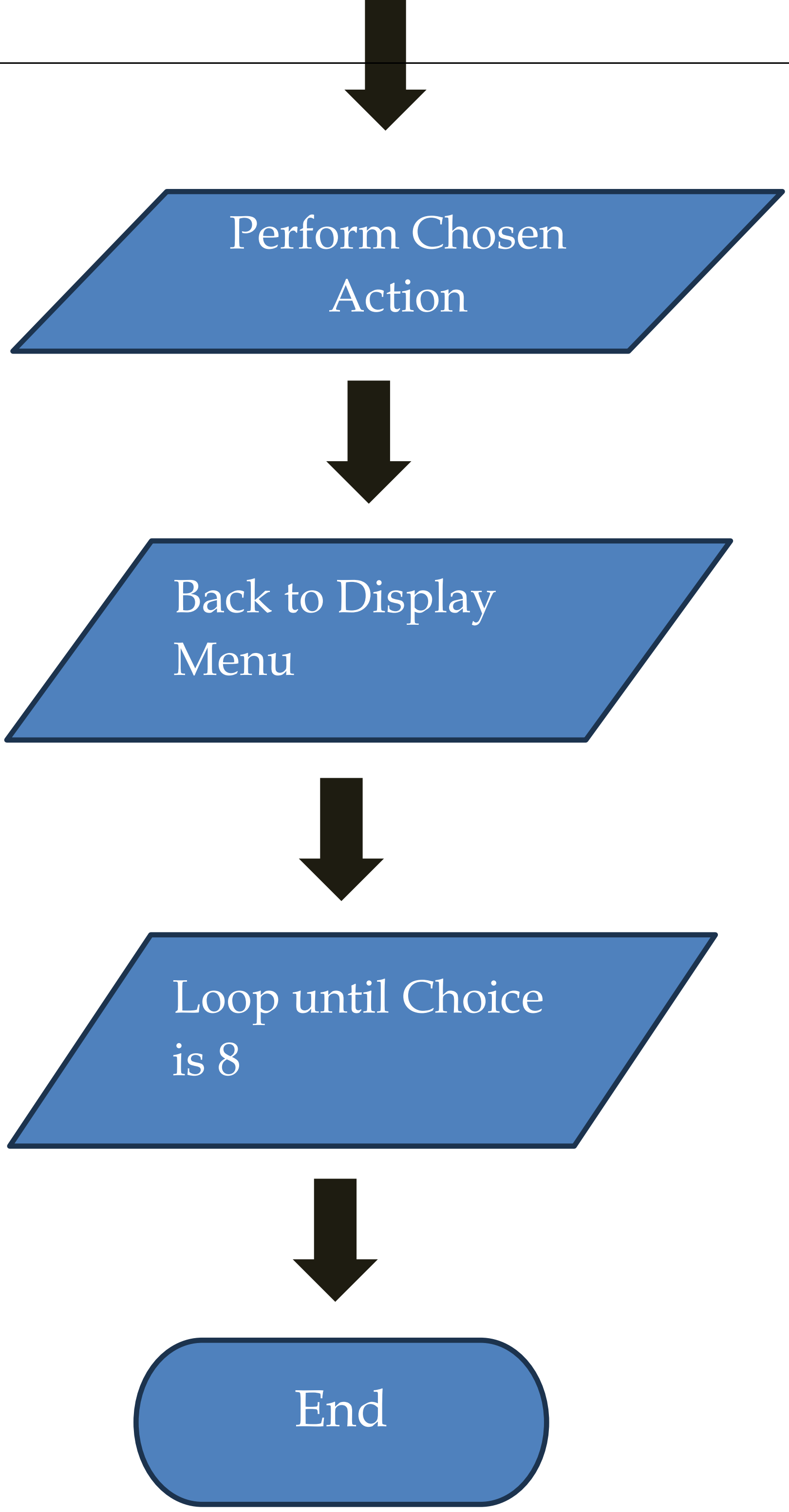
List of Tables

#include<iostream>	Standard library for input/output operations, includes "cin" and "cout," facilitates stream handling.
#include<vector>	The vector header file, <vector>, is part of the C++ Standard Template Library (STL) and provides a class template for dynamic arrays, also known as vectors. Vectors are a convenient and efficient way to store and manage collections of data. They automatically grow or shrink as needed, and they provide a variety of methods for accessing and manipulating their elements.
#include<algorithm>	The algorithm header file, <algorithm>, is part of the C++ Standard Template Library (STL) and provides a variety of generic algorithms that operate on ranges of elements. These algorithms are designed to be efficient and easy to use, and they can be applied to a wide variety of data structures,

	including vectors, arrays, and lists.
Classes & Object	Class - Blueprint for objects, encapsulate data and functions,support inheritance and polymorphism for abstraction. Object – Instance of a class, encapsulates data and methods, promotes modularity and reusability.

List of figures





List of Equations

- ISBN = isbn: This equation assigns the value of the isbn variable to the ISBN member variable of the ConcreteBook object.

- title = t: This equation assigns the value of the t variable to the title member variable of the ConcreteBook object.

- author = a: This equation assigns the value of the a variable to the author member variable of the ConcreteBook object.

- quantity = q: This equation assigns the value of the q variable to the quantity member variable of the ConcreteBook object.

- price = p: This equation assigns the value of the p variable to the price member variable of the ConcreteBook object.

- genre = g: This equation assigns the value of the g variable to the genre member variable of the ConcreteBook object.

- books.push_back(book): This equation adds the book object to the end of the books vector.

- it = find_if(books.begin(), books.end(), id: const ConcreteBook& book { return book.getISBN() == id; }): This equation uses the find_if algorithm to find the first element in the books vector where the ISBN member variable is equal to the id variable.

- books.erase(it, books.end()): This equation removes the element from the books vector at the position of the iterator it.

- found = true: This equation sets the found variable to true, which indicates that a book has been found.

1.Introduction

The provided C++ code presents a simple yet effective library management system, utilizing fundamental principles of object-oriented programming (OOP) such as abstraction, encapsulation, inheritance, and polymorphism. The system is designed to manage a collection of books, allowing users to perform various operations such as adding, searching, updating, and deleting books.

At its core, the code defines an abstract base class Book, from which a concrete book class, ConcreteBook, is derived. This hierarchy exemplifies the concept of inheritance, providing a blueprint for creating specific book objects while ensuring a standardized interface for displaying book information. The virtual functions in the base class enable polymorphic behavior, allowing for the implementation of specialized functionality in the derived class.

The BookCollection class serves as a container for storing instances of ConcreteBook. This class encapsulates the vector of books, demonstrating the encapsulation principle by hiding the internal implementation details of book storage. Users can interact with the library system through a console-based menu, making choices ranging from adding new books to searching for them based on ISBN, author, genre, or price range. The system employs standard C++ features, including input/output streams, vectors, and algorithms from the Standard Template Library (STL).

The code's modular design promotes maintainability and extensibility. Users can easily expand the system by adding new functionalities or incorporating additional attributes to the book objects. Overall, this library management system provides a foundational example of OOP principles in action, fostering code reusability, flexibility, and organization.

2. Methodology

Object-Oriented Design:

The system employs object-oriented principles, utilizing classes and inheritance. The Book class acts as an abstract base class, promoting abstraction and providing a blueprint for derived classes.

Concrete Book Representation:

The ConcreteBook class represents a specific book, inheriting from the Book class. It encapsulates essential details such as ISBN, title, author, quantity, price, and genre.

Book Collection Management:

The BookCollection class encapsulates a vector of ConcreteBook objects, providing functions to add, search, update, and delete books. This class demonstrates encapsulation by hiding the internal implementation details of book storage.

Polymorphic Display:

The displayInfo function in the ConcreteBook class is implemented polymorphically. It allows for the display of book information with or without showing the quantity based on user preference.

User Interaction through Console Menu:

The main function provides a console-based menu system allowing users to interact with the library management system. The menu offers options to add a book, search for a book by ISBN, update book quantity, delete a book, search by author, search by genre, search by price range, and exit the system.

Input Validation:

The code incorporates basic input validation, ensuring that user inputs are of the correct data type. For instance, when entering ISBN, quantity, or price, the program checks for valid integer or float inputs.

Algorithms from the Standard Template Library (STL):

The code utilizes algorithms from the C++ Standard Template Library (STL), such as find_if to search for a book by ISBN and remove_if to delete a book by ISBN. This demonstrates the use of existing, well-tested functionality to enhance code readability and maintainability.

Error Handling:

The system includes error-handling mechanisms, providing user-friendly messages when a book is not found, or an invalid choice is entered in the menu.

Modularity and Extensibility:

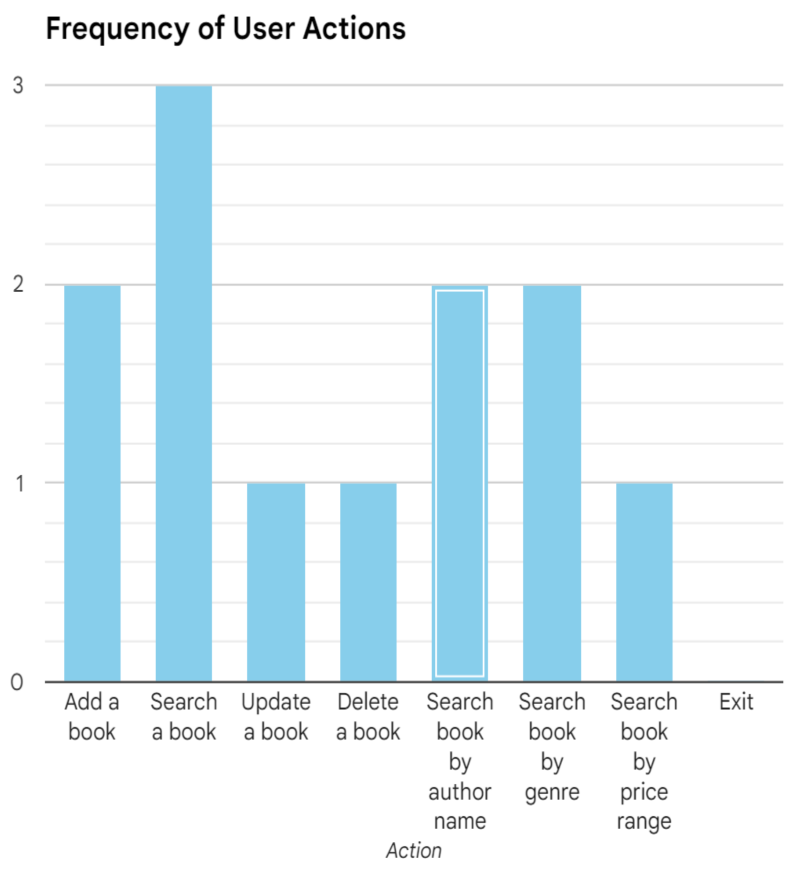
The code is designed in a modular fashion, allowing for easy extension and maintenance. Users can add new functionalities or modify existing ones without affecting the entire codebase.

Graceful Exit:

The program allows users to gracefully exit the system by choosing the appropriate menu option, ensuring a smooth user experience.

This methodology emphasizes principles of object-oriented programming, encapsulation, and user interaction, providing a foundation for building and extending a library management system.

3)Discussion



This graph consists-

1,add book

2,search a book

3,update a book

4,delete a book.

5,search book by author name

6,search book by genre

7,search a book by price rage

8,exit

Concluding Remarks

Abstraction and Inheritance:

The use of an abstract base class Book and a derived class ConcreteBook demonstrates the concept of abstraction and inheritance. This allows you to represent both the abstract idea of a book and specific instances of concrete books.

Encapsulation:

The encapsulation of book details within the ConcreteBook class and the vector of books within the BookCollection class helps organize and hide the internal implementation details.

Polymorphism:

Polymorphism is demonstrated through the use of virtual functions, such as displayInfo(), which is overridden in the derived class. This allows you to treat objects of derived classes uniformly through a pointer to the base class.

User Interface:

The user interface with a menu-driven system is clear and easy to understand. Users can perform various operations such as adding, searching, updating, and deleting books.

Input Validation:

Consider adding input validation to ensure that users enter valid data. For example, you might check if the ISBN is a positive integer, if the quantity is non-negative, and if the price is a positive float.

Error Handling:

Implement more robust error handling, especially in cases where user inputs may not be as expected. For instance, when the user enters a non-existent ISBN for searching, updating, or deleting a book.

Modularization:

Consider breaking down the program into smaller functions to improve readability, maintainability, and reusability. Each function should ideally perform a specific task, making the code more modular.

Persistence:

Currently, the data is stored in memory during program execution. If you want to persistently store data between program runs, you might consider file I/O to save and load the book collection data from a file.

Comments:

The code is generally well-commented, which is good for understanding the logic. Consider adding comments to describe the purpose and usage of each function, especially if the project grows.

Testing:

Ensure that the program is thoroughly tested with different scenarios to identify and address potential issues

Future work:

Future Work: Enhancements and Features for Library Management System

User Authentication and Authorization:

Implement a secure user authentication system to ensure that only authorized users can perform certain operations, such as adding, updating, or deleting books. This will enhance the system's security and control.

Database Integration:

Integrate a database system to persistently store book information. This will allow the library management system to retain data across sessions, improving data management and providing scalability.

Graphical User Interface (GUI):

Develop a graphical user interface for the library management system to enhance the user experience. A GUI can make the system more intuitive and visually appealing, providing users with an interactive platform.

Book Borrowing and Returning:

Extend the functionality to include features for book borrowing and returning. Implement a mechanism to track book availability, due dates, and user borrowing history, transforming the system into a more comprehensive library management solution.

Fine Calculation System:

Introduce a fine calculation system for overdue books. Implement a mechanism to calculate fines based on the duration a book is overdue, providing a fair and automated approach to managing late returns.

Notification System:

Develop a notification system to alert users about upcoming due dates, overdue books, or important library announcements. This can be implemented through email, SMS, or in-app notifications.

Book Reviews and Ratings:

Allow users to submit reviews and ratings for books they have read. Implementing a rating and review system can help other users make informed decisions about which books to borrow.

Multiple Copies and Editions:

Extend support for multiple copies of the same book and different editions. This feature can be useful for libraries that stock multiple copies of popular books or have different editions available.

Advanced Search Options:

Enhance the search functionality with advanced options, such as searching by publication year, language, or a combination of criteria. This will provide users with more flexible and powerful search capabilities.

Data Analytics and Reporting:

Implement data analytics features to generate reports on book popularity, user preferences, and other relevant metrics. This data can be valuable for library administrators in making informed decisions.

Cross-Platform Compatibility:

Ensure the library management system is compatible with multiple platforms, including desktops, web browsers, and mobile devices. This will increase accessibility for users.

Internationalization and Localization:

Implement internationalization (i18n) and localization (l10n) features to support multiple languages and regional preferences. This can make the system more inclusive and accessible to a broader user base.

References:

1. Thinking in C++, Bruce, Eckel, Pearson, Second edition, Volume 1, 2002.

2. Object-oriented programming in C++, Robert Lafore, Course Sams Publishing, Fourth edition,

2001.

3. Lischner, Ray. STL Pocket Reference: Containers, Iterators, and Algorithms. " O'Reilly Media,

Inc.", 2003.