FINAL PROJECT REPORT GROUP 1

TOPIC: Book Recommendation System

1.0 Introduction

The Book Recommendation System is a command-line tool (CLI) developed in C++ designed to help users discover new books based on their interests. The system allows users to input preferences such as genre or author and receive recommendations accordingly. With a simple CLI interface, the Book Recommendation System provides a seamless experience for users to explore a wide range of books tailored to their tastes.

2.0 Objectives and Purposes

The primary objectives and purposes of the Book Recommendation System are:

- **Assist Users:** Help users discover new books that match their preferences.
- **Personalization:** Provide personalised book recommendations tailored to each user's specific criteria.
- **Engagement:** Enhance user engagement with books by suggesting titles they may enjoy.
- **Encouragement:** Foster a love for reading by introducing users to new genres and authors they may not have explored before.

3.0 System Design and Architecture

The system design follows an object-oriented approach with the following key components:

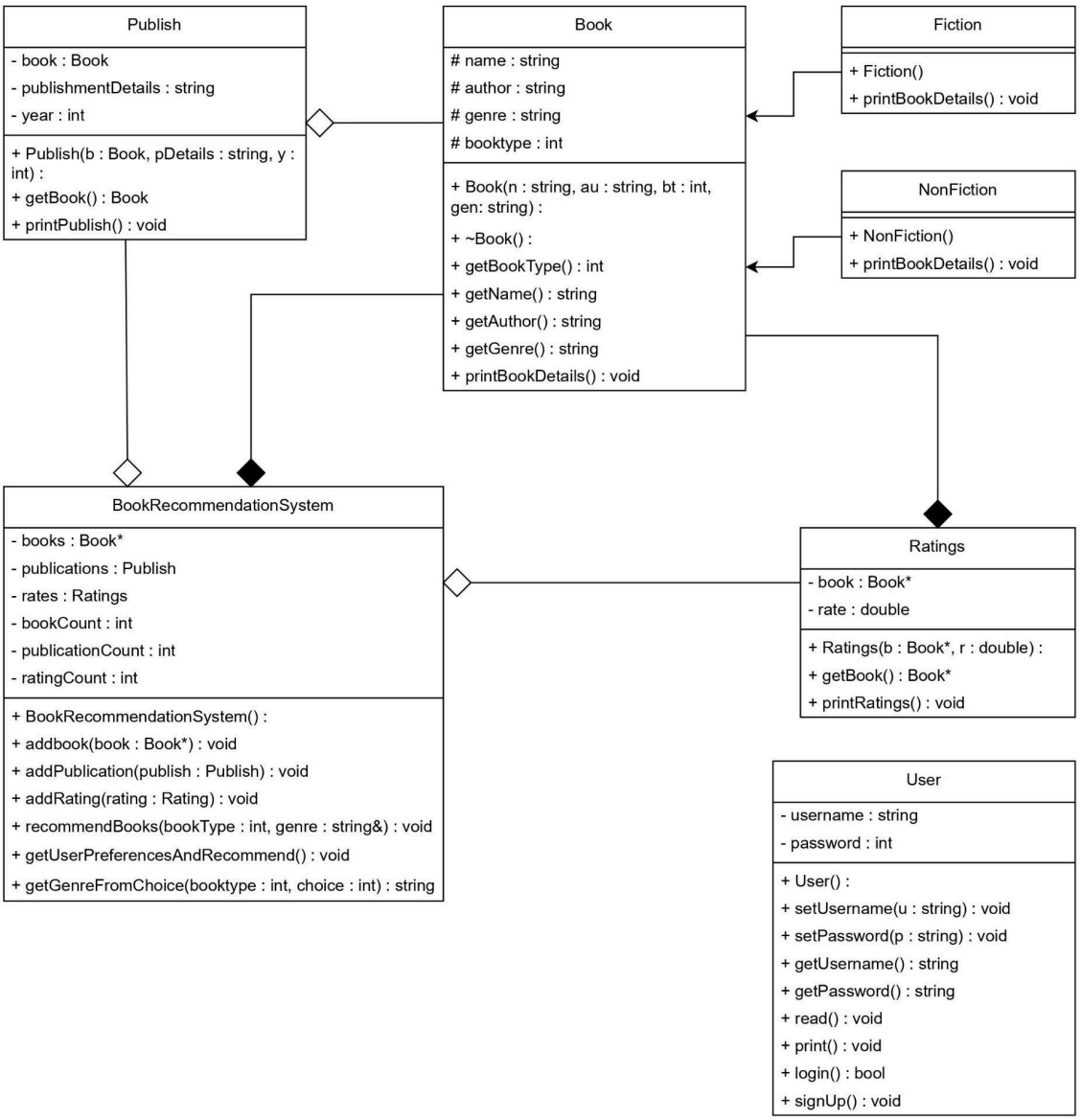
- **User Management:** Handles user registration and authentication.
- **Book Management:** Manages book details, including their types (Fiction, NonFiction), authors, genres, and ratings.
- **Recommendation Engine:** Analyses user preferences to recommend books.

Key Classes

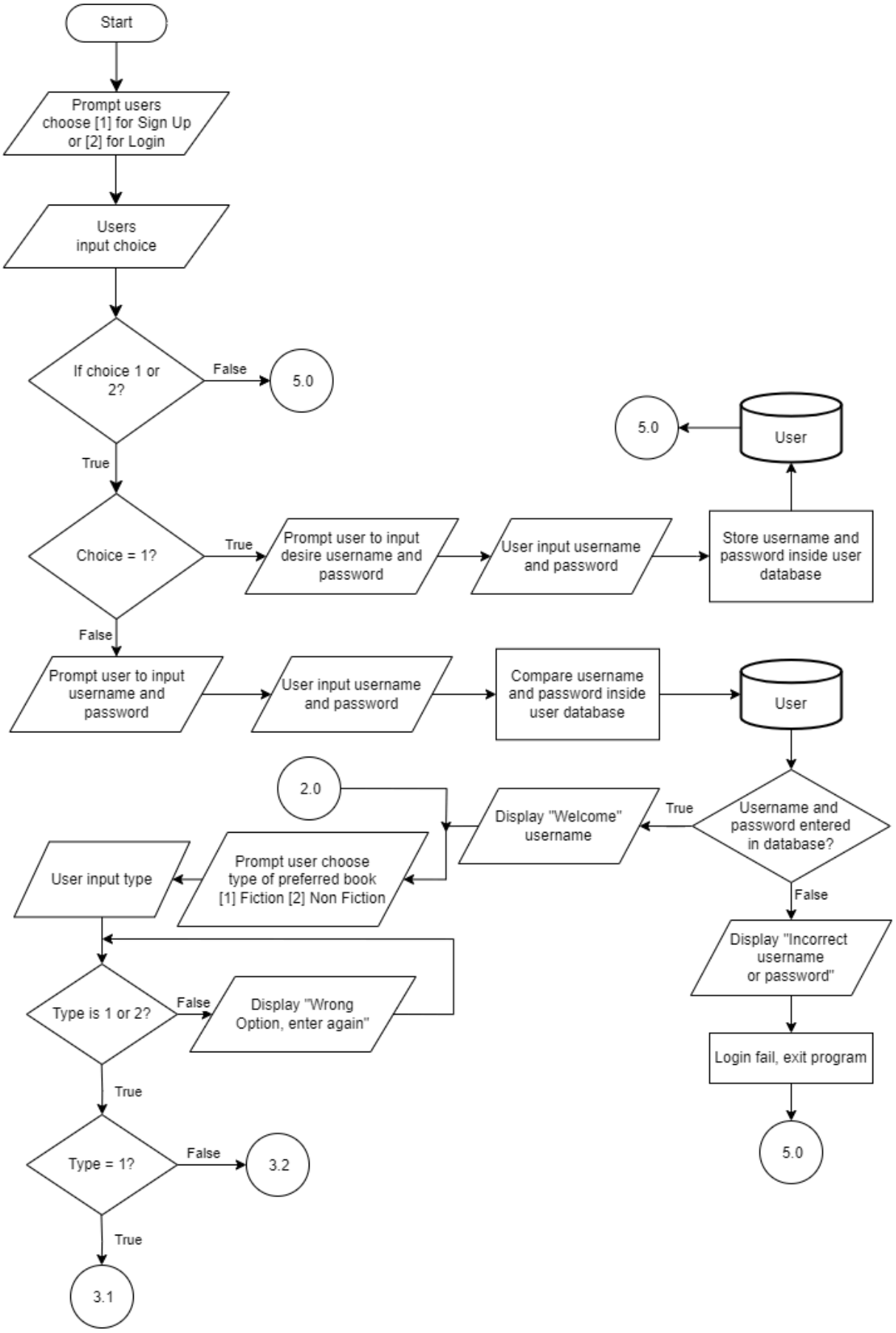
- **User Class:** Manages user data, including username, password, and preferences.
- **Book Class:** Represents books, with derived classes for different types of books (e.g., Fiction, NonFiction).
- **Publish Class:** Manages publication details associated with each book.
- **Ratings Class:** Handles ratings associated with books.
- **BookRecommendationSystem Class:** Integrates all functionalities, managing books, publications, ratings, and providing book recommendations.

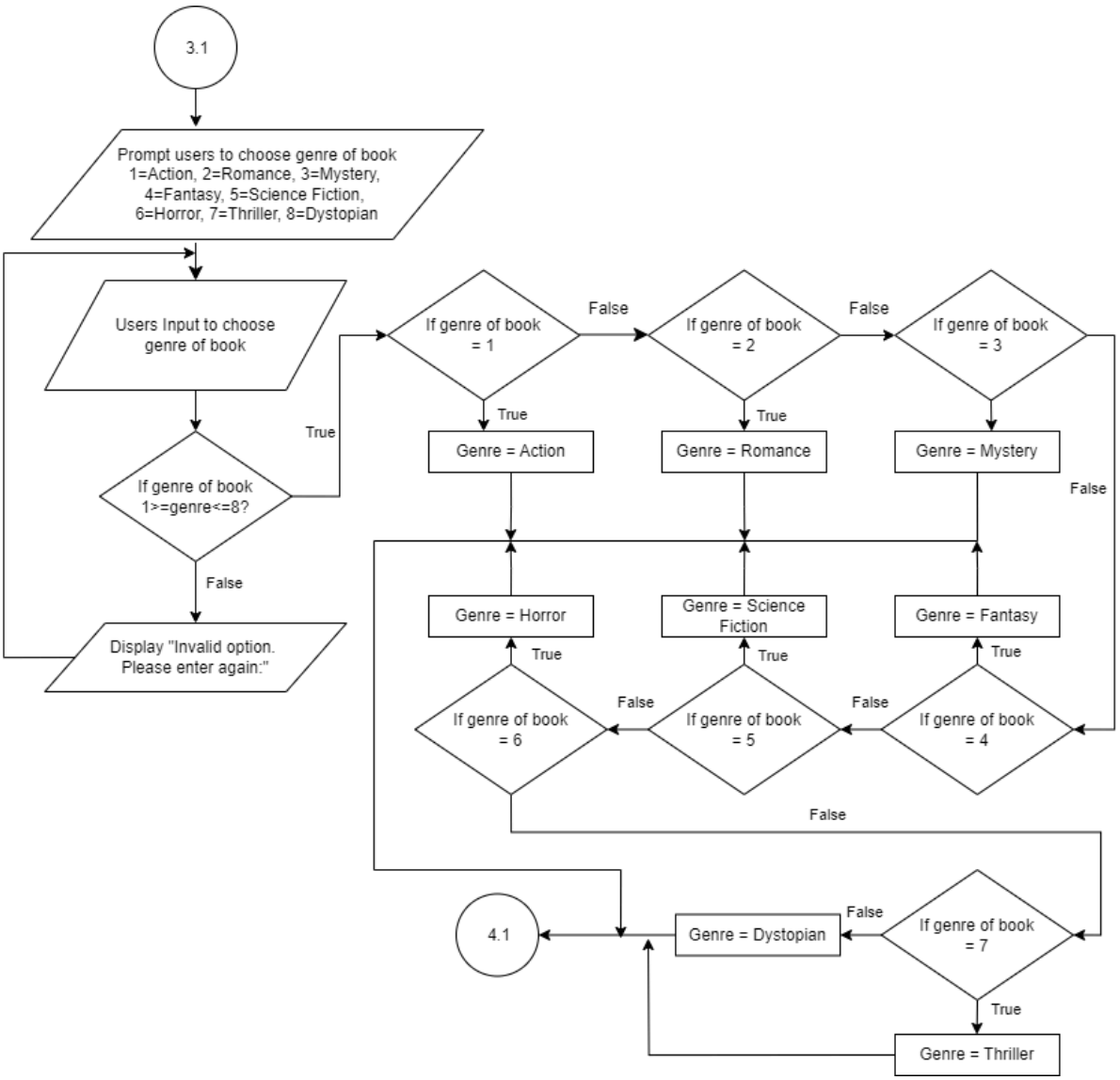
4.0 Flowchart and UML Class Diagram

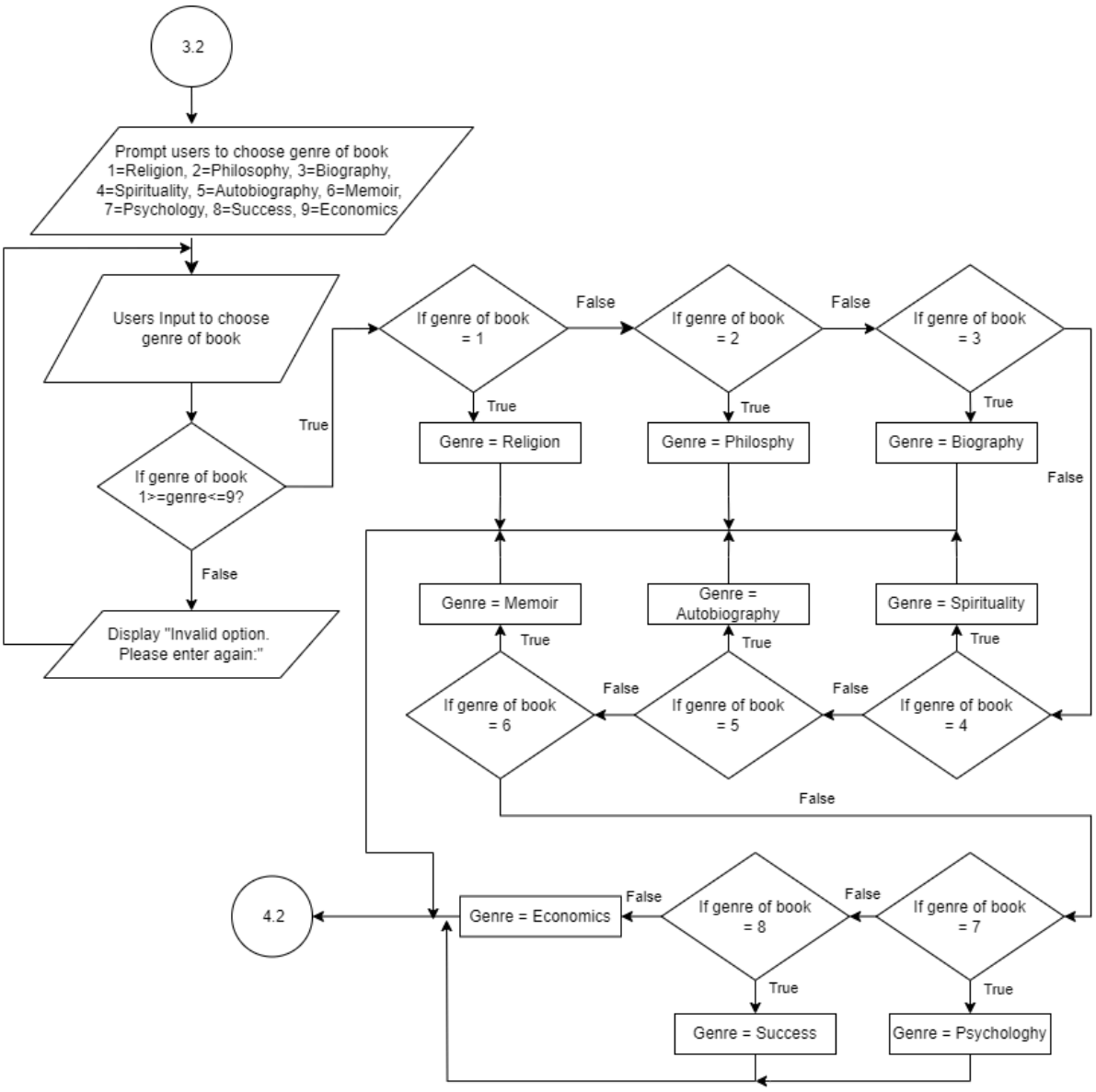
UML Class Diagram

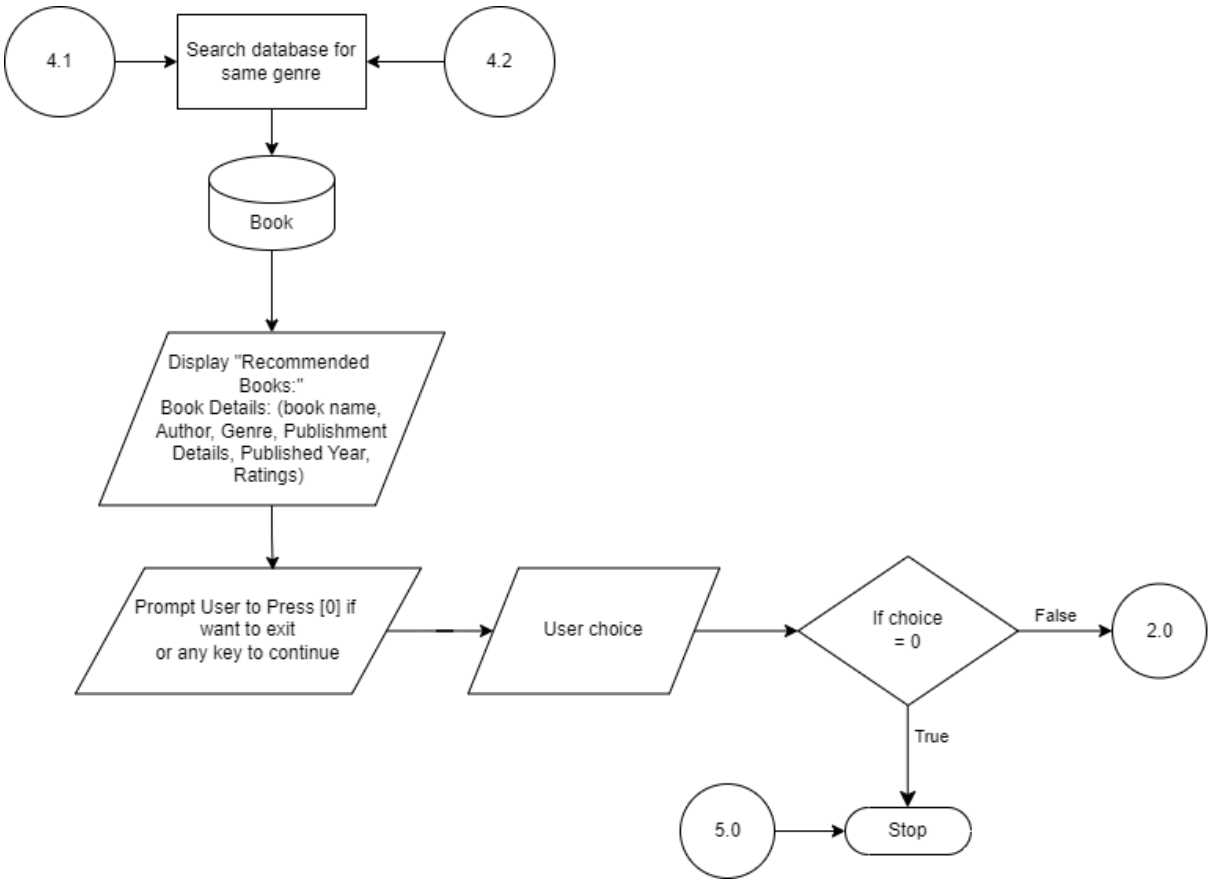


Flowchart


Start
Prompt users choose [1] for Sign Up or [2] for Login
Users input choice
If choice 1 or 2?
False
5.0
True
Choice = 1?
True
Prompt user to input desire username and password
User input username and password
Store username and password inside user database
User
5.0
False
Prompt user to input username and password
User input username and password
Compare username and password inside user database
User
Username and password entered in database?
True
Display "Welcome" username
2.0
Prompt user choose type of preferred book [1] Fiction [2] Non Fiction
User input type
False
Display "Incorrect username or password"
Login fail, exit program
5.0
Type is 1 or 2?
False
Display "Wrong Option, enter again"
True
Type = 1?
False
3.2
True
3.1


3.1
Prompt users to choose genre of book
1=Action, 2=Romance, 3=Mystery,
4=Fantasy, 5=Science Fiction,
6=Horror, 7=Thriller, 8=Dystopian
Users Input to choose genre of book
If genre of book 1>=genre<=8?
True
False
Display "Invalid option. Please enter again:"
If genre of book = 1
False
If genre of book = 2
False
If genre of book = 3
True
True
True
Genre = Action
Genre = Romance
Genre = Mystery
False
Genre = Horror
Genre = Science Fiction
Genre = Fantasy
True
True
True
If genre of book = 6
False
If genre of book = 5
False
If genre of book = 4
False
4.1
Genre = Dystopian
False
If genre of book = 7
True
Genre = Thriller


3.2
Prompt users to choose genre of book
1=Religion, 2=Philosophy, 3=Biography,
4=Spirituality, 5=Autobiography, 6=Memoir,
7=Psychology, 8=Success, 9=Economics
Users Input to choose genre of book
If genre of book 1>=genre<=9?
True
False
Display "Invalid option. Please enter again:"
If genre of book = 1
False
If genre of book = 2
False
If genre of book = 3
True
Genre = Religion
Genre = Philosphy
Genre = Biography
False
Genre = Memoir
Genre = Autobiography
Genre = Spirituality
If genre of book = 6
If genre of book = 5
If genre of book = 4
False
4.2
Genre = Economics
If genre of book = 8
If genre of book = 7
Genre = Success
Genre = Psychologhy


4.1
Search database for same genre
4.2
Book
Display "Recommended Books:"
Book Details: (book name, Author, Genre, Publishment Details, Published Year, Ratings)
Prompt User to Press [0] if want to exit
or any key to continue
User choice
If choice = 0
False
2.0
True
5.0
Stop

5.0 Class Details

User Class

The **User** class is responsible for managing user information and authentication. It includes methods for setting and getting user details, reading user data from input, printing user details, and handling login and sign-up processes. User data is stored in files to maintain persistence.

Book Class

The **Book** class serves as the base class for all book objects. It includes attributes for book name, author, genre, and book type, along with methods to retrieve these details. Derived classes **Fiction** and **NonFiction** extend the Book class to handle specific types of books. Methods in these classes allow for printing detailed information about each book.

Fiction Class

Derived from Book, represents fiction books specifically. It inherits attributes from Book and adds a specialisation for fiction books. It overrides printBookDetails() to display details specific to fiction books.

NonFiction Class

Also derived from Book, represents non-fiction books. It inherits attributes from Book and specialises for non-fiction books. Similar to Fiction, it overrides printBookDetails() to display details specific to non-fiction books.

Publish Class

The **Publish** class manages publication details, including the publisher's name and the year of publication. It associates these details with a book and provides methods to retrieve and print publication details.

Ratings Class

The **Ratings** class handles book ratings. It associates a rating value with a book and includes methods to retrieve the book associated with a rating and to print rating details.

BookRecommendationSystem Class

The **BookRecommendationSystem** class is the core class that integrates all functionalities. It maintains arrays of books, publications, and ratings, along with counters to track the number of entries. The class provides methods to add books, publications, and ratings to the system, recommend books based on user preferences, and retrieve genres based on user choices.

6.0 Implementation of Concept

1.**Encapsulation**

Explanation: Encapsulation in object-oriented programming bundles data (attributes) and methods (functions) that operate on the data into a single unit (class). It hides the internal state of objects from direct access by clients and allows controlled access via public methods.

Justification: The `Book` class encapsulates attributes such as `name`, `author`, `genre`, and `booktype`, providing public methods like `getName()`, `getAuthor()`, `getGenre()` to access these attributes. This prevents direct modification of internal data and ensures data integrity

```
class Book { // Parent class
protected:
    string name, author, genre;
    int booktype; // 1 for Fiction, 2 for Non-Fiction
public:
    Book() : name(""), author(""), genre(""), booktype(0) {} // Default constructor
    Book(string n, string au, int bt, string gen) : name(n), author(au), booktype(bt), genre(gen) {}
```

2.**Aggregation**

Explanation: Aggregation represents a "has-a" relationship where one class (the container or aggregator) contains references to objects of another class (the aggregate). The aggregated objects can exist independently of the container.

Justification: The `Ratings` class aggregates a `Book` object using a pointer (`Book* book`). It maintains a relationship where `Ratings` can refer to a `Book` object without owning it, allowing flexibility in object relationships and data management

```
class Ratings {
private:
    Book* book; // aggregation
    double rate;
public:
    Ratings() : book(NULL), rate(0.0) {}
    Ratings(Book* b, double r) : book(b), rate(r) {}

    Book* getBook() const {
        return book;
    }
```

3.**Composition**

Explanation: Composition is a stronger form of aggregation where the lifetime of the contained object is managed by the container. If the container object is destroyed, all contained objects are also destroyed.

Justification: In your design, the `Publish` class demonstrates composition with `Book` as a member object (`Book book`). The `Publish` object owns its associated `Book` object, ensuring that a `Publish` instance cannot exist without a corresponding `Book` instance

```
class Publish {
private:
    Book book; // composition
    string publishmentDetails;
    int year;
public:
    Publish() : publishmentDetails(""), year(0) {}
    Publish(Book b, string pDetails, int y) : book(b), publishmentDetails(pDetails), year(y) {}

    Book getBook() const {
        return book;
    }
```

4.**Inheritance**

Explanation: Inheritance allows one class (derived class) to inherit properties and behaviours from another class (base or parent class). It promotes code reuse and supports the "is-a" relationship.

Justification: Both `Fiction` and `NonFiction` classes inherit from the `Book` class. They inherit attributes and methods such as `printBookDetails()` from `Book`, enabling specialisation (e.g., specific genre handling) while leveraging common functionality defined in `Book`

```
class Fiction : public Book {
public:
    Fiction() {}
    Fiction(string n, string au, string gen) : Book(n, au, 1, gen) {}

    void printBookDetails() const {
        cout << "Fiction Book Details:" << endl;
        Book::printBookDetails();
    }
};
```

```
class NonFiction : public Book {
public:
    NonFiction() {}
    NonFiction(string n, string au, string gen) : Book(n, au, 2, gen) {}

    void printBookDetails() const {
        cout << "Non-Fiction Book Details:" << endl;
        Book::printBookDetails();
    }
};
```

5.**Polymorphism**

Explanation: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables methods to be defined in multiple derived classes with the same name but different implementations.

Justification: The `printBookDetails()` method in the `Book` class is declared as `virtual`, allowing `Fiction` and `NonFiction` classes to override this method with their specific implementations (`void printBookDetails() const`). This supports dynamic method binding at runtime based on the actual object type.

```
31      virtual void printBookDetails() const {
32          cout << "Book Name: " << name << endl;
33          cout << "Author: " << author << endl;
34          cout << "Genre: " << genre << endl;
35      }
```

```
void printBookDetails() const {
    cout << "Fiction Book Details:" << endl;
    Book::printBookDetails();
}
```

```
void printBookDetails() const {
    cout << "Non-Fiction Book Details:" << endl;
    Book::printBookDetails();
}
```

6.**Array of Object**

Explanation: An array of objects refers to storing multiple instances of a class in a contiguous block of memory, allowing efficient access and manipulation of objects as a group.

Justification: In `BookRecommendationSystem`, arrays (`adbooks[]`, `addpublications[]`, `addrating[]`) store multiple instances of `Book`, `Publish`, and `Ratings` objects respectively. This facilitates managing and accessing collections of related objects (e.g., books, publication details, ratings) within the system.

```
class BookRecommendationSystem {
private:
    Book* books[MAX_BOOKS];
    Publish publications[MAX_PUBLICATIONS];
    Ratings rates[MAX_RATINGS];
    int bookCount, publicationCount, ratingCount;

public:
    BookRecommendationSystem() : bookCount(0), publicationCount(0), ratingCount(0) {}

    void addBook(Book* book) {
        if (bookCount < MAX_BOOKS) {
            books[bookCount++] = book;
        }
    }

    void addPublication(Publish publish) {
        if (publicationCount < MAX_PUBLICATIONS) {
            publications[publicationCount++] = publish;
        }
    }

    void addRating(Ratings rating) {
        if (ratingCount < MAX_RATINGS) {
            rates[ratingCount++] = rating;
        }
    }
```

In `main()`, we use `BookRecommendationSystem` to populate these arrays

```
// Adding Fiction Books
bookSystem.addBook(new Fiction("The Great Gatsby", "F. Scott Fitzgerald", "Action"));
bookSystem.addBook(new Fiction("1984", "George Orwell", "Mystery"));
bookSystem.addBook(new Fiction("To Kill a Mockingbird", "Harper Lee", "Romance"));
bookSystem.addBook(new Fiction("Pride and Prejudice", "Jane Austen", "Romance"));
bookSystem.addBook(new Fiction("The Catcher in the Rye", "J.D. Salinger", "Action"));
bookSystem.addBook(new Fiction("The Hobbit", "J.R.R. Tolkien", "Fantasy"));
bookSystem.addBook(new Fiction("Harry Potter and the Philosopher's Stone", "J.K. Rowling", "Fantasy"));
bookSystem.addBook(new Fiction("The Hunger Games", "Suzanne Collins", "Action"));
bookSystem.addBook(new Fiction("Dune", "Frank Herbert", "Science Fiction"));
bookSystem.addBook(new Fiction("Brave New World", "Aldous Huxley", "Science Fiction"));
bookSystem.addBook(new Fiction("The Martian", "Andy Weir", "Science Fiction"));
bookSystem.addBook(new Fiction("The Shining", "Stephen King", "Horror"));
bookSystem.addBook(new Fiction("Gone Girl", "Gillian Flynn", "Thriller"));
bookSystem.addBook(new Fiction("The Girl with the Dragon Tattoo", "Stieg Larsson", "Mystery"));
bookSystem.addBook(new Fiction("The Road", "Cormac McCarthy", "Dystopian"));
```

7.File handling

Explanation: File handling in C++ provides mechanisms to read from and write to files. It allows data to be stored persistently and retrieved as needed by the program.

Justification: The `User` class utilises file handling (`fstream`) in methods like `login()` to read and write user credentials (`username` and `password`) to a file (`users.txt`). This enables user authentication and registration functionalities while persisting user data across program executions.

```
bool login() {
    ifstream userFile("users.txt");
    if (!userFile) {
        cerr << "Unable to open user file 'users.txt'." << endl;
        return false;
    }

    string storedUsername;
    string storedPassword;
    bool loginSuccessful = false;

    while (userFile >> storedUsername >> storedPassword) {
        if (username == storedUsername && password == storedPassword) {
            loginSuccessful = true;
            break;
        }
    }

    userFile.close();
```

7.0 Conclusion

The Book Recommendation System developed using C++ provides a user-friendly way to discover new books based on personal preferences. By leveraging object-oriented programming principles, we have created a robust and flexible system that can be easily extended with new features and functionalities. This project not only enhances user engagement with books but also encourages exploration of new genres and authors, fostering a deeper love for reading.