SECJ1023
PROGRAMING TECHNIQUE II
SECTION 08
LECTURER: MS LIZAWATI BINTI MI YUSUF

Group Project Deliverable 4: Project Finale

Group

Name	**Matric No**
Chuah Hui Wen	A23CS0219
Lim Xin Rou	A23CS0240
Chen Wei Jay Nickolas	A23CS5028

TABLE OF CONTENT

Section A
Project description

Introduction

Nowadays, there are more books available than ever before, which can make it hard for people to find books they will enjoy. Traditional ways of recommending books, like suggestions from friends or bestseller lists, often don't meet the varied tastes of readers. To solve this problem, creating a smart book recommendation system is essential.

The book recommendation system suggests books to the users that they might like based on their preferred genre. The main goal of this project is to design a book recommendation system that can accurately match books to a user's preferences, helping them find new and interesting reads.

This report explains our book recommendation system, including its goals and purposes. It covers the analysis and design process, including flow charts and UML class diagrams, as well as the steps taken to create the system. Additionally, it includes the implementation of the concepts.

By developing a reliable and user-friendly book recommendation system, we aim to make reading more enjoyable and help people discover books they might otherwise miss. This project benefits readers by saving them time and effort, and also supports authors and publishers by increasing the visibility of their books.

Objectives of the system

The main objective of this book recommendation system is to create a user-friendly platform that could provide personalized book recommendations to users based on the users' favorite genres. The system leverages user preferences to suggest books that align with their interests, ensuring a tailored reading experience. This approach helps users to discover the books that match their taste. Enhancing their experience on finding the book they are interested in.

Additionally, the system enables users to explore trending books and various genres. By providing popular books in different genres, users can stay updated with the reading trends. These features ensure that users have access to a diverse range of books, encouraging exploration of new genres.

Moreover, the system also provides functionality for users to create and customize their own book list. This allows them to save their personalized collections of books they are interested in, providing a convenient way to organize and manage their reading choice.

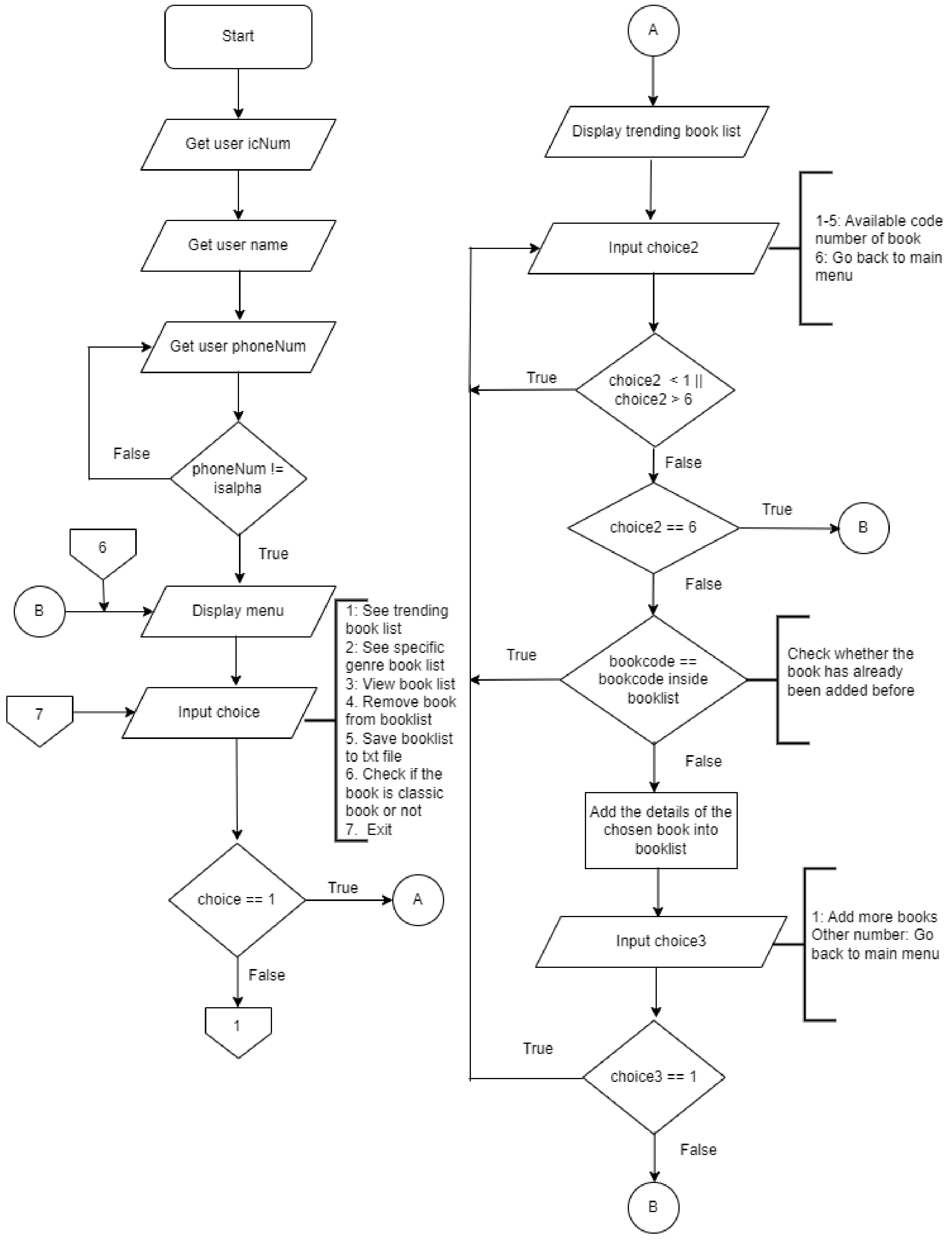
Lastly, this book recommendation system aims to cultivate and spark interest of people to indulge in reading habits by making it easy and enjoyable for users to find and access new books. In a nutshell, the goal is to contribute to a culture of reading by making it easier to access different books.

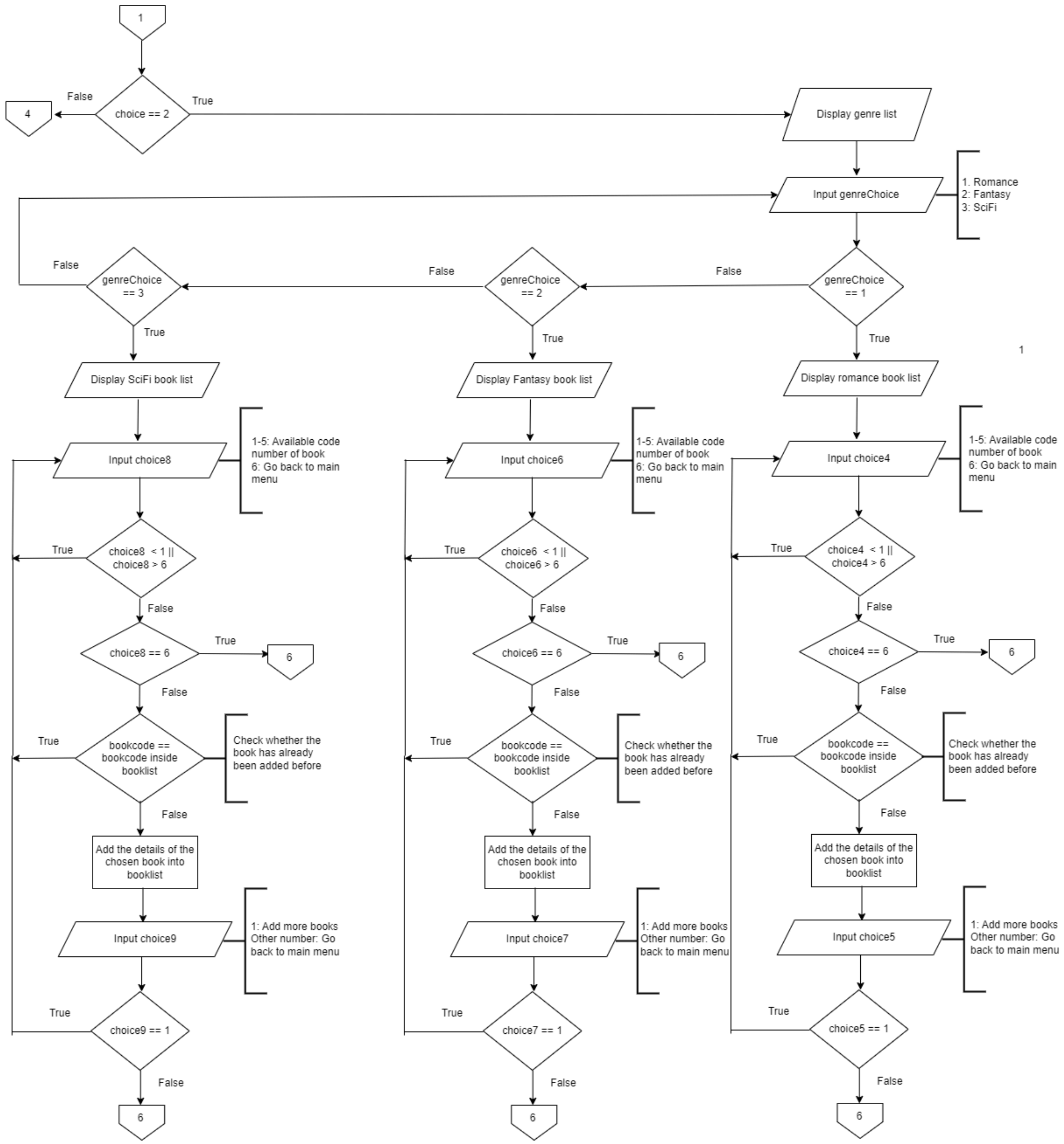
Purpose of the system

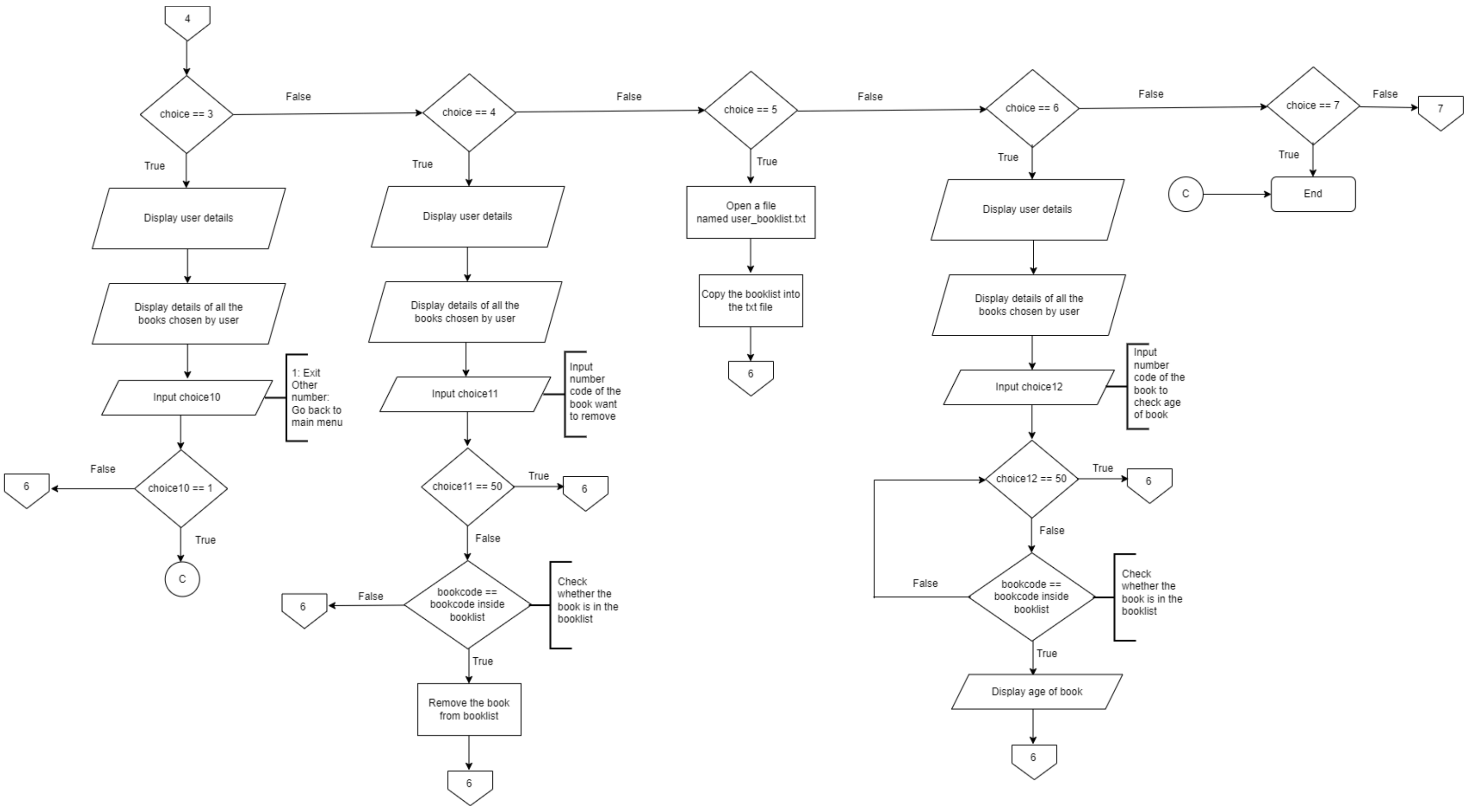
The purpose of this book recommendation system is to let users view the books sorted by genre. Then, users can choose the book that they like and add it to their personalized book list. If they don't want the book anymore, they can also choose to remove the book from their book list. The users can also check the age of the book in their booklist. Users also can save their book list as a txt file, so they can have a copy of their booklist for future use.

Analysis and Design

Flow Chart

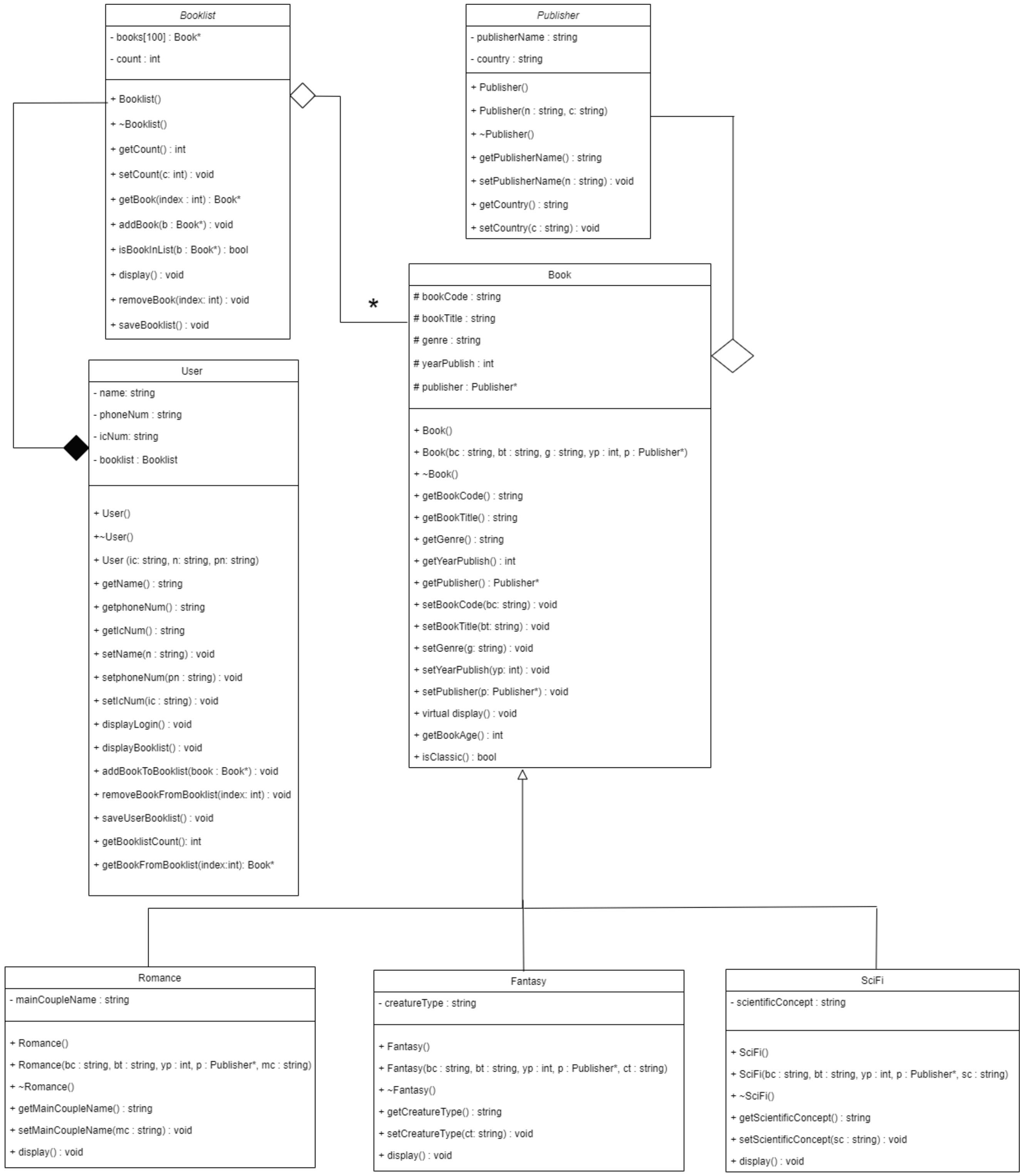

Start
Get user icNum
Get user name
Get user phoneNum
phoneNum != isalpha
False
True
6
B
Display menu
1: See trending book list
2: See specific genre book list
3: View book list
4. Remove book from booklist
5. Save booklist to txt file
6. Check if the book is classic book or not
7. Exit
7
Input choice
choice == 1
True
A
False
1
A
Display trending book list
Input choice2
1-5: Available code number of book
6: Go back to main menu
choice2 < 1 || choice2 > 6
True
False
choice2 == 6
True
B
False
bookcode == bookcode inside booklist
True
Check whether the book has already been added before
False
Add the details of the chosen book into booklist
Input choice3
1: Add more books
Other number: Go back to main menu
choice3 == 1
True
False
B


1
choice == 2
False
4
True
Display genre list
Input genreChoice
1. Romance
2: Fantasy
3: SciFi
genreChoice == 1
False
genreChoice == 2
False
genreChoice == 3
False
True
Display romance book list
Display Fantasy book list
Display SciFi book list
1
Input choice4
Input choice6
Input choice8
1-5: Available code number of book
6: Go back to main menu
choice4 < 1 || choice4 > 6
choice6 < 1 || choice6 > 6
choice8 < 1 || choice8 > 6
choice4 == 6
choice6 == 6
choice8 == 6
6
bookcode == bookcode inside booklist
Check whether the book has already been added before
Add the details of the chosen book into booklist
Input choice5
Input choice7
Input choice9
1: Add more books
Other number: Go back to main menu
choice5 == 1
choice7 == 1
choice9 == 1
6



4
choice == 3
False
True
Display user details
Display details of all the books chosen by user
Input choice10
1: Exit Other number: Go back to main menu
choice10 == 1
False
6
True
C
choice == 4
False
True
Display user details
Display details of all the books chosen by user
Input choice11
Input number code of the book want to remove
choice11 == 50
True
6
False
bookcode == bookcode inside booklist
Check whether the book is in the booklist
False
6
True
Remove the book from booklist
6
choice == 5
False
True
Open a file named user_booklist.txt
Copy the booklist into the txt file
6
choice == 6
False
True
Display user details
Display details of all the books chosen by user
Input choice12
Input number code of the book to check age of book
choice12 == 50
True
6
False
bookcode == bookcode inside booklist
Check whether the book is in the booklist
False
True
Display age of book
6
choice == 7
False
7
True
C
End

UML Class Diagram


Booklist
- books[100] : Book*
- count : int
+ Booklist()
+ ~Booklist()
+ getCount() : int
+ setCount(c: int) : void
+ getBook(index : int) : Book*
+ addBook(b : Book*) : void
+ isBookInList(b : Book*) : bool
+ display() : void
+ removeBook(index: int) : void
+ saveBooklist() : void
Publisher
- publisherName : string
- country : string
+ Publisher()
+ Publisher(n : string, c: string)
+ ~Publisher()
+ getPublisherName() : string
+ setPublisherName(n : string) : void
+ getCountry() : string
+ setCountry(c : string) : void
*
Book
# bookCode : string
# bookTitle : string
# genre : string
# yearPublish : int
# publisher : Publisher*
+ Book()
+ Book(bc : string, bt : string, g : string, yp : int, p : Publisher*)
+ ~Book()
+ getBookCode() : string
+ getBookTitle() : string
+ getGenre() : string
+ getYearPublish() : int
+ getPublisher() : Publisher*
+ setBookCode(bc: string) : void
+ setBookTitle(bt: string) : void
+ setGenre(g: string) : void
+ setYearPublish(yp: int) : void
+ setPublisher(p: Publisher*) : void
+ virtual display() : void
+ getBookAge() : int
+ isClassic() : bool
User
- name: string
- phoneNum : string
- icNum: string
- booklist : Booklist
+ User()
+~User()
+ User (ic: string, n: string, pn: string)
+ getName() : string
+ getphoneNum() : string
+ getIcNum() : string
+ setName(n : string) : void
+ setphoneNum(pn : string) : void
+ setIcNum(ic : string) : void
+ displayLogin() : void
+ displayBooklist() : void
+ addBookToBooklist(book : Book*) : void
+ removeBookFromBooklist(index: int) : void
+ saveUserBooklist() : void
+ getBooklistCount(): int
+ getBookFromBooklist(index:int): Book*
Romance
- mainCoupleName : string
+ Romance()
+ Romance(bc : string, bt : string, yp : int, p : Publisher*, mc : string)
+ ~Romance()
+ getMainCoupleName() : string
+ setMainCoupleName(mc : string) : void
+ display() : void
Fantasy
- creatureType : string
+ Fantasy()
+ Fantasy(bc : string, bt : string, yp : int, p : Publisher*, ct : string)
+ ~Fantasy()
+ getCreatureType() : string
+ setCreatureType(ct: string) : void
+ display() : void
SciFi
- scientificConcept : string
+ SciFi()
+ SciFi(bc : string, bt : string, yp : int, p : Publisher*, sc : string)
+ ~SciFi()
+ getScientificConcept() : string
+ setScientificConcept(sc : string) : void
+ display() : void

Section B
Implementation of the Concepts
Encapsulation

In this project, all classes have applied the concept of encapsulation, in which the classes bind the attributes and data into one entity and define itself and the scope of its operations. Encapsulation is demonstrated through the use of classes with private and public access specifiers.

1. class Publisher
   Attributes :
   - string publisherName
   - string country

   publisherName and country are private member variables of the class. They are accessible only within the class and its member functions.

   Methods:
   - Publisher() is a default constructor
   - Publisher(string n, string c) is a parameterized constructor that initializes publisherName and country with the provided values
   - ~Publisher() is a destructor
   - string getPublisherName() is an accessor that returns the value of publisherName
   - void setPublisherName(string n) is a mutator that set value for the publisherName
   - string getCountry() is an accessor that returns the value of country
   - void setCountry(string c) is a mutator that set value for the country

2. class Book
   Attributes :
   - string bookCode
   - string bookTitle
   - string genre
   - int yearPublish
   - Publisher* publisher

   All attributes are protected member variables of the class. They are accessible within the class and its derived classes. bookCode, bookTitle, genre, and yearPublish store information about the book, while publisher is a pointer to a Publisher object, indicating the publisher of the book.

Methods :

- Book() is a default constructor
- Book(string bc, string bt, string g, int yp, Publisher* p) is a parameterized constructor that initializes bookCode, bookTitle, genre, yearPublish and publisher with the provided values
- ~Book() is a destructor
- string getBookCode() is an accessor that returns the value of bookCode
- string getBookTitle() is an accessor that returns the value of bookTitle
- string getGenre() is an accessor that returns the value of genre
- int getYearPublish() is an accessor that returns the value of yearPublish
- Publisher* getPublisher() is an accessor that returns a pointer that points to the book's publisher
- void setBookCode(string bc) is a mutator that sets the value of bookCode
- void setBookTitle(string bt) is a mutator that sets the value of bookTitle
- void setGenre(string g) is a mutator that sets the value of genre
- void setYearPublish(int yp) is a mutator that sets the value of yearPublish
- void setPublisher(Publisher *p) is a mutator that sets the value of publisher
- virtual void display() is a virtual function that displays the information of the book. It is marked as virtual, indicating that it can be overridden by derived classes.
- int getBookAge() is a function to calculate the book age
- bool isClassic() is a function to check whether the book is a classic book

3. class Romance

   Attributes :

   - string mainCoupleName

   mainCoupleName is a private member variable of the Romance class. mainCoupleName is accessible only within the class and its member functions.

   Methods :

   - Romance() is a default constructor
   - Romance(string bc, string bt, int yp, Publisher* p, string mc) is a parameterized constructor that initializes the Romance object with the provided values
   - ~Romance() is a destructor
   - string getMainCoupleName() is an accessor that returns the value of mainCoupleName
   - void setMainCoupleName(string mc) is a mutator that sets the value of mainCoupleName

- void display() is a function that overrides the display() function of the base class Book. It first calls the display() function of the Book class to display book information, and then prints the mainCoupleName.

4. class Fantasy
   Attributes :
   - string creatureType

   creatureType is a private member variable of the Fantasy class. creatureType is accessible only within the class and its member functions.

   Methods :
   - Fantasy() is a default constructor
   - Fantasy(string bc, string bt, int yp, Publisher* p, string ct) is a parameterized constructor that initializes the Fantasy object with the provided values.
   - ~Fantasy() is a destructor
   - string getCreatureType() is a accessor that returns the value of creatureType
   - void setCreatureType(string ct) is a mutator that sets the value of creatureType
   - void display() is a function that overrides the display() function of the base class Book. It first calls the display() function of the Book class to display book information, and then prints the creatureType

5. class SciFi
   Attributes :
   - string scientificConcept

   scientificConcept is a private member variable of the SciFi class. scientificConcept is accessible only within the class and its member functions.

   Methods :
   - SciFi() is a default constructor
   - ~SciFi() is a destructor
   - SciFi(string bc, string bt, int yp, Publisher* p, string sc) is a parameterized constructor that initializes the SciFi object with the provided values.
   - string getScientificConcept() is an accessor that returns the value of scientificConcept
   - void setScientificConcept(string ct) is a mutator that sets the value of scientificConcept
   - void display() is a function that overrides the display() function of the base class Book. It first calls the display() function of the Book class to display book information, and then prints the scientificConcept

6. class Booklist
   Attributes :
   - Book* books[100]
   - int count

   Both attributes are private member variables of the Booklist class. Both attributes are accessible only within the class and its member functions. books is an array of pointers to Book objects. The array can hold up to 100 Book pointers. count is an integer that tracks the number of Book objects currently in the list.

   Methods:
   - Booklist() is a constructor
   - ~Booklist() is a destructor
   - int getCount() is an accessor that returns the count of books
   - void setCount(int c) is a mutator that set the value of count
   - Book* getBook(int index) is an accessor that returns a pointer that points to the book
   - void addBook(Book* b) is a mutator that adds book objects to the booklist object
   - bool isBookInList(Book* b) is a method that checks whether the book that the user wants to add is in the book list already
   - void display() is a method that displays the the list of books
   - void removeBook(int index) is a method to remove the book from booklist
   - void saveBooklist() is a method to save book list to a txt file

7. class User
   Attributes :
   - string name
   - string phoneNum
   - string icNum
   - Booklist booklist

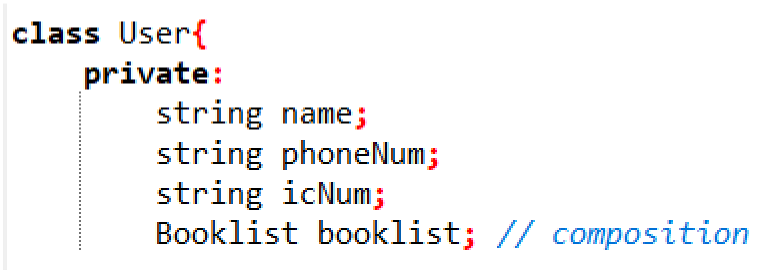
   All attributes are private member variables of the User class. All attributes are accessible only within the class and its member functions.

   Methods :
   - User() is a default constructor
   - User(string ic, string n, string pn) is a parameterized constructor that initializes a User object with the given IC number, name, and phone number.
   - ~User() is a destructor
   - string getName() is an accessor that return the name of the user
   - string getPhoneNum() is an accessor that returns the phoneNum of the user
   - string getIcNum() is an accessor that return the icNum of the user

- void setName(string n) is a mutator that sets the name of the user
- void setPhoneNum(string p) is a mutator that sets the phoneNum of the user
- void setIcNum(string ic) is a mutator that sets the icNum of the user
- void displayLogin() is a method to prompt the user to enter their IC number, name, and phone number
- void displayBooklist() is a method to display the user details and user's list of books.
- void addBookToBooklist(Book* book) is a method that adds the user's interested book to their booklist
- void removeBookFromBooklist(int index) is a method that remove book from user's personal booklist
- void saveUserBooklist() is a method that save user personalize booklist to txt file
- int getBooklistCount() is a method to get the total count of books in user booklist
- Book* getBookFromBooklist(int index) is a method to access a book in user booklist

<u>Composition</u>

Composition is a "has-a" relationship between objects, where one object contains or is composed of another. In this project, the user class consists of a booklist object, which is created by the Booklist class. In this case, the composition is applied as in the book recommendation system, the user can choose for the books they are interested in. Hence, the user must have a personalized booklist so they can add books to it. Employing composition, it means that the user class can also access to the public methods of the booklist class. Other than that, A User consists of a Booklist object, indicating that the Booklist cannot exist independently if there is no User. The booklist strongly depends on the User. If the User object is destroyed, then the Booklist object is also destroyed.



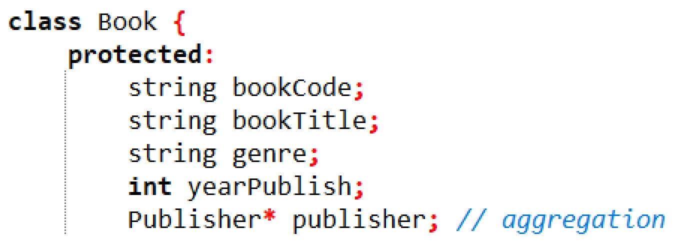
```
class User{
    private:
        string name;
        string phoneNum;
        string icNum;
        Booklist booklist; // composition
```

Aggregation

Aggregation implies a relationship where the child can exist independently of the parent.

1. Book and Publisher

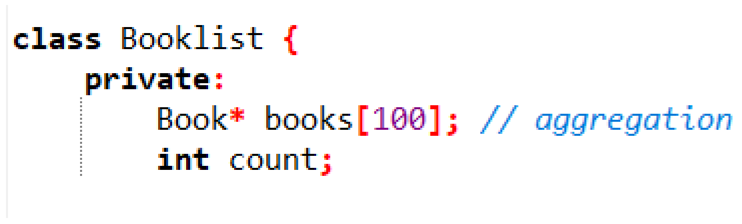
Book and Publisher will have an aggregation relationship. The Book class holds a pointer to a Publisher object. A Book has a Publisher. The Publisher object is created outside the Book and is passed to it, showing a "has-a" relationship. Publisher can exist independently. If the Book object is destroyed, the Publisher object will not be destroyed.



```
class Book {
    protected:
        string bookCode;
        string bookTitle;
        string genre;
        int yearPublish;
        Publisher* publisher; // aggregation
```

2. Booklist and Book

Booklist and Book will have an aggregation relationship. The Booklist class aggregates Book pointers, showing a "has-a" relationship, meaning booklist has a collection of books. Books can exist independently. If the Booklist object is destroyed, the Book object will not be destroyed.



```
class Booklist {
    private:
        Book* books[100]; // aggregation
        int count;
```

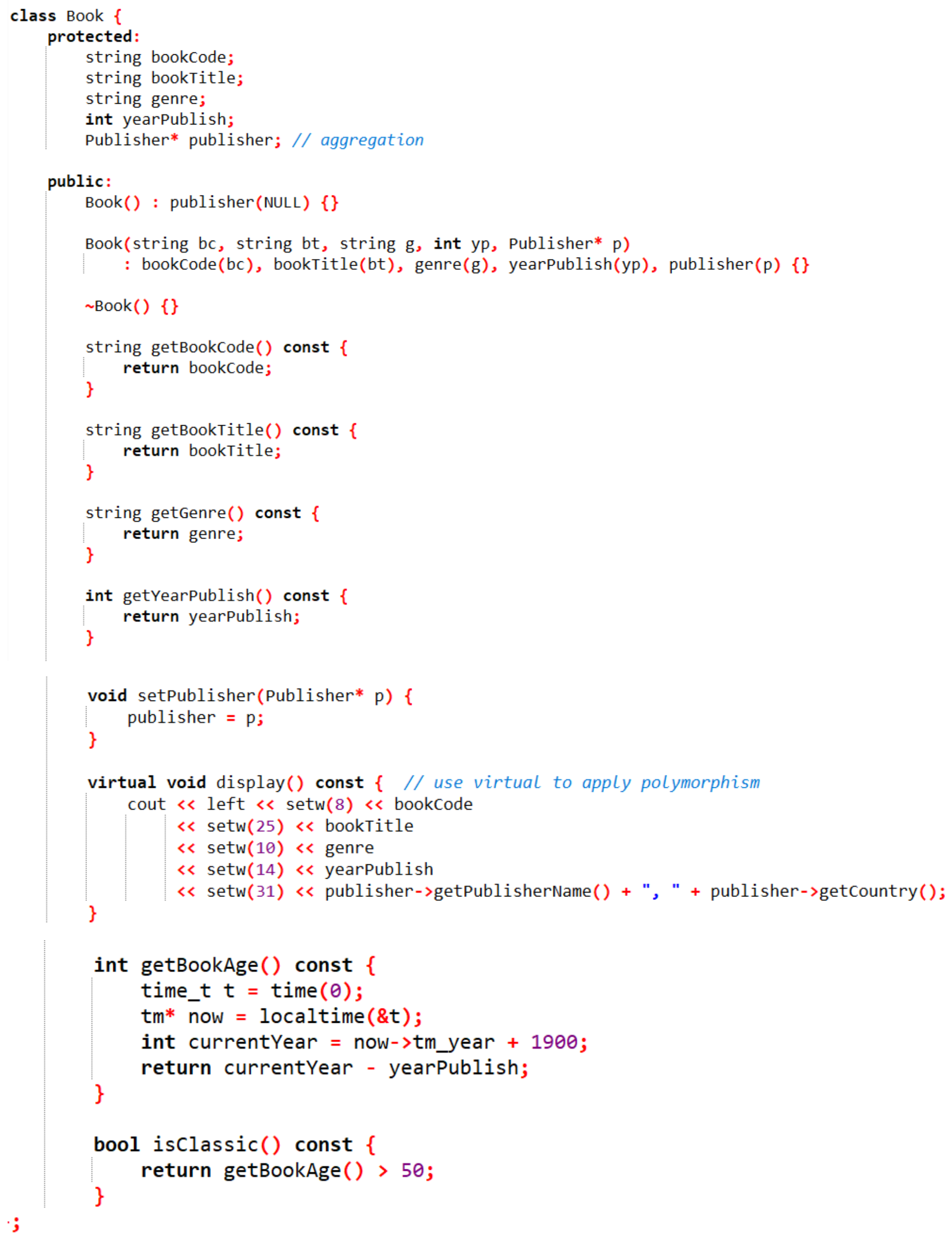
Inheritance

Inheritance allows derived classes to inherit properties and behaviors from the base class. In this project, inheritance is demonstrated through the Book class and its derived classes: Romance, Fantasy, and SciFi.

When an object of the Romance, Fantasy, or SciFi class is created, each object will contain all the attributes and methods of the Book class, plus any additional attributes and methods specific to the derived class. Each derived class object, for example: Romance class object, Fantasy class object, and SciFi class object can access the protected and public members of the Book class directly.

Base Class: Book

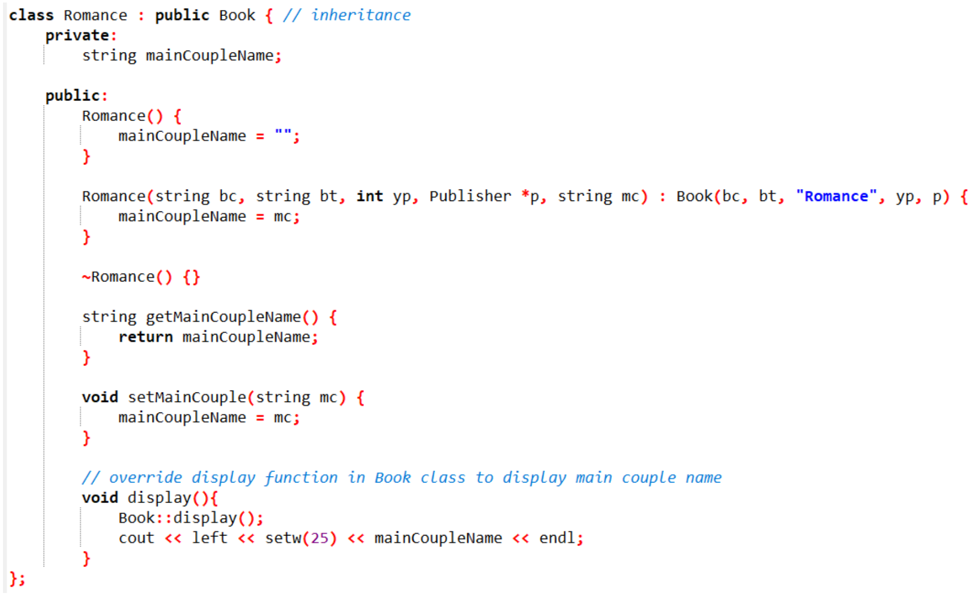
The Book class contains attributes and methods that are common to all types of books.



```cpp
class Book {
    protected:
        string bookCode;
        string bookTitle;
        string genre;
        int yearPublish;
        Publisher* publisher; // aggregation

    public:
        Book() : publisher(NULL) {}

        Book(string bc, string bt, string g, int yp, Publisher* p)
            : bookCode(bc), bookTitle(bt), genre(g), yearPublish(yp), publisher(p) {}

        ~Book() {}

        string getBookCode() const {
            return bookCode;
        }

        string getBookTitle() const {
            return bookTitle;
        }

        string getGenre() const {
            return genre;
        }

        int getYearPublish() const {
            return yearPublish;
        }

        void setPublisher(Publisher* p) {
            publisher = p;
        }

        virtual void display() const {  // use virtual to apply polymorphism
            cout << left << setw(8) << bookCode
                 << setw(25) << bookTitle
                 << setw(10) << genre
                 << setw(14) << yearPublish
                 << setw(31) << publisher->getPublisherName() + ", " + publisher->getCountry();
        }

        int getBookAge() const {
            time_t t = time(0);
            tm* now = localtime(&t);
            int currentYear = now->tm_year + 1900;
            return currentYear - yearPublish;
        }

        bool isClassic() const {
            return getBookAge() > 50;
        }
};
```

Derived Class: Romance

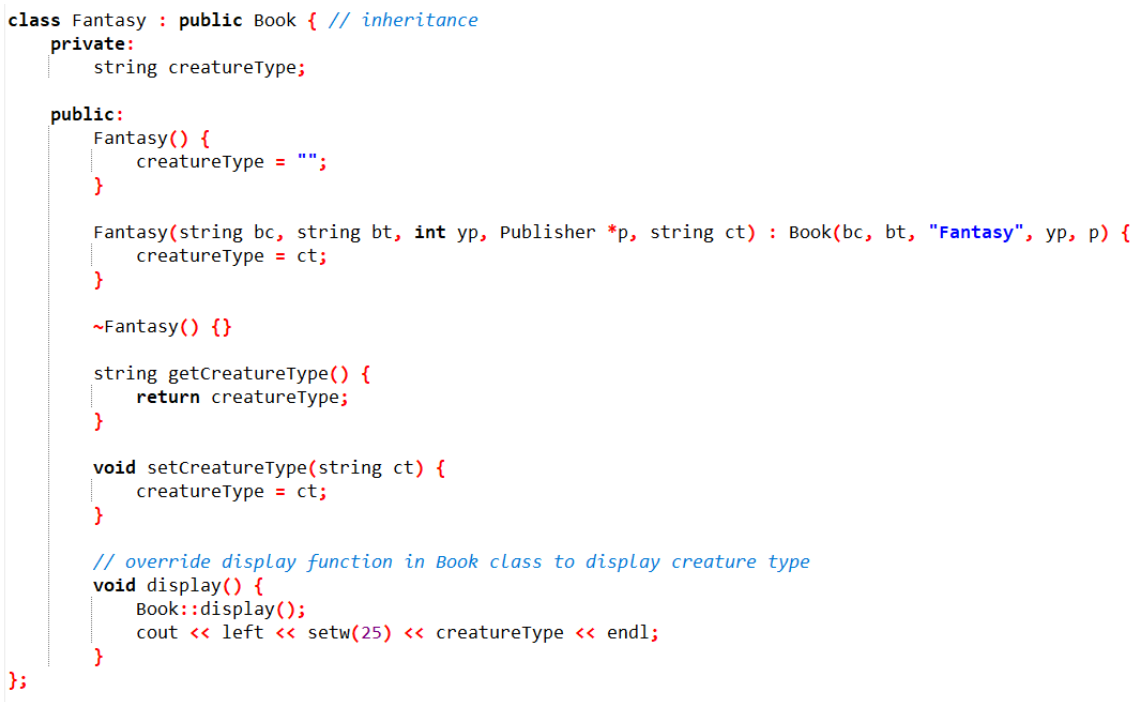
The Romance class inherits from Book and adds an attribute specific to romance books. The attribute specific to the Romance class is mainCoupleName.



```cpp
class Romance : public Book { // inheritance
    private:
        string mainCoupleName;

    public:
        Romance() {
            mainCoupleName = "";
        }

        Romance(string bc, string bt, int yp, Publisher *p, string mc) : Book(bc, bt, "Romance", yp, p) {
            mainCoupleName = mc;
        }

        ~Romance() {}

        string getMainCoupleName() {
            return mainCoupleName;
        }

        void setMainCouple(string mc) {
            mainCoupleName = mc;
        }

        // override display function in Book class to display main couple name
        void display(){
            Book::display();
            cout << left << setw(25) << mainCoupleName << endl;
        }
};
```

Derived Class: Fantasy

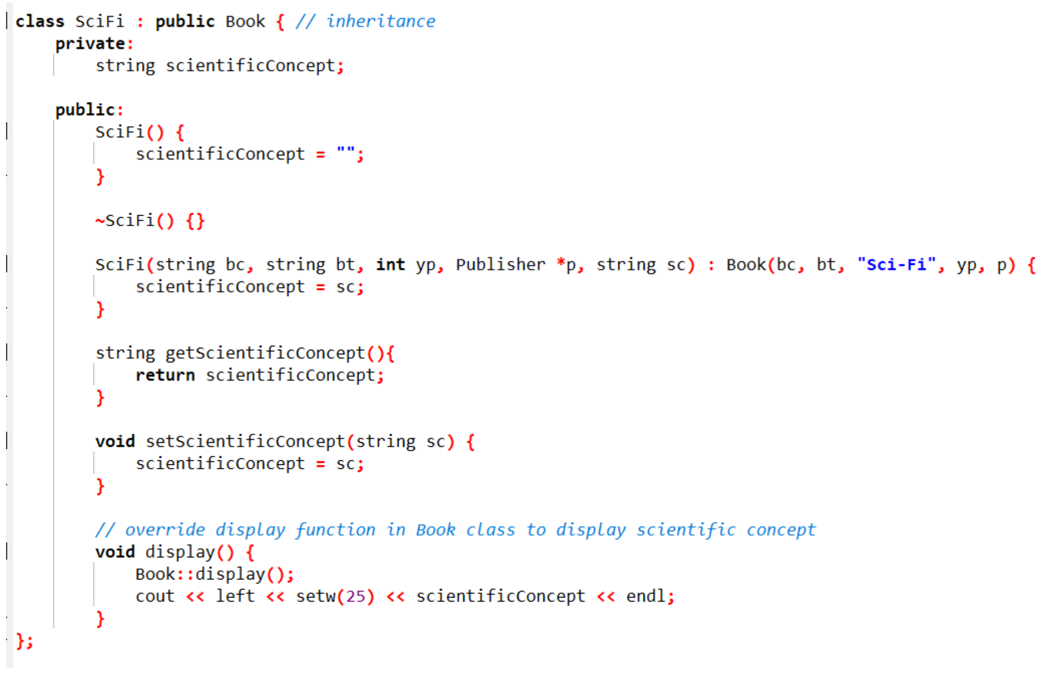
The Fantasy class inherits from Book and adds an attribute specific to fantasy books. The attribute specific to the Fantasy class is creatureType.



```cpp
class Fantasy : public Book { // inheritance
    private:
        string creatureType;

    public:
        Fantasy() {
            creatureType = "";
        }

        Fantasy(string bc, string bt, int yp, Publisher *p, string ct) : Book(bc, bt, "Fantasy", yp, p) {
            creatureType = ct;
        }

        ~Fantasy() {}

        string getCreatureType() {
            return creatureType;
        }

        void setCreatureType(string ct) {
            creatureType = ct;
        }

        // override display function in Book class to display creature type
        void display() {
            Book::display();
            cout << left << setw(25) << creatureType << endl;
        }
};
```

Derived Class: SciFi

The SciFi class inherits from Book and adds an attribute specific to sci-fi books. The attribute specific to the SciFi class is scientificConcept.



```cpp
class SciFi : public Book { // inheritance
    private:
        string scientificConcept;

    public:
        SciFi() {
            scientificConcept = "";
        }

        ~SciFi() {}

        SciFi(string bc, string bt, int yp, Publisher *p, string sc) : Book(bc, bt, "Sci-Fi", yp, p) {
            scientificConcept = sc;
        }

        string getScientificConcept(){
            return scientificConcept;
        }

        void setScientificConcept(string sc) {
            scientificConcept = sc;
        }

        // override display function in Book class to display scientific concept
        void display() {
            Book::display();
            cout << left << setw(25) << scientificConcept << endl;
        }
};
```

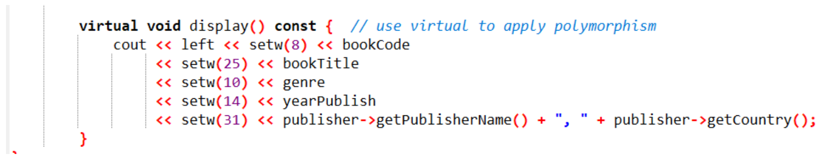
Polymorphism

In this project, polymorphism is applied using a virtual function. In the Book class, the display() function is declared as virtual void display() const. This means that derived classes which are Romance, Fantasy, SciFi can override this function with their specific implementations.

Each derived class overrides the display function in the Book class to display all the common book attributes and every derived class specific attribute. This allows each genre of book to display additional details, for example mainCoupleName for Romance books, creatureType for Fantasy books and scientificConcept for Sci-Fi books while still leveraging common behavior defined in the Book class.

Base Class: Book

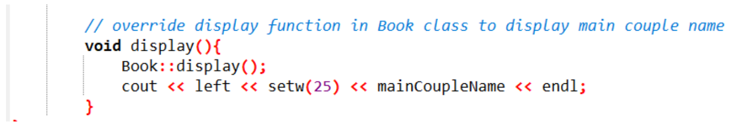
Virtual display method is declared in Book class to display the common attribute of a book which includes the bookCode, bookTitle, genre, yearPublish and publisher. This virtual display method can be overridden by derived classes.



```
virtual void display() const {  // use virtual to apply polymorphism
    cout << left << setw(8) << bookCode
         << setw(25) << bookTitle
         << setw(10) << genre
         << setw(14) << yearPublish
         << setw(31) << publisher->getPublisherName() + ", " + publisher->getCountry();
}
```

Derived Class: Romance

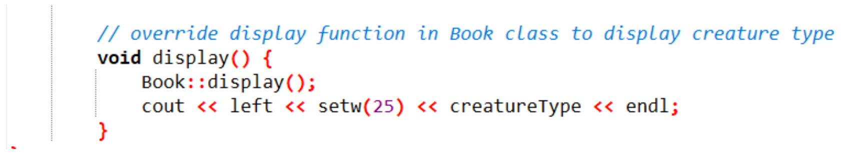
Romance class overrides the display() function to display genre-specific information which is mainCoupleName in addition to the general book details.



```
// override display function in Book class to display main couple name
void display(){
    Book::display();
    cout << left << setw(25) << mainCoupleName << endl;
}
```

Derived Class: Fantasy

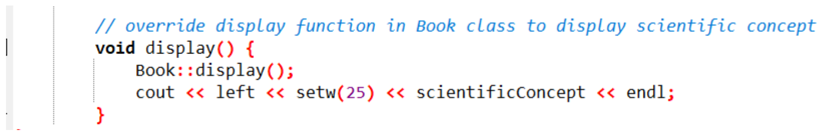
Fantasy class overrides the display() function to display genre-specific information which is creatureType in addition to the general book details.



```
// override display function in Book class to display creature type
void display() {
    Book::display();
    cout << left << setw(25) << creatureType << endl;
}
```

Derived Class: SciFi

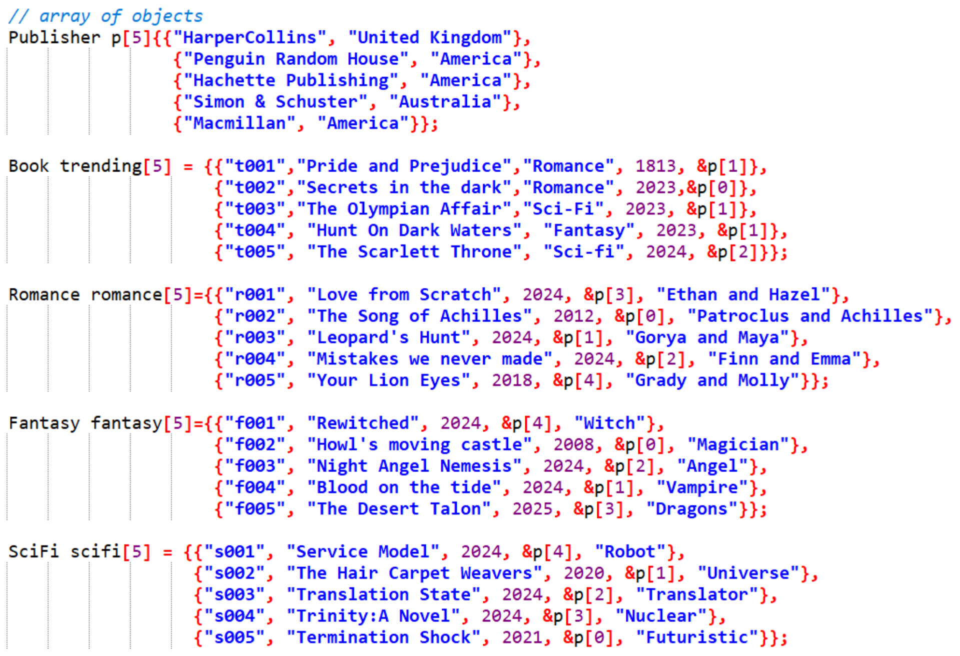
SciFi class overrides the display() function to display genre-specific information which is scientificConcept in addition to the general book details.



```
// override display function in Book class to display scientific concept
void display() {
    Book::display();
    cout << left << setw(25) << scientificConcept << endl;
}
```

Array of objects

In this project, arrays of objects are initialized directly in the main function. Arrays of objects in this code are used to manage collections of books and publishers. For example, p is an array of Publisher objects, trending is an array of Book objects, romance is an array of Romance objects, fantasy is an array of Fantasy objects and scifi is an array of SciFi objects. Each element in these arrays represents an instance of the respective class.

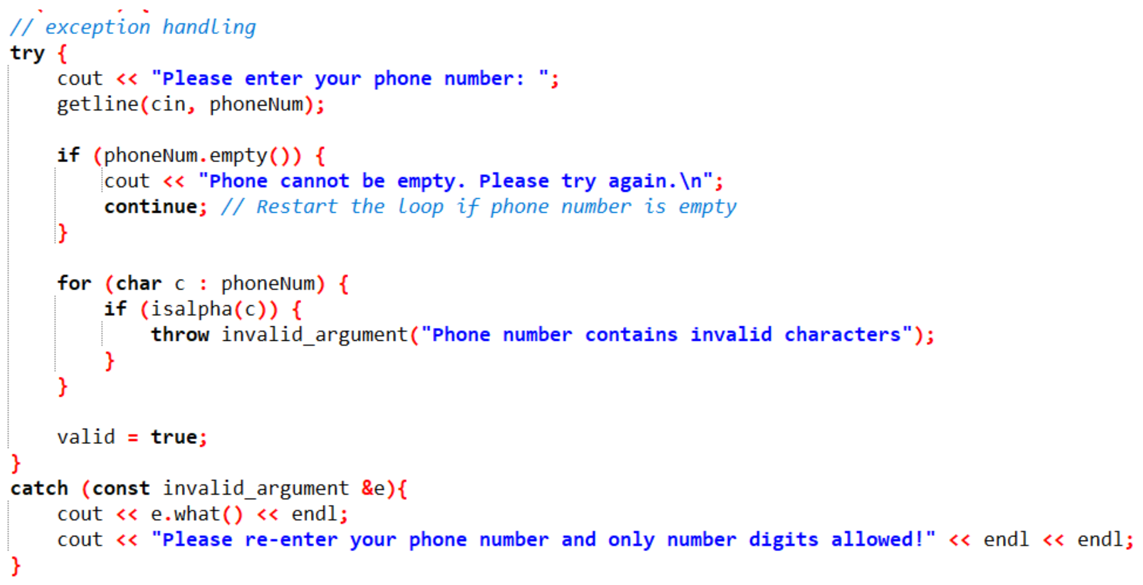


```
// array of objects
Publisher p[5]{{"HarperCollins", "United Kingdom"},
                {"Penguin Random House", "America"},
                {"Hachette Publishing", "America"},
                {"Simon & Schuster", "Australia"},
                {"Macmillan", "America"}};

Book trending[5] = {{"t001","Pride and Prejudice","Romance", 1813, &p[1]},
                    {"t002","Secrets in the dark","Romance", 2023,&p[0]},
                    {"t003","The Olympian Affair","Sci-Fi", 2023, &p[1]},
                    {"t004", "Hunt On Dark Waters", "Fantasy", 2023, &p[1]},
                    {"t005", "The Scarlett Throne", "Sci-fi", 2024, &p[2]}};

Romance romance[5]={{"r001", "Love from Scratch", 2024, &p[3], "Ethan and Hazel"},
                    {"r002", "The Song of Achilles", 2012, &p[0], "Patroclus and Achilles"},
                    {"r003", "Leopard's Hunt", 2024, &p[1], "Gorya and Maya"},
                    {"r004", "Mistakes we never made", 2024, &p[2], "Finn and Emma"},
                    {"r005", "Your Lion Eyes", 2018, &p[4], "Grady and Molly"}};

Fantasy fantasy[5]={{"f001", "Rewitched", 2024, &p[4], "Witch"},
                    {"f002", "Howl's moving castle", 2008, &p[0], "Magician"},
                    {"f003", "Night Angel Nemesis", 2024, &p[2], "Angel"},
                    {"f004", "Blood on the tide", 2024, &p[1], "Vampire"},
                    {"f005", "The Desert Talon", 2025, &p[3], "Dragons"}};

SciFi scifi[5] = {{"s001", "Service Model", 2024, &p[4], "Robot"},
                  {"s002", "The Hair Carpet Weavers", 2020, &p[1], "Universe"},
                  {"s003", "Translation State", 2024, &p[2], "Translator"},
                  {"s004", "Trinity:A Novel", 2024, &p[3], "Nuclear"},
                  {"s005", "Termination Shock", 2021, &p[0], "Futuristic"}};
```

<u>Exception handling</u>

In this project, exception handling is used to manage errors related to user input phone number. In the User class, exception handling is applied specifically to validate the phone number input. This is to ensure that the phone number entered consists only of digits.

The try block attempts to read and validate the phone number. If the phone number contains any alphabetic characters (isalpha(c)), an invalid_argument exception is thrown with a descriptive message. The catch block catches the exception (invalid_argument) and displays an error message, prompting the user to retry entering the phone number. The loop continues until a valid phone number (containing only digits) is entered.



```
// exception handling
try {
    cout << "Please enter your phone number: ";
    getline(cin, phoneNum);

    if (phoneNum.empty()) {
        cout << "Phone cannot be empty. Please try again.\n";
        continue; // Restart the loop if phone number is empty
    }

    for (char c : phoneNum) {
        if (isalpha(c)) {
            throw invalid_argument("Phone number contains invalid characters");
        }
    }

    valid = true;
}
catch (const invalid_argument &e){
    cout << e.what() << endl;
    cout << "Please re-enter your phone number and only number digits allowed!" << endl << endl;
}
```