C++ Assignment

Library Management System

PG-DAC A25

Project Overview

Build a **Library Management System** in C++ that allows managing books, members, and book transactions (issue/return). This project will help you apply all C++ concepts learned in the course.

Features to Implement

1. **Book Management** - Add, remove, search, and display books
2. **Member Management** - Register, remove, search members
3. **Issue/Return Books** - Track which member has which book
4. **Fine Calculation** - Calculate fine for late returns
5. **Data Persistence** - Save and load data from files

Project Structure

Create the following folder structure:

```
LibraryManagementSystem/
├── headers/          (header files .h)
├── src/              (source files .cpp)
├── data/             (text files for storage)
└── README.md
```

Classes to Create

1. Person (Base Class)

- **Purpose**: Base class for Member and Librarian
- **Members**: id, name, phone
- **Key Points**:
  - Make this an abstract class using pure virtual function `displayInfo()`
  - Implement **virtual destructor** to ensure proper cleanup when deleting derived objects through base pointer

> **Important**: Without virtual destructor, only base class destructor will be called causing resource leak!

2. Member (Derived from Person)

- **Purpose**: Represents library members who can borrow books
- **Inheritance Mode**: Use `public` inheritance
- **Additional Members**:
  - Number of books currently issued
  - Array/list of issued book IDs (use dynamic memory)
- **Key Points**:
  - Implement copy constructor (deep copy)
  - Implement destructor to free dynamic memory
  - Override `displayInfo()` from base class
  - Overload `=` and `==` operators
  - Overload `<<` and `>>` operators

3. Librarian (Derived from Person)

- **Purpose**: Represents staff who manage the library
- **Inheritance Mode**: Use `public` inheritance
- **Additional Members**: employeeId, salary
- **Key Points**:
  - Override `displayInfo()` from base class
  - Has special permissions to add/remove books

4. Book

- **Purpose**: Represents a book in the library
- **Members**: bookId, title, author, isbn, status, quantity, available copies
- **Key Points**:
  - Use `enum class` for BookStatus (AVAILABLE, ISSUED, etc.)
  - Use `enum class` for BookGenre
  - Overload operators: `++` (pre/post), `--`, `==`, `<`, `[]`, `<<`, `>>`

> **Hint for `[]` operator**: Return different book details based on index (0=id, 1=title, 2=author, etc.)

5. Container (Template Class)

- **Purpose**: Generic container to store Books or Members
- **Members**: Dynamic array, size, capacity
- **Key Points**:
  - Use `new` in constructor, `delete[]` in destructor
  - Implement copy constructor and assignment operator
  - Use function overloading for `find()` - search by ID or by name

> **Important**: Template class implementation must be in header file, not in separate .cpp file

6. Transaction

- **Purpose**: Track book issue/return transactions
- **Members**: transactionId, memberId, bookId, issueDate, dueDate, returnDate, fineAmount
- **Key Points**:
  - Use `static` member for auto-incrementing transaction ID

- Use `static const` for fine rate per day
- Calculate fine based on days overdue

7. Library

- **Purpose**: Main class that manages everything
- **Members**:
  - Container of Books (your template class)
  - Container of Members (your template class)
  - Vector of Transactions (use STL)
  - Map of bookId to memberId (use STL)
- **Key Points**:
  - Implement file handling to save/load data
  - Use `fstream` for file operations

Runtime Polymorphism Requirements

Your project must demonstrate **runtime polymorphism**:

1. **Virtual Functions**: `displayInfo()` in Person class must be virtual
2. **Function Overriding**: Member and Librarian must override `displayInfo()`
3. **Base Class Pointers**: Store Person* pointers that can point to Member or Librarian objects
4. **Late Binding**: When calling `displayInfo()` through Person pointer, correct derived class version should execute

> **Test This**: Create `Person* p = new Member(...)` and call `p->displayInfo()`. It should print Member's version, not Person's.

Custom Exceptions to Create

Create a base `LibraryException` class inheriting from `std::exception`, then create derived exceptions:

1. `BookNotFoundException`
2. `MemberNotFoundException`
3. `BookNotAvailableException`
4. `MaxBooksExceededException`

> **Tip**: Store error message, function name, and line number in exceptions. Use `__FUNCTION__` and `__LINE__` macros.

Namespace

Wrap all your classes inside a namespace called `LibrarySystem`.

Main Program

Create a menu-driven program with options:

```
1.  Add New Book
2.  Remove Book
3.  Search Book
4.  Display All Books
5.  Register Member
6.  Remove Member
7.  Search Member
8.  Display All Members
9.  Issue Book
10. Return Book
11. View Overdue Books
12. Save Data
13. Load Data
0.  Exit
```

> **Important**: Wrap menu operations in try-catch blocks to handle exceptions gracefully.

Concepts Checklist

Make sure your project uses all these concepts:

Concept	Where to Use
Pointers & Dynamic Memory	Container class, Member's book list
References	Function parameters
Classes & Objects	All classes
Access Specifiers	private, protected, public appropriately
Constructors	Default, Parameterized, Copy
Destructors	Container, Member
Inheritance (public mode)	Person → Member, Person → Librarian
Virtual Functions	displayInfo() in Person
Pure Virtual Function	Makes Person abstract
Function Overriding	Member/Librarian override displayInfo()
Virtual Destructor	Person destructor must be virtual
Runtime Polymorphism	Person* pointing to derived objects
Upcasting	Derived to Base (implicit)
this pointer	Setter methods, operator=
const member functions	All getters
static members	Transaction ID counter

Concept	Where to Use
Namespaces	LibrarySystem
Function Overloading	find() methods
Operator Overloading	At least 5 operators
Templates	Container class
STL (vector, map)	Library class
File Handling	Save/Load functions
Exception Handling	Custom exceptions
Enums	BookStatus, BookGenre

Important Instructions

Memory Management

- Every `new` must have a corresponding `delete`
- Implement destructor wherever you use dynamic allocation
- Implement copy constructor if class has pointers (Rule of Three)

File Format

Save data in simple text format. Example for books:

```
101|Introduction to C++|Bjarne Stroustrup|978-0321958327|AVAILABLE|5|5
102|Data Structures|Mark Weiss|978-0132847377|ISSUED|3|2
```

Getting Started Steps

1. **Start with simple classes first** - Create Book class with basic members and constructors
2. **Add operators one by one** - Test each operator before moving to next
3. **Build Container template** - Test with int first, then with Book
4. **Create inheritance hierarchy** - Person → Member
5. **Add file handling** - Save/load one class at a time
6. **Add exceptions** - Replace error messages with proper exceptions
7. **Build Library class** - Integrate everything
8. **Create main menu** - Connect all functionality

Sample Test Scenario

Your program should handle this flow:

1. Add 3 books to library

2. Register 2 members
3. Issue a book to member 1
4. Try to issue same book to member 2 (should show not available)
5. Return the book
6. Issue same book to member 2 (should succeed now)
7. Save all data to files
8. Exit and restart program
9. Load data from files (all previous data should be restored)

Happy Coding!