Library Management System
– ESE 224 2022 Fall Mid-Term Version

Xin Wang
Stony Brook University
x.wang@stonybrook.edu

Xi Cheng
Stony Brook University
cheng.xi@stonybrook.edu

1 Introduction

This is the first version for your first submission (mid-term version). The second version (final version) will be updated depending on the following lectures. You have to submit the first part in the midst of the semester and the second part by the end of the semester.

1.1 Project Objective

- Practice analyzing and debugging techniques.
- Develop good coding habits:
  - Use separate code files for the declarations and implementations of different classes.
  - Use functions to make the code concise and modular.
  - Use comments where necessary.
  - Use proper indent.
- Read/write data from/to files.
- Basic interface design with user selections.
- Implement object-oriented programming concepts such as inheritance, overloading.

1.2 Team Spirit

The project has to be completed by a group of 3/4 students. If you cannot group with others, TAs will help you to group. In the report, you will have to detail the tasks each students finish. A student who has done 99% of the work is not guaranteed to have a high score if other teammates have not done anything. So get everyone involved in discussions, coding, writing, etc. Note: Github[*] is recommended. Here is a tutorial https://guides.github.com/activities/hello-world/.

1.3 Library Management System

In this project, you will learn to design a Library Management System (LMS). A LMS is a software that uses to maintain the record of the library. It contains work like the number of available books in the library, the number of books is issued or returning or renewing a book or late penalty record, etc. LMS can help to maintain a database that is useful to enter new books & record books borrowed by the members, with the respective submission dates. Moreover, it also reduces the manual record burden of the librarian. LMS allows the librarian to maintain library resources in a more operative manner that will help to save their time. It is also convenient for the librarian to manage the process of books allotting and making payment. LMS is also useful for users as well as a librarian to keep the constant track of the availability of all books in a store. In this project, a student/teacher will be able to use his own username and password to access the system and perform corresponding operations.

[*]https://github.com/

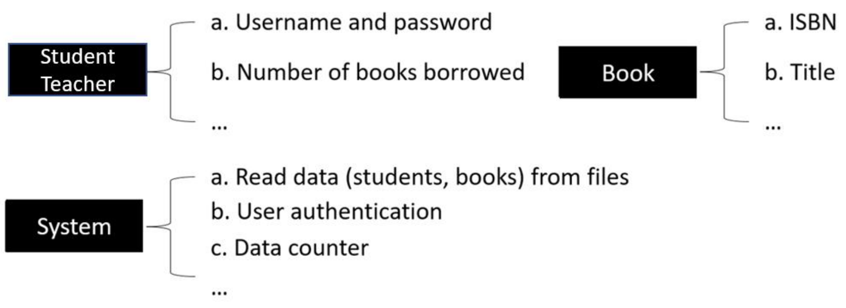


Figure 1: The definition of the classes.

2 Class Definitions

2.1 Classes Overview

Figure 1 shows the three classes that you need to define and implement for the LMS (**You don't need to implement System as a class**).

2.2 Student

Student class has the following attributes:

- Username.
- Password.
- Maximum number of copies that is allowed to keep. A student can keep ≤ 5 copies.
- Maximum borrowing periods. A student can keep ≤ 30 days for each copy.
- List of copies borrowed by the student.

2.3 Teacher

Teacher class has the exact same attribute as **Student** class, except:

- Maximum number of copies that is allowed to keep. A teacher can keep ≤ 10 copies.
- Maximum borrowing periods. A student can keep ≤ 50 days for each copy.

2.4 Book

Book class is used to represent a set of copies that share the same unique International Standard Book Number (ISBN). There are many copies sharing the same ISBN but they can have different IDs. Each object of this class should have the following information:

- ISBN.
- Title.
- Author. For simplicity, each book has only one author.
- Category, such as math, chemistry, engineering, etc.
- ID. This is the unique identification for each copy.
- Reader's name, indicating who borrows this copy.
- Start date of borrowing periods, indicating when the reader starts to keep the copy.
- Expiration date, indicating the expected date for the reader to return the copy.

2.5 System

System is not an actual class (it doesn't have any public or private attributes). You don't need to implement it as a class. It represents a bunch of utility functions that make you LMS running (see below).

3 Functions

To make LMS works successfully, the following functions need to be implemented. Some of them are class member functions (3.4, 3.5 and 3.6), and others are standalone functions (3.1, 3.2 and 3.3).

3.1 File Operation

In your project, all the data should be stored in files, specifically, student.txt, book.txt. Each time you run the system, it firstly reads these files and all the data in the files should be loaded into proper data structures, such as vectors and lists. When the program is ended, the data should be updated back to the same files if any updates have been operated (not in this version).

3.2 User Authentication

The user of the system has his own username and password. Each time he wants to access the system, he is asked to enter them for authentication. Only if it's successful, he can access the system and perform operations. Otherwise, the program will be directly ended. If access successfully, a menu like Figure 2 will be shown to the user.

3.3 Date Counter

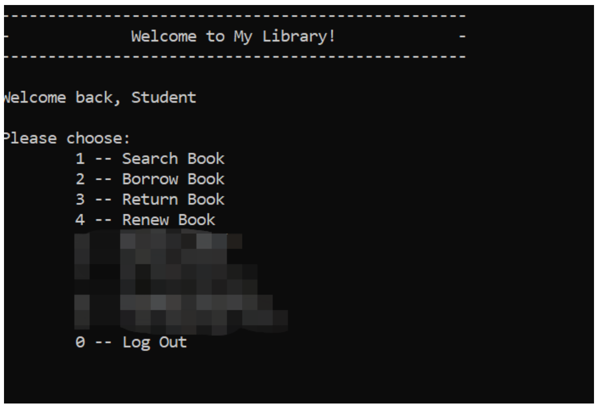
In your system, you have to simulate the calendar by setting up a **Date Counter**. This counter adds 1 at the end of each day. You may call the function *clock()* of <*ctime*> library to get the clock ticks. Refer to http://www.cplusplus.com/reference/ctime/.

Requirements:

- In LMS, the length of a "day" is 5 (any pre-defined value is fine) seconds.
- The TAs will evaluate your codes in one shot. Date counter is continuous for simplicity. No need to consider what will happen when run the program the 2nd time.

3.4 Student member functions

The operations a **Student** can perform are shown in Figure 2.



```
------------------------------------------------
-              Welcome to My Library!          -
------------------------------------------------

Welcome back, Student

Please choose:
        1 -- Search Book
        2 -- Borrow Book
        3 -- Return Book
        4 -- Renew Book
        [redacted]
        0 -- Log Out
```

Figure 2: Reader Menu.

3.4.1 Search Books

A student can search for books by key like **ISBN**, **title, category** and **ID**. The output should include all the information of the matched books **except Reader's name**.

Requirements:

- If there are several books fits the searching conditions, the system should display them all.
- If a student chooses searching the book copy through **ID**, the system should use **binary search** to search for the corresponding copy. You should not use the index in the vector/array to access the corresponding copy as a book copy maybe removed. (see 3.5.2)
- The system will display books in the order based on several properties, the priority is as follows:
    - Whether the books are available or not. The books that are available to rent will always before books that are already be borrowed.
    - For the books that are unavailable to rent, they should be sorted by the expiration date (the books that expired sooner should before the books that expired later).
    - Alphabetical order of the book **Title**.
    - Alphabetical order of the book **Author**.
    - Alphabetical order of the book **ISBN**.
    - Alphabetical order of the book **ID**.

3.4.2 Borrow Books

A student can borrow a copy of a book by identifying its ID. This student should be denoted as the current borrower of this copy. Meanwhile, the copy should be added to the student's list of borrowed copies.

Requirements:

- If the student has overdue copies, he/she cannot borrow new copies. The system shows a reminder message to the student if he/she has any.
- A student cannot borrow more than the maximum.
- A student cannot borrow a copy that has been lent to others.

3.4.3 Return Books

A student can return a copy of a book through the book ID. The borrowed copy becomes available to others again once it is returned. The copy should be removed from this user's list of borrowed copies.

3.4.4 Renew Books

A student can renew a copy of a book through the book ID to extend its expiration date. You should reset the expiration date of the book as if it were borrowed today. When the student chooses Renew books in the main menu, the system should display all the books the student has borrowed but not returned, sorted by the expiration date. Then the student can enter the ID of the book copy he/she wants to renew.

3.4.5 Operator overloading

You're required to implement overloading of the stream operator << and >> for each class (object) to make them read in or output an object. For example, in Book.cpp, you will have something similar to:

```
ostream& operator << (ostream& output, const Book* book) {
string studentname = "NONE";
if (!book)
    return output;
if (book->getStudent())
    studentname = book->getStudent()->getUsername();
output << "\tID:\t\t" << book->getID() << endl
...
...
istream& operator >> (istream& input, Book* book) {
int id, borrow, expire;
string isbn, studentname;
bool available;
input >> id >> isbn >> studentname >> ... >> expire;
...
```

3.4.6 Accessors and mutators

The *get()* and *set()* functions you used in labs, for all the student attributes such as the username, passwords, etc.

3.5 Teacher member functions

The teacher class will have all the member function as the student class. Besides, it will also include two additional functions.

3.5.1 Request a new book copy

A teacher can request a new book copy and add it to the library. After choosing this option, the system will ask the teacher to input **the ISBN, title, author,** and **category** of the new book. The new book will be added to the library and assign a new **ID**. **You should only add the book to the data structure you use to save books at runtime, not write the book to the book.txt file.**

3.5.2 Delete an existing copy

A teacher can delete an existing copy from the library using the book **ID**. After choosing this option, the system will ask the teacher to input the ID of the book copy. The book copy will be removed from the library. **You should only delete the book copy to the data structure you use to save books at runtime, not remove the book copy from the book.txt file.**

3.6 Book member functions

The book class doesn't have any special actions, it only needs to include accessors, mutators and operator overloading similar to 3.4.4 and 3.4.5.

4 Submission & Grading

4.1 Submission Requirements

Each group only submit one version of project. But in the project, the contribution of each member must be clarified, so that TAs are able to grade depends on the contributions. In general, members in one group have pretty close scores unless some members do not contribute much.

4.1.1 Code

- Write your codes within the group and not share inter-groups.
- Make sure your codes can be compiled successfully before you submit them.
- Your codes should be concise and modular by the class. All files, including .cpp, .h and data files, should be submitted in a project folder, and all codes in one main file is not recommended.
- Provide comments in your codes where necessary.
- Explain how to use your system to borrow or return a book.

4.1.2 Report

TAs will first read your report and then check your codes accordingly. Your project report should describe clearly:

- Architecture of your project and the explanation of each section it has. Please explain how some of the functions/classes work. Explain the designing logic behind the codes.
- Specification of classes, including the attributes and functions of each class.
- System functions you implement, including how parameters are passed when each function is called.
- All the highlights of your projects. Some of the students may have some features that are not exactly the same with the requirements from the project and they think they are good to be included. That is totally fine. But the students need to explain well.
- Important: always explain why you code like that.
- Notice the system you are using (Windows, Mac, MS, or xcode).

4.2 Grading

The grading scale of the first submission is:

- Required part (65 points)
  - Implementation of all classes (10 points)
  - File operation (10 points)
  - User authentication (10 points)
  - Date counter (10 points)
  - Student and teacher member function (20 points: search 10 points, others 10 points)
  - Book member function (5 points)
- Report (30 points)
- Concise and modular coding style (5 points)
  - Putting different classes declaration and implementation in separate files.
  - Giving comments in your codes.
  - Using functions instead of putting all lines in *main()* function.
- Bonus (10 points)
  - Extra functions (10 points).
    - Add a new function that allows the student and teacher see the top 10 popular book ranking. (The popularity of a book is how many times the book has been borrowed, different book copies contribute to the same book)