Library Management System
– ESE 224 2020 Fall Mid-Term Version

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

Xuewen Yang
Stony Brook University
xuewen.yang@stonybrook.edu

Jingwei Li
Stony Brook University
jingwei.li@stonybrook.edu

1 Introduction

This is the final version of the project. Please submit 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. You will have to finish grouping by the end of Friday night. 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 are 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 user will be able to use his own username and password to access the system and perform corresponding operations according to his user type - reader or librarian. A reader can borrow a book and a librarian can add a new reader.

*https://github.com/

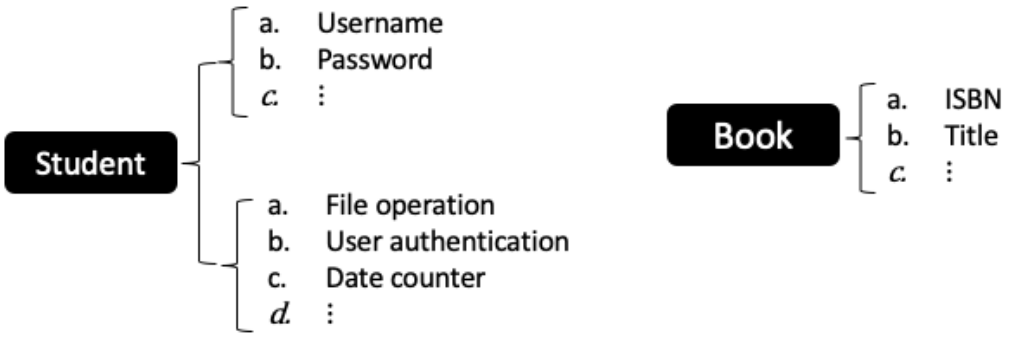

Figure 1: The hierarchical relations among the Classes for LMS.

2 Class Definitions

2.1 Classes Overview

Figure 1 shows the three classes that you need to define and implement for the LMS.

In LMS, a **User** is someone who has the access to the system. Depending on the different needs, a **User** can be someone who uses LMS to borrow or return books, which is a **Reader**, or someone who provides the access and information to the **Reader**s, which is a **Librarian/Administrator**. Most LMS gives more priorities and resources to the **Teacher** users, they might be able to borrow more books or keep a book for longer time than general **Student** users. Most LMS gives more priorities and resources to the **Teacher** users, they might be able to borrow more books or keep a book for longer time than general **Student** users.

When a **User** is borrowing a book, you are actually borrowing a copy of a **Book**. A **Book** contains several copies.

2.2 User

User is a base class used to represent all the users of the system. Either a **Reader** or a **Librarian** can be called as a **User**. So, User class should have those common member variables and functions of all kinds of users. Specifically, users' username and password should be included.

2.3 Reader

Reader is a derived class of **User** class[†]. Each object of this class represents a reader of the system. A reader can perform operations such as borrowing, returning and reserving a book. The Reader class should keep information such as:

- Username.
- Password.
- Maximum number of copies that a reader is allowed to keep. This number might be changed because of penalties due to late returns.
- Maximum days that a reader is allowed to keep.
- List of copies borrowed by the reader.
- List of books reserved by the reader.
- Number of penalties, late-return copies will lead to penalty.

Reader class can be divided into two types: **Student** and **Teacher**.

[†]If you don't understand what is a *derived class* that is okay. We will cover this later in the lectures.

2.4 Student

Student class should have the following attributes:

- Username.
- Password.
- Maximum number of copies that a student is allowed to keep. A student can keep ≤ 5 copies. This number might be changed because of penalties due to late returns.
- Maximum borrowing periods. A student can keep ≤ 30 days for each copy.
- List of copies borrowed by the student.
- List of books reserved by the student.
- Number of penalties, late-return will lead to penalty.

2.5 Teacher

Teacher class should have the following attributes:

- Username.
- Password.
- Maximum number of copies that a teacher is allowed to keep. A teacher can keep ≤ 10 copies. This number might be changed because of penalties due to late returns.
- Maximum borrowing periods. A teacher can keep ≤ 50 days for each copy.
- List of copies borrowed by the teacher.
- List of books reserved by the teacher.
- Number of penalties, late-return will lead to penalty.

2.6 Librarian

Librarian is the other derived class of **User** class. Each object of this class represents a librarian of the library. A librarian can perform more operations such as adding and deleting books or users.

Librarian class should have the following attributes:

- Username.
- Password.

2.7 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.
- Index. This is the unique index for each copy.
- Count. How many available copies are there.
- Favor. How many people like this book.
- ~~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.~~

3 Copy

A copy of a book has the following info:

- ID.
- Book. Which book this copy belongs to.
- A reader.
- A reserver.
- Available.
- Borrowed date.
- Reserve date.
- Expiration date.

4 Functions

To make LMS works successfully, the following functions need to be implemented.

4.1 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.

Requirements:

- Operator overloading. You're required to implement overloading of the stream operator $\ll$ and $\gg$ 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;
...
```

- You're recommended to use function template here, because given the filename, the loading and saving are the same with the help of overloading.

4.2 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. According to the user's type, your system should provide different menus. If access successfully, a menu like Figure 2 will be shown to the user.

Requirements:

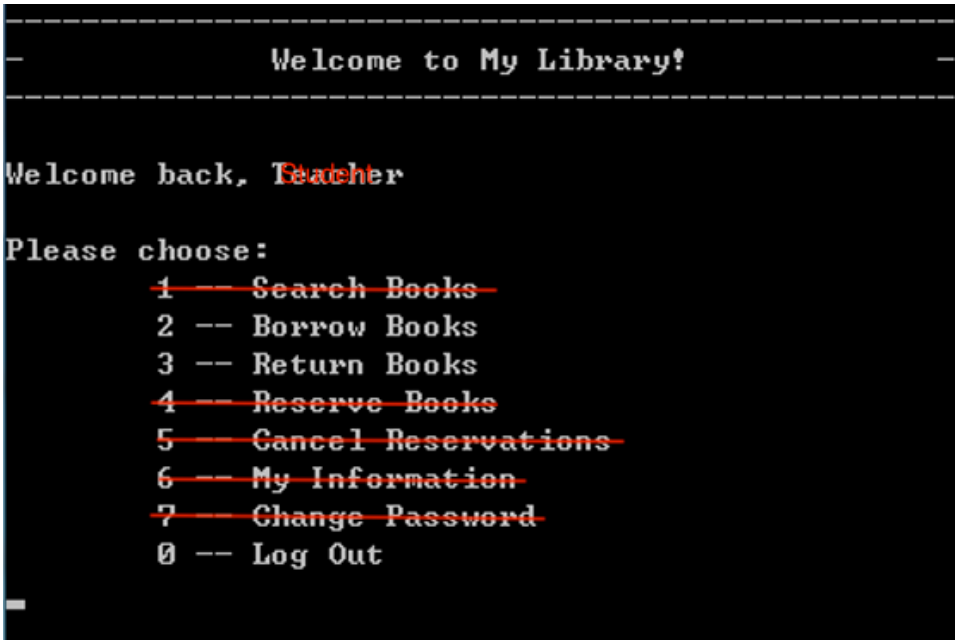

Figure 2: Reader Menu.

- Password Mask: $*$ instead of actual character of the passwords should be shown on the screen. (bonus)

4.3 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.
- ~~Each time you access the system, the date should be counted continuously from the ending date of your last access.~~ 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.

4.4 Student Operations

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

Notes:

- The "book" here means a **Copy** of books except in Section 4.4.1 and 4.4.7 where it means a **Book**.

4.4.1 Search Books

A user is allowed to search for books by keys, including: ISBN, title, author's name and category. The output should include all the basic information of the searching result such as ISBN, title, author's name and category. The IDs of available copies of the books have to be listed.

Requirements:

- If several books share the same author or belong to the same category (these search methods are not one-one mapping), the system should display them all. In this case, the searching

results should be sorted by their popularity. The popularity is measured by the number of reservations. Larger number of reservations means higher popularity.

4.4.2 Borrow Books

A student is able to 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 cannot borrow new copies. The system shows a reminder message to the student if he has any.
- A student cannot borrow more than the maximum.
- A student cannot borrow a copy which is reserved by others. Only if the other reservees have cancelled their reservations, or he/she is the first to reserve this copy (depending on the order of the reservee list), he/she can borrow it.
- If several copies of a book is available, the first reservee can borrow any of them.
- A reader cannot borrow a copy that has been lent to others.
- The real maximum borrowing period is determined by the popularity of the book. For example, if no one reserve the book, a student can keep a copy for 30 days. With the number of the current reservees increases by 20, the maximum borrowing period decreases by 1 day.

4.4.3 Return Books

The borrowed copy becomes available to others again once it is returned. The copy should be removed from this student's list of borrowed copies.

Requirements:

- Although the student returns the overdue copies, the number of his penalty should increase by one. If the number of penalty exceeds 5, the maximum number of copies he can borrow will be reduced by one for every 5 penalties.

4.4.4 Reserve Books

In LMS, a reader needs to make a reservation for a book if no available copies left. The reservation is completed once the reader is added to the reservation list of the book.

Requirements:

- If a reader has overdue copies, he cannot reserve any books. The system has to show a reminder message for that.
- The borrowing order is *first come first serve*.
- If the first reservee does not borrow the book within 5 days when there are available copies, his reservation is cancelled automatically.

4.4.5 Cancel Reservations

When a reader cancels his/she reservation, he/she is removed from the book's reservee list. Meanwhile, the book should be removed from the reader's reservation list.

4.4.6 Renew Books

Renewal can happen anytime before the borrowing period ends if there is no reservation from other readers. A reader can re-borrow the book if the borrowing period ends. A reader cannot renew a book when there are readers in the book's reservee list.

Requirements:

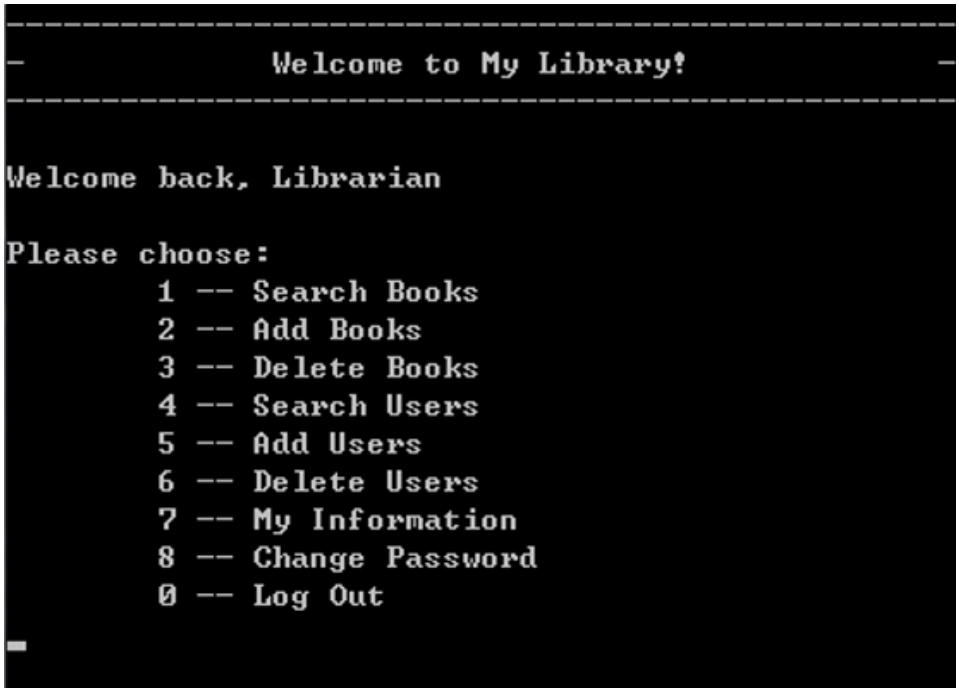

Figure 3: Librarian Menu.

- Only if the book's reservee list is empty can the reader makes the renewal.
- A reader cannot reserve a book if he/she is keeping it.
- A reader cannot execute the renewal operation if he has copies overdue, including the one he wants to renew.

4.4.7 Recommend Books

(bonus) LMS is able to recommend books to readers. If the reader's list of borrowed copies is not empty, then according to the category of the last copy he borrowed, LMS recommends the 10 most popular books with the same category. If the list is empty, the 10 most popular books should be recommended randomly.

4.4.8 Other Notes

Each class should have accessors and mutators (the *get()* and *set()* functions you used in labs), including the username, passwords, etc.

4.5 Librarian Operations

The operations a **Librarian** can perform are shown in Figure 3. Notes:

- The "book" here also means a copy of books, except in Section 4.5.1.

4.5.1 Search Books

It's the same as the one for readers.

4.5.2 Add New Books (Copies)

A librarian has to add new copies to the library, including the information of the new copy, (ISBN, book's title, author's name, category). Then the system should assign a new ID to this new copy.

Requirements:

- If there are no other copies of the same book as the added one, the system creates a new object of this book.

4.5.3 Delete Old Books

When a copy is accidentally broken or for other reasons, a librarian can delete it from the library by the ID.

Requirements:

- If the copy is lent out, it cannot be deleted;
- If the copy is deleted, it should be also removed from the readers' reservation list;
- If the target is the last copy of the book, the system also deletes the book.

4.5.4 Search Users

A librarian can search users by their usernames. All the information of the searching results should be displayed. Requirements:

- If the target is a librarian, show his username and password only.
- If the target is a reader, show his/her name, password, reader type as well as the copies he/she is keeping currently.

4.5.5 Add New Users

The librarian has the access to create accounts for new users. After choosing the type of the new user, he/she only needs the username and password of this user.

Requirements:

- The name of the new user should be different from those of existing users. The system shows a reminder message if this happens.

4.5.6 Delete Old Users

A librarian can delete old users from the system by providing their names.

Requirements:

- If the reader keeps any book copies, he can't be deleted.
- After deletion, the reader needs to be removed from the corresponding books' reservee list.

4.5.7 Other Notes

A **Librarian** class should have accessors and mutators (the *get()* and *set()* functions you used in labs), including the username, passwords, etc.

5 Submission & Grading

5.1 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.

5.1.1 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.

5.1.2 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.
- Usage of advanced concepts, such as template, overloading.
- 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).

5.2 Grading

The grading scale of the first submission is:

- Required part (80 points)
  - Implementation of all classes (20 points)
  - File operation (10 points)
  - User authentication (10 points)
  - Date counter (10 points)
  - Student operations (20 points)
  - Librarian operations (25 points)
- Report (30 points)
- Bonus (10 points)
  - Concise and modular coding style (10 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.
  - Usage of/ template (5 points).
  - Extra functions (5 points/each).
    - Recommend books.
    - Adjust ending date according to book's popularity.