Kelvin Watson
OSU ID 932540242

# FINAL PROJECT REPORT

## PROJECT COMPONENT (Library Simulator)

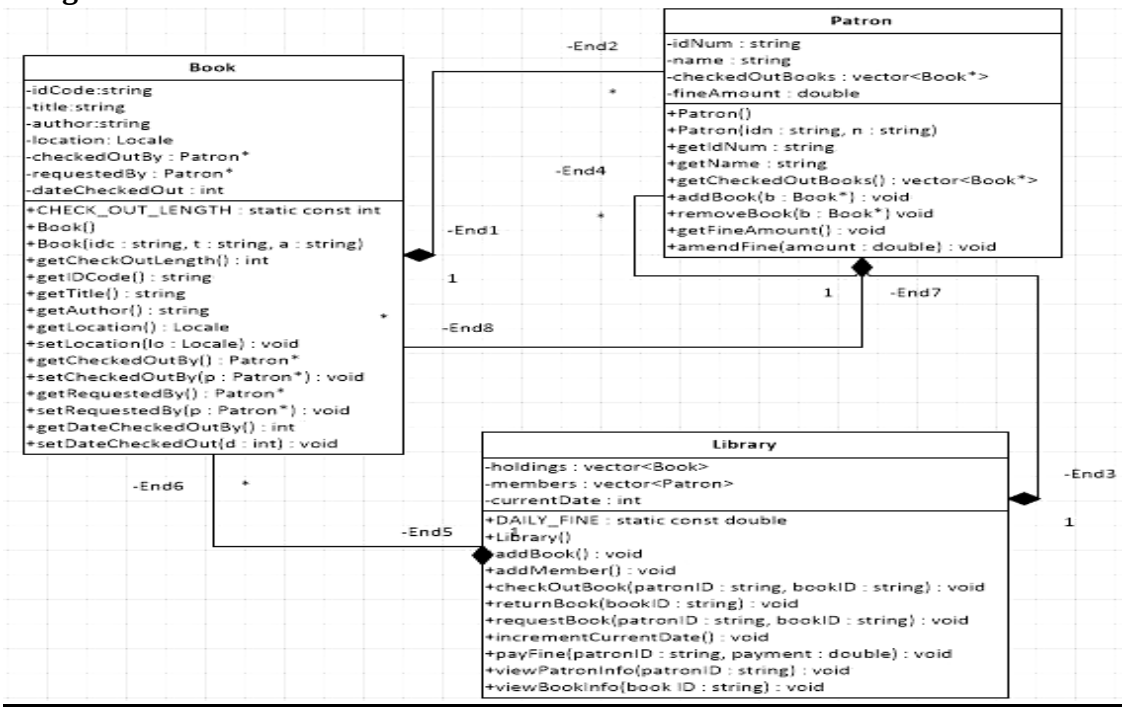
**Components: Book.h, Book.cpp, Patron.h, Patron.cpp, Library.h, Library.cpp, Menu.cpp**

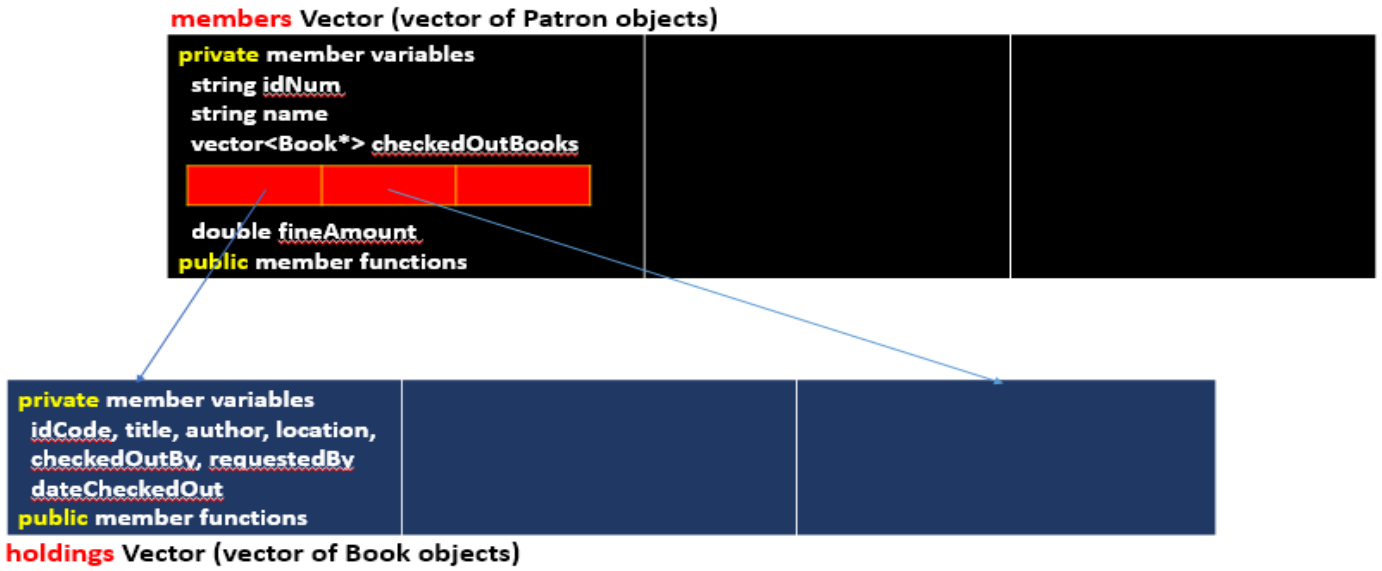
### Understanding

In this final project, I am being asked to employ classes, vectors, and pointers build a library simulator. More specifically, I am being asked to:

- make use of 3 classes: Book, Patron, Library
  - Because all three classes have private member variables, I will need to build public accessor and mutator functions to access the variables.
- The Patron class has private vector of Book pointers, so I will need to assign pointers to point to the vector containing Book objects, and ensure that I set the pointers to NULL when the Patron has returned the books to the library.
- allows the user to select from 10 functionalities:
  - add a Book object to the holdings vector
  - add a Patron object to the members vector
  - check out a book, which would entail having a pointer from the checkedOutBooks vector point to the Book object, setting the location of the Book object to CHECKED_OUT, setting the currentDate, and prohibiting another patron from checking out the book once it has been checked out
  - .return a book, which would entail erasing a vector element from the checkedOutBooks vector, setting the pointer within that vector to NULL, setting the location of the Book object to ON_SHELF
  - request a book, which would entail setting the location of the Book object to ON_HOLD if the book is on the shelf or checked out, and prohibiting the book to be requested by another patron once requested.
  - allow a patron to pay fines, which would involve using the static constant DAILY_FINE, multiplying it with the number of days overdue, and amending the fine accordingly
  - increment the date to "fast-forward" to the point when a book would potentially be overdue.
  - view Patron information
  - view Book information
  - exit the program
- apply the concepts of classes, vectors and pointers

### Design






**Input:** The user enters one of 10 commands at a time by entering an integer between 1 and 10. For the addBook function, the user enters a unique book ID code, a title and an author. For the addMember function, the user enters a unique patron ID and a name. For the checkOutBook and requestBook functions, the user enters a book ID and patron ID for the checkout. For the returnBook function, the user enters a book ID. For the payFine function, the user enters the patron ID and an amount (double) to input the fine payment for late fees.

**Processing**: Processing is done by several functions.

1. In the addBook, the user-inputted strings (book ID. title, author) are used to call a constructor to create an instance of the Book object, which is passed into the function addBook, where the object is added to the Library class' holdings vector.
2. In the addMember, the user-inputted strings (patron ID and name) are used to call the function addMember, where a local instance of the Patron object is created and pushed back to the Library's members vector.
3. In the checkedOutBook function, the patron ID and book ID are passed in as arguments, which are used to check if a patron ID and book ID are a match to ones in the library. If the book and patron exists in the holdings and members vectors respectively, the book's location (of the Locale enumerated data type) determines whether the book can be checked out or not. If the book is ON_SHELF, then it can be checked out. If the book is ON_HOLD by another patron or already checked out, it cannot be checked out.
4. In the returnBook function, if there is a match between the book ID and an element of the holdings vector, then whether the book can be returned is determined based on the location of the book. If the book is already ON_SHELF, it cannot be returned. If the book is CHECKED_OUT, then it can be returned. If the book has been requested by a patron while checked out, when it is returned, the book is immediately placed ON_HOLD for the requester.
5. In the requestBook, patron ID and book IDs are passed as arguments. If there is a match for the patron ID and book IDs in the member and holdings vectors, then the book can be placed ON_HOLD as long as it is not already ON_HOLD for another patron.
6. The incrementCurrentDate function does not take in arguments, but passes the DAILY_FINE to the amendFine function to calculate the fineAmount for the patron.
7. In the payFine function, the user-inputted patron ID and payment amounts are used to update the user's late fees.
8. Each of the Library, Book and Patron implementation files contain various accessor and mutator functions to access the private member variables.

**Output**: The program outputs the menu after each command/functionality has finished executing. It also outputs information about specific Patrons and Books if the viewPatronInfo and viewBookInfo functions are called.

**Implementation** See header and implementation files Book.h, Book.cpp, Patron.h, Patron.cpp, Library.h, Library.cpp, Menu.cpp

**See next page for Testing and Reflection**

**Testing** I will test various inputs in a variety of sequences to try and break the program. Below are some examples of possible scenarios:

| Input | Expected Output | Actual Output | Discussion |
|---|---|---|---|
| request a book that has already been requested by someone else | `Unable to reserve as gone with wind has already been requested by patron1` | `Unable to reserve as gone with wind has already been requested by patron1` | Output as expected |
| return a book that has never been checked out | `Cannot return Gone with the Wind - this book is on the library shelf.` | `Cannot return Gone with the Wind - this book is on the library shelf.` | Output as expected |
| request a book that is already checked out | `Gone with the Wind has been placed on hold for patron2.` | `Gone with the Wind has been placed on hold for patron2.` | Output as expected |
| request a book that is on the shelf | `Gone with the Wind has been placed on hold for patron1` | `Gone with the Wind has been placed on hold for patron1` | Output as expected |
| check out a book that is on hold | `Sorry, Gone with the Wind is on hold for patron1` | `Sorry, Gone with the Wind is on hold for patron1` | Output as expected |
| payment exceeds fines | `kelvin now has a credit balance of $6.80` | `kelvin now has a credit balance of $6.80` | Credit balance is returned as expected |
| view patron information with multiple books checked out and fines | `You have selected to view patron information.`<br>`ID: K`<br>`Name: Kelvin`<br>`This patron has 2 checked-out book(s):`<br>`Book ID: 1 Title: gone Author: a`<br>`Book ID: 2 Title: hunger Author: a`<br>`Current Fines: $1.60` | `You have selected to view patron information.`<br>`ID: K`<br>`Name: Kelvin`<br>`This patron has 2 checked-out book(s):`<br>`Book ID: 1 Title: gone Author: a`<br>`Book ID: 2 Title: hunger Author: a`<br>`Current Fines: $1.60` | Multiple books displays correctly, along with current fines |
| view book information | `Book ID: 1`<br>`Title: Gone with the Wind`<br>`Author: a`<br>`Location: On Hold`<br>`Requested by: Kelvin (ID: K)` | `Book ID: 1`<br>`Title: Gone with the Wind`<br>`Author: a`<br>`Location: On Hold`<br>`Requested by: Kelvin (ID: K)` | Output as expected |
| returning a book that was placed on hold | `patron3 has now checked out Gone with the Wind` | `patron3 has now checked out Gone with the Wind` | Once returned, the patron who placed book on hold has checked out the book. |

## Reflection

When I first started the project, I struggled with the setup of the 7 files because I kept getting compile errors due to forgetting to include header files, and using the angled brackets instead of the double quotations for the .h files. However, after the files were set up correctly, building the library simulator program became much more efficient than having all of the Class implementation code in one .cpp file. I learned from this experience that a good approach is to implement each function one-at-a-time, and test before moving on to another function. I found this approach more efficient than trying to code and implement everything at once, which could lead to difficult-to-find bugs.

One of the more challenging aspects of the project was figuring out how to correctly use the pointers in the checkedOutBooks vector and ensuring that they were set to NULL when a book was returned to the library.

After repeated testing, I realized that the members and holdings vectors changed their address after each push back. I did not understand this originally until I did some research. Eventually, I narrowed this down to the fact that because the vector had no specified starting size, it must allocate new memory each time it pushed back a new element. Because the address of the vector itself kept changing (due to the new memory allocation), the Book pointers in the checkedOutBooks vector no longer pointed to the correct addresses, leading to segmentation faults that were difficult to pinpoint. I used the vector.reserve function to allocate a minimum amount of memory for the checkedOutBooks, members, and holdings vectors to prevent their addresses from continually changing after each Book or Patron object was pushed back.

Input validation was done by parsing input using C-style strings, then converting the validated input to int or double using atoi or atof respectively.

## References

1. http://www.cplusplus.com/reference/cstdlib/atoi/

2. http://www.cplusplus.com/reference/cstdlib/atof/
3. http://www.cplusplus.com/reference/vector/vector/reserve/
4. http://stackoverflow.com/questions/2447392/does-stdvector-change-its-address-how-to-avoid