CSCI 1300 - Starting Computing
Instructor: Fleming
Homework 8: Part III

**This assignment is a project!

Due Monday, November 5th, by 11:55 pm

**No bonus for project II !

This assignment is due **Monday, November 5th, by 11:55 pm**

- ***All components*** **(Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by Monday, November 5th, by 11:55 pm for your homework to receive points.**

Objectives:

- Practice implementing classes
- Develop proper techniques for object-oriented programming
- Manipulate arrays of objects
- Implement classes which have objects from other classes as data members

Problem Set

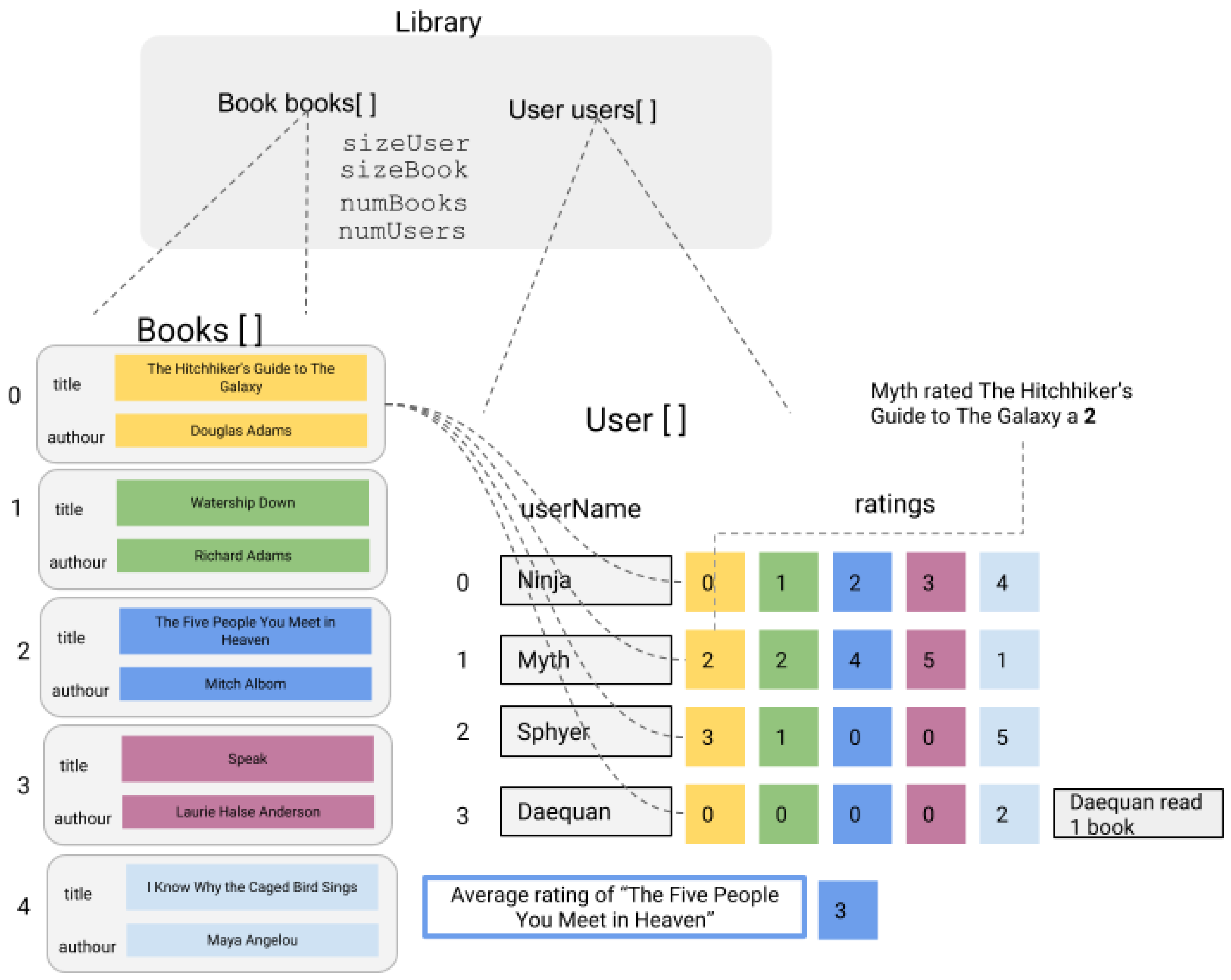
***All the examples and values used in examples are imaginary and randomly generated.*

In part 3 you will streamline your implementation with the addition of a library class. This class will centralize many of the methods you implemented in parts 1 and 2 and provide more functionality, including the ability to recommend books based on the similarity of two users.

Specifications

- Create a new class Library. Define the class in a header file and implement the class in a cpp file.
- The Book and User classes from Homework 7 will be part of Homework 8 as well
- In `main()` create an instance of Library object and a menu as specified
- Students should have seven files (Book.h, Book.cpp, User.h, User.cpp, Library.h, Library.cpp, Hmwk8.cpp)
- The name of each member function should be exactly as specified. If you modify the function names your solution will not pass the autograder.

Visualization of various elements in HW7



Library Class

**You should have separate files for class definition and implementation: Library.h and Library.cpp

Create a class `Library`, with separate interface and implementation, comprised of the following attributes:

Data members (private):	
`Book` array: `books`	An array of `Book` objects
`User` array: `users`	An array of `User` objects
int: `numBooks`	Number of books in the database (library)
int: `numUsers`	Number of users in the database (library)
int: `sizeBook`	The capacity of the `books` array (200). Constant
int: `sizeUser`	The capacity of the `users` array (200). Constant
Member functions (public):	
Default constructor	Sets both `numBooks` and `numUsers` to value 0.
`readBooks(string)`	Takes a string (the name of the file to be read) and populates the `books` array. Returns the total number of books in `books` array as an integer
`readRatings(string)`	Takes a string (the name of the file to be read) and populates the `users` array. Returns the total number of users in `users` array as an integer
`printAllBooks()`	Prints all books stored in `books` array.
`getCountReadBooks(string)`	Takes a string (username) and returns the number of books read by that user as an integer.
`calcAvgRating(string)`	Takes a string (the title of a book) and returns the average rating of the specified book as a double
`addUser(string)`	Takes a string (username) and returns *True* if the username is successfully added to the database. Otherwise, returns *False*.

`checkOutBook(string, string, int)`	Takes two strings and an integer for username, title of book, and a new rating, respectively (in this order). Returns *True* if the rating is successfully updated. Otherwise, returns *False*.
`viewRatings(string)`	Takes a string (username) and prints all the books a user has provided ratings for.
`getRecommendations(string)`	Takes a string username and prints the first 5 book recommendations from the most similar (other) user.

It is advisable to write your own test cases for each class. Test your class in Cloud9 before submitting to the autograder, as the autograder has a **submission limit** of **20 tries every 24 hours**.

Problem 1 - the member function `readBooks`

The member function `readBooks` populates an array of `Book` objects with the title and author data found in a file similar to the file `books.txt`. The array of `Book` objects is one of the data members of the `Library` class. This function should:

- Accept one input argument:
  - string: the name of the file to be read
- Use ifstream, split(), and getline to read and parse data from the file.
- For each line in the file:
  - fill in the `author` and `title` data members for a `Book` object, at the appropriate index in the array of `Book` objects.
- Also:
  - Update the total number of books in the system(from all the files read so far, see the “Important” note below)
- Return the total number of books in the system, as an integer.
- If the file contains more books than “empty” slots available in the `books` array, only write enough books so that you **do not exceed the capacity of the array.**
- If the file cannot be opened, return -1

Important: when testing your `readBooks` function, make sure it supports multiple reads from different .txt files in a row. For example, you should be able to call the function to read the file *books1.txt* **, and then call the function again to read the file**

book2.txt. **The result should be an array of Book objects, with the books from the first file, followed by the books from the second file.**

Problem 2 - the member function `readRatings`

The member function `readRatings` populates an array of `User` objects with the user name and the ratings data found in a file similar to the file `ratings.txt`. The array of `User` objects is one of the data members of the `Library` class. This function should:

- Accept one input argument:
  - string: the name of the file to be read
- Use ifstream, split(), and getline to read and parse data from the file.
- For each line in the file:
  - fill in the `username` and `ratings` data members for a `User` object, at the appropriate index in the array of `User` objects.
- Also:
  - Update the total number of users in the system(from all the files read so far, see the "Important" note below)
- Return the total number of users in the system, as an integer.
- If the file contains more users than "empty" slots available in the `users` array, only write enough books so that you **do not exceed the capacity of the array.**
- If the file cannot be opened, return -1

Important: when testing your `readRatings` function, make sure it supports multiple calls in a row. For example, you should be able to call the function to read the file *ratings1.txt* **, and then call the function again to read the file** *ratings2.txt*. **The result should be an array of User objects, with the users from the first file, followed by the users from the second file.**

Expected output:
cynthia... diane... joan... barbara... (etc.)

Problem 3 - the member function `printAllBooks`

The member function `printAllBooks` prints the list of all the books in the database (library). This function should:

- Accept no arguments
- Not return anything
- If the program has not read any books files **or** any ratings files, print the following message:

```
Database has not been fully initialized
```

- Otherwise, print "`Here is a list of books`", followed by each book in the following format

```
<book title > by <book author>
```

Expected output (assuming you have read the data **only** from `books.txt`)

```
Here is a list of books
The Hitchhiker's Guide To The Galaxy by Douglas Adams
Watership Down by Richard Adams
The Five People You Meet in Heaven by Mitch Albom
Speak by Laurie Halse Anderson
(etc.)
```

Problem 4 - the member function `getCountReadBooks`

The member function `getCountReadBooks` determines how many books a particular user has read and reviewed. This function should:

- Accept one argument:
  - string: username
- Return the number of books read/reviewed by the specified user, as an integer.
- If the program has not read any books files **or** any ratings files, it should return -1 after printing the following message:

```
Database has not been fully initialized
```

- If the database is initialized (at least one file of books and at least one file of ratings has been read), but the user name is not found, return -2 after printing the following message :

```
<username> does not exist in the database
```

Problem 5 - the member function `calcAvgRating`

The member function `calcAvgRating` returns the average (mean) rating for a particular book. This function should:

- Accept one argument:
    - string: book title
- Return the average rating of the specified book as a `double`
- If the program has not read any books files **or** any ratings files, it should return -1 after printing the following message:

```
Database has not been fully initialized
```

- If the database is initialized (at least one file of books and at least one file of ratings has been read), but the book is not found, return -2 after printing the following message:

```
<bookTitle> does not exist in the database
```

*Note: Books that haven't been read **shouldn't** be counted in calculating the average.*

Problem 6 - the member function `addUser`

The member function `addUser` adds a new user to the database. This function should:

- Accept one argument:
    - string: user name
- Fill in the `username` and `ratings` data members for a `User` object, at the appropriate index in the array of `User` objects. :
- Update the total number of users in the system(from all the files read so far, see the "Important" note below)
- The name of the user is case insensitive (e.g. Ben. BEN, ben are all same as ben)
- Return *True* if the user is successfully added.
- If the user already exists in the database, print the following message and return *False*

```
<username> already exists in the database
```

- If the database is full (the array of User objects is full), print the following message and return *False*

```
Database full
```

Problem 7 - the member function `checkOutBook`

The member function `checkOutBook` updates the rating of the book for the user. This function should:

- Accept three arguments in this order
  - string : username
  - string: book title
  - int: new rating
- Find the index of the user and the index for the book, then update the new rating if the new rating value is valid. The rating scheme follows the one provided in homework 6.

Rating	Meaning
0	Did not read
1	Hell No - hate it!!
2	Don’t like it.
3	Meh - neither hot nor cold
4	Liked it!
5	Mind Blown - Loved it!

- If the program has not read any books files **or** any ratings files, it should return *False* after printing the following message:

```
Database has not been fully initialized
```

- If the rating is successfully updated, then returns *True*. Otherwise, it shows one (or more) of the following messages in the given order and return *False*.
  - If the user is not found:

```
<username> does not exist in the database
```

  - If the title is not found:

```
<bookTitle> does not exist in the database
```

  - If the rating value is not valid:

```
<newRatingValue> is not valid
```

Problem 8 - the member function `viewRatings`

The member function `viewRatings` prints all the books a user has provided ratings for. Recall that a rating of 0 means a user has not rated that book and hence shouldn't be displayed. This function should:

- Accept one input argument:
  - string: username
- Not return anything.
- If the program has not read any books files **or** any ratings files, print the following message:

  ```
  Database has not been fully initialized
  ```

- If the user is not found in the database, print:

  ```
  <username> does not exist in the database
  ```

- If the user is found in the database, but has not rated any books, print:

  ```
  <username> has not rated any books yet
  ```

- If the user exists in the database, and it has rated at least one book, display the user's ratings in the following format:

Expected output (assuming you have read the data **only** from `books.txt,ratings.txt`)

```
Here are the books that megan rated
Title : The Hitchhiker's Guide To The Galaxy
Rating : 5
-----
Title : The Five People You Meet in Heaven
Rating : 2
-----
(...)
```

Problem 9 - the member function `getRecommendations`

The member function `getRecommendations` will recommend book titles a user might enjoy, based on the user's similarity of ratings with another user. This function should:

- Accept one input argument:
  - string: username

- Not return anything.
- If the program has not read any books files **or** any ratings files, it should print the following message:

  ```
  Database has not been fully initialized
  ```

- If the database is initialized (at least one file of books and at least one file of ratings has been read), but the user name is not found, it should print the following message :

  ```
  <username> does not exist in the database
  ```

To generate recommendations for a user, for example the user named “ben”:

1. Find the most similar user to “ben”. Let’s say we found “claire” to be most similar.
2. Recommend to “ben” the first 5 books in the database “claire” has rated with a rating of 3, 4 or 5, that “ben” has not yet read (rating 0)
   - If there are less than 5 books to recommend, recommend as many as possible. “ben” will be presented with 0 to 5 recommendations.

To find out who the most similar user is to “ben”, we need to compare “ben”’s ratings with the ratings from all the other users. We need to look at differences in the rating values between the two users for a certain book, add them up, and come up with a *similarity value*. The user who has the smallest *similarity value*, will be the user most similar to “ben”.
Note 1: A new user, who has not rated any books, cannot be chosen as the most similar user. We recommend you use the getCountReadBooks function to weed out the new users.
Note 2: You can assume there will be no ties between two users.

The similarity metric you should use is the **sum of squared differences (SSD)**.
The **sum of squared differences** is calculated by summing the squares of the differences between corresponding elements in two ratings arrays from two users. Follow the example below.

Let *A* represent ben's ratings, and *B* represent claire's ratings.
A_i is ben's rating for book *i*, and B_i is claire's rating for book *i*

$$SSD = \sum_i (A_i - B_i)^2$$

For example:

$john's\ ratings:\ [0,\ 1,\ 3,\ 5]$

$claire's\ ratings:\ [3,\ 0,\ 5,\ 0]$

$SSD = (0 - 3)^2 + (1 - 0)^2 + (3 - 5)^2 + (5 - 0)^2$

$SSD = (-3)^2 + (1)^2 + (-2)^2 + (5)^2$

$SSD = 9 + 1 + 4 + 25 = 39$

Users with very different ratings will get a high SSD.

$john's\ ratings:\ [5,\ 1,\ 0,\ 0,\ 5]$

$david's\ ratings:\ [1,\ 5,\ 0,\ 5,\ 1]$

$SSD = (5 - 1)^2 + (1 - 5)^2 + (5 - 0)^2 + (5 - 1)^2$

$SSD = 4^2 + 4^2 + 5^2 + 4^2$

$SSD = 16 + 16 + 25 + 16 = 73$

Two users with very similar ratings will get a low SSD.

$john's\ ratings:\ [5,\ 0,\ 5,\ 3]$

$claire's\ ratings:\ [5,\ 0,\ 4,\ 2]$

$SSD = (5 - 5)^2 + (5 - 4)^2 + (3 - 2)^2$

$SSD = 0^2 + 1^2 + 1^2$

$SSD = 0 + 1 + 1 = 2$

For example (this example is different than the data in ratings.txt):
Let's say we're generating recommendations for John. Here are the books:

```
Douglas Adams,The Hitchhiker's Guide To The Galaxy
Richard Adams,Watership Down
Mitch Albom,The Five People You Meet in Heaven
Laurie Halse Anderson,Speak
```

```
Liz:   [5, 0, 5, 3]
John:  [5, 0, 3, 0]
David: [4, 1, 0, 5]
```

To generate recommendations for John:

1. **find the most similar user**
   John has a SSD of 13 with Liz and an SSD of 36 with David, so John is more similar to Liz.

2. **find 10 books Liz (the most similar user) has rated as a 3, 4, or 5 that John has not yet read (rating 0)**
   We look at Liz's list to find books that she's rated that John hasn't rated yet:
   - ❏ Liz has rated `The Hitchhiker's Guide To The Galaxy` as a 4, but John has already rated this book.
   - ❏ Liz has rated `Watership Down as 1,` but the rating is too low.
   - ❏ Liz hasn't rated `The Five People You Meet in Heaven`
   - ❏ Liz has rated `Speak` as a 5. John hasn't read that book yet, so we add it to the list of recommendations.

There are no more books that Liz has rated, so we're done. Our final list of recommendations will be:

```
speak by laurie halse anderson
```

Here are examples of output in different situations:

- If there are no books to recommend for a certain user, print the following:

  ```
  There are no recommendations for <username> at the present
  ```

- If there is at least one book to recommend for a certain user, print the following:

```
Here are the list of recommendations:
<book_title_1> by <author1>
<book_title_2> by <author2>
…
…
<book_title_5> by <author5>
```

Driver function

In addition to the menu functionality used in the Homework 6 and 7, we need to add 4 more options. You can find the updated displayMenu function in Hmwk8.cpp on Moodle.

Menu options:

1. Read book file
    - Prompt the user for a file name
    - Pass the file name to your `readBooks` function.
    - After the function returns, print the total number of books in the database:

      ```
      Total books in the database: <numberOfBooks>
      ```

    - If no books are saved to the database due to wrong file name, then print the following message:

      ```
      No books saved to the database
      ```

2. Read user file
    - Prompt the user for a file name.
    - Pass the file name to your `readRatings` function
    - After the function returns, print the total number of users in the database:

      ```
      Total users in the database: <numberOfUsers>
      ```

    - If no books are saved to the database due to wrong file name, then print the following message:

      ```
      No users saved to the database
      ```

3. Print book list
    - Call your `printAllBooks` function.

4. Find number of books user rated
    - Prompt the user for a username.
    - Pass the username to your `getCountReadBooks` function
    - If the user exists in the system, print the result in the following format:

```
<name> rated <numBookRead> books
```

5. Get average rating
   - Prompt the user for a title.
   - Pass the title to your `calcAvgRating` function
   - If the title exists in the database, print the result in the following format:

     ```
     The average rating for <bookTitle> is <value>
     ```

   `Note: <value>` is a double with 2 decimal points.

6. Make a new account
   - Prompt the user for a username:
     ```
     Enter a username:
     ```
   - Pass the username to your `addUser` function
   - Based on the return value from the `addUser` function, you need to print a message. If the user has successfully been added to the database, print:

     ```
     Welcome to the library <user name>
     ```

   - If the user was not added to the database, print:

     ```
     <user name> could not be added in the database
     ```

7. Check Out Book
   - Prompt the user for the username, the title of a book, and the new rating
     ```
     Enter a username:

     Enter a book title:

     Enter a rating for the book:
     ```
   - Pass the username, the title, and the new rating to your `checkOutBook` function
   - Based on the return value from the `checkOutBook` function, you need to print a message. If the book has successfully been read and rated, print:

   ```
   We hope you enjoyed your book. The rating has been updated
   ```

   - If the book has not successfully been read and rated, print:

```
<user_name> could not check out <book_title>
```

8. View Ratings
    - Prompt the user for a username
      ```
      Enter a username:
      ```
    - Pass the username and the title to your `viewRating` function
9. Get recommendations
    - Prompt the user for a username
      ```
      Enter a username:
      ```
    - Pass the username to your `getRecommendations` function

10. Quit