Homework 7 : Part II

Homework 7 is a project that counts for 10% of your overall grade.

This homework has two parts, Part II is **due Sunday March 25th, by 6pm**

Points for Homework 7

- 15 points for Part I **due Sunday March 18th, by 6pm**
- 45 points for Part II **due Sunday March 25th, by 6pm**
- 40 points for *Interview Grading* **week of April 2nd**

Notes on Interview Grading:

We will be conducting interview grading the week after spring break (the week of April 2nd).

If you do not attend an interview grading session, you will automatically receive a 0 on this homework.

Any cancellations must be made at least 24 hours in advance of your scheduled time. You may reschedule *once* the whole semester for a missed interview grade (across the three interview grades) with a **25 point penalty** for the reschedule.

- All components for Part II (Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by **Sunday March 25th, by 6pm** for your solution to receive points.
- **Develop in Cloud9:** For this assignment, write and test your solution using Cloud9.
- **Submission:** All three steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.
    1. Make sure your Cloud9 workspace is shared with your TA: Your recitation TA will review your code by going to your Cloud9 workspace. TAs will check the last version that was saved before the submission deadline.
        - Create a directory called *Hmwk7* and place all your file(s) for this assignment in this directory.

2. **Submit to the Moodle Autograder:** Head over to Moodle to the link *Hmwk 7 Part II*. You will find one programming quiz question for each problem in the assignment. You can modify your code and re-submit (press Check again) as many times as you need to, up until the assignment due date.
3. **Submit a zip file to Moodle:** After you have completed all the questions and checked them on Moodle, you must submit a zip file with the .cpp file(s) you wrote in Cloud9. Submit this file going to *Hmwk 7 Part II(File Submission)* on moodle.

You are not required to write test cases for this assignment.

Book Recommender System

Write a *Library* class such that *libraryDriver.cpp* can make use of functions in the *Library* class to create a book recommendation system.

The *Library* class should have the following two private data members:

- an array of *Book* objects *books*
- an array of *User* objects *users*

The *Library* class should have member functions. It is up to you to declare and define the member functions for the *Library* class, in order to achieve the features below.

The *libraryDriver,* essentially the main program using all the 3 classes we just created will need to call appropriate functions of the 3 classes to provide the following features:

1. Load book and user data from text files

Load book data and user data from text files. A user must be created for every user in the **ratings.txt** file, and a book must be created for every book in the **books.txt**. The data loading should happen at the beginning of the driver program, *libraryDriver.cpp.*

The input files will always be named **books.txt** and **ratings.txt**. (i.e., You can hardcode these names into your program.) Load your book data and user data from these files.

Format of books.txt

Every non-empty line of **books.txt** will have the following format:

```
<Author>,<Title>
```

For example:

```
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
Maya Angelou,I Know Why the Caged Bird Sings
```

Book data should be case insensitive. Convert all titles and author names to lowercase before assigning them to the member data for the Book objects. For example:

```
Douglas Adams,The Hitchhiker's Guide To The Galaxy
```

should become

```
douglas adams,the hitchhiker's guide to the galaxy
```

Format of ratings.txt

Every line of **ratings.txt** will have the following format: a row with the user's name, followed by a comma, followed by this user's ratings for all books.

Username and ratings are separated by comma, and each rating is separated by a single space.

```
<name>,<book_1_rating> <book_2_rating> <book_3_rating> ...
```

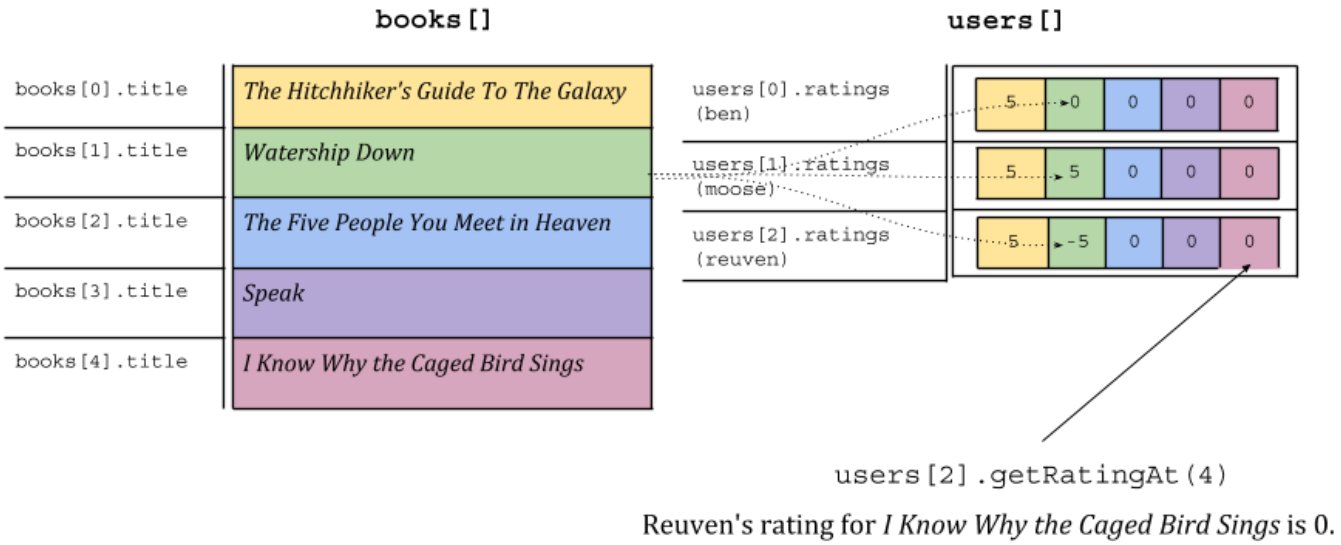
The number of ratings a user has will always be equal to the number of books. For example, if there were 5 books in **books.txt**, every user will have 5 ratings in **ratings.txt**.

Convert names to lowercase before creating the *User* object. For example, "Ben" should become "ben".

For example: The ratings.txt will look like:

```
Ben,5 0 0 0 0
Moose Potter,5 5 0 0 0
Reuven ravenclaw,5 -5 0 0 0
```

The i^{th} rating for each user in **ratings.txt** corresponds to the i^{th} book in **books.txt**. In this example, a user's second rating is always for the *Watership Down.*



If the data is loaded successfully, print a message of acknowledgment:

```
Data Loaded successfully!
```

2a. Login

After the data is loaded, welcome the user to the library and ask them to enter their name (see some examples in the **sampleRuns.txt** file).

(There are no passwords. Anyone can log in as anyone else simply by providing their name.)

If a user has interacted with the system before, the system should remember the user's previous ratings.

```
Data Loaded successfully!
Welcome to the Library, What is your name?:
Ben
Welcome back, Ben
```

Login should be case insensitive. Convert all logins to lowercase before searching for the user. If a user "ben" or "Ben" was in the array of users, all of the following logins should be successful:

- Ben
- BEN
- ben
- bEN

Keep in mind the user might make mistakes. Your program should account for user error at any step. For example, if upon asking the user to entering their name, the user accidentally presses ENTER, your program should recognize the fact that the input is basically “empty” and prompt the user to enter their name again.

```
Data Loaded successfully!
Welcome to the Library, What is your name?:

You provided an empty username, Please provide a valid user name to login or register:
Enter your name again:
AlBus DumbledOrE
Welcome back, AlBus DumbledOrE
```

2b. Add a new user

If it is a user's first time interacting with the system, they should be added to the list of users.

```
Data Loaded successfully!
Welcome to the Library, What is your name?:
Anya
Welcome to the Library, Anya
```

After welcoming the user, present the user with 4 choices:

- 1. View your ratings
- 2. Rate a book
- 3. Get book recommendations
- 4. Quit

The user is asked to choose one of the menu options by entering one character for each option: ‘v’, ‘r’, ‘g’, ‘q’, like in the example below:

```
Data Loaded successfully!
Welcome to the Library, What is your name?:
BeN
Welcome back, BeN
Would you like to (v)iew your ratings, (r)ate a book,
(g)et recommendations, or (q)uit?:
```

Keep in mind the user might make mistakes. Your program should account for user error at this step also. If the user enters something other than ‘v’, ‘r’, ‘g’, ‘q’ (lower or upper case), please prompt the user to enter their choice again.

3: View all your own ratings.

A user should be able to see all the books they have provided ratings for. Recall that a rating of 0 means a user has not rated that book and hence shouldn’t be displayed. Here is an example of how to display the user’s rating values:

```
Here are the books that you have rated:
Title : the hitchhiker's guide to the galaxy
Rating : 5
------------------------------
Title : the sisterhood of the travelling pants
Rating : 1
------------------------------
Title : the da vinci code
Rating : 1
------------------------------
Title : the princess diaries
Rating : -3
------------------------------
Title : ender's game
Rating : 5

...
```

4: Rate a book

A user should be able to rate books. The rating scheme follows the one provided last week.

Rating	Meaning
-5	Hated it!
-3	Didn't like it
0	Haven't read it
1	ok - neither hot nor cold about it
3	Liked it!
5	Really liked it!

If the user chooses to rate a book, the user will be prompted first to enter the title of the book. Keep in mind that we will treat the used input as case insensitive. All of the following user input values should be successful:

- `Ender's Game`
- `ender's game`
- `ENDER'S GAME`

- `Ender's GAME`

If the title of the book already exists in the system, then the user will be asked to provide the rating value. If the title does not exists, prompt the user to enter a new title. Even if the user has already rated a book, they should be able to enter a new rating value. This will result in an update of the ratings array for the user. **Note:** You might want to make a helper function to find the index of the book.

Here is an example of the dialog with the user.

```
Would you like to (v)iew your ratings, (r)ate a book,
(g)et recommendations, or (q)uit?:
r
Enter the name of the book that you would like to
rate:
The Princess Diaries
Enter your rating of the book:
3
Success!
Thank-you. The rating for the book titled "the
princess diaries" has been updated to 3
```

If the book doesn't exist in your Library, or if the user presses ENTER accidentally, then print an error message, like below:

```
Would you like to (v)iew your ratings, (r)ate a book, (g)et
recommendations, or (q)uit?:
r
Enter the name of the book that you would like to rate:
Wings of Fire
Enter your rating of the book:
5
The book you requested does not exist in the database
```

5. See top 10 recommended books

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. Find at most 10 books “claire” has rated with a rating of 3 or 5 that “ben” has not yet read (rating 0)
    - If there are less than 10 books to recommend, recommend as many as possible. Ben will be presented with 0 to 10 recommendations.

For the data currently in **ratings.txt,** there are no books to recommend for cust1:

```
Data Loaded successfully!
Welcome to the Library, What is your name?:
cust1
Welcome back, cust1
Would you like to (v)iew your ratings, (r)ate a book, (g)et
recommendations, or (q)uit?:
g
There are no recommendations for you at present
```

For the data currently in **ratings.txt,** there are five book recommendations for Jim C:

```
Data Loaded successfully!
Welcome to the Library, What is your name?:
Jim C
Welcome back, Jim C
Would you like to (v)iew your ratings, (r)ate a book, (g)et
recommendations, or (q)uit?:
g
Here are some of the books that we think you would like
nineteen eighty-four (1984) by george orwell
harry potter series by j.k. rowling
bone series by jeff smith
the war of the worlds by h g wells
the chrysalids by john wyndham
```

The similarity metric you should use is the **sum of squared differences**.

The **sum of squared differences** is calculated by summing the squares of the differences between corresponding elements in the ratings array.

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.

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

$$SSD = (5 - (-5))^2 + (5 - (-5))^2 + (0 - (-5))^2 + (-5 - 5)^2$$
$$SSD = (10)^2 + (10)^2 + (5)^2 + (-10)^2$$
$$SSD = 100 + 100 + 25 + 100 = 325$$

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

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

$$SSD = (5 - 5)^2 + (0 - 0)^2 + (5 - 3)^2 + (-5 - (-5))^2$$
$$SSD = (0)^2 + (0)^2 + (2)^2 + (0)^2$$
$$SSD = 0 + 0 + 4 + 0 = 4$$

For example (this example is different than the data in ratings.txt):

Let's say we're generating recommendations for John.

```
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: [-5, 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 135 with David, so John is more similar to Liz.

2. **find 10 books Liz (the most similar user) has rated as a 3 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 5, but John has already rated this book.

- ❏ Liz hasn't rated `Watership Down`.
- ❏ Liz has rated `The Five People You Meet in Heaven` as a 5, but John has already rated this book.
- ❏ Liz has rated `Speak` as a 3. 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
```

6. Quit

If a user decides to quit the program, the user information stored in **ratings.txt** should be updated to reflect the new state of the ratings and/or new users.

Note: We recommend you to create a helper function which saves all the data about user and their ratings back in the file from where they were read (in the same format).

The file **sampleRuns.txt** contain sample runs from our program testing most of the features.

Your program should generate the same output given the input files provided on moodle.

The test cases on the autograder might correspond to some of these sample runs.

Submitting to the Autograder:

Because of the current format of moodle, you will need to put all your code (*User, Book*, *Library*, and *libraryDriver*) together in the answer box.

Submitting the assignment:

- 45 points for Part II **due Sunday March 25th, by 6pm**
  - Submit your code to the Autograder on Moodle
  - Zip the files containing your *Book* class, *User* class, Library class, and *libraryDriver.cpp* and upload it to the file submission link. The files can be

organized in any format you would like as long as the code compiles and runs.

Extra Credit

Do not submit your code with the extra credit to the autograder, your TA will evaluate your implementation of the extra credit during the grading interview.

Create a special admin account that can add new books. Your admin will be the very first user in the user file. For example, if your **ratings.txt** file looked like this, **Ben** would be the admin.

```
Ben,5 0 0 0 0 0 0 1 0 1
Moose,5 5 0 0 0 0 3 0 0 1
Reuven,5 -5 0 0 0 0 -3 -5 0 1
Cust1,3 3 5 0 0 0 3 0 0 3
Cust2,3 0 0 0 0 0 0 0 0 0
Francois,3 3 5 0 0 0 3 0 0 3
```

You should add a fifth option to the menu for all users that is "add new book".

Normal users cannot add new books, only admins can, so if a normal user chooses "add a new book" you will display the message: "Sorry, only administrators can add books".

If the admin adds any new books, the new books should be added to the end of the **books.txt** file in the order the admin added them in.

The **users.txt** file will also need to be updated so that each user has the correct number of ratings. For example, for the **ratings.txt** file above, if a new book was added, each user would have another rating.

```
ben,5 0 0 0 0 0 0 1 0 1 0
moose,5 5 0 0 0 0 3 0 0 1 0
reuven,5 -5 0 0 0 0 -3 -5 0 1 0
cust1,3 3 5 0 0 0 3 0 0 3 0
cust2,3 0 0 0 0 0 0 0 0 0 0
francois,3 3 5 0 0 0 3 0 0 3 0
```