# Homework 7 : Part I

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

This homework has two parts, Part I is due **Sunday, March 18th, by 6 pm**

**Points for Homework 7**

- 15 points for Part I **due Sunday March 18th, by 6pm**
    - 10 points for autograder questions on *Book* and *User* classes
    - 5 points for the *libraryDriver.cpp* file
- 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 the 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 I (Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by **Sunday, March 18th 6:00 pm** 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*. 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 (File Submission)* on moodle.

***You are not required to write test cases for this assignment, and points will not be deducted for a lack of comments.*** 😱

*However, we ask that you please still include the comments at the top identifying you and the assignment for the sanity of your favorite TA.* 😊

```
// CS1300 Spring 2018
// Author:
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/…
// Homework 7
```

# Background

Adapted from http://nifty.stanford.edu/2011/craig-book-recommendations/cs1/handout.shtml

If you've ever bought a book online, the bookseller's website probably told you what other books you might like. This is handy for customers, but also very important for business. In September, online movie-rental company Netflix awarded one million dollars to the winners of the Netflix Prize. The competition simply asked for an algorithm that would perform 10% better than their own algorithm. Making good predictions about people's preferences was that important to this company. It is also a very current area of research in machine learning, which is part of the area of computer science called artificial intelligence.

So how might we write a program to make recommendations for books?

Consider a user named Rabia. How is it that the program should predict books Rabia might like?

The simplest approach would be to make almost the same prediction for every customer. In this case the program would simply calculate the average rating for all the books in the database, sort the books by rating and then from that sorted list, suggest the top 5 books that Rabia hasn't already rated. With this simple approach, the only information unique to Rabia used by the prediction algorithm was whether or not Rabia had read a specific book.

We could make a *better* prediction about what Rabia might like by considering her actual ratings in the past and how these ratings compare to the ratings given by other customers. Consider how you decide on movie recommendations from friends. If a friend tells you about a number of movies that s(he) enjoyed and you also enjoyed them, then when your friend recommends another movie that you have never seen, you probably are willing to go see it. On the other hand, if you and a different friend always tend to disagree about movies, you are not likely to go to see a movie this friend recommends.

A program can calculate how similar two users are by treating each of their ratings as a vector and calculating a similarity value based on some calculation (like a dot product, or a sum of squared differences) using the two vectors.

Once you have calculated the pair-wise similarity between Rabia and every other customer, you can then identify whose ratings are most similar to Rabia's. In this case Suelyn is most similar to Rabia, so we would recommend to Rabia the top books from Suelyn's list that Rabia hadn't already rated.

Your task for this project is to write a program that takes people's book ratings and makes book recommendations to them using this technique. We will refer to the people who use the program as "users".

# Problem Set

Questions 1 and 2 will be autograded. To encourage you to practice testing your own classes (which you will need to do for the next project), the autograder questions for these two classes will not be released until **Wednesday, March 14th at 9:00 AM**.

# Question 1 - 5 points

Write a class *Book* that has the following private data members:

- a string *title*
- a string *author*

1. Write getters and setters for each of the data members. They should follow the naming convention getAttribute/setAttribute. For example, *title* would have a getter **getTitle** and a setter **setTitle**.
2. Write a public constructor for the class that takes a *title* and *author* and creates a new book object.
3. Write a default constructor for the class that takes no parameters and creates a new book object with a *title* "NONE" and *author* "NONE".

I should be able to create a new book by doing:

```
Book cs_textbook("Problem Solving with C++ (9th Edition)", "Walter  Savitch");
```

An alternate syntax to construct a new book is:

```
Book cs_textbook =  Book("Problem Solving with C++ (9th Edition)", "Walter  Savitch");
```

# Question 2 - 5 points

Write a class *User* that has the following private data members:

- a string *name* that uniquely identifies a user
- an array *ratings* that can hold 100 integers. Elements in *ratings* must be one of the values in the chart below.

- an integer *numRatings* for the number of ratings in the *ratings* array. This will correspond with the total number of books.

| 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! |

1. Write a public constructor for the class that takes as parameters a *name*, an array of integers *userRatings*, and an integer *numberRatings*, and creates a new user object. Each rating in the *userRatings* array should be copied into the user's *ratings* array starting at *ratings[0]*.

   I should be able to create a new user by doing the following:

```
int user1Ratings[] = {0, 1, 3, -1, 0, -5, -3};
User user1 = User("Vipra", user1Ratings, 7);
```

2. Write a default constructor for the class that takes no parameters and sets a *name* "NONE", an array *ratings* of 0s, and a *numRatings* of 0.
3. Write a getter and setter for *name, getName* and *setName* respectively.
4. Write a getter and setter for *numRatings*, *getNumRatings* and *setNumRatings* respectively.
5. For the data member *ratings* you will need to write your methods a bit differently since it is not possible to return an entire array. Write the following functions:
   - **getRatingAt** which takes a single integer parameter for the index and returns the rating at that index. If the index is greater than or equal to *numRatings* return -1000.
   - **setRatingAt** a function which takes two parameters in this order:
     - the position at which to add the rating
     - the rating

This function should set the rating at the position to the new rating.

If the position is greater than or equal to *numRatings* keep the existing rating and return -1000.

We want to prevent users from entering ratings that aren't -5, -3, 0, 1, 3, or 5. If **setRatingAt** is called with an invalid rating, print "Invalid Input!" and return -1. Otherwise print "Success!" and return 0 to indicate success.

Given the example above, I should be able to get ratings of book at index 2 for user1:

```
cout << user1.getRatingAt(2) << endl;
```

**output**

```
3
```

and set ratings for the user:

```
cout << user1.setRatingAt(3, -5) << endl;
cout << user1.getRatingAt(3) << endl;
cout << user1.setRatingAt(3, -10000) << endl;
cout << user1.getRatingAt(3) << endl;
cout << user1.getRatingAt(20) << endl;
cout << user1.setRatingAt(25, 3) << endl;
```

**output**

```
Success!
0
-5
Invalid Input!
-1
-5
-1000
-1000
```

# Question 3 - 5 points

****You are not writing code for this question, only algorithms**.**

Next week, you will have to write a class *Library* and a driver file *libraryDriver.cpp*.

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

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

Unlike the other two classes, we will not be providing specific data members or methods we want your *Library* class to have. Part II will require you to design a *Library* class. The program *libraryDriver.cpp* will need to make use of functions in the Library class to allow users to interact with a book recommendation system.

The *libraryDriver* program will need to have the following features:

- Load book data and user data from text files. A user must be created for every user in the users file, and a book must be created for every book in the book file. Full details on the format of these files will be provided in Part II.
- Add a new user.
- Login. (There are no passwords. Anyone can log in as anyone else simply by providing their name.)
- Quit. This must automatically save all updated data to your users file.
- View all your own ratings.
- Rate a book.
- See recommended books. (A specific metric to compare users to each other will be provided in Part II).

For **5 points** we would like you to fill the *libraryDriver.cpp* file we have provided with pseudocode (in comments) describing the algorithms you think you will need to implement the above features. You should specifically address *all* of the features outlined above to receive full points.

You will be allowed to change your approach as you move forward into Part II without penalty.

To help you out, here are a few hypothetical runs representing what we want to be possible by the end of Part II. (Don't pay too much attention to the specific wording, as that may change).

These runs all show the default scenario, or the happy paths where nothing goes wrong. You, however, should also consider what will happen when errors occur. For example, what happens if a user accidentally inputs a book rating of 10000?

**Greeting a new user:**

```
vipra_gupta:~/workspace $ ./libraryDriver
Data successfully loaded!
Welcome to the library! What is your name?: Vipra
Welcome to the library, Vipra.
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: q
Data successfully saved. Goodbye!
vipra_gupta:~/workspace $
```

**Greeting an existing user:**

```
vipra_gupta:~/workspace $ ./libraryDriver
Data successfully loaded!
Welcome to the library! What is your name?: Vipra
Welcome back, Vipra.
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: q
Data successfully saved. Goodbye!
vipra_gupta:~/workspace $
```

**Rating a book:**

```
vipra_gupta:~/workspace $ ./libraryDriver
Welcome to the library! What is your name?: Vipra
Welcome back, Vipra.
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: r
Please enter the title of the book: Problem Solving with C++ (9th Edition)
Please enter your rating: 3
Thank-you. The rating for Problem Solving with C++ (9th Edition) by Walter Savitch
has been updated to 3.
```

```
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: q
Data successfully saved. Goodbye!
vipra_gupta:~/workspace $
```

### Getting recommendations:

```
vipra_gupta:~/workspace $ ./libraryDriver
Welcome to the library! What is your name?: Vipra
Welcome back, Vipra.
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: g
Here are your recommendations:
Hitchiker's Guide to the Galaxy by Douglas Adams
Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L.
Rivest, and Clifford Stein
The Hunger Games by Suzanne Collins
The Fault in Our Stars by John Green
The Secret Garden by Frances Hodgson Burnett
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: q
Data successfully saved. Goodbye!
vipra_gupta:~/workspace $
```

### Viewing ratings:

```
vipra_gupta:~/workspace $ ./libraryDriver
Welcome to the library! What is your name?: Vipra
Welcome back, Vipra.
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: v
Problem Solving with C++ (9th Edition) 3
Would you like to (v)iew your ratings, (r)ate a book, (g)et recommendations, or
(q)uit?: q
Data successfully saved. Goodbye!
vipra_gupta:~/workspace $
```

### Submitting the assignment:

- Part I **due Sunday March 18th, by 6pm**
    - Question 1 (Autograder)
    - Question 2 (Autograder)
    - Fill the *libraryDriver.cpp* with pseudocode

- Zip the files containing your *Book* class and *User* class and *libraryDriver.cpp* and upload it to the file submission link

## Extra Credit (included in Part II)

Create a special admin account that can add new books. Your admin will be the very first user in the user file.

You should add another 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 tell them "Sorry, only administrators can add books".