CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 7
Due: Saturday, October 26th, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm Oct. 25th)

2 components (Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, October 26th, 6:00 pm for your homework to receive points.

---

0. Table of Contents

0. Table of Contents
1. Objectives
2. Background
Header and Source files
3. Submission Requirements
4. Rubric
5. Problem Set
    Problem 1 (10 points): User Class
    Problem 2 (30 points): readRatings
    Problem 3 (20 points): getRating
    Problem 4 (20 points): getCountReadBooks
    Problem 5 (20 points): calcAvgRating
    Problem 6 (30 points): put them together
6. Homework 7 checklist
7. Homework 7 points summary

---

1. Objectives

- Use filestream objects to read data from text files
- Create objects
- Array operations: initialization, search
- Improve proficiency with loops and strings
- Get started with part of your Project 2

---

2. Background

Header and Source files

Creating our own classes with various data members and functions increases the complexity of our program. Putting all of the code for our classes as well as the main functionality of our program into one .cpp file can become confusing for you as a programmer, and we need ways of reducing the visual clutter that this creates. This is why, as we increase the complexity of a program, we might need to create multiple files: **header** and **source** files.

Header file

Header files will have “.h” as their filename extensions. In a header file, we *declare* one or more of the complex structures (classes) we want to develop. In a class, we define member functions and member attributes. These functions and attributes are the building blocks of the class.

```
#include <iostream>
using namespace std;

class className
{
     public:
     .
     .
     .
     private:
     .
     .
     .
};
```

Source file

Source files are recognizable by the “.cpp” extension. In a source file we *implement* the complex structures (class) defined in the header file. Since we are splitting the development of the actual code for the class into a definition (header file) and an implementation (source file), we need to link the two somehow.

```
#include “className.h”
```

In the source file we will include the header file that defines the class so that the source file is “aware” of where we can retrieve the definition of the class. We must define the class definition in every source that wants to use our user defined data type (our class). When implementing each member function, our source files must tell the compiler that these functions are actually the methods defined in our class definition using the syntax that we showed earlier.

How to compile multiple .cpp and .h files:
In this homework, it will be necessary to write multiple files (.h and .cpp) and test them before submitting them. You need to compile and execute your code **using the command line**. This means you need to type commands into a **bash** window instead of pressing the *Run* button as you may have been doing.

Make sure that you start by changing directories so that you are in the folder where your solution’s files are stored. In this example, our folder will be called hmwk7. To change to this directory, use:

```
cd hmwk7/
```

When compiling from the command line, you need to specify each of the .cpp files in your project. This means that when you call the g++ compiler, you need to explicitly name the files you’re compiling:

```
g++ -std=c++11 file1.cpp file2.cpp main.cpp
```

The compiling command results in the creation of an executable file. If you did not specify a name for this executable, it will be named a.out by default. To execute this file, use the command:

```
./a.out
```

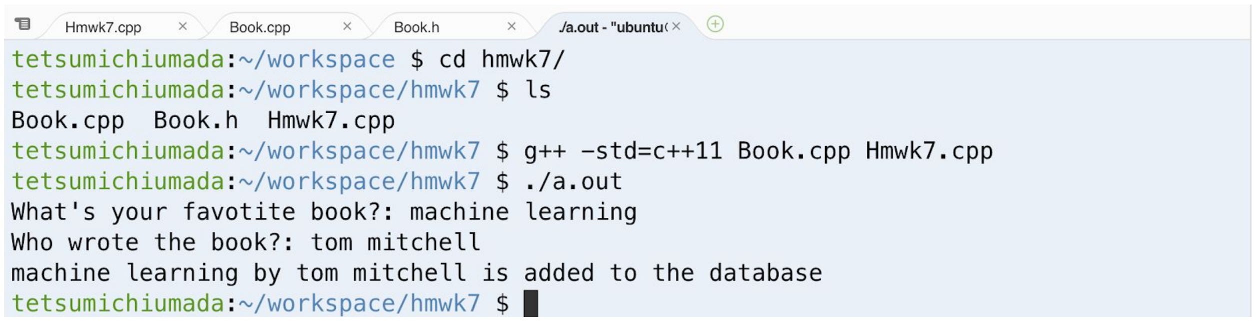
You can add the -o flag to your compiling command to give the output file a name:

```
g++ -o myName.out -std=c++11 file1.cpp file2.cpp main.cpp
```

And then run the file with

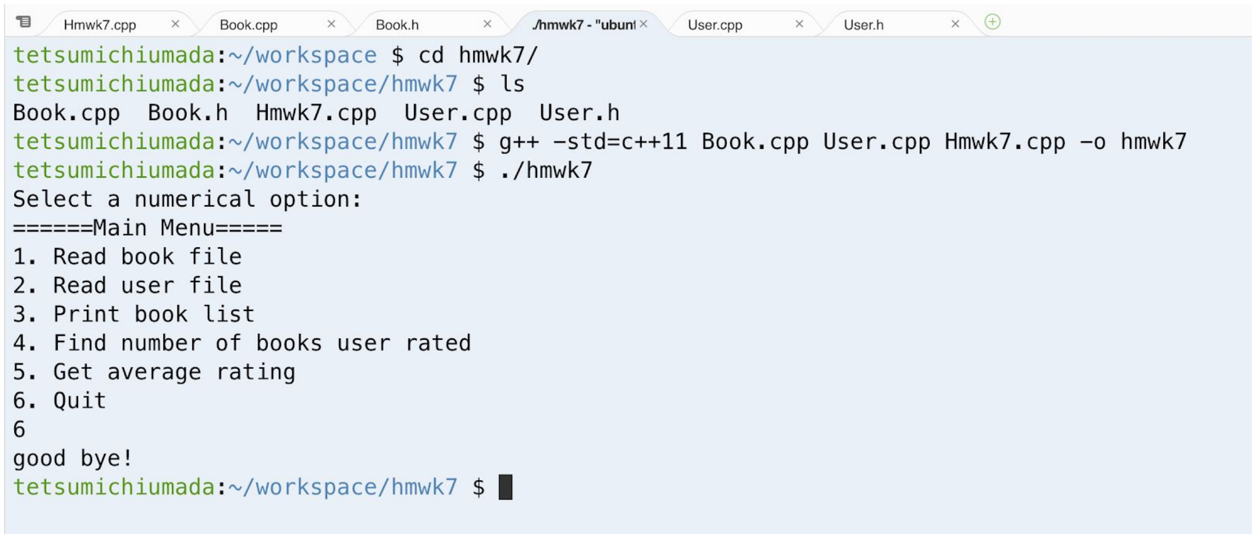
```
./myName.out
```

Example 1:
Compiling book.cpp and Hmwk7.cpp. The picture below shows a **bash** window.



```
tetsumichiumada:~/workspace $ cd hmwk7/
tetsumichiumada:~/workspace/hmwk7 $ ls
Book.cpp  Book.h  Hmwk7.cpp
tetsumichiumada:~/workspace/hmwk7 $ g++ -std=c++11 Book.cpp Hmwk7.cpp
tetsumichiumada:~/workspace/hmwk7 $ ./a.out
What's your favotite book?: machine learning
Who wrote the book?: tom mitchell
machine learning by tom mitchell is added to the database
tetsumichiumada:~/workspace/hmwk7 $
```

Example 2:
Here, we have named the executable hmwk7



```
tetsumichiumada:~/workspace $ cd hmwk7/
tetsumichiumada:~/workspace/hmwk7 $ ls
Book.cpp  Book.h  Hmwk7.cpp  User.cpp  User.h
tetsumichiumada:~/workspace/hmwk7 $ g++ -std=c++11 Book.cpp User.cpp Hmwk7.cpp -o hmwk7
tetsumichiumada:~/workspace/hmwk7 $ ./hmwk7
Select a numerical option:
======Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit
6
good bye!
tetsumichiumada:~/workspace/hmwk7 $
```

3. Submission Requirements

All three steps must be fully completed by the submission deadline for your homework to be graded.

1. ***Work on questions on your Cloud 9 workspace:*** You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **hmwk7** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 7 CodeRunner**. You will find one programming quiz question for each problem in the assignment. Submit your solution for the first problem and press the Check button. You will see a report on how your solution passed the tests, and the resulting score for the first problem. You can modify your code and re-submit (press *Check* again) as many times as you need to, up until the assignment due date. Continue with the rest of the problems.

3. ***Submit a .zip file to Moodle:*** After you have completed all 6 questions from the Moodle assignment, zip all 10 solution files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 7** link on Moodle.

---

4. Rubric

Aside from the points received from the **Homework 7 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 7** link on Moodle and assign points for the following:

Style, Comments, Algorithm (10 points):

Style:

- Your code should be well-styled, and we expect your code to follow some basic guidelines on whitespace, naming variables and indentation, to receive full credit. Please refer to the **CSCI 1300 Style Guide** on Moodle.

Comments:

- Your code should be well-commented. Use comments to explain what you are doing, especially if you have a complex section of code. These comments are intended to help other developers understand how your code works. These comments should begin with two backslashes (`//`) or the multi-line comments (`/* … comments here… */`) .
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Homework 7 - Problem # ...
```

Algorithm:

- Before each function that you define, you should include a comment that describes the inputs and outputs of your function and what algorithms you are using inside the function.
- This is an example C++ solution. Look at the code and the algorithm description for an example of what is expected.

Example 1:

```
/*
 * Algorithm: convert money from U.S. Dollars (USD) to Euros.
 *   1. Take the value of number of dollars involved
 *      in the transaction.
 *   2. Current value of 1 USD is equal to 0.86 euros
 *   3. Multiply the number of dollars got with the
 *      currency exchange rate to get Euros value
 *   4. Return the computed Euro value
 * Input parameters: Amount in USD (double)
 * Output (prints to screen): nothing
 * Returns: Amount in Euros (double)
 */
```

Example 2:

```
double convertUSDtoEuros(double dollars)
{
     double  exchange_rate  =  0.86;  //declaration  of  exchange
rate
     double euros = dollars * exchange_rate; //conversion
     return euros; //return the value in euros
}
```

The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

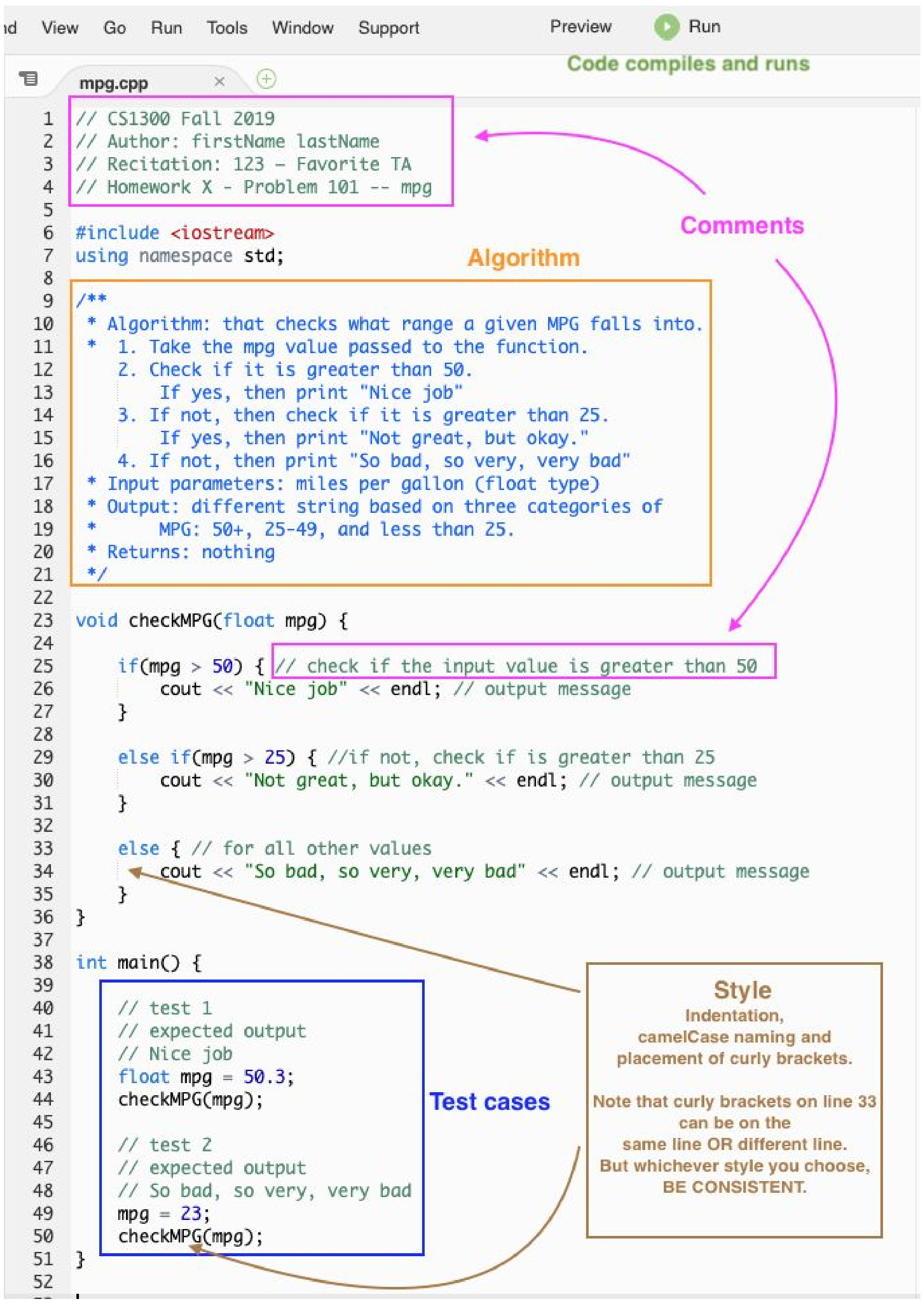
```
/*
 * conversion
 */
```

```
double convertUSDtoEuros(double dollars)
{
      double euros = dollars * 0.86;
      return euros;
}
```

Test Cases (20 points)

1. *Code compiles and runs* (**6 points**):
    - The zip file you submit to Moodle should contain **7** full programs (with a `main()` function), saved as .cpp files. It is important that your programs can be compiled and run on Cloud9 with no errors. The functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**14 points**):
   For this week's homework, all 5 problems are asking you to create a function. In your solution file for each function, you should have at least 2 test cases present in their respective `main()` function, for a total of at least 14 test cases (see the diagram on the next page). Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.



```cpp
1  // CS1300 Fall 2019
2  // Author: firstName lastName
3  // Recitation: 123 – Favorite TA
4  // Homework X - Problem 101 -- mpg
5
6  #include <iostream>
7  using namespace std;
8
9  /**
10  * Algorithm: that checks what range a given MPG falls into.
11  *  1. Take the mpg value passed to the function.
12     2. Check if it is greater than 50.
13         If yes, then print "Nice job"
14     3. If not, then check if it is greater than 25.
15         If yes, then print "Not great, but okay."
16     4. If not, then print "So bad, so very, very bad"
17  * Input parameters: miles per gallon (float type)
18  * Output: different string based on three categories of
19  *      MPG: 50+, 25-49, and less than 25.
20  * Returns: nothing
21  */
22
23 void checkMPG(float mpg) {
24
25     if(mpg > 50) { // check if the input value is greater than 50
26         cout << "Nice job" << endl; // output message
27     }
28
29     else if(mpg > 25) { //if not, check if is greater than 25
30         cout << "Not great, but okay." << endl; // output message
31     }
32
33     else { // for all other values
34         cout << "So bad, so very, very bad" << endl; // output message
35     }
36 }
37
38 int main() {
39
40     // test 1
41     // expected output
42     // Nice job
43     float mpg = 50.3;
44     checkMPG(mpg);
45
46     // test 2
47     // expected output
48     // So bad, so very, very bad
49     mpg = 23;
50     checkMPG(mpg);
51 }
52
```

Style

Indentation, camelCase naming and placement of curly brackets.

Note that curly brackets on line 33 can be on the same line OR different line. But whichever style you choose, BE CONSISTENT.

5. Problem Set

Note: To stay on track for the week, we recommend to finish/make considerable progress on problems 1-3 by Wednesday. Students with recitations on Thursday are encouraged to come to recitation with questions and have made a start on all of the problems.

Problem 1 (10 points): User Class

Create a `User` class, with separate interface (`User.h`) and implementation (`User.cpp`), comprised of the following attributes:

Data members (private):	
string: `username`	
int array: `ratings`	Number of elements should be `size`
int: `numRatings`	Number of books in the database
int: `size`	The capacity of the `ratings` array (50). Constant
Member functions (public):	
Default constructor	Sets `username` to an empty string, `numRatings` to 0, `size` to 50, and all the elements of `ratings` array to the value 0
Parameterized constructor	Takes a string, an array of integers, and one integer for initializing `username`, `ratings`, and `numRatings` respectively. Make sure the value passed into the constructor for `numRatings` does not exceed the value of the data member `size`
`getUsername()`	Returns `username` as a string
`setUsername(string)`	(void) Assigns `username` the value of the input string
`getRatingAt(int)`	Parameter: `int index`. Returns the rating stored at the specified index. If `index` is larger than the size of the `ratings` array, or less than 0, returns -1.

`setRatingAt(int,int)`	Parameters: `int index, int value`. Sets the rating to `value` at the specified `index,` if `index` is within the bounds of the array and `value` is between 0 and 5. Returns a boolean value, `true` if the rating is successfully updated and `false` otherwise.
`getNumRatings()`	Returns `numRatings` as an integer
`setNumRatings(int)`	(void) Assigns `numRatings` the value of the input int
`getSize()`	Returns `size` as an integer

The zip submission should have three files for this problem: **User.h**, **User.cpp**, and a driver called **userDriver.cpp**, with a `main()` function to test your member functions. For **Coderunner**, paste your **User class and its implementation** (the contents of User.h and User.cpp). Do not include the `main()` function.

In your `main()` function, the test cases should include the creation of class objects with both the default and parameterized constructors. You must also test each of the getter and setter member functions by creating and manipulating class objects and displaying output to verify that things are working properly. For reference, follow the `cashRegister` example from lecture 21 materials, and the *Worked Example 9-1* from the textbook (`Implementing a Bank Account Class` - step 6: Test your class).

Problem 2 (30 points): readRatings

We will now update the **readRatings** from Homework 5 to use an array of `User` objects instead of having a `usernames` array and a `ratings` array. The functionality stays the same as the one from Homework 5.

Write a function **readRatings** that loads the user ratings by reading the `ratings.txt` file. The first value of each line in `ratings.txt` is the username. Each username is followed by a list of ratings of the user for each book in `books.txt`.

For example, let us say there are in total of 3 books. The `ratings.txt` file would be of the format:

```
Ritchie,3,3,3
stroustrup,0,4,5
gosling,2,2,3
rossum,5,5,5
...
```

This function should:

- Accept five parameters in this order:
    - **string** `filename`: the name of the file to be read
    - **array** `users`: array of **User** objects
    - **int** `numUsersStored` number of users currently stored in the arrays
    - **int** `usersArrSize`: capacity of the `users` arrays. The default value for this data member is 100.
    - **int** `maxCol:` maximum number of columns. The default value for this data member is 50.
- Use `ifstream` and `getline` to read data from the file, making an instance of a `User` object for each line, and placing it in the `users` array.
- **Hint**: You can use the `split()` - function from Homework 4, with comma (",") as the delimiter. When you copy your code in the Answer Box on Moodle Coderunner, make sure you copy **Users** class, **readBooks()** function, and **split()** function.
- You can use `stoi` to convert each rating value (a string, as read from the text file) into an integer value.
- Empty lines should not be added to the arrays.
- The function should return the following values depending on cases:
    - Case1: When `numUsersStored` is greater than or equal to the `usersArrSize`, return -2.
    - Case2: If the file cannot be opened, return -1.
    - Case3: When `numUsersStored` is smaller than the `size` of `users` array, keep the existing elements in `users` array, then read data from file and add (append) the data to the arrays. The number of users stored in the arrays cannot exceed the size of the `users` array.
    - Case4: Return the total number of users in the system, as an integer.
    - Your function must check these cases in the order specified above.

Example 1: The `users` array is empty, so `numUsersStored` is 0.

ratings.txt	`Ninja,0,1,2,3,4`
	`Myth,2,2,4,5,1`

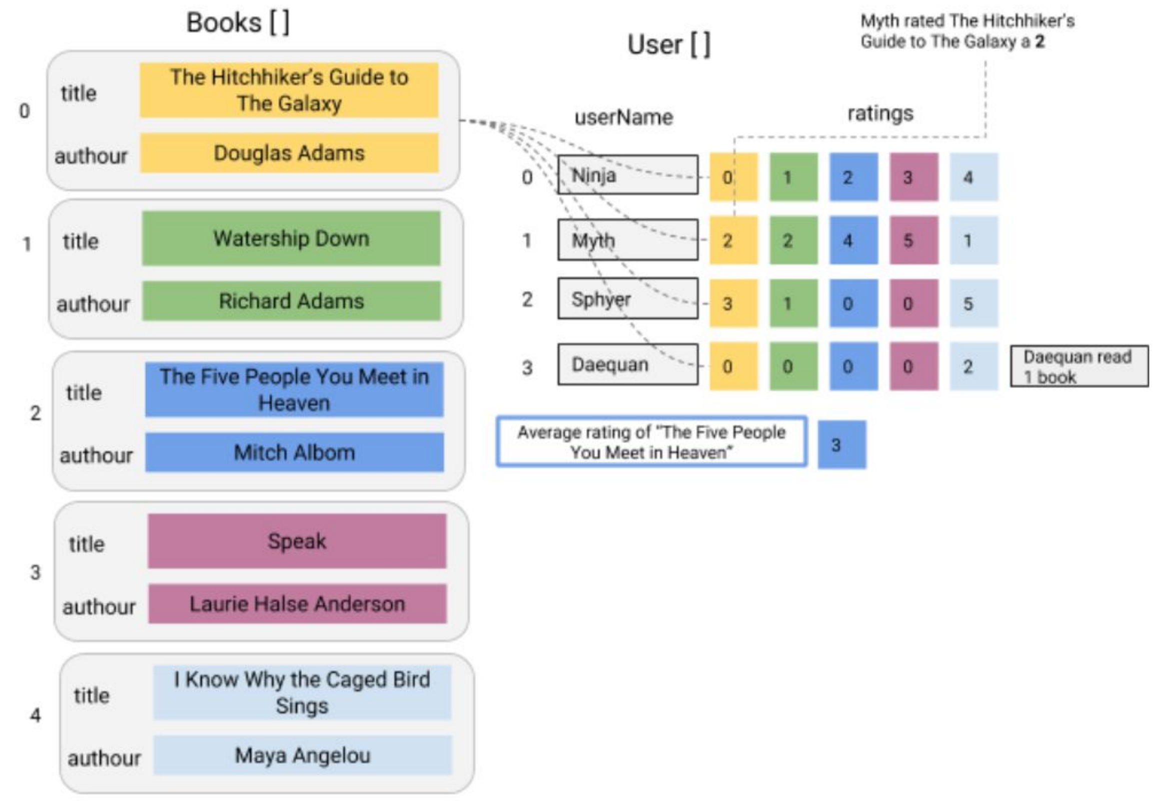
	`Sphyer,3,1,0,0,5`
	`Daequan,0,0,0,0,2`

Function call	`User users[10];`
	`int numUsers = 0;`
	`int usersArrSize = 10;`
	`readRatings(“ratings.txt”, users, numUsersStored, usersArrSize, 50);`

Return value	`4`

Testing the data member `username`	`// Code to print the values`
	`cout<<users[0].getUsername()<<endl;`
	`cout<<users[1].getUsername()<<endl;`
	`cout<<users[2].getUsername()<<endl;`
	`cout<<users[3].getUsername()<<endl;`
	`// Expected Output`
	`Ninja`
	`Myth`
	`Sphyer`
	`Daequan`

Testing the data member `ratings`	`// Code to print the values`
	`cout<<users[0].getRatingAt(0)<<endl;`
	`cout<<users[0].getRatingAt(1)<<endl;`
	`cout<<users[0].getRatingAt(2)<<endl;`
	`cout<<users[0].getRatingAt(3)<<endl;`
	`cout<<users[0].getRatingAt(4)<<endl;`
	`.`
	`.`
	
	`// Expected Output`
	`0`
	`1`
	`2`
	`3`
	`4`
	`.`
	`.`



Example 2: The `users` array is empty, so `numUserStored` is 0 and a bad file is given

Function call	`User users[10];`
	`int numUsersStored = 0;`
	`int usersArrSize = 10;`
	`readRatings("badFile.txt", users, numUserStored, usersArrSize, 50);`

Return value	`-1`

Testing the data member `username`	`// Code to print the values`
	`cout<<users[0].getUsername()<<endl;`
	`cout<<users[1].getUsername()<<endl;`
	`.`
	`.`
	`// Expected Output`
	`""`
	`""`
	`.`
	`.`

Testing the data member `ratings`

```
 // Code to print the values
cout<<users[0].getRatingAt(0)<<endl;
cout<<users[0].getRatingAt(1)<<endl;
cout<<users[0].getRatingAt(2)<<endl;
cout<<users[0].getRatingAt(3)<<endl;
cout<<users[0].getRatingAt(4)<<endl;
.
.

// Expected Output
0
0
0
0
0
.
.
```

Example 3: The `users` array is already full, so `readRatings` returns -2.

moreRatings.txt

```
alpha,0,1,2,3,4
beta,0,1,2,3,4
gamma,0,1,2,3,4
delta,0,1,2,3,4
```

Function call

```
User users[2] = {"Ninja", "Myth"};

users[0].setUsername("Ninja");
users[1].setUsername("Myth");

users[0].setRatingAt(0,0);
users[0].setRatingAt(1,1);
users[0].setRatingAt(2,2);
users[0].setRatingAt(3,3);
users[0].setRatingAt(4,4);
users[1].setRatingAt(0,2);
users[1].setRatingAt(1,2);
users[1].setRatingAt(2,4);
users[1].setRatingAt(3,5);
users[1].setRatingAt(4,5);

int numUsersStored = 2;
int usersArrSize = 2;
```

	`readRatings("moreRatings.txt", users, numUsersStored,`
	`usersArrSize, 50);`

Return value	`-2`

Testing the data member `username`	`// Code to print the values`
	`cout<<users[0].getUsername()<<endl;`
	`cout<<users[1].getUsername()<<endl;`
	
	`// Expected Output`
	`Ninja`
	`Myth`

Testing the data member `ratings`	`// Code to print the values`
	`cout<<users[0].getRatingAt(0)<<endl;`
	`cout<<users[0].getRatingAt(1)<<endl;`
	`cout<<users[0].getRatingAt(2)<<endl;`
	`cout<<users[0].getRatingAt(3)<<endl;`
	`cout<<users[0].getRatingAt(4)<<endl;`
	`cout<<users[1].getRatingAt(0)<<endl;`
	`.`
	`.`
	
	`// Expected Output`
	`0`
	`1`
	`2`
	`3`
	`4`
	`2`
	`.`
	`.`

Example 4: There is already 1 user in the `users` array, so the value of `numUsers` is 1. However, the array `size` is only two, so only the first line of the file is stored and the function returns the `size` of the array.

ratings.txt	`stroustrup,0,4,5`
	`gosling,2,2,3`
	`rossum,5,5,5`

Function calls	`User users[2];`

	`users[0].setUsername("ritchie");`
	`users[0].setRatingAt(0,0);`
	`users[0].setRatingAt(1,1);`
	`users[0].setRatingAt(2,2);`
	` int numUsers = 1;`
	`int usersArrSize = 2;`
	`readRatings("ratings.txt", users, numUsers, usersArrSize, 50);`

Return value	`2`

Testing the data member `username`	` // Code to print the values`
	`cout<<users[0].getUsername()<<endl;`
	`cout<<users[1].getUsername()<<endl;`
	
	`// Expected Output`
	`ritchie`
	` stroustrup`

Testing the data member `ratings`	`// Code to print the values`
	`cout<<users[0].getRatingAt(0)<<endl;`
	`cout<<users[0].getRatingAt(1)<<endl;`
	`cout<<users[0].getRatingAt(2)<<endl;`
	`cout<<users[1].getRatingAt(0)<<endl;`
	`cout<<users[1].getRatingAt(1)<<endl;`
	`cout<<users[1].getRatingAt(2)<<endl;`
	
	`// Expected Output`
	`0`
	`1`
	`2`
	`0`
	`4`
	`5`

The zip submission should have three files for this problem: **User.h**, **User.cpp**, and a driver called **readRatingsDriver.cpp**, with a `main()` function to test your member functions. For CodeRunner, paste your User class and its implementation (the contents of **User.h** and **User.cpp**), and your **readRatings** function. Do not include the `main()` function. After developing in Cloud9, this function will be one of the functions you include at the top of **HW7.cpp**.

Problem 3 (20 points): getRating

We now have a list of books (in the form of an array of `Book` objects) and a list of users and their ratings for these books (in the form of an array of `User` objects).

Write a function that, given a user's name and a book's title, returns the rating that the user gave for that book.

- Your function **MUST** be named **getRating**.
- Your function should take 6 parameters in the following order:
    - **string**: username
    - **string**: title of the book
    - **Array of User objects**: `users`
    - **Array of Book objects**: `books`
    - **int**: number of users currently stored in the `users` array
    - **int**: number of books currently stored the `books` array
- The username and book title search should be case insensitive. For example, "Ben, "ben" and "BEN" are one and the same user.
- If both the user name and the book title are found in the arrays, then the function should return the user's rating value for that book title.
- The function should return the following values depending on cases:
    - Return the rating value if both user and title are found
    - Return -3 if the user or the title are not found

Sample code to generate `books` and `users` array.

Setting the values in `books`	`//Creating 3 books`
	`Book books[3];`
	
	`//Setting book title and author for book 1`
	`books[0].setTitle("Title1");`
	`books[0].setAuthor("Author1");`
	
	`//Setting book title and author for book 2`
	`books[1].setTitle("Title2");`
	`books[1].setAuthor("Author2");`
	
	`//Setting book title and author for book 3`
	`books[2].setTitle("Title3");`
	`books[2].setAuthor("Author3");`

Printing the values in `books`	`//Printing values in each book object`
	`cout<<"Books: "<<endl;`
	
	`cout<<books[0].getTitle()<<” by "<<books[0].getAuthor()<<endl;`
	
	`cout<<books[1].getTitle()<<" by "<<books[1].getAuthor()<<endl;`
	
	`cout<<books[2].getTitle()<<" by "<<books[2].getAuthor()<<endl;`

`//Expected Output`	`Books:`
	`Title1 by Author1`
	`Title2 by Author2`
	`Title3 by Author3`

Setting the values in `users`	`//Creating 2 users`
	`User users[2];`
	
	`//Setting username and ratings for User1`
	`users[0].setUsername("User1");`
	`users[0].setNumRatings(3);`
	`users[0].setRatingAt(0,1);`
	`users[0].setRatingAt(1,4);`
	`users[0].setRatingAt(2,2);`
	
	`//Setting username and ratings for User2`
	`users[1].setUsername("User2");`
	`users[1].setNumRatings(3);`
	`users[1].setRatingAt(0,0);`
	`users[1].setRatingAt(1,5);`
	`users[1].setRatingAt(2,3);`

Printing the values in `users`	`//Values in each user object`
	`cout<<"Users and their ratings: "<<endl;`
	`cout<<users[0].getUsername()<<": "<<users[0].getRatingAt(0)<<", "<<users[0].getRatingAt(1)<<", "<<users[0].getRatingAt(2)<<endl;`

	`cout<<users[1].getUsername()<<": "<<users[1].getRatingAt(0)<<", "<<users[1].getRatingAt(1)<<", "<<users[1].getRatingAt(2)<<endl;`

`// Expected Output`	`Users and their ratings:`
	`User1: 1, 4, 2`
	`User2: 0, 5, 3`

Example 1: Both the `userName` and `bookTitle` exists, and the value of rating is non-zero

Function call	`getRating("User1", "Title2", users, books, 2, 3);`
Return value	`4`

Example 2: The `userName` does not exist, it returns - 3

Function call	`getRating("User4", "Title1", users, books, 2, 3);`
Return value	`-3`

Example 3: The `bookTitle` does not exist, it returns - 3

Function call	`getRating("User1", "Title10", users, books, 2, 3);`
Return value	`-3`

Example 4: The `userName` and the `bookTitle` do not exist

Function call	`getRating("User12", "Title10", users, books, 2, 3);`
Return value	`-3`

The zip submission should have five files for this problem: **Book.h**, **Book.cpp**, **User.h**, **User.cpp**, and a driver called **readRatingDriver.cpp**, with a `main()` function to test your member and **getRating** functions. For **CodeRunner**, paste your User and Book classes (the contents of the four Book and User files mentioned above) and your **getRating** function. Do not include the `main()` function. After developing in Cloud9, this function will be one of the functions you include at the top of HW7.cpp.

Problem 4 (20 points): getCountReadBooks

Write a getCountReadBooks function that counts how many books a given user has rated.

- Your function **MUST** be named **getCountReadBooks**.
- Your function should take 4 parameters:
    - **string**: username
    - **Array of User objects**: `users`
    - **int**: number of users currently stored in the `users` array
    - **int**: number of books currently stored the books array
- The username search should be case insensitive. For example, “Ben, “ben” and “BEN” are one and the same user.
- The function should return the following values depending on cases:
    - Return the number of books read/rated by the specified user if the user is found.
    - Return -3 if the username is not found or there is no book.

Example 1: The library is initialized

bookFile.txt	`Author1,Title1`
	`Author2,Title2`
	`Author3,Title3`

ratingFile.txt	`User1,1,4,2`
	`User2,0,5,3`
	`User3,0,0,0`

Function calls	`//Creating 2 users`
	`User users[2];`
	
	`//Setting username and ratings for User1`
	`users[0].setUsername("User1");`
	`users[0].setNumRatings(2);`
	`users[0].setRatingAt(0,2);`
	`users[0].setRatingAt(1,2);`
	
	`//Setting username and ratings for User2`
	`users[1].setUsername("User2");`
	`users[1].setNumRatings(4);`
	`users[1].setRatingAt(0,4);`
	`users[1].setRatingAt(1,4);`

	`// getCountReadBooks for User2`
	`cout << getCountReadBooks(“User2”,users,2,3);`

outputs	`2`

Example 2: The user does not exist

bookFile.txt	`Author1,Title1`
	`Author2,Title2`
	`Author3,Title3`

ratingFile.txt	`User1,1,4,2`
	`User2,0,5,3`
	`User3,0,0,0`

Function calls	`User users[10];`
	
	`int numUsers = 0;`
	`int usersArrSize = 10;`
	`readRatings(“ratingFile.txt”, users, numUsers, usersArrSize, 50);`
	
	`// getCountReadBooks for User4`
	`cout <<`
	`getCountReadBooks(“User4”,users,numUsers,3);`

outputs	`-3`

Example 3: The user has not rated any book yet

bookFile.txt	`Author1,Title1`
	`Author2,Title2`
	`Author3,Title3`

ratingFile.txt	`User1,1,4,2`
	`User2,0,5,3`
	`User3,0,0,0`

Function calls	`User users[10];`
	
	`int numUsers = 0;`

	`int usersArrSize = 10;` `readRatings(“ratingFile.txt”, users, numUsers, usersArrSize, 50);` `// getCountReadBooks for User3` `cout << getCountReadBooks(“User3”,users,numUsers,3);`
outputs	`0`

The zip submission should have five files for this problem: **User.h**, **User.cpp**, and a driver called **getCountReadBooks.cpp**, with a `main()` function to test your member and **getCountReadBooks** functions. For **CodeRunner**, paste your User class and your **getCountReadBooks** function. Do not include the `main()` function. After developing in Cloud9, this function will be one of the functions you include at the top of HW7.cpp.

Problem 5 (20 points): calcAverageRating

Create a function `calcAvgRating` which returns the average (mean) rating for a particular book.

- Your function **MUST** be named **calcAverageRating**.
- Your function should take 5 input arguments in the following order:
  - **Array of User objects**: `users`
  - **Array of Book objects**: `books`
  - **int**: number of users currently stored in the `users` array
  - **int**: number of books currently stored the `books` array
  - **string**: title of the book
- The book title search should be case insensitive. For example, “Harry Potter”, “harry potter” and “HARRY POTTER” are one and the same book.
- The function should return the following values depending on cases:
  - Return the average rating of the specified book as a `double` if title is found
  - Return -3 if the title is not found or there are no users
  - Return 0 if the book has not been read by anyone. Poor book!

Note: Books that haven’t been read (have a rating value of 0) ***shouldn’t*** *be counted in calculating the average.*

Example 1: The library is initialized, and we can calculate an average ratings for the book.

Function calls	`//Create list of books`
	`Book books[2];`
	`books[0].setTitle("Title1");`
	`books[0].setAuthor("Author1");`
	`books[1].setTitle("Title2");`
	`books[1].setAuthor("Author2");`
	
	`//Create list of users`
	`User users[2];`
	
	`//Setting username and ratings for User1`
	`users[0].setUsername("User1");`
	`users[0].setNumRatings(2);`
	`users[0].setRatingAt(0,2);`
	`users[0].setRatingAt(1,2);`
	
	`//Setting username and ratings for User2`
	`users[1].setUsername("User2");`
	`users[1].setNumRatings(4);`
	`users[1].setRatingAt(0,4);`
	`users[1].setRatingAt(1,4);`
	
	`// calcAvgRating for Title2`
	`cout <<calcAverageRating(books, users, 2, 2, “title2”);`

outputs	`3.0`

The function returns 4.5 because User 1 rated 4 and User 2 rated 5, which means (4 + 5) / 2 = 4.5. Since User 3’s rating is 0, it is not included for the calculation.

Example 2: The title does not exist in the library. No user has read a particular title

bookFile.txt	`Author1,Title1`
	`Author2,Title2`
	`Author3,Title3`

ratingFile.txt	`User1,0,4,2`
	`User2,0,5,3`
	`User3,0,0,0`

Function calls	`//Create list of books`
	`Book books[10] = {};`

	`//add books to Library`
	` readBooks("fileName.txt",books, 0, 10);`
	
	`//Create list of users`
	` User users[10] = {};`
	
	`//add users`
	`readRatings("ratingFile.txt");`
	
	`// calcAvgRating for Title4`
	`cout <<calcAverageRating(books, users, 3, 3, "Title4");`
	
	`// calcAvgRating for Title1`
	`cout <<calcAverageRating(books, users, 3, 3, "Title1");`

outputs	`-3`
	`0`

Problem 6 (30 points): put them together

Now combine your Book class, User class, and functions you wrote so far. Create a file called **hw7.cpp**. In this file build a program that gives the user a menu with 6 options:

1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit

Note: The Book class definition should be in Book.h and Book.cpp, The User class definition should be in User.h and User.cpp. All the other functions used for Problems 2-5 in Hmwk7 and readBooks() from hmwk6 will go in hw7.cpp. For Problem 7, you need to submit the entire program hw7.cpp (including Book class and User class) in the answer box of the CodeRunner auto-grader on Moodle.

The menu will run on a loop, continually offering the user four options until they opt to quit. You should make use of the functions you wrote previously, call them, and process the values they return.

Size of arrays

In your driver function, you must declare your arrays with the appropriate size. The capacity of the `books` array is 50. The capacity of the `users` array is 100.

Option 1: Read books

- Prompt the user for a file name.
  - `Enter a book file name:`
- Pass the file name to your `readBooks` function.
- Print the total number of books in the database in the following format:
  - `Total books in the database: <numberOfBooks>`
- If the function returns -1, then print the following message:
  - `No books saved to the database.`
- If the function returns -2, print
  - `Database is already full. No books were added.`
- If the function returns a value equal to the size of the array of Books print the following message:
  - `Database is full. Some books may have not been added.`

Option 2: Read ratings

- Prompt the user for a file name.
  - `Enter a user file name:`
- Pass the file name to your `readRatings` function.
- Print the total number of users in the database in the following format:
  - `Total users in the database: <numberOfUsers>`
- If the function returned -1, then print the following message:
  - `No users saved to the database.`
- If the function returned -2, print
  - `Database is already full. No users were added.`
- If the function returns a value equal to the size of the array of Users print the following message:
  - `Database is full. Some users may have not been added.`

Option 3: Get rating

- Prompt the user for a user name.
  - Enter a user name:
- Prompt the user for a title
  - Enter a book title:

- Pass the user name and title to your `getRating` function.
- Print the result in the following format:
    - `<name> rated <title> with <rating>`
- If the function returned 0, print the result in the following format:
    - `<name> has not rated <title>`
- If the function returned -3, print the result in the following format:
    - `<name> or <title> does not exist.`

Option 4: Find number of books user rated

- Prompt the user for a user name.
    - Enter a user name:
- Pass the username, and other parameters to your `getCountReadBooks` function.
- Print the result in the following format:
    - `<name> rated <number> books.`
- If the function returned 0, print the result in the following format:
    - `<name> has not rated any books.`
- If the function returned -3, print the result in the following format:
    - `<name> does not exist.`

Option 4: Get average rating

- Prompt the user for a book title.
    - Enter a book title:
- Pass the title, and other parameters to your `calcAverageRating` function.
- Print the result in the following format:
    - `The average rating for <title> is <avg rating>`
- If the function returned -3, print the result in the following format:
    - `<title> does not exist.`

Option 6: Quit

- Print `Good bye!` and then stop the program.

Invalid input

If the user input is not the above values print `Invalid input.`

Below is an example of running the `HW7` program:

```
Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
```

```
6. Quit
Good bye!
1
Enter a book file name:
badFile.txt
No books saved to the database.

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
1
Enter a book file name:
books.txt
Database is full. Some books may have not been added.

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
1
Enter a book file name:
books2.txt
Database is already full. No books were added.

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
2
```

```
Enter a user file name:
ratings.txt
Total users in the database: 86

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
3
Enter a user name:
megan
Enter a book title:
The Hunger Games
megan rated The Hunger Games with 4


Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
3
Enter username:
amy
Enter book title:
The Lord of the Rings
amy rated The Lord of the Rings with 2

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
```

```
6. Quit
3
Enter a user name:
Punith
Enter a book title:
intro to OS
Punith or intro to OS does not exist.

======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
4
Enter a user name:
Malvika
Malvika does not exist.

======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
4
Enter a user name:
tiffany
tiffany rated 41 books.

======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
5
Enter a book title:
Naruto
```

```
The average rating for Naruto is 2.67105

======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
5
Enter a book title:
Introduction to Algorithms
Introduction to Algorithms does not exist.

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
11
Invalid input.

Select a numerical option:
======Main Menu=====
1. Read books
2. Read ratings
3. Get rating
4. Find number of books user rated
5. Get average rating
6. Quit
6
good bye!
```

6. Homework 7 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 7 CodeRunner**
2. Submit one zip file to **Homework 7**. The zip file should be named, **hmwk7_lastname.zip**. It should have the following 10 files:
    - Book.h (from hmwk6)
    - Book.cpp (from hmwk6)
    - User.h
    - User.cpp
    - userDriver.cpp
    - readRatingsDriver.cpp
    - getRating.cpp
    - getCountReadBooks.cpp
    - calcAverageRating.cpp
    - hw7.cpp

7. Homework 7 points summary

Criteria	**Pts**
CodeRunner*	140
Style, Comments, Algorithms	10
Test cases	20
Recitation attendance (week 9)*	-30
Total	170
5% early submission bonus	+5%

* You also need to submit `Book` class and `ReadBooks` from Hwmk5 to codeRunner
* if your attendance is not recorded, you will lose points. Make sure your attendance is recorded on Moodle.