CSCI 1300 CS1: Starting Computing
Instructor: Fleming/Wong, Spring 2019
Homework 7
Due Wednesday, March 13, by 11 pm
+5% bonus if submitted by Tuesday, March 12, 11:59 pm

****The work you do in this assignment will be used as part of the upcoming assignments and Project 2.**

All 3 components (Cloud9 workspace, Moodle Coderunner attempts, and zip file) must be completed and submitted by Wednesday, March 13, at 6 pm for your homework to receive points.

---

1. Objectives

- Define classes and create objects
- Array operations: initialization, search
- Create arrays of an object type
- Use filestream objects to read data from text files
- Continue building on what will become your Project 2

---

2. Submission Requirements

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

1. ***Create Hmwk7 directory on your Cloud 9 workspace:*** 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.
    - Make sure to *save* the final version of your code (File > Save). Verify that this version displays correctly by going to File > File Version History.
    - The file(s) should have all of your functions, test cases for the functions in `main()` function(s), and adhere to the style guide. Please read the

submission file instructions under Week 4. **You must include a test case for each one of your member functions for your classes.**

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 9 questions from the Moodle assignment, zip all 15 files you compiled in Cloud9, and submit the zip file through the **Homework 7 (File Submission)** link on Moodle.

---

3. 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 (File Submission)** link on Moodle and assign points for the following:

Style and Comments **(5 points):**

- The style guide is posted on Moodle under Week 6.
- Your code should be well-commented. Please review the standard for well-commented code, presented in more detail in previous homework write-ups.
- Please also include a comment at the top of your solution with the following format:

```
// CS1300 Spring 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Cloud9 Workspace Editor Link: https://ide.c9.io/…
// Homework 7 - Problem # ...
```

Global variables **(use will result in a 5 point deduction):**

- Later in the semester, we will learn about global variables and the joys and dangerous therein. To keep things simple, straightforward, and easy to debug and test, **you may not use global variables in this homework.**

Algorithm **(5 points):**

- 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. Please review the standard for including your algorithm for each function, presented in more detail in previous homework write-ups.

Test Cases **(30 points):**

1. *Code compiles and runs* (**10 points**):
    - All the files you submit on Moodle under the zip file submission should compile and be free of bugs. It is important that your programs can be compiled and run on Cloud9 with no errors. The class files and the functions included in these programs should match those submitted to the CodeRunner on Moodle.

2. *Test cases* (**20 points**):
   For this week's homework, every problem is asking you to define a class and/or some of its member functions, or functions to make use of your classes. In addition to the class files (.h and .cpp), you will need to create a driver/tester program, with a `main()` function, where you must have test cases for each function or member function. Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.

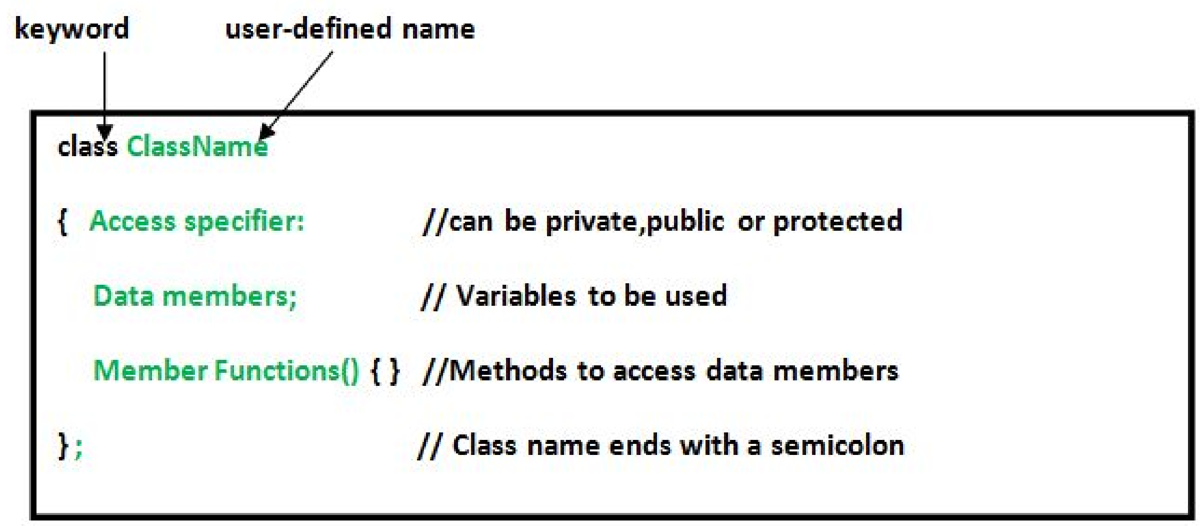
   Please make sure that your submission files follow the the **submission file instructions** under Week 6.

---

4. Part One: Syntax

Classes

When writing complex programs, sometimes the built in data types (such as int, char, string) don't offer all of the functionality or flexibility that may be desired or necessary for the problem being solved. A solution to this is for the developer (you - yes, ***you***!) to create your own custom data types to use, called **classes**. Classes are user-defined data types, which hold their own **data members** and **member functions**, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object, customized for whatever particular problem a programmer is working on. Below is an example of the basic definition of a class in C++.



```
class ClassName
{  Access specifier:            //can be private,public or protected
   Data members;                // Variables to be used
   Member Functions() { }       //Methods to access data members
};                              // Class name ends with a semicolon
```

Let’s break down the main components of this diagram:

Class Name:
A class is defined in C++ using the keyword **class** followed by the name of class. The body of the class is defined inside the curly brackets and terminated by a semicolon at the end. This class name will be how you reference any *variables* or *objects* you create of that type. For example:

```
ClassName objectName;
```

The above line would create a new object (variable) with name **`objectName`** and of type **`ClassName`**, and this object would have all the properties that you have defined within the class **`ClassName`**.

Access Specifiers:
Access specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by outside functions. Let’s consider an analogy to describe access specifiers. Imagine an intelligence agency, like the CIA, that has 10 senior members. Such an organization would have various sorts of information, and would need to have some way of determining who has access to any given piece of information.

Some information may only be accessible to the 10 senior members of the agency, and any other person would be unable to access it directly. This may be the secret operations of the agency or information that would be dangerous if everyone could see it. This data would be **private**. In a class, like in our intelligence agency, **private** data is available only to specific data members and member functions. They are not allowed to be accessed directly by any object or function outside the class. Only member functions are allowed to access the private data members of a class.

Some other information may be available to anyone who wants to know about it. This would be **public** data. Even people outside the CIA would be able to know about it, such as press releases by the agency. In terms of our class, this **public** data can be accessed by any member function of the class, as well as outside functions and objects. The data members and member functions declared **public** can be accessed by other classes too. The **public** members of a class can be accessed from anywhere in the program using the direct member access operator (**.**) with the object of that class.

Data Members and Member Functions:
Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class. The data members which are declared in any class definition are created by using any fundamental data types (like int, char, float etc). For example, strings objects the data member would be the **`char array[]`** that holds all of the individual letters of your string. Some of a string's member functions would be functions like `.substr()`, `.find()`, and `.length()`.

Accessing Data Members:
Keeping with our intelligence agency example, the code below would define a class that holds information for any general agency. In main, we then create a new `intelligenceAgency` object called `CIA`, and we set its `organizationName` to "CIA" by using the access operator (.) and the corresponding data member's name. However, if we wanted to access the `classifiedIntelligence` data member, we would not be able to do it in the same way. Not everyone has access to that data. We would need to use a member function of the agency `getClassifiedIntelligence()` in order to see that information. However, this function may modify the information such that more sensitive bits are removed. This makes sure that there is still some control of the private information that is released by our `intelligenceAgency`.

```
class intelligenceAgency
{
     public:
     intelligenceAgency();          // Default constructor
     intelligenceAgency(string s); // Parameterized constructor
     string organizationName;
     string getClassifiedIntelligence();
     void setClassifiedIntelligence(string s);

     private:
     string classifiedIntelligence;
};

int main()
{
     intelligenceAgency CIA;
     CIA.organizationName = "CIA";
     cout << CIA.classifiedIntelligence; //gives an error
     cout << CIA.getClassifiedIntelligence();
}
```

Defining Member Functions:

Something you may have noticed is that we *declared* various member functions in our class definition, but that we didn't specify how they will work when called. The body of the function still needs to be written. The immediate solution is to define a function, such as `getClassifiedIntelligence()`, corresponding to our class's functions. But how does our program know that these functions are the same as the ones we declared in our class? You as the developer need to explicitly tell the computer that these functions you are defining are the same ones you declared earlier.

```
string intelligenceAgency::getClassifiedIntelligence()
{
     return classifiedIntelligence;
}

void intelligenceAgency::setClassifiedIntelligence(string s)
{
     classifiedIntelligence = s;
}
```

We start the definition as we do any function: with the return type. Then, we have to add a specifier `intelligenceAgency::` that lets our program know that this function and the one we declared inside the class are the same. We can see that this function returns the class's `classifiedIntelligence` to the user. Functions like `getClassifiedIntelligence()` are called accessor, or "**getter**", functions. This is because they retrieve or 'get' the private data members of a class object and return it to the program so that these values may be used elsewhere. The second function, `setClassifiedIntelligence(string s)`, is called a mutator, or "**setter**", function. These allow functions from other parts of our program to modify or 'set' the private data members of our class objects. Getters and setters are the functions that our program uses to interact with the private data members of our class objects.

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 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
```

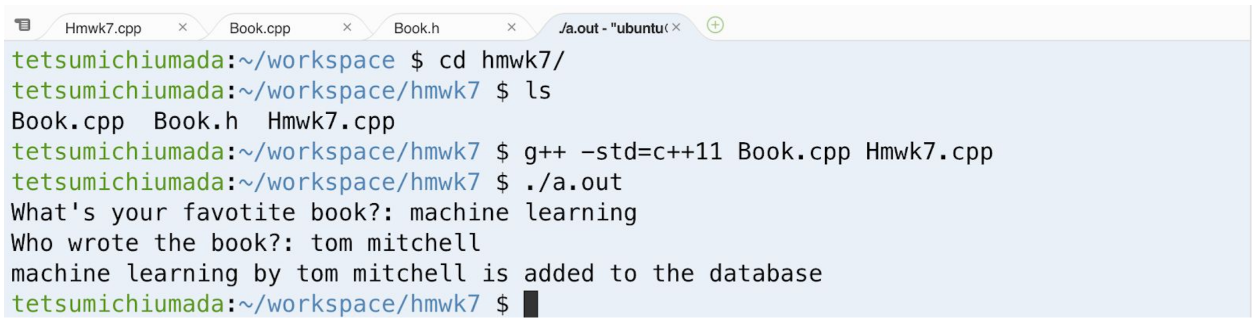
You can add the -o flag to your compiling command to give the output file a name:

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

And then run the file with

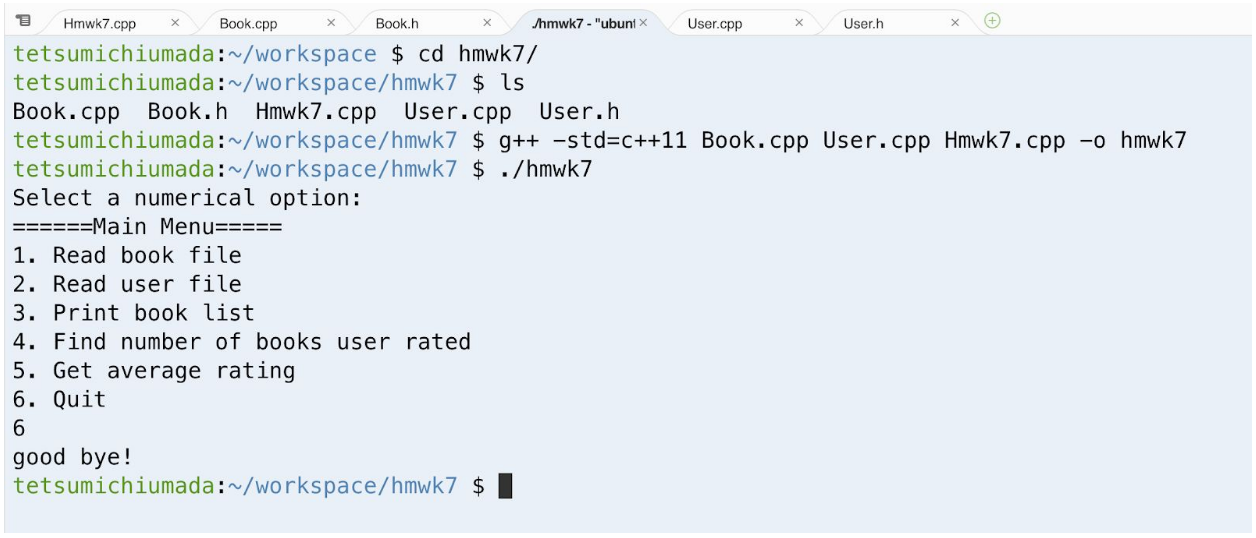
```
./myName.out
```

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 $
```

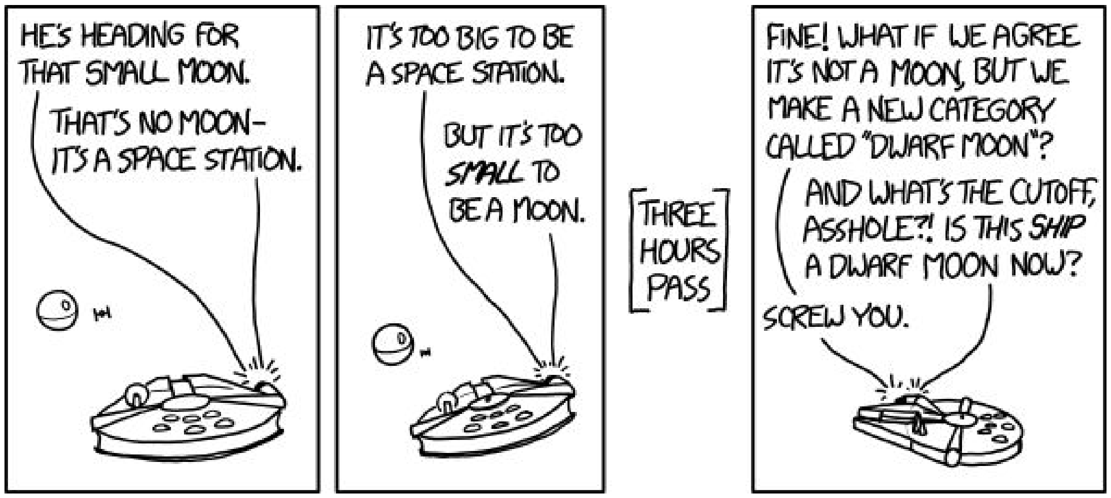
5. Problem Set

<u>Note: To stay on track for this homework set, we recommend to finish/make considerable progress on Problems 1-6 by Friday. Come to recitation with questions!</u>

The first 2 problems of this homework have been designed to help you practice creating and using classes, as well as to build upon the work that you've been doing with arrays. These problems will require you to make separate header and source files for your class, as well as a "driver" file which will test the class and functions that you've created. For these first 2 problems, you will need to turn in 4 files: your header file for the Planet class (**Planet.h**), a source file for the definitions of the getters and setters of the Planet class (**Planet.cpp**), and a source file for each individual problem in this section (**planetDriver.cpp** (Problem 1) and **maxRadiusDriver.cpp** (Problem 2)**).**

5.1. Stretch first!

Problem 1 (10 points) planet class


HE'S HEADING FOR THAT SMALL MOON.
THAT'S NO MOON- IT'S A SPACE STATION.
IT'S TOO BIG TO BE A SPACE STATION.
BUT IT'S TOO SMALL TO BE A MOON.
[THREE HOURS PASS]
FINE! WHAT IF WE AGREE IT'S NOT A MOON, BUT WE MAKE A NEW CATEGORY CALLED "DWARF MOON"?
AND WHAT'S THE CUTOFF, ASSHOLE?! IS THIS SHIP A DWARF MOON NOW?
SCREW YOU.

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

Data members (private):	
string: `planetName`	Name of the planet
double: `planetRadius`	Radius of the planet

Member functions (public):	
Default constructor	Sets `planetName` to empty string, and `planetRadius` to 0.0
Parameterized constructor	Takes a string and double initializing `planetName` and `planetRadius`, in this order
`getName()`	Returns `planetName` as a string
`setName(string)`	(void) Assigns `planetName` the value of the input string
`getRadius()`	Returns `planetRadius` as a double
`setRadius(double)`	(void) Assigns `planetRadius` the value of the input double
`getVolume()`	Calculates and returns the volume of the planet as a double

It is advisable to write your own test cases for each class. Test your class in Cloud9 before submitting to the CodeRunner autograder.

The zip submission should have three files for this problem: **Planet.h**, **Planet.cpp**, and a driver called **planetDriver.cpp** to test your member functions. For Coderunner, in the Answer Box, paste your **Planet class and its implementation** (contents of Planet.h and Planet.cpp). Do not include your **main()** function from planetDriver.cpp that you used for testing.

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 22 materials, and the *Worked Example 9-1* from the textbook (`Implementing a Bank Account Class` - step 6: Test your class).

Problem 2 (10 points) maxRadius

Write a function, `maxRadius`, to find the index of the planet with the largest radius in the array. The function should:

- Be named **maxRadius**
- Take **two** arguments in the following order
  - An **array of planet** objects
  - An **integer**, the size of the array (possibly partially filled, so this would be the number of actual planet elements in the array)

- Return the index of the planet in the array with the largest radius.
- If multiple planets in the array share the largest radius, your function should return the index of the first planet.
- If the array of Planet objects is empty, your function should return -1

Examples:

Function call	**Output**

`Planet planets[5];`
`planets[0] = Planet(“On A Cob Planet”,1234);`
`planets[1] = Planet(“Bird World”,4321);`
`int idx = maxRadius(planets, 2);`
`cout << planets[idx].getName() << endl;`
`cout << planets[idx].getRadius() << endl;`
`cout << planets[idx].getVolume() << endl;`
	`Bird World`
	`4321`
	`3.37941e+11`

`Planet planets[3];`
`planets[0] = Planet(“Nebraska”,13.3);`
`planets[1] = Planet(“Flarbellon-7”,8.6);`
`planets[2] = Planet(“Parblesnops”,6.8);`
`int idx = maxRadius(planets, 3);`
`cout << planets[idx].getName() << endl;`
`cout << planets[idx].getRadius() << endl;`
`cout << planets[idx].getVolume() << endl;`
	`Nebraska`
	`13.3`
	`9854.7`

`Planet planets[3];`
`int idx = maxRadius(planets, 0);`
`cout << idx << endl;`
	`-1`

`Planet planets[3];`
`planets[0] = Planet(“Planet Squanch”,6.8);`
`planets[1] = Planet(“Delphi 6”,8.6);`
`Planet newPlanet;`
`newPlanet.setName(“Cronenberg World”);`
`newPlanet.setradius(8.6);`
`planets[2] = newPlanet;`
`int idx = maxRadius(planets, 3);`
`cout << planets[idx].getName() << endl;`
	`Delphi 6`
	`8.6`
	`2664.31`

```
cout << planets[idx].getRadius() << endl;
cout << planets[idx].getVolume() << endl;
```

The zip submission should have three files for this problem: **Planet.h**, **Planet.cpp**, and a driver called **maxRadiusDriver.cpp**, with your **maxRadius()** function and a **main()** function to test your **maxRadius()** function.

For the Problem 2 Coderunner, paste **your Planet class, its implementation, and your maxRadius function**. (Submit what you did for Problem 1, *and* add your maxRadius function.) Do not include your **main()** function from maxRadiusDriver.cpp that you used for testing your function.

5.2. Part Two: Let's do this.

For the remaining 7 problems, we will be updating our solutions from Homework 6 in preparation for Project 2, where we will create a book recommender system. For the purposes of this assignment, we will be re-organizing the functions you've created to read and store users and ratings by creating classes for both users and books. For this portion of the homework, you will be asked to submit 11 separate files (*a recap of the list of required files can be found on Chapter 6, page 35 of this write up*): a header and source for **both** the user and books classes (**Book.h and Book.cpp, and User.h and User.cpp**), a driver to test the basic functionality of each class (**bookDriver.cpp, userDriver.cpp**), as well as a source file that will be the driver for each of the individual functions you write for problems 3-9 (**readBooksDriver.cpp, printAllBooksDriver.cpp, readRatingsDriver.cpp, getRatingDriver.cpp**). Lastly, you will need to submit a driver that will use all of the functionality of your classes (**HW7.cpp**).

Problem 3 (10 points) Book Class

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

Data members (private):	
string: `title`	
string: `author`	

Member functions (public):	
Default constructor	Sets both `title` and `author` to empty strings
Parameterized constructor	Takes two strings for initializing `title` and `author`, in this order
`getTitle()`	Returns `title` as a string
`setTitle(string)`	(void) Assigns `title` the value of the input string
`getAuthor()`	Returns `author` as a string
`setAuthor(string)`	(void) Assigns `author` the value of the input string

The zip submission should have three files for this problem: **Book.h**, **Book.cpp**, and a driver called **bookDriver.cpp**, with a **main()** function to test your member functions. For Coderunner, paste your Book class and its implementation (the contents of Book.h and Book.cpp). Do not include the `main()` function, nor the line `#include "Book.h”`

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 the Lecture 22 materials, and the *Worked Example 9-1* from the textbook (`Implementing a Bank Account Class` - step 6: Test your class).

Problem 4 (20 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, 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 22 materials, and the *Worked Example 9-1* from the textbook (`Implementing a Bank Account Class` - step 6: Test your class).

Problem 5 (15 points) readBooks

Update the **readBooks** function from Homework 6 to now fill an array of `Book` objects instead of having separate `titles` array and `authors` array. The functionality stays the same as the one from the previous homework. This function should:

- Accept four input arguments in this order:
  - **string** `fileName`: the name of the file to be read.
  - **array** `books`: array of **Book** objects.
  - **int** `numBooksStored`: the number of books currently stored in the arrays. You can always assume this is the correct number of actual elements in the arrays.
  - **int** `booksArrSize`: capacity of the `books` array. The default value for this data member is 50.
- Use `ifstream` and `getline` to read data from the file, making an instance of the Book object for each line, and placing it into the `books` array.
- You can use the `split()` function from Problem 3 in Homework 6, with comma (‘,’) as the delimiter. When you copy your code to the coderunner, make sure you put in the Answer Box your **Book** class, **readBooks()** function, **split()** function.
- The function should return the following values depending on cases:
  - Return the total number of `books` in the system, as an integer.
  - When the file is not opened successfully, return -1.
  - When `numBooksStored` is equal to the `size`, return -2.
  - The priority of the return code -2 is higher than -1, i.e., in cases when `numBooksStored` is equal to the `size` and the file cannot be opened, the function should return -2.
  - When `numBooksStored` is smaller than `size`, keep the existing elements in `books`, then read data from the file and add (append) the data to the array. The number of books stored in the array cannot exceed the `size` of the `books` array.
- Empty lines should not be added to the arrays.

Example 1: The `books` array is empty, so for this example `numBooksStored` is zero.

fileName.txt	`Author A,Book 1`
	`Author B,Book 2`

Function call	`Book books[10] = {};`
	`readBooks(“fileName.txt”,books, 0, 10);`

Return value	`2`

Testing the data member `author`	`// Code to print the values`
	`cout<<books[0].getAuthor()<<endl;`
	`cout<<books[1].getAuthor()<<endl;`
	
	`// Expected Output`
	`Author A`
	`Author B`

Testing the data member `title`	`// Code to print the values`
	`cout<<books[0].getTitle()<<endl;`
	`cout<<books[1].getTitle()<<endl;`
	
	`// Expected Output`
	`Book 1`
	`Book 2`

Example 2: Suppose there's already one book in the `books` array, so `numBooksStored` is one. Since there is room for 9 more books, all the data read from the new file will be added to the array.

fileName.txt	`Author A,Book 1`
	`Author B,Book 2`

Function call	`Book books[10];`
	`books[0].setAuthor(“Author Z”);`
	`books[0].setTitle(“Book N”);`
	`readBooks(“fileName.txt”,books, 1, 10);`

Return value	`3`

Testing the data member `author`	`// Code to print the values`
	`cout<<books[0].getAuthor()<<endl;`
	`cout<<books[1].getAuthor()<<endl;`
	`cout<<books[2].getAuthor()<<endl;`
	
	`// Expected Output`
	`Author Z`
	`Author A`
	`Author B`

Testing the data member `title`	`// Code to print the values`
	`cout<<books[0].getTitle()<<endl;`
	`cout<<books[1].getTitle()<<endl;`
	`cout<<books[2].getTitle()<<endl;`

	`// Expected Output`
	`Book N`
	`Book 1`
	`Book 2`

Example 3: There's already one book in the `books` array, so `numBooksStored` is one. However, the array size is only two, so only the first line of the new file is stored.

fileName.txt	`Author A,Book 1`
	`Author B,Book 2`
	`Author C,Book 3`

Function calls	`Book books[2];`
	`books[0].setAuthor(“Author Z”);`
	`books[0].setTitle(“Book N”);`
	`readBooks(“fileName.txt”, books, 1, 2);`

Return value	`2`

Testing the data member `author`	`// Code to print the values`
	`cout<<books[0].getAuthor()<<endl;`
	`cout<<books[1].getAuthor()<<endl;`
	`// Expected Output`
	`Author Z`
	`Author A`

Testing the data member `title`	`// Code to print the values`
	`cout<<books[0].getTitle()<<endl;`
	`cout<<books[1].getTitle()<<endl;`
	`// Expected Output`
	`Book N`
	`Book 1`

Example 4: file does not exist.

Function call	`Book books[2];`
	`books[0].setAuthor(“Author Z”);`
	`books[0].setTitle(“Book N”);`
	`readBooks(“badFileName.txt”,books, 1, 2);`

Return value	`-1`

Testing the data member `author`	`// Code to print the values`
	`cout<<books[0].getAuthor()<<endl;`
	`// Expected Output`
	`Author Z`

Testing the data member `title`	`// Code to print the values`
	`cout<<books[0].getTitle()<<endl;`
	`// Expected Output`
	`Book N`

Example 5: `numBookStored` equals `size` means the array is already full.

fileName.txt	`Author A,Book 1`
	`Author B,Book 2`
	`Author C,Book 3`

Function call	`Book books[1];`
	`books[0].setAuthor(“Author Z”);`
	`books[0].setTitle(“Book N”);`
	`readBooks(“fileName.txt”,books, 1, 1);`

Return value	`-2`

Testing the data member `author`	`// Code to print the values`
	`cout<<books[0].getAuthor()<<endl;`
	`// Expected Output`
	`Author Z`

Testing the data member `title`	`// Code to print the values`
	`cout<<books[0].getTitle()<<endl;`
	`// Expected Output`
	`Book N`

The zip submission should have three files for this problem: **Book.h**, **Book.cpp** and a driver program called **readBooksDriver.cpp** to test your member functions and readBooks function. For **Coderunner**, paste your **Book class** (both the header and implementation)**, and your readBooks function**, not the entire program. After developing in Cloud9, this function will be one of the functions you include at the top of HW7.cpp.

Problem 6 (10 points) printAllBooks

The **printAllBooks** function from Homework 6 was very useful, so let's do it again, but with an array of Book objects this time! Write a *new* **printAllBooks** function which will be useful in displaying the contents of your library. This function should:

- Accept two arguments in this order:
    - **array** `books`: array of **Book** objects.
    - **int**: number of books in the array (*Note: this value might be less than the capacity of 50 books*)

- This function does **not** return anything
- If the number of books is 0, print "`No books are stored`"
- Otherwise, print "`Here is a list of books`" and then each book in a new line using the following statement
  ```
  cout << books[i].getTitle() << " by ";
  cout << books[i].getAuthor() << endl;
  ```

Note: In the test case, you can always assume that the number of books matches the number of elements in the `books` array.

Expected output (assuming you have read the data 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
...
```

The zip submission should have three files for this problem: **Book.h**, **Book.cpp** and a driver program called **printAllBooksDriver.cpp** to test your member functions and printAllBooks function. For **Coderunner**, paste **only your Book class implementation and printAllBooks function**, not the entire program. After developing in Cloud9, this function will be one of the functions you include at the top of HW7.cpp.

Problem 7 (20 points) readRatings

We will now update the **readRatings** from last week 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 last week.

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 3 books. The `ratings.txt` file would be of the format:

```
ratings.txt
```

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

This function should:

- Accept five arguments 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 Problem 3 in Homework 6, 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.
- The function should return the following values depending on cases:
    - Return the total number of users in the system, as an integer.
    - If the file cannot be opened, return -1
    - When `numUsersStored` is greater than or equal to the `usersArrSize`, return -2
    - The priority of the return code -2 is higher than -1, i.e., in cases when `numUsersStored` is equal to the `size` and the file cannot be opened, the function should return -2
    - 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.
    - Empty lines should not be added to the arrays.

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

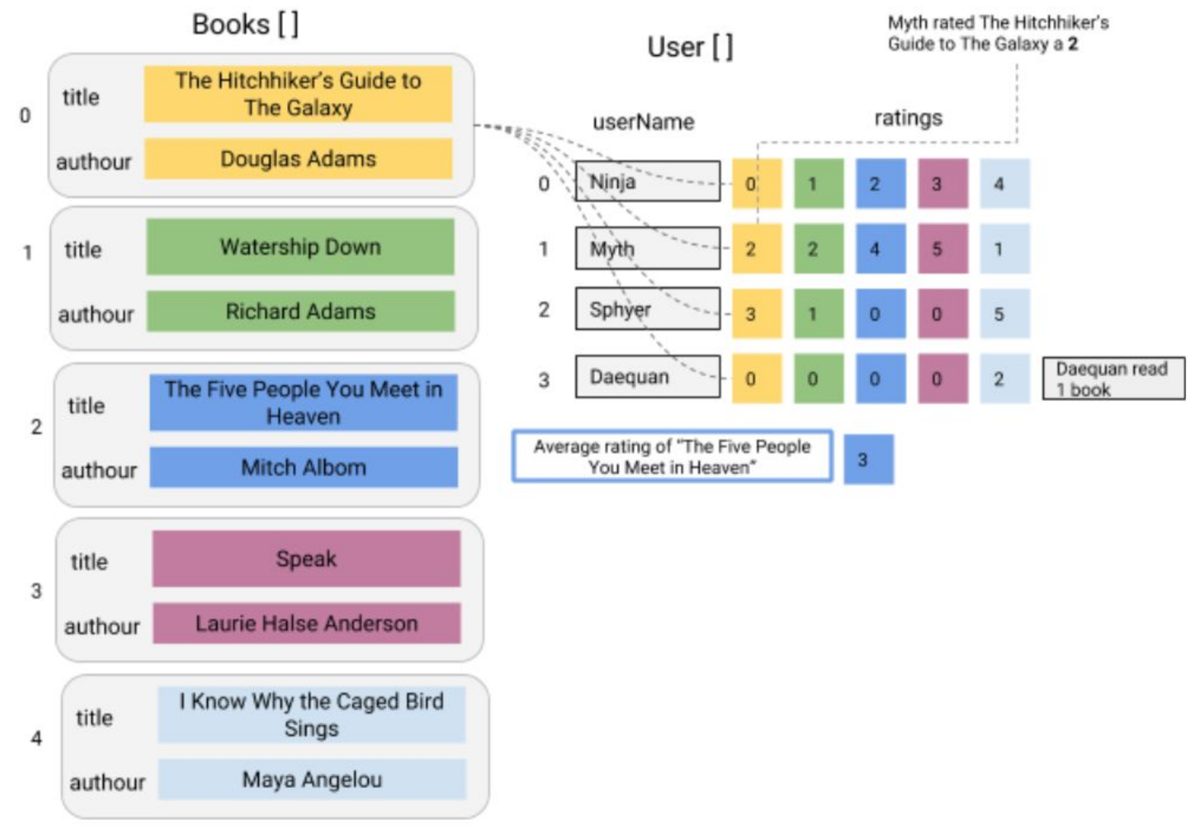
ratings.txt	`Ninja,0,1,2,3,4`
	`Myth,2,2,4,5,1`
	`Sphyer,3,1,0,0,5`
	`Daequan,0,0,0,0,2`

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 8 (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).

We now update the **getRating** function from Homework 6 to use these arrays of `User` and `Book` objects , to search for a particular user's rating for a particular book title. 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 input arguments 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

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 9 (10 points) driver

Now, let's modify our **HW6.cpp** from Homework 6 to use the `Book` class, `User` class, and the other functions we have updated for Homework 7. Make a copy of your old HW6.cpp and rename it to **HW7.cpp**, to modify for this problem so we can show off to our parents, significant others, pets or annoyed roommates how much cool stuff we're doing!

Note that the function definitions for Problems 5-8 will go in this file as well. The zip file submission should have five files for this problem: **Book.h**, **Book.cpp**, **User.h**, **User.cpp**, and a driver called **HW7.cpp**, with a main() function to test your menu interface. Note that the submitted HW7.cpp file should ***not*** have the class definitions in it. They should be contained in their respective modules (Book.h+Book.cpp, or User.h+User.cpp) and **#include**'ed in HW7.cpp.

For **CodeRunner**, however, paste your entire HW7.cpp driver function ***with*** the class definitions from Problems 3 and 4. For Problem 9, you need to submit the the entire program HW7.cpp, including the Book and User classes, in the answer box of the Coderunner auto-grader on Moodle.

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

- In your driver function, you must declare your arrays with the appropriate size. The capacity of the `books` array is 50 and `users` array is `100`.
  - In your template function the sizes have been declared. Match them accordingly
    - `const static int  usersArrSize = 100;`
    - `const static int  booksArrSize = 50;`

- **Option 1:** Initialize library
  - Prompt the user for a 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 returned -1, then print the following message:
    - `No books saved to the database.`

   - If the function returned -2, print
     - `Database is already full. No books were added.`

   - When `numberOfBooks` is equal to size of the array print the following message:
     - `Database is full. Some books may have not been added.`

- **Option 2:** Initialize user catalog
   - Prompt the user for a 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: <numUsers>`

   - 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.`

   - When `numUsers` is equal to size of the array print the following message:
     - `Database is full. Some users may have not been added.`

- **Option 3:** Display library
   - Call your `printAllBooks` function.

- **Option 4:** Get a rating
   - Prompt the user for a username.
   - Prompt the user for a title
   - Pass the username and the title to your `getRating` function
   - If the user exists in the system, print the result in the following format:
     - `<name> rated <title> with <rating>`

   - If the function returns 0, print the result in the following format:
     - `<name> has not rated <title>`

   - If the function returns -3, print the result in the following format:
     - `<name> or <title> does not exist`

- **Option 5:** Quit
  - Print "`good bye!`" before exiting

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

```
Select a numerical option:
======Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Get rating
5. Quit
1
Enter a book file name:
badFile.txt
No books saved to the database.

Select a numerical option:
======Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Get rating
5. 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 book file
2. Read user file
3. Print book list
4. Get rating
5. 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 book file
2. Read user file
3. Print book list
4. Get rating
5. Quit
2
Enter a user file name:
ratings.txt
Total users in the database: 86

Select a numerical option:
======Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Get rating
5. Quit
3
Here is a list of books
The Hitchhiker's Guide To The Galaxy by Douglas Adams
Watership Down by Richard Adams
...
...
...
Maus: A Survivor's Tale by Art Spiegelman
The Joy Luck Club by Amy Tan
The Lord of the Rings by J R R Tolkien
```

```
Select a numerical option:
======Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Get rating
5. Quit
4
Enter username:
Punith
Enter book title:
XKCD 1739
Punith or XKCD 1739 does not exist

Select a numerical option:
======Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Get rating
5. Quit
4
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 book file
2. Read user file
3. Print book list
4. Get rating
5. Quit
```

```
5
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 (File Submission). The zip file should be named, **<firstName>_<lastName>_homework7.zip**., and have following 15 files:
    1. `Planet.h`
    2. `Planet.cpp`
    3. `planetDriver.cpp`
    4. `maxRadiusDriver.cpp`
    5. `Book.h`
    6. `Book.cpp`
    7. `bookDriver.cpp`
    8. `User.h`
    9. `User.cpp`
    10. `userDriver.cpp`
    11. `readBooksDriver.cpp`
    12. `printAllBooksDriver.cpp`
    13. `readRatingsDriver.cpp`
    14. `getRatingDriver.cpp`
    15. `HW7.cpp`

7. Homework 7 point summary

Criteria	**Pts**
CodeRunner (problem 1 - 9)	125
Style and Comments	5
Algorithms	5
Test cases	30
Recitation attendance (Mar 5 or Mar 7)*	-40
Not using classes in the driver	-10
Using global variables	-5
Total	165
5% early submission bonus	+5%

* if your attendance is not recorded, you will lose points. Make sure your attendance is recorded on Moodle.