CSCI 1300 CS1: Starting Computing
Instructor: Fleming/Wong, Spring 2019
Homework 6
Due Monday, March 4, by 6 pm
+5% bonus if submitted by Sunday, March 3, 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 Monday, March 4, by 6 pm for your homework to receive points.

---

1. Objectives

- Array operations: initialization, search
- Improve proficiency with loops and strings
- Use filestream objects to read data from text files
- Get started with part of 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 Hmwk6 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 **Hmwk6** 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.

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 6 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 9 files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 6 (File Submission)** link on Moodle.

---

3. Rubric

Aside from the points received from the **Homework 6 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 6 (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 6 - Problem # ...
```

Global variables (use will attract a 5 points 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 (20 points):

1. *Code compiles and runs* (**6 points**):
    - The zip file you submit to Moodle should contain **9** full programs (each 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, 7 problems are asking you to create a function (Problems 2 through 8). In your Cloud9 solution file for each function, you should have 2 test cases present in their respective `main()` function, for a total of 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.

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

---

4. Part One: Syntax

4.1. Background

4.1.1. Arrays

An array is a *data structure* which can store primitive data types like floats, ints, strings, chars, and booleans.

Arrays have both a **type** and a **size**.

- **How to Declare Arrays**

```
data_type array_name[declared_size];
bool myBooleans[10];
string myStrings[15];
int myInts[7];
```

- **How to Initialize Arrays (Method 1)**

```
bool myBooleans[4] = {true, false, true, true};
```

If you do not declare the size inside the square brackets, the array size will be set to however many entries you provide on the right.

```
bool myBooleans[] = {true, false, true}; // size = 3
```

Note: the size specified in the brackets needs to match the number of elements you wrote in the curly brackets.

Example 1: When the specified size is larger than the actual number of elements, the elements provided in the curly brackets will be the first several elements in the array, while the additional elements will be filled with default values. If it's an integer/double array, the default values are zero, while if it's a string array, the default values are empty strings.

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

int main() {
    int intArray[5] = {1,2,3};
    for (int i = 0; i < 5; i ++) {
        cout << intArray[i] << " ";
    }
}
```

Output:

```
1 2 3 0 0
```

Example 2: When the specified size is smaller than the actual number of elements, there will be a compilation error.

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

int main() {
    int intArray[3] =
              {1,2,3,4,5};
}
```

Output (Error message):

```
error: excess elements in array
initializer
int intArray[3] = {1,2,3,4,5};
                           ^

1 error generated.
```

- **How to Initialize Arrays (Method 2)**

  You can also initialize elements one by one using a for or while loop:

```
int myInts[10];
int i = 0;
while (i < 10) {
    myInts[i] = i;
    i++;
}
//{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- **How to Access Elements in an Array**

  We have essentially already had practice with accessing elements in arrays, as in C++, strings are arrays of characters.

  You can access elements in arrays using the same syntax you used for strings:

```
string greetings[] = {"hello", "hi", "hey", "what's up?"};
cout << greetings[3] << endl;
```

"hello"	"hi"	"hey"	**"What's up?"**
0	1	2	**3**

Arrays can be iterated in the same way we iterated over strings last week. Below we are iterating through an array of strings:

```
string greetings[] = {“hello”, “hi”,
“hey”, “what’s up?”};
int size = 4;
int i = 0;
while (i < size){
      cout << greetings[i] << endl;
      i++;
}
```

Output:

```
hello
hi
hey
what's up?
```

4.1.2 Two-dimensional Arrays

Two-dimensional arrays are also arrays, but they are arrays of arrays. When you declare a one-dimensional array of integers, it will look like this:

```
int integer_array[10];
```

To declare a two-dimensional array, simply add a new dimension as follows,

```
int integer_array[10][20];
```

which will create a 10 x 20 matrix. The (i, j) element of this matrix is accessed by

```
integer_array[i][j];
```

Initializing values for two-dimensional arrays is similar to the one-dimension case. For example, if you want to initialize a two-dimensional array with some values, here’s what you might want to do:

```
int integer_array[2][2] = { {10, 20}, {30, 40} };
```

However, be careful when you write something like this,

```
int integer_array[2][2] = { {10, 20} };
```

which does not give the matrix as

```
10 20
10 20
```

Instead, it gives you

```
10 20
0 0
```

This is similar to the example 1 in Method 1 of initializing arrays (see here). Because the second dimension is specified as 2, but we only provide the array in the first dimension, *i.e.,* `{10,20}`, the second array is filled with default values.

4.1.3. Passing arrays to functions

- **Passing By Value**

  Up until now, when calling functions, we have always *passed by value.* When a parameter is passed in a function call, a new variable is declared and initialized to the value *passed* in the function call.

  Observe that the variable x in **main** and variable x in **addOne** are *separate* variables in memory. When **addOne** is called with x on line 9, it is the *value* of x (i.e. 5) that is *passed* to the function. This *value* is used to initialize a new variable x that exists only in **addOne**'s scope. Thus the value of the variable x in main's scope remains unchanged even after the function **addOne** has been called.

```
1     void addOne(int x){
2         x = x + 1;
3         cout << x << endl;
4     }
5
6     int main(){
7        int x = 5;
8        cout << x << endl;
9        addOne(x);
10       cout << x << endl;
11    }
```

Output:

```
5
6
5
```

- **Passing By Reference**

  Arrays, on the other hand, are *passed by reference* (to the original array's location in the computer's memory. So, when an array is passed as a parameter, the original array is used by the function.

  Observe that there is only *one* array X in memory for the following example. When the function **addOne** is called on line 9, a *reference* to the original array X is *passed* to **addOne**. Because the array X is *passed by reference*, any modifications done to X in **addOne** are done to the original array. These modifications persist and are visible even after the flow of control has exited the function and we return to main.

```
1     void addOne(int X[]){
2         X[0] = X[0] + 1;
3         cout << X[0] << endl;
4     }
5     int main(){
6        int X[4] = {1, 5, 3, 2};
7        cout << X[0] << endl;
8        addOne(X);
9        cout << X[0] << endl;
10    }
```

Output:

```
1
2
2
```

When we pass a one-dimensional array as an argument to a function we also provide its length. For two-dimensional arrays, in addition to providing the length (or number of rows), we will also assume that we know the length of each of the subarrays (or the number of columns). A function taking a two-dimensional array with 10 columns as an argument then might look something like this:

```
void twoDimensionalFunction(int matrix[][10], int rows){ … }
```

4.1.4 File Input/Output

During this class so far, we have been using the iostream standard library. This library provided us with methods like cout and cin. cin is a method is for reading from standard input (i.e. in the terminal via a keyboard) and cout is for writing to standard output.

In this part, we will discuss file input and output, which will allow you to read from and write to a file. In order to use these methods, we will need to use another C++ standard library, fstream (file stream).

These headers must be included in your C++ file before you are able to process files.

```
#include <iostream>
#include <fstream>
```

Reading lines from a file:
To read lines from a file, you need to take the following steps.

Step 1: Make a stream object. Create an object (a variable) of file stream type. If you want to open a file for reading only, then the ifstream object should be used ("input file stream").

Example:

```
//create an output file stream for writing to file
ifstream myfile;
```

Step 2: Open a file. Once you have an object (a variable) of file stream type, you need to open the file. To open the file, you use the method (function) named open. For ifstream objects, the method takes only one parameter: the file name as a string (surrounded by " " if giving file name directly)

Example:

```
//open the file file1.txt with the file stream
myfile.open("file1.txt");
```

Step 3: Checking for opening file. It is always good practice to check if the file has been opened properly and send a message if it did not open properly. To check if a file stream successfully opened the file, you can use `fileStreamObject.is_open()`. This method will return a Boolean value true if the file has successfully opened and false otherwise.

Example:

```
if (myfile.is_open()){
    // do things with the file
} else {
    cout << "file open failed" << endl;
}
```

Step 4: Read lines from the files. To read a line from the file, you can use `getline(filestream, line)` which returns true as long as an additional line has been successfully assigned to the variable line. Once no more lines can be read in, `getline` returns false. So we can set up a while loop where the condition is the call to `getline`.

Example:

```
string line = "";
int lineidx = 0;
// read each line from the file
while (getline(myfile, line)) {
    // print each line read from the file
    cout << lineidx << ": " << line << endl;
    // increment index(count of lines in the file)
    Lineidx++;
}
```

Step 5: Closing a file. When you are finished processing your files, it is recommended to close all the opened files before the program is terminated. The standard syntax for closing your file is `myfilestream.close();`

Example:

```
// closing the file
myfile.close();
```

Step 6: Putting it all together. Here is the temple for reading lines from a file

```
//create an output file stream for writing to file
ifstream myfile;
//open the file file1.txt with the file stream
```

```
myfile.open("file1.txt");
if (myfile.is_open()){
     // do things with the file
     string line = "";
     int lineidx = 0;
     // read each line from the file
     while (getline(myfile, line)) {
          // print each line read from the file
          cout << lineidx << ": " << line << endl;
          // increment index(count of lines in the file)
          Lineidx++;
     }
} else {
     cout << "file open failed" << endl;
}
// closing the file
myfile.close();
```

- **Other functions**

  Here are some useful functions for this week's assignment. We've provided links to documentation on how to use them. These pages provide explanation on the function as well as examples on how to use them:

  - `string.length()` (find the length of a string)
  - `string.empty()` (determine if a string is empty or not)
  - `getline` (extract characters from a stream and place into a string)
  - `fstream::open` (open a file)
  - `fstream::is_open` (determine if a file successfully opened)
  - `ios::fail` (determine if a file successfully opened)
  - `ios::eof` (determine if at the end of a file)
  - `stoi` (convert string to integer)
  - `stod` (convert string to double)

5. Problem Set

Note: To stay on track for the week, we recommend to finish/make considerable progress on Problems 1-4 by Wednesday. Students with recitation on Thursday are encouraged to come to recitation with questions and have made a start on* all *of the problems.

The first 4 problems are to get some practice working with arrays. You will turn in a separate .cpp file for each of the first 4 problems. The remaining 5 problems are to get a running start on Project 2, wherein we will develop a book recommendation system (description below). You will turn in a version of the function that is tested in its own program, as well as *one single* .cpp program that includes all of Problems 5-9. All 9 problems are required. Note however that there are still separate Coderunner problems on Moodle for testing your codes.

5.1. Underhand tosses

Problem 1 (5 points) arrayPilgrimage

Write a program `arrayPilgrimage.cpp` to **declare and populate** the four arrays listed below. For testing in Coderunner, you will paste the `main()` function from your program.

- `temps` - an array of 10 floating point numbers initialized with -459.67 (absolute zero in Fahrenheit)
- `colors` - an array of the strings “Red”, “Blue”, “Green”, “Cyan”, and “Magenta”, in that order.
- `sequence` - an array of the first 100 positive integers in order; 1, 2, 3, 4, … etc.
- `letters` - an array of all uppercase and lowercase characters in order, A, a, B, b, C, c, … **Hint:** the ASCII table will be helpful here!

In order to test that you correctly created and populated the arrays, print the values of each of their elements, in order, one element per line. **Hint:** use a loop to traverse each array.

Expected output:

```
-459.67
-459.67
…
-459.67
```

```
Red
Blue
…
Magenta

1
2
…
100

A a
B b
…
Z z
```

In Cloud9 the file should be called **arrayPilgrimage.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 1, in the Answer Box, paste **the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 2 (10 points) floodMap

Write a function **floodMap** which accepts three input arguments, in this order: (1) a two-dimensional **array of double.** Each element of the array indicates the height of the terrain at a particular point (assume that there are 4 columns). (2) An **integer** indicating the number of rows in the map, and (3) a **double** indicating the current water level. The function should print out a "map" of which points in the array are below the water level. In particular, each position at or below the water level will be represented with a * and each position above the water level will be represented with an underscore _.

- Your function should be named **floodMap**
- Your function should take three parameters, in this order:
    - a two-dimensional **array of double** with 4 columns,
    - the **integer** number of rows in the array, and
    - a **double** indicating the current water level.
- Your function should print a flood map as described below.
- Your function should not return anything

Examples:

Function call	Output
`double map[1][4] = {{5.0, 7.6, 3.1, 292}};` `floodMap(map, 1, 6.0);`	`*_*_`

`double map[2][4] = {{0.2, 0.8, 0.8, 0.2},`
`{0.2, 0.2, 0.8, 0.5}};`
`floodMap(map, 2, 0.0);`
	`____`
	`____`

`double map[2][4] = {{0.2, 0.8, 0.8, 0.2},`
`{0.2, 0.2, 0.8, 0.5}};`
`floodMap(map, 2, 0.5);`
	`*__*`
	`**_*`

`double map[2][4] = {{0.2, 0.8, 0.8, 0.2},`
`{0.2, 0.2, 0.8, 0.5}};`
`floodMap(map, 2, 1.0);`
	`****`
	`****`

`double map[4][4] = {{1.0, 5.0, 1.0, 1.0},`
`{1.0, 5.0, 5.0, 1.0},`
`{5.0, 1.0, 5.0, 5.0},`
`{1.0, 1.0, 1.0, 1.0}};`
`floodMap(map, 4, 3.14);`
	`*_**`
	`*__*`
	`_*__`
	`****`

In Cloud9 the file should be called **floodMap.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 2, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 3 (10 points) split

The **split** function from Homework 5 was such a blast, let's do it again, but with an array this time! Write a *new* function **split** which takes four input arguments: a string to be split, a character to split on ("a delimiter"), an array of strings to fill with the split pieces

of the input string, and an integer representing the maximum number of split string pieces. The function will split the input string in to pieces separated by the delimiter, and populate the array of strings with the split pieces up to the provided maximum number of pieces. Your function will return the number of pieces the string was split into.

- Your function should be named **split**
- Your function takes four input arguments:
  - The **string** to be split.
  - A delimiter **character**, which marks where the above string should be split up.
  - An **array of string**, which you will use to store the split-apart string pieces.
  - The **int** length of the given array
- Your function returns the number of pieces the input string was split into as an integer.
- Your function does not print anything.
- If the input string is split into more pieces than the array of string can hold (more than the indicated length), your function should fill only as many words as it can, and return -1.
- **Note:** *On Sunday Feb 24, the solution to the **split()** function from Homework 5 will be posted.*

Example: in the following examples, `printArray` function will output all the elements in the array, one per line.

Function calls	Output

`string words[6];`
`cout << split("one small step",' ',words,6) << endl;`
`printArray(words);`
	`3`
	`one`
	`small`
	`step`

`string words[6];`
`cout << split(" one small step  ",' ',words,6) << endl;`
`printArray(words);`
	`3`
	`one`
	`small`
	`step`

`string words[6];`
`cout << split("cow/big pig/fish",'/',words,6) << endl;`
`printArray(words);`
	`3`
	`cow`
	`big pig`
	`fish`

`string words[6];`
`cout << split("cow/big pig//fish",'/',words,6) << endl;`
`printArray(words);`
	`3`
	`cow`
	`big pig`

	`fish`

`string words[6];`
`cout << split("unintentionally",'n',words,6) << endl;`
`printArray(words);`
	`5`
	`u`
	`i`
	`te`
	`tio`
	`ally`

`string words[2];`
`cout << split("one small step",' ',words, 2) << endl;`
`printArray(words);`
	`-1`
	`one`
	`small`

In Cloud9 the file should be called **split.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 3, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 4 (15 points) getLinesFromFile

Write a function called **getLinesFromFile** that reads from a text file and stores its contents in an array. Each line in the file will either contain a single integer or be empty.

- Your function should be named **getLinesFromFile**
- Your function should take three parameters in the following order:
  - A **string** fileName
  - An array of integers **int arr[]**
  - The **int length** of the given array
- Your function will open and read the file specified by the **fileName** parameter. This string should include both the name and extension of the file (what type of file it is). There are some good and bad examples below:
  - Good: "books.txt"
  - Bad: "authors"
- Your function should read the file line by line, and at each line if there is a number it should store it in the array specified by the **arr[]** parameter.

- If a line is empty it should continue reading until it sees another integer before adding to the array. Empty lines should not be added to the array.
- Once the array is full (**length** integers have been added), subsequent integers in the file should be ignored.
- If your function has added any integers to the array (meaning that the specified file exists and could be read), it should return the number of integers added to the array.
- If the file does not exist, return -1.
- Your function **does not** print anything

Example:

fileName.txt	`1`
	`5`
	`23`
	
	`18`

Function call	`int arr[4];`
	`cout << getLinesFromFile("fileName.txt", arr, 4) << endl;`
	`printArray(arr);`

Output	`4`
	`1`
	`5`
	`23`
	`18`

Value of `arr`	`{1, 5, 23, 18}`

In Cloud9 the file should be called **getLinesFromFile.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 4, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

5.2. Part Two: Fast-pitch this time

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 2009, 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. Rabia loves a good book! How can we write program that can predict books she that might like?

The simplest approach would be to simply calculate the average rating for all of 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. In this case, the program would make almost the same prediction for every customer. 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 that this friend recommends.

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

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. If another

user, Suelyn, is most similar to Rabia, we would recommend to Rabia the top books from Suelyn's list that Rabia hadn't already rated.

Your task for this preliminary part of Project 2 is to develop logistical methods for initializing the database and manipulating its contents. In upcoming assignments, you will exploit these operations for making recommendations.

Problems leading up to Project 2

For each of Problems 5-8, you will turn in a version of the function that is tested in its own program, as well as all together in Problem 9. Problem 9 builds off of a template **HW6.cpp** program (provided on Moodle), which will include your functions from Problems 5-8.

Problem 5 (20 points) readBooks

Write a function **readBooks** that populates a pair of arrays with the titles and authors found in `books.txt`. This function should:

- Accept five input arguments in this order:
  - **string** `fileName`: the name of the file to be read.
  - **Array of string**: `titles`.
  - **Array of string**: `authors`.
  - **int** `numBookStored`: 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** `size`: capacity of the `authors`/`titles` arrays. This number should always be greater or equal to the number of books currently stored in the arrays.
- Use ifstream and getline to read data from the file, placing author names in the `authors` array and titles in the `titles` array.
- You can use the `split()` function from Problem 3, with comma (‘,’) as the delimiter. When you copy your code to the coderunner, make sure you put in the answer box both functions **readBooks** and **split**.
- Special cases to consider:
  - When the file is not opened successfully, return -1.
  - When `numBookStored` is equal to the `size`, return -2.
  - When `numBookStored` is smaller than `size`, keep the existing elements in `titles` and `authors`, then read data from file and add the data to the array. The number of books stored in the array cannot exceed to the `size` of the array.

- Empty lines should not be added to the arrays.

Example 1: The `authors` and `titles` arrays are empty, so `numBookStored` is zero.

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

Function call	`string authors[10] = {};`
	`string titles[10] = {};`
	`readBooks("fileName.txt",titles,authors, 0, 10);`

Return value	`2`
Value of `authors`	`{"Author A", "Author B"}`
Value of `titles`	`{"Book 1", "Book 2"}`

Example 2: There's already one book and one author in the arrays, so the data read from the file will be added to the array.

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

Function call	`string authors[10] = {"Author Z"};`
	`string titles[10] = {"Book N"};`
	`readBooks("fileName.txt",titles,authors, 1, 10);`

Return value	`3`
Value of `authors`	`{"Author Z", "Author A", "Author B"}`
Value of `titles`	`{"Book N", "Book 1", "Book 2"}`

Example 3: There's already one book and one author in the arrays, so `numBookStored` is one. However, the array size is only two, so only the first line of the file is stored.

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

Function call	`string authors[2] = {"Author Z"};`
	`string titles[2] = {"Book N"};`
	`readBooks("fileName.txt", titles, authors, 1, 2);`

Return value	`2`
Value of `authors`	`{"Author Z", "Author A"}`
Value of `titles`	`{"Book N", "Book 1"}`

Example 4: file does not exist.

Function call	`string authors[2] = {"Author Z"};`
	`string titles[2] = {"Book N"};`
	`readBooks("badFileName.txt",titles,authors, 1, 2);`

Return value	`-1`
Value of `authors`	`{"Author Z"}`
Value of `titles`	`{"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	`string authors[1] = {"Author Z"};`
	`string titles[1] = {"Book N"};`
	`readBooks("fileName.txt", titles, authors, 1, 1);`

Return value	`-2`
Value of `authors`	`{"Author Z"}`
Value of `titles`	`{"Book N"}`

In Cloud9 the file should be called **readBooks.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle. After developing in Cloud9, this function will be one of the functions needed at the top of **HW6.cpp**.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 5, in the Answer Box, paste **only your function definition, not the entire program**.

Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 6 (15 points) printAllBooks

It will be useful to display the contents of your library. Next, make a function **printAllBooks** that meets the following criteria:

- Accept three arguments in this order:
  - **Array of string**: `titles`
  - **Array of string**: `authors`
  - **int**: number of 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 << titles[i] << " by " << authors[i] << endl;
```

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

In Cloud9 the file should be called **printAllBooks.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle. After developing in Cloud9, this function will be one of the functions needed at the top of **HW6.cpp**.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 6, in the Answer Box, paste **only your function definition, not the entire program**.

Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 7 (30 points) readRatings

Write a function **readRatings** that loads 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
...
```

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

Your function should:

- Accept six arguments in this order:
  - **string**: the name of the file to be read
  - **Array of string**: `users`

    - 2 dimensional **int array**: `ratings` - list of ratings for each user.The number of rows corresponds to the number of users, and the number of columns corresponds to the number of books.
    - **int**: `numUsers` number of users currently stored in the arrays
    - **int**: `maxRows` maximum number of rows of the `ratings` 2D array (convention: `array[row][column]`) *[assume to be 100]*
    - **int**: `maxColumns` maximum number of columns of the `ratings` 2D array *[assume to be 50]*
- Use `ifstream` and `getline` to read data from the file, placing each username in the `users` array, and each set of ratings in a row of the `ratings` 2D array.
- **Hint**: You can use the `split()` - function from problem 3, with comma (“,”) as the delimiter. When you copy your code in the answer box on Moodle Coderunner, make sure you put both the functions `readRatings` and `split`.
- 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 `numUsers` is equal to the maximum number of rows in the `ratings` 2D array, return -2
    - When `numUsers` is smaller than the maximum number of rows in the `ratings` 2D array, keep the existing elements in `users` array and `ratings` 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 arrays.
    - Empty lines should not be added to the arrays.

Example 1: The `users` and `ratings` arrays are empty, so `numUsers` is 0.

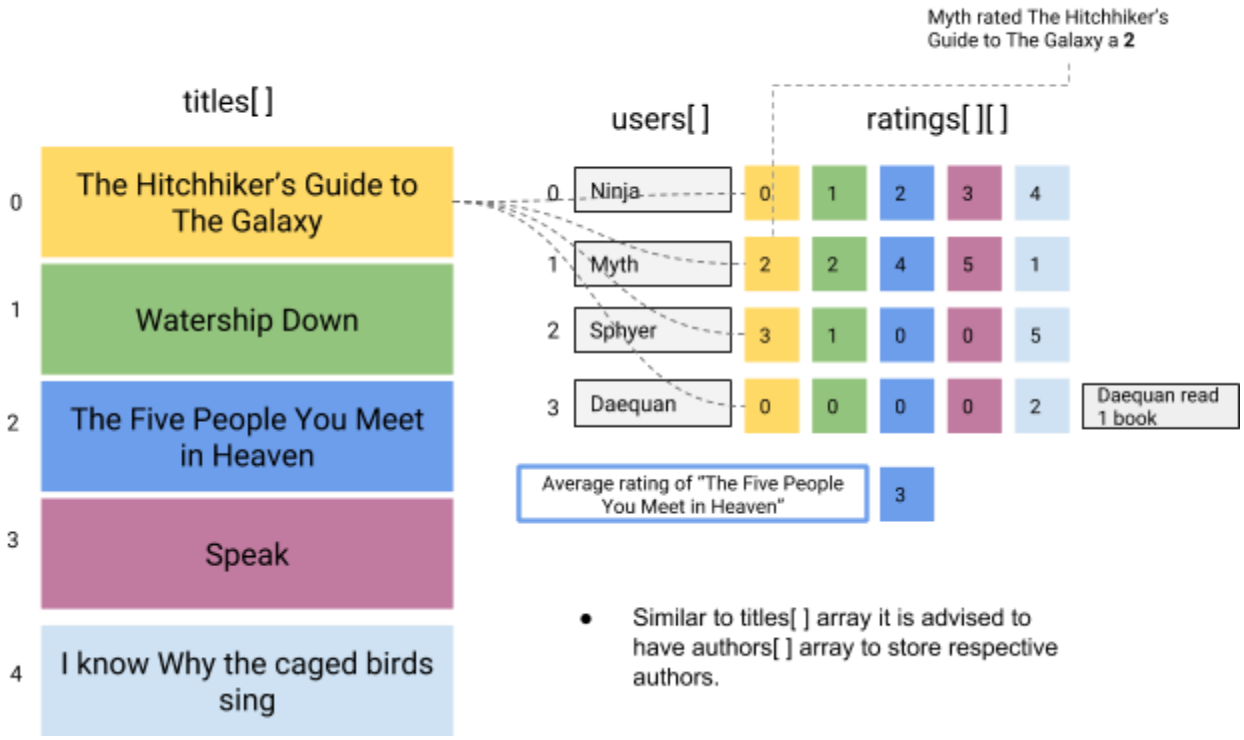
ratings.txt	`Ninja,0,1,2,3,4`
	`Myth,2,2,4,5,1`
	`Sphyer,3,1,0,0,5`
	`Daequan,0,0,0,0,2`

Function call	`string users[10] = {};`
	`int ratings[10][50] = {{0}};`
	`int numUsers = 0;`
	`int maxRows = 10;`
	`int maxColumns = 50;`
	`readRatings("ratings.txt", users, ratings,`
	`numUsers, maxRows, maxColumns);`

Return value	`4`
Value of `users`	`{"Ninja", "Myth", "Sphyer", "Daequan", "", "", "", "", "", ""}`

Value of `ratings`	`{`
	`  {0, 1, 2, 3, 4, …, 0, 0},`
	`  {2, 2, 4, 5, 6, …, 0, 0},`
	`  {3, 1, 0, 0, 5, …, 0, 0},`
	`  {0, 0, 0, 0, 2, …, 0, 0},`
	`  {0, 0, 0, 0, 0, …, 0, 0},`
	`  {0, 0, 0, 0, 0, …, 0, 0},`
	`  {0, 0, 0, 0, 0, …, 0, 0},`
	`  {0, 0, 0, 0, 0, …, 0, 0},`
	`  {0, 0, 0, 0, 0, …, 0, 0},`
	`}`

Visualization of various elements in HW6



Example 2: The `authors` and `titles` arrays are empty, so `numBookStored` is 0 and a bad file is given

Function call	`string users[] = {};`
	`int ratings[10][50] = {{0}};`
	`int numUsers = 0;`
	`int maxRows = 10;`
	`int maxColumns = 50;`
	`readRatings("badFile.txt", users, ratings, numUsers, maxRows, maxColumns);`

Return value	`-1`
Value of `users`	`{"", "", "", "", "", "", "", "", "", ""}`

Value of `ratings`	`{`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`  {0, 0, …, 0, 0},`
	`}`

Example 3: The `users` and `ratings` arrays are 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	`string users[4] = {"Ninja", "Myth", "Sphyer", "Daequan"};`
	`int ratings[4][50] = {{0, 1, 2, 3, 4, …, 0, 0},`
	`{2, 2, 4, 5, 6, …, 0, 0},`
	`{3, 1, 0, 0, 5, …, 0, 0},`
	`{0, 0, 0, 0, 2, …, 0, 0}};`
	
	`int numUsers = 4;`
	`int maxRows = 4;`
	`int maxColumns = 50;`

	`readRatings("moreRatings.txt",users, ratings, numUsers, maxRows, maxColumns);`
Return value	`-2`
Value of `users`	`{"Ninja", "Myth", "Sphyer", "Daequan"}`

Value of `ratings`	`{`
	`  {0, 1, 2, 3, 4, …, 0, 0},`
	`  {2, 2, 4, 5, 6, …, 0, 0},`
	`  {3, 1, 0, 0, 5, …, 0, 0},`
	`  {0, 0, 0, 0, 2, …, 0, 0}`
	`}`

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 call	`string users[2] = {"ritchie"};`
	`int ratings[2][50] = {{0, 1, 2, 0, 0, …, 0, 0}`
	`  {0, 0, 0, 0, 0, …, 0, 0}};`
	`int numUsers = 1;`
	`int maxRows = 2;`
	`int maxColumns = 50;`
	`readRatings("ratings.txt", users, ratings, numUsers, maxRows, maxColumns);`

Return value	`2`
Value of `users`	`{"ritchie", "stroustrup"}`

Value of `ratings`	`{`
	`  {0, 1, 2, 0, 0, …, 0, 0},`
	`  {0, 4, 5, 0, 0, …, 0, 0},`
	`}`

In Cloud9 the file should be called **readRatings.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle. After

developing in Cloud9, this function will be one of the functions needed at the top of **HW6.cpp**.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 7, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 8 (25 points) getRating

We now have a list of usernames and a list of ratings. Now, we want 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 7 input arguments in the following order:
  - **string**: username
  - **string**: title of the book
  - **Array of string**: `users`
  - **Array of string**: `titles`
  - 2-Dimensional **int array**: `ratings` - list of ratings for each user *[assume it has 100 rows and 50 columns]*
  - **int**: number of users whose names/ratings are currently stored in the string array/2D array respectively
  - **int**: number of books whose titles/ratings are currently stored in the string array/2D array respectively

- The username and title search should be case insensitive. "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

Value of `users`	`string users[2] = {"User1", "User2"};`
Value of `titles`	`string titles[3] = {"Title1", "Title2", "Title3"};`

Value of `ratings`	`int ratings[2][3] = {{1,4,2},{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, titles, ratings, 2, 3);`
Return value	`4`

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

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

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

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

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

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

In Cloud9 the file should be called **getRating.cpp** and it will be one of 9 files you need to zip together for the **Homework 6 (File Submission)** on Moodle. After developing in Cloud9, this function will be one of the functions needed at the top of **HW6.cpp**.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 8, in the Answer Box, paste **only your function definition, not the entire program**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

Problem 9 (20 points) put it all together

Menu functionality has been provided for you in the starter code on Moodle in the file **HW6.cpp** . Download the file and fill in the missing parts in the code which are indicated by comments in the `main()` function.

Note: the function definitions for Problems 3, 5, 6, 7 and 8 will go in this file as well. For Problem 9, you need to submit the the entire program HW6.cpp 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 above (Problems 3 and 5-8), call them, and process the values they return.

- In your driver function, you must declare your arrays with the appropriate size. The size of `titles` and `authors` array is 50. The size of `users` array is 100, and the `ratings` array is 2 dimensional where `maxRows` is 100 (max number of users) and `maxColumns` is 50 (max number of books).
    - In your template function the sizes have been declared. Match them accordingly
        - `const static int  ratingArrSize = 100;`
        - `const static int  bookArrSize = 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 `HW6` 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!
```

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner**. For Problem 9, in the Answer Box, **you need to submit the the entire program HW6.cpp**. Press the Check button. You can modify your code and re-submit (press Check again) as many times as you need to.

6. Homework 6 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 6 CodeRunner**
2. Submit one zip file to **Homework 6 (File Submission)**. The zip file should be named, **<firstName>_<lastName>_homework6.zip**. It should have following 9 files:
    - **arrayPilgrimage.cpp**
    - **floodMap.cpp**
    - **Split.cpp**
    - **getLinesFromFile.cpp**
    - **readBooks.cpp**
    - **printAllBooks.cpp**
    - **readRatings.cpp**
    - **getRating.cpp**
    - **HW6.cpp**

7. Homework 6 point summary

Criteria	Pts
CodeRunner (problem 1 - 9)	150
Style and Comments	5
Algorithms	5
Test cases	20
Recitation attendance (Feb 26 or Feb 28)*	-30
Using global variables	-5
Total	180
5% early submission bonus	+5%

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