CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 5
Due: Saturday, October 12th, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm Oct. 11th)

2 components (Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, October 12th, 6:00 pm for your homework to receive points.

---

0. Table of Contents

---

1. Objectives

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

---

2. Background

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();
```

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.. 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.

---

3. Submission Requirements

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

1. ***Work on questions on your Cloud 9 workspace:*** You need to write your code on Cloud 9 workspace to solve questions and test your code on your Cloud 9 workspace before submitting it to Moodle. (Create a directory called **hmwk5** and place all your file(s) for this assignment in this directory to keep your workspace organized)

2. ***Submit to the Moodle CodeRunner:*** Head over to Moodle to the link **Homework 5 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 7 questions from the Moodle assignment, zip all 7 solution files you compiled in Cloud9 (one cpp file for each problem), and submit the zip file through the **Homework 5** link on Moodle.

---

4. Rubric

Aside from the points received from the **Homework 5 CodeRunner** quiz problems, your TA will look at your solution files (zipped together) as submitted through the **Homework 5** link on Moodle and assign points for the following:

Style, Comments, Algorithm (10 points):

Style:

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

Comments:

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

```
// CS1300 Fall 2019
// Author: my name
// Recitation: 123 – Favorite TA
// Homework 5 - Problem # ...
```

Algorithm:

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

Example 1:

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

Example 2:

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

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

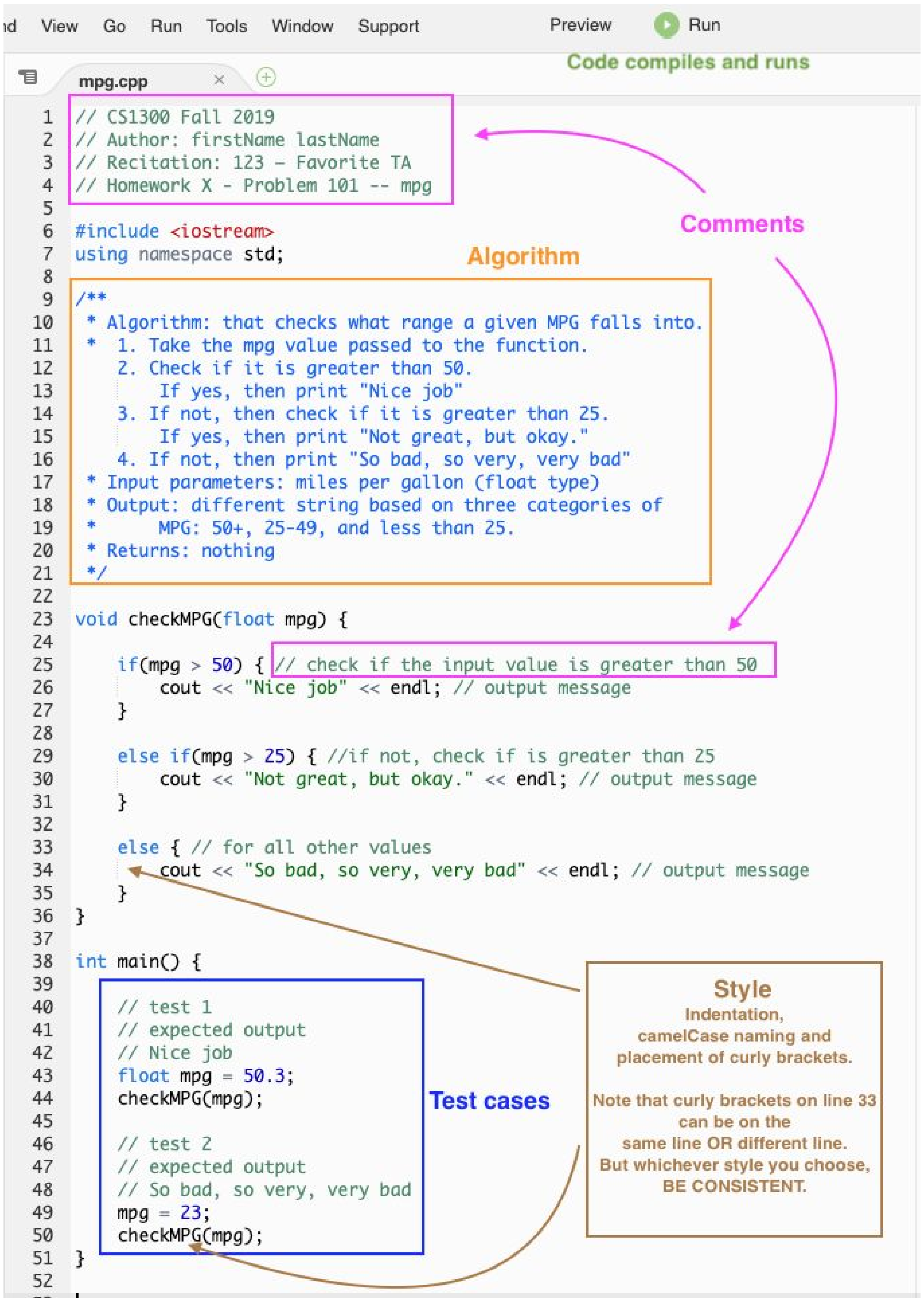
```
/*
 * conversion
 */
double convertUSDtoEuros(double dollars)
{
     double euros = dollars * 0.86;
     return euros;
}
```

Test Cases (20 points)

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

2. *Test cases* (**14 points**):
   For this week's homework, all 7 problems are asking you to create a function. In your solution file for each function, you should have at least 2 test cases present in their respective `main()` function, for a total of 14 test cases (see the diagram on the next page). If you created your own files as the test cases, please include and submit them.

Your test cases should follow the guidelines, **Writing Test Cases**, posted on Moodle under Week 3.



Indentation, camelCase naming and placement of curly brackets.

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

```cpp
// CS1300 Fall 2019
// Author: firstName lastName
// Recitation: 123 – Favorite TA
// Homework X - Problem 101 -- mpg

#include <iostream>
using namespace std;

/**
 * Algorithm: that checks what range a given MPG falls into.
 *   1. Take the mpg value passed to the function.
    2. Check if it is greater than 50.
        If yes, then print "Nice job"
    3. If not, then check if it is greater than 25.
        If yes, then print "Not great, but okay."
    4. If not, then print "So bad, so very, very bad"
 * Input parameters: miles per gallon (float type)
 * Output: different string based on three categories of
 *      MPG: 50+, 25-49, and less than 25.
 * Returns: nothing
 */

void checkMPG(float mpg) {

    if(mpg > 50) { // check if the input value is greater than 50
        cout << "Nice job" << endl; // output message
    }

    else if(mpg > 25) { //if not, check if is greater than 25
        cout << "Not great, but okay." << endl; // output message
    }

    else { // for all other values
        cout << "So bad, so very, very bad" << endl; // output message
    }
}

int main() {

    // test 1
    // expected output
    // Nice job
    float mpg = 50.3;
    checkMPG(mpg);

    // test 2
    // expected output
    // So bad, so very, very bad
    mpg = 23;
    checkMPG(mpg);
}
```

5. Problem Set

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

Problem 1 (5 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
  - 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];` `int x = getLinesFromFile("fileName.txt", arr, 4);` `cout << "Function returned: " << x << "\n";` `printArray(arr);`
Output	`Function returned: 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 the files you need to zip together for the **Homework 5** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 CodeRunner**. For Problem 1, 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 2 (10 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 strings**: `titles`.
  - **Array of strings**: `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 homework 4, 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**.
- Your function should return the number of books stored in the array, after you populate them.
- Empty lines should not be added to the arrays.
- Unexpected values might be passed into the function and your function should be able to handle them without crashing the program:

- Case1: When `numBookStored` is equal to the `size`, return `-2`.
- Case2: When the file is not opened successfully, return `-1`.
- Case3: 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 the `size` of the array. The function returns the size of the array when the array is filled while reading a file.
- Your function must check these cases in the order specified above.

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 //1 book was already present, and we added 2 more`
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 the files you need to zip together for the **Homework 5** on Moodle.

Don’t forget to head over to Moodle to the link **Homework 5 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): 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 strings**: `titles`
    - **Array of strings**: `authors`
    - **int**: number of books
- This function does **not** return anything
- If the number of books is 0 or less than 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.

Example 1: `printAllBooks` is called with arrays of books and authors.

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

Function call	`string authors[10] = {};`
	`string titles[10] = {};`
	`int nb = readBooks(“books.txt”,titles,authors, 0, 10);`
	`printAllBooks(titles, authors, nb);`

Expected output	`Here is a list of books`
	`Book 1 by Author A`
	`Book 2 by Author B`

Example 2: no books stored in the arrays

Function call	`string authors[10] = {};`
	`string titles[10] = {};`
	`printAllBooks(titles, authors, 0);`

Expected output	`No books are stored`

In Cloud9 the file should be called **printAllBooks.cpp** and it will be one of the files you need to zip together for the **Homework 5** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 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 (10 points): printBooksByAuthor

Some user wants to know if there are books by their favorite author. So let's write a function **printBooksByAuthor** that should do the following things:

- Your function should be named **printBooksByAuthor**
- Your function should take four parameters in the following order:
    - **Array of strings**: `titles`
    - **Array of strings**: `authors`
    - **int**: `number of books`
    - **string:** `author name`
- You function should **not return** anything.
- You can assume that the number of elements in both arrays is the same. The number of books represents the number of elements in each array.
- If the number of books is 0 or less than 0, print "`No books are stored`"
- If there are no books by the author then you should print the following, "`There are no books by <author>`"
- If there are one or multiple books by the same author, you should print the following statement, "`Here is a list of books by <author>`" followed by, each book's title by this author in a new line.

Example 1: There are two books by `Author A`.

Function Call:

```
string titles[3] = {"Book 1", "Book 2", "Book 3"};
string authors[3] = {"Author A", "Author B", "Author A"
int numBooks = 3;
string author = "Author A";
printBooksByAuthor(titles, authors, numBooks, author);
```

Expected Output:

```
Here is a list of books by Author A
Book 1
Book 3
```

Example 2: Both the arrays are empty and `numBooks` is 0.

Function Call:

```
string titles[5] = {};
string authors[5] = {};
int numBooks = 0;
string author = "Author A";
```

	`printBooksByAuthor(titles, authors, numBooks, author);`
Expected Output:	`No books are stored`

Example 3: There are no books by the `Author A`.

Function Call:	`string titles[3] = {"Book 1", "Book 2", "Book 3"};`
	`string authors[3] = {"Author B", "Author C", "Author D"`
	`int numBooks = 3;`
	`string author = "Author A";`
	`printBooksByAuthor(titles, authors, numBooks, author);`

Expected Output:	`There are no books by Author B`

In Cloud9 the file should be called **printBooksByAuthor.cpp** and it will be one of the files you need to zip together for the **Homework 5** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 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.

Problem 5 (20 points): floodMap

Write a function **floodMap** that prints out a "map" of which points in the array are below or above the water level.The function takes a two-dimentional array where each element indicates the height of the terrain at a particular point (assume that there are 4 columns). Each point 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 not return anything
- Your function should print a flood map as described below.

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 the files you need to zip together for the **Homework 5** on Moodle.

Don't forget to head over to Moodle to the link **Homework 5 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 (20 points): readFloatMap

Write a function called **readFloatMap** that reads from a text file, stores its contents in a 2D array, and returns the number of lines it reads.

- Your function should be named **readFloatMap**
- Your function should take three parameters in the following order:
  - A **string** `fileName`
  - A **2D array of doubles with 4 columns** `double arr[][4]`
  - The **int** number of rows 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. Each line contains 4 floating point numbers which should be stored in the corresponding columns of the 2D array.
- If your function has added any rows to the array (meaning that the specified file exists and could be read), it should return the number of rows added to the array.
- If the file does not exist, return -1.
- Your function does not print anything

Example 1: The file exists and the function can read.

fileName.txt	`23.5,34.3,1.3,2.3`
	`1.3,0.23,11.5,6.7`

Function call	`double floatMap[2][4];`
	`readFloatMap("fileName.txt", floatMap, 2);`

Return value	`2`

Value of `floatMap`	`{{23.5,34.3,1.3,2.3},`
	` {1.3,0.23,11.5,6.7}}`

Example 1: The file exists and the function can read.

Function call	`double floatMap[2][4];`
	`readFloatMap("notThisFile.txt", floatMap, 2);`

Return value	`-1`

In Cloud9 the file should be called **readFloatMap.cpp** and it will be one of the files you need to zip together for the **Homework 5** on Moodle.

Don’t forget to head over to Moodle to the link **Homework 5 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(25 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 strings**: `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 homework 4, 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.
- Empty lines should not be added to the arrays (i.e. if no ratings are listed).
- The function should return the following values depending on cases:
    - Case1: Return the total number of users in the system, as an integer.
    - Case2: When `numUsers` is equal to the maximum number of rows in the `ratings` 2D array, return -2
    - Case3: 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.
    - Case4: If the file cannot be opened, return -1
    - Your function must check these cases in the order specified above.

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`

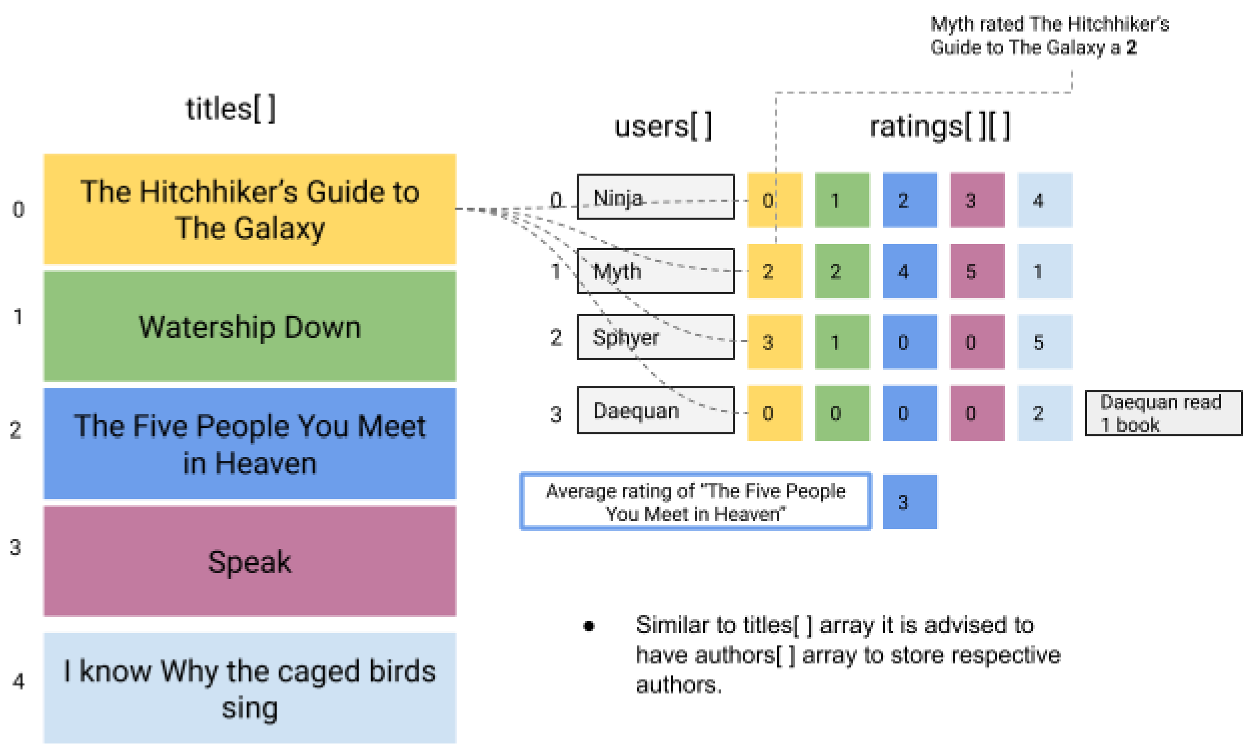
Function call	`string users[10] = {};`
	`int ratings[10][50] = {{0}};`
	`int numUsers = 0;`
	`int maxRows = 10;`
	`int maxColumns = 50;`
	`readRatings(“ratings.txt”, users, ratings, numUsers, maxRows, maxColumns);`

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},`
	`{0, 0, 0, 0, 0, …, 0, 0},`
	`}`

Visualization of various elements in HW5


Myth rated The Hitchhiker's Guide to The Galaxy a **2**
titles[ ]
users[ ]
ratings[ ][ ]
0 The Hitchhiker's Guide to The Galaxy
1 Watership Down
2 The Five People You Meet in Heaven
3 Speak
4 I know Why the caged birds sing
0 Ninja 0 1 2 3 4
1 Myth 2 2 4 5 1
2 Sphyer 3 1 0 0 5
3 Daequan 0 0 0 0 2
Daequan read 1 book
Average rating of "The Five People You Meet in Heaven" 3

- Similar to titles[ ] array it is advised to have authors[ ] array to store respective authors.

Example 2: The `authors` and `titles` arrays are empty, so `numUsers` 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 the files you need to zip together for the**Homework 5** on Moodle.

Don’t forget to head over to Moodle to the link **Homework 5 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.

6. Homework 5 checklist

Here is a checklist for submitting the assignment:

1. Complete the code **Homework 5 CodeRunner**
2. Submit one zip file to **Homework 5**. The zip file should be named, **hmwk5_lastname.zip**. It should have the following 7 files:
    - getLinesFromFile.cpp
    - readBooks.cpp
    - printAllBooks.cpp
    - printBooksByAuthor.cpp
    - floodMap.cpp
    - readFloatMap.cpp
    - readRatings.cpp

7. Homework 5 points summary

Criteria	**Pts**
CodeRunner (problem 1 - 7)	100
Style, Comments, Algorithms	10
Test cases	20
Recitation attendance (week 6)*	-30
Total	130
5% early submission bonus	+5%

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