CSCI 1300 CS1: Starting Computing
Ashraf, Fleming, Correll, Cox, Fall 2019
Homework 6
Due: Saturday, October 19th, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm Oct. 18th)

2 components (Moodle CodeRunner attempts, and zip file) must be completed and submitted by Saturday, October 19th, 6:00 pm for your homework to receive points.

---

0. Table of Contents

---

1. Objectives

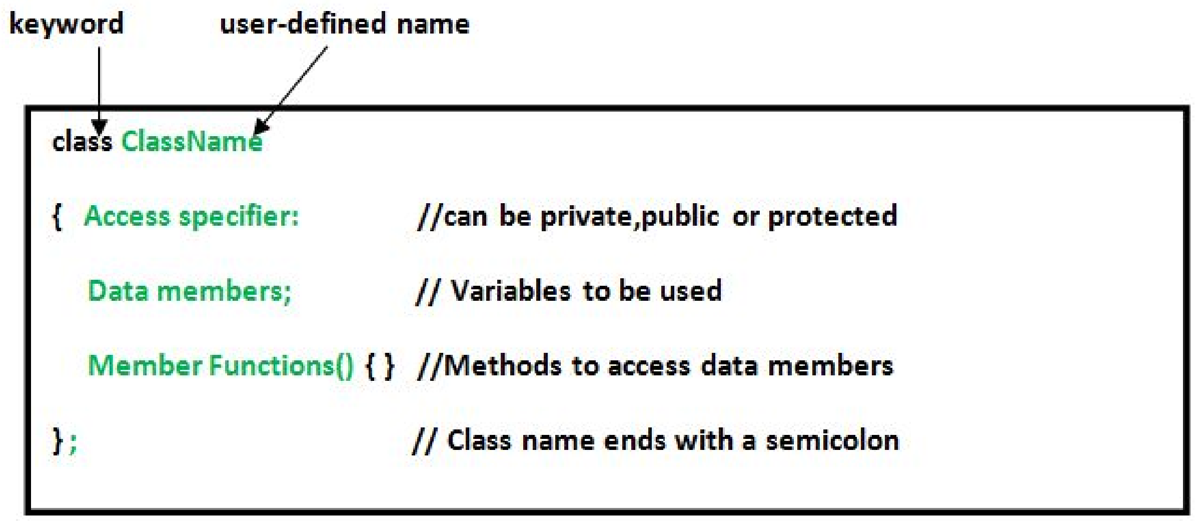
- Use filestream objects to read data from text files
- Array operations: initialization, search
- Improve proficiency with loops and strings
- Get started with part of your Project 2

---

2. Background

Classes

When writing complex programs, sometimes the built in data types (such as *int*, *char*, *string*) don't offer developers enough functionality or flexibility to solve their problems. A solution 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 the 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, they restrict access by outside functions to the class members. Consider the following analogy. Imagine an intelligence agency like the CIA, that has 10 senior members. Such an organization holds various sorts of information, and needs some way of determining who has access to any given piece of information.

Some information concerning classified or dangerous operations may only be accessible to the 10 senior members of the agency, and not directly accessible by any other person. 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 and not directly accessible by any object or function outside the class.

Some other information may be available to anyone who wants to know about it. This is **public** data. Even people outside the CIA can know about it, and the agency might release this information through press releases or other means. 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, even other classes. 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 define the properties and behavior of the objects in a Class. The data members declared in any class definition are fundamental data types (like int, char, float etc). For example, for strings objects, the data member is a **`char array[]`** that holds all of the individual letters of your string. Some of a string's member functions are functions like `.substr()`, `.find()`, and `.length()`.

Accessing Data Members

Keeping with our intelligence agency example, the code below defines 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, cannot access the `classifiedIntelligence` data member in the same way. Not everyone has access to that private data. Instead, we need to use a member function of the agency `getClassifiedIntelligence()` in order to see that information. However, we may write this function remove sensitive information from the string. This allows us to control the release of private information 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

You may have noticed 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 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 a **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 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.

className.h

```
#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 hmwk6. 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:

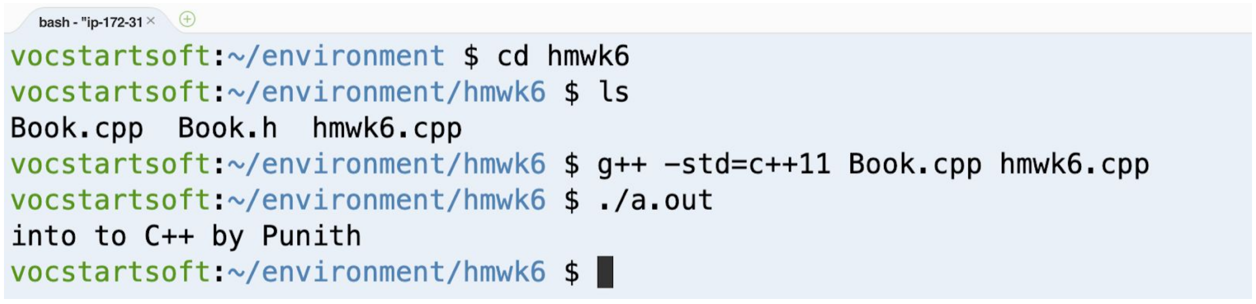
```
g++ -o myName.out -std=c++11 file1.cpp file2.cpp main.cpp
```

And then run the file with

```
./myName.out
```

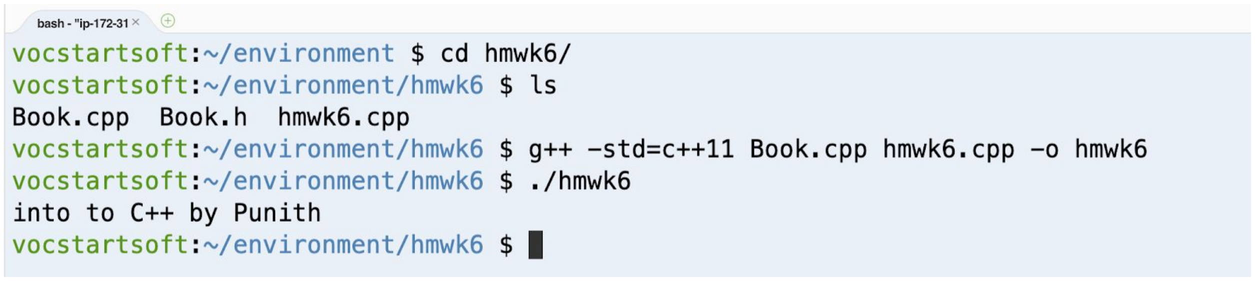
Example 1

Compiling book.cpp and Hmwk6.cpp. The picture below shows a **bash** window.



```
vocstartsoft:~/environment $ cd hmwk6
vocstartsoft:~/environment/hmwk6 $ ls
Book.cpp  Book.h  hmwk6.cpp
vocstartsoft:~/environment/hmwk6 $ g++ -std=c++11 Book.cpp hmwk6.cpp
vocstartsoft:~/environment/hmwk6 $ ./a.out
into to C++ by Punith
vocstartsoft:~/environment/hmwk6 $ █
```

Example 2
Here, we have named the executable hmwk6



```
vocstartsoft:~/environment $ cd hmwk6/
vocstartsoft:~/environment/hmwk6 $ ls
Book.cpp  Book.h  hmwk6.cpp
vocstartsoft:~/environment/hmwk6 $ g++ -std=c++11 Book.cpp hmwk6.cpp -o hmwk6
vocstartsoft:~/environment/hmwk6 $ ./hmwk6
into to C++ by Punith
vocstartsoft:~/environment/hmwk6 $ 
```

---

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 **hmwk6** 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 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 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 6** link on Moodle.

---

4. 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** 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 6 - 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
}
```

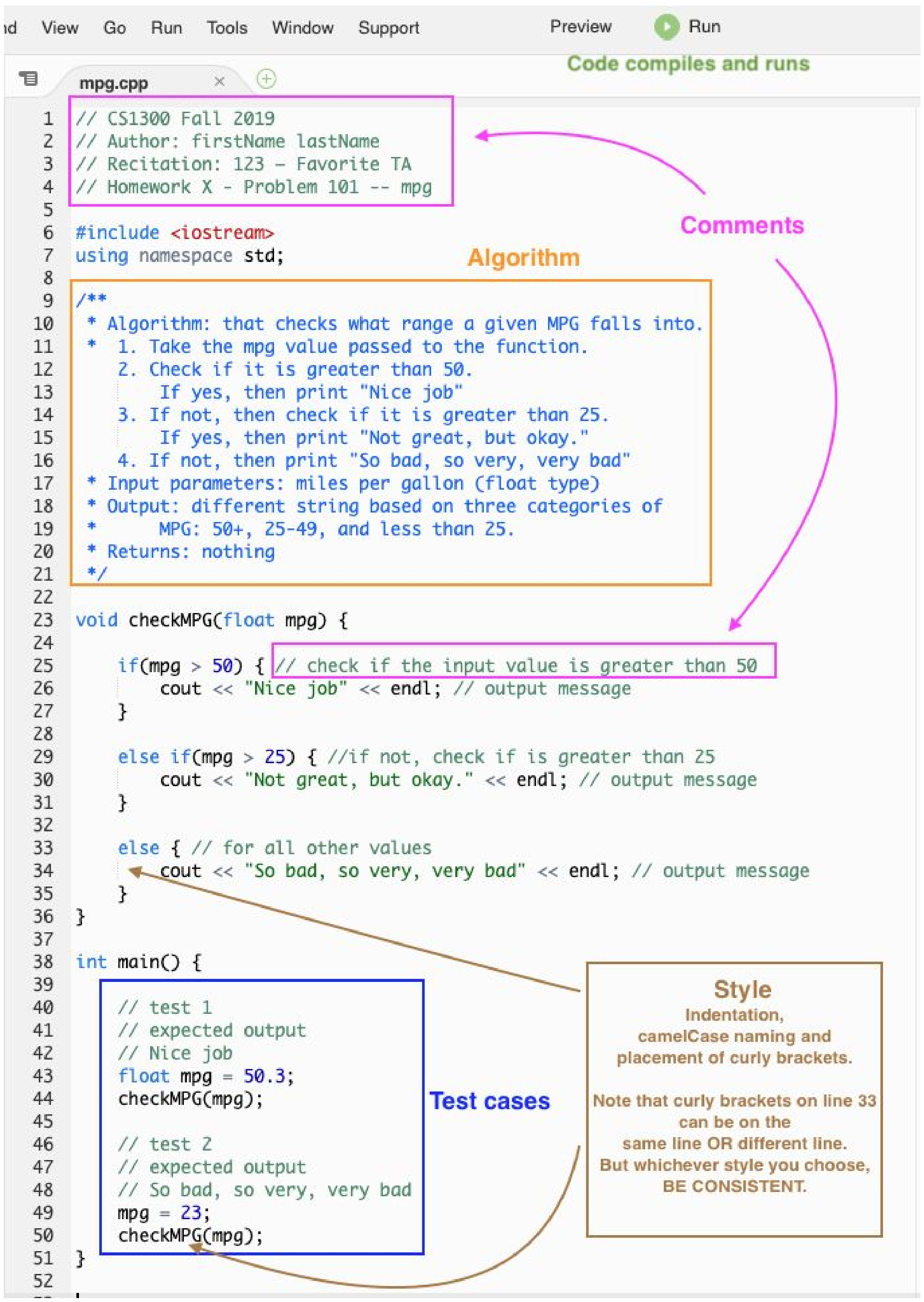
The algorithm described below does not mention in detail what the algorithm does and does not mention what value the function returns. Also, the solution is not commented. This would work properly, but would not receive full credit due to the lack of documentation.

```
/*
 * 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, 6 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 12 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. Test cases are not required for problem 7.



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

Style

Indentation, camelCase naming and placement of curly brackets.

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

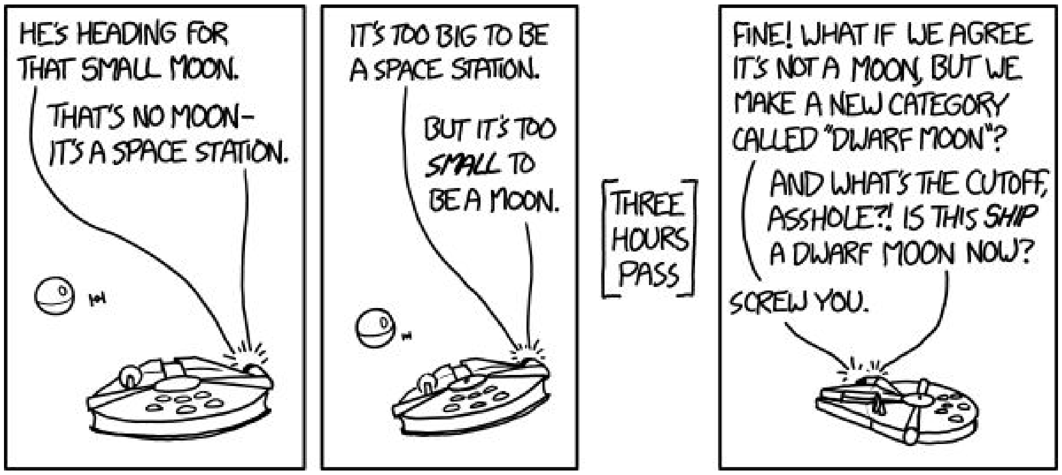
5. Problem Set

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

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): planetClass


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 21 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**
- Have **two** parameters 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);`
	`Delphi 6`
	`8.6`

`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;`
`cout << planets[idx].getRadius() << endl;`
`cout << planets[idx].getVolume() << endl;`
	`2664.31`

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 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.

Problem 3 (15 points): BookClass

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

Problem 4 (10 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:

- Have four parameters 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 6 in Homework 4, 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.
- Empty lines should not be added to the arrays.
- The function should return the following values depending on cases:
  - Return the total number of `books` in the system, as an integer.
  - When `numBooksStored` is equal to the `size`, return -2.
  - When the file is not opened successfully, return -1.

- 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.

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 hw6.cpp.

Problem 5 (10 points): printAllBooks

The **printAllBooks** function from Homework 5 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:

- Have two parameters 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 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 << 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 hw6.cpp.

Problem 6 (15 points): printBooksByAuthor

As we have seen in Homework 5 that some users want to know if there are books by their favorite author. Instead of printing all books, we will print the list of books that are written by the given author. So let's write a function **printBooksByAuthor** that fulfills the following criteria

- Your function should be named **printBooksByAuthor**
- Your function should take three parameters in the following order:
    - **Array of book objects**: `books`
    - **int**: `number of books` (*Note: this value might be less than the capacity of 50 books array*)
    - **string:** `author name`
- You function should **not return** anything.
- 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.

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

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

Function Call:	`// two books by author A`
	`Book book1 = Book("Book 1", "Author A");`
	`Book book2 = Book("Book 2", "Author B");`
	`Book book3 = Book("Book 3", "Author A");`
	`Book listOfBooks[3] = {book1, book2, book3};`
	`int numberOfBooks = 3;`
	`string author = "Author A";`
	`printBooksByAuthor(listOfBooks, numberOfBooks, author);`

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

Example 2: Books array is empty and `numberOfBooks` is 0.

Function Call:	`// no books stored with numBooks 0`
	`Book listOfBooks[5] = {};`
	`int numberOfBooks = 0;`

	`string author = "Dan Brown";`
	`printBooksByAuthor(listOfBooks, numberOfBooks, author);`

Expected Output:	`No books are stored`

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

Function Call:	`// No books by author A`
	`Book book1 = Book("Book 1", "Author B");`
	`Book book2 = Book("Book 2", "Author C");`
	`Book book3 = Book("Book 3", "Author D");`
	`Book listOfBooks[3] = {book1, book2, book3};`
	`int numberOfBooks = 3;`
	`string author = "Author A";`
	`printBooksByAuthor(listOfBooks, numberOfBooks, author);`

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

The zip submission should have three files for this problem: **Book.h**, **Book.cpp** and a driver program called **printBooksByAuthorDriver.cpp** to test your member functions and printBooksByAuthor function. For **Coderunner**, paste **only your Book class implementation and printBooksByAuthor function**, not the entire program. After developing in Cloud9, this function will be one of the functions you include at the top of hw6.cpp.

Problem 7 (20 points): put them together

Now combine your answers from problems 3, 4, 5 and 6. Create a file called **hw6.cpp**. In this file build a program that gives the user a menu with 4 options:

1. Read books
2. Print all books
3. Print books by author
4. Quit

Note: The Book class definition in Problem 3 and the function definitions for Problems 4, 5, and 6 will go in this file as well. For Problem 7, you need to submit 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 four options until they opt to quit. You should make use of the functions you wrote previously, call them, and process the values they return.

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

Option 1: Read books

- Prompt the user for a file name.
  - `Enter a book file name:`
- Pass the file name to your `readBooks` function.
- Print the total number of books in the database in the following format:
  - `Total books in the database: <numberOfBooks>`
- If the function 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 the size of the array print the following message:
  - `Database is full. Some books may have not been added.`

Option 2: Print all books

- Call your `printAllBooks` function.

Option 3: Print all books by author

- Prompt the user to enter the name of an author (be sure it can handle spaces in author names)
  - `Enter name of author:`
- Pass the name of the author to your `printBooksByAuthor` function
- Print all the books by that author.

Option 4: Quit

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

Invalid input

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

Below is an example of running the `HW6` program:

```
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
```

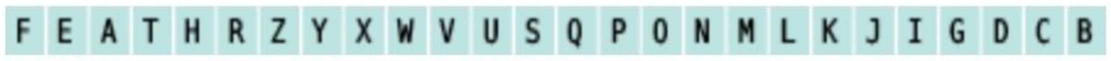
```
1
Enter a filename: fakeFile.txt
No books saved to the database.
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
1
Enter a filename: books1.txt
Total books in the database: 4
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
2
Here is a list of books
Book 1 by Author A
Book 2 by Author B
Book 3 by Author B
Book 4 by Author B
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
3
Enter name of author: Author A
Here is a list of books by Author A
Book 1
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
3
Enter name of author: Author B
Here is a list of books by Author B
Book 1
Book 2
Book 3
```

```
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
3
Enter name of author: Michael
There are no books by Michael
======Main Menu=====
1. Read books
2. Print all books
3. Print books by author
4. Quit
4
Good bye!
```

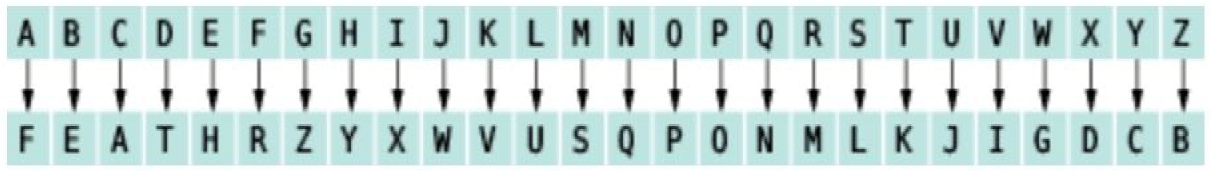
Extra Credit(10 points): keywordCipher

The Caesar cipher shifts all letters by a fixed number. Here is a better idea. As the key, don't use numbers but words. A keyword cipher is a form of monoalphabetic substitution. A keyword is used as the key, and it determines the letter matchings of the cipher alphabet to the plain alphabet.

Suppose the keyword is FEATHER. Then first remove duplicate letters, yielding FEATHR, and append the other letters of the alphabet in reverse order:



Now encrypt the letters as follows:


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
F E A T H R Z Y X W V U S Q P O N M L K J I G D C B

With FEATHER as the keyword, all A's become F's, all B's become E's, all C's become A's and so on. Hence, a message "UGLY DUCKLING", will be encrypted to "JZUC TJAVUXQZ" using the above cipher. And an encrypted message "EFKSFQ", will be decrypted to "BATMAN".

Write a function keywordCipher which will encrypt or decrypt a message using keyword cipher.

- Your function MUST be named **keywordCipher**.
- Your function takes three parameters:
  - message - a string in all capital letters, and can also have spaces. For e.g, message can be "ZOMBIE HERE", "BATMAN".
  - key - a string which is used to encrypt the message. It should be in all capital letters, and can have spaces. For e.g "SECRET", "STAR WARS".
  - flag - a bool variable that controls whether your function will be encrypting or decrypting the given message. A value of true will mean your function is encrypting the given message, and a value of false will mean your function is decrypting it.
- Your function MUST RETURN the resulting encrypted or decrypted string.

Examples:

- If the arguments are `keywordCipher ("ZOMBIE HERE", "SECRET", true)`, the function should return ANPEWT XTKT as the encrypted message.
- If the arguments are `keywordCipher ("ANPEWT XTKT", "SECRET", false)`, the function should return ZOMBIE HERE as the decrypted message.

Don't forget to head over to Moodle to the link **Homework 6 CodeRunner(ec, cipher)**. For this problem, 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.

Extra Credit(10 points): vigenereCipher

The *Vigenere cipher* is a type of polyalphabetic cipher. It is somewhat similar to caesar cipher i.e., each alphabet is substituted by a new alphabet based on the *shift*. The difference being, in *vigenere*, the *shift* is calculated based on a letter from the keyword (explained below) rather than a fixed number. And, the letter from the keyword is chosen in a dynamic fashion based on the position of the input letter to be encrypted.

Let’s look at the encryption algorithm through an example. There are two components to this type of encryption, keyword and the input text. If the keyword is TIGER and the input text we are trying to encode is BATMAN, encryption happens as follows:

The first letter ‘B’ is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S

The encrypted alphabet is a regular alphabet just shifted to start from T (first letter of the keyword). Therefore, the letter ‘B’ becomes ‘U’ on encryption.

The second letter ‘A’ is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H

Again, the encrypted alphabet is a regular alphabet just shifted to start from I (second letter of the keyword). Hence, the letter ‘A’ becomes ‘I’ on encryption.

The third letter ‘T’ is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F

The letter ‘T’ becomes ‘Z’ on encryption.

The fourth letter ‘M’ is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D

The letter ‘M’ becomes ‘Q’ on encryption.

The fifth letter ‘A’ is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q

The letter ‘A’ becomes ‘R’ on encryption.

The sixth letter ‘N’ is encrypted using the rules:

Input Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted Letter	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S

The letter ‘N’ becomes ‘G’ on encryption.

We can summarize this process in three steps:

1. For each input letter find the corresponding letter from the keyword. This is done by looking at the position of the letter in the input sequence, then, locating the letter from the keyword at the same position.

   Note: if the keyword is shorter than input text we need to loop back to the start. In the above example we see ‘N’ from BATMAN follows the same rules as ‘B’. This is because TIGER is shorter than BATMAN.

2. Create a new alphabet sequence shifted to start from the above computed letter.

3. Find the ‘encrypted letter’ for the input letter from the above created alphabet sequence.

- Your function MUST be named **vigenereCipher**.
- Your function takes three parameters:
  - message - a string in all capital letters with spaces. Unencoded messages will be strings such as “I LIKE CHOCOLATE” or “HELLO WORLD”, and encoded messages will be strings such as “B TOOV VPUGFEIZI” or “AMRPF PWXPU”.
  - key - a string which is used to encrypt the message. It should be all capital letters. For e.g “TIGER”, “CATS”.
  - flag - a bool variable that controls whether your function will be encrypting or decrypting the given message. A value of true will mean your function is encrypting the given message, and a value of false will mean your function is decrypting it.
- Your function should not print anything.
- Your function should return the encrypted/decrypted message: a string value.

Examples:

- If the input arguments are `vigenereCipher(“UNICORNS”, “TIGER”, true)`, the function should return NVOGFKVY as the encrypted message.
- If the input arguments are `vigenereCipher(“UNI CORNS”, “TIGER”, true)`, the function should return NVO GFKVY as the encrypted message.
- If the input arguments are `vigenereCipher(“NVOGFKVY”, “TIGER”, false)`, the function should return UNICORNS as the encrypted message.

Don’t forget to head over to Moodle to the link **Homework 6 CodeRunner(ec, cipher)**. For this problem, 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.

7. 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**. The zip file should be named, **hmwk6_lastname.zip**. It should have the following 7 files:
    - Planet.h
    - Planet.cpp
    - planetDriver.cpp
    - maxRadiusDriver.cpp
    - Book.h
    - Book.cpp
    - bookDriver.cpp
    - readBooksDriver.cpp
    - printAllBooksDriver.cpp
    - printBooksByAuthorDriver.cpp
    - hw6.cpp

8. Homework 6 points summary

Criteria	**Pts**
CodeRunner (problem 1 - 7)	90
Style, Comments, Algorithms	10
Test cases	20
Recitation attendance (week 8)*	-30
Total	120
Extra credit: Cipher	20
5% early submission bonus	+5%

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