Recitation 6

This recitation is due **Saturday, February 24th, by 6 pm**

- All components (Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by **Saturday, February 24th 6:00 pm** for your solution to receive points.
- Recitation attendance is required to receive credit.

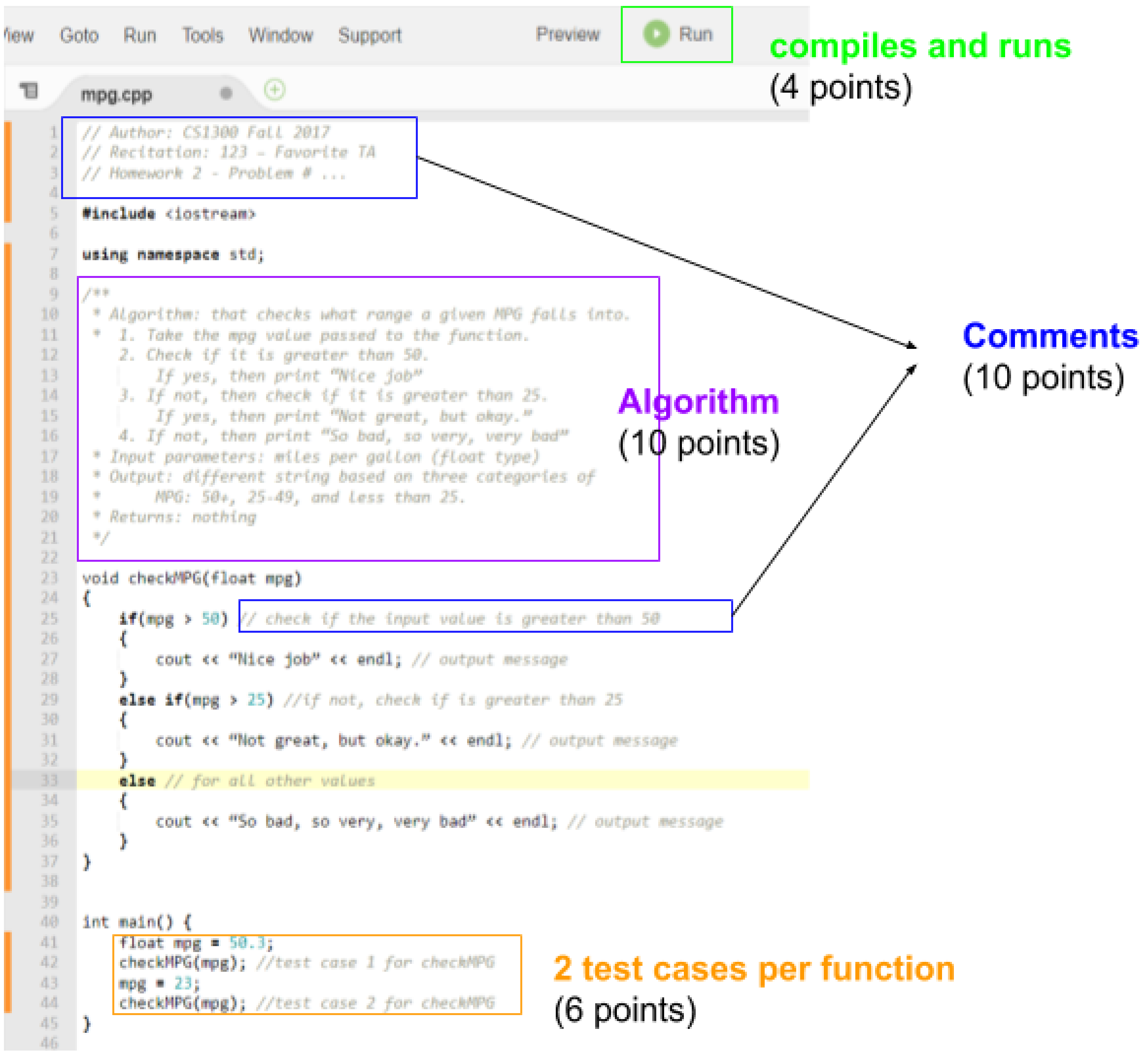
- **Develop in Cloud9:** For this recitation assignment, write and test your solution using Cloud9.
- **Submission:** All three steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.
    1. Make sure your Cloud9 workspace is shared with your TA: 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 *Rec6* and place all your file(s) for this assignment in this directory.
    2. **Submit to the Moodle Autograder:** Head over to Moodle to the link *Rec 6*. You will find one programming quiz question for each problem in the assignment. You can modify your code and re-submit (press Check again) as many times as you need to, up until the assignment due date.
    3. **Submit a zip file to Moodle:** After you have completed all the questions and checked them on Moodle, you must submit a zip file with the .cpp file(s) you wrote in Cloud9. Submit this file going to *Rec 6 (File Submission)* on moodle.

Please follow the same submission guidelines first outlined in Homework 3. Here is a summary:

- 4 points for <u>code compiling and running</u>
- 6 points for <u>two test cases</u> for each function in main
- 10 points for an <u>algorithm description</u> for each function
- 10 points for comments
    - required comments at top of file

      ```
      // CS1300 Spring 2018
      // Author:
      // Recitation: 123 – Favorite TA
      // Cloud9 Workspace Editor Link: https://ide.c9.io/…
      // Homework 3 - Problem # ...
      ```

    - comments for functions / test cases



```
// Author: CS1300 Fall 2017
// Recitation: 123 - Favorite TA
// Homework 2 - Problem # ...

#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() {
    float mpg = 50.3;
    checkMPG(mpg); //test case 1 for checkMPG
    mpg = 23;
    checkMPG(mpg); //test case 2 for checkMPG
}
```

Arrays

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

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
```

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

In C++, strings are arrays of characters.

```
//accessing character at position 1
string greeting = “hello”;
cout << greeting[1] << endl;
```

‘h’	**‘e’**	‘l’	‘l’	‘o’
0	**1**	2	3	4

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**

Remember how we iterated through individual characters in a string:

```
string greeting = “hello”;
int i = 0;
while (i < greeting.length()){
      cout << greeting[i] << “ “;
   i++;
}
```

```
Output:

h e l l o
```

You can iterate through arrays of other types in the same way. 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?
```

Passing By Reference vs. Passing By Value

Passing By Value

Up until now, 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 (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 are *passed by reference.* 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
6       int main(){
7             int X[4] = {1, 5, 3, 2};
8             cout << X[0] << endl;
9             addOne(X);
10            cout << X[0] << endl;
11      }
```

Output:

```
1
2
2
```

Problem Set

Please include the code for Question 1 in your main function (you do not need tests for this problem).

For Questions 2 - 4 you will be writing functions. Please make sure to provide 2 test cases each in main as well as comments and algorithm descriptions.

Question 1

Declare the following arrays:

```
arrayOne - array of integers with 49 elements
arrayTwo - floating point values, one for each practicum
arrayTri - booleans for attendance in each of the 14 recitations
arrayFor - array of strings, one for each month
```

Question 2

Write a function to print all the integers in an array, one per line. Your function will take two parameters, the integer array and the array length (in this order).

Use the function declaration:

```
void printValues(int myArray[], int length);
```

Example

Input:

```
int myArray[3] = {2, 5, 24};
```

Output:

```
2
5
24
```

Question 3

Write a function **getArraySum** that calculates the sum of the elements in an integer array. Your function should take two parameters, an integer array and the array length (in this order). It should return the sum of the elements as an integer.

Example

```
int myArray[3] = {4, 6, 10};        →     return 20

int myArray[3] = {1, 1, 1};         →     return 3

int myArray[4] = {0, 0, 0, 0};      →     return 0
```

Question 4

Write a function **replaceNums** that replaces each element except the first and last by the larger of its two neighbors. Your function should take two parameters, an integer array and the array length (in this order). Your function should not return or print anything.

Examples

```
int myArray[2] = {1, 5};            →     {1, 5}

int myArray[1] = {1};               →     {1}

int myArray[3] = {1, 3, 5};         →     {1, 5, 5}

int myArray[3] = {5, 3, 1};         →     {5, 5, 1}

int myArray[5] = {1, 2, 3, 4, 5};   →     {1, 3, 4, 5, 5}

int myArray[5] = {5, 4, 3, 2, 1};   →     {5, 5, 4, 3, 1}
```

Question 5 has been removed from this recitation. You are free to attempt it on the quiz, but it has been changed to be worth 0 points.

~~Question 5~~

~~Write a function **shiftNums** that shifts all elements by one to the right and moves the last element into the first position. Your function should take two parameters, an integer array and the array length (in this order). Your function should not return or print anything.~~

~~Examples~~

~~`int myArray[1] = {1};`~~ → ~~`{1}`~~

~~`int myArray[2] = {1, 5};`~~ → ~~`{5, 1}`~~

~~`int myArray[3] = {1, 3, 5};`~~ → ~~`{5, 1, 3}`~~

~~`int myArray[5] = {3, 3, 3, 3, 1};`~~ → ~~`{1, 3, 3, 3, 3}`~~

~~int myArray[5] = {1, 2, 3, 4, 0};~~ → ~~{0, 1, 2, 3, 4}~~

Submitting Your Code to the Autograder on Moodle

You must name the functions as indicated in each problem description. The cout formats provided for each problem are not suggestions – they *MUST* be followed precisely, word-for-word and including all punctuation marks, otherwise the autograder will not recognize your results and you will not receive credit.

If there are errors in your solution to a particular problem, a button labeled “Show differences” will appear below the table of tests after you hit “check”. This can be a very useful tool in helping you find small typos, especially in cout statements.