Recitation 9: Classes

This recitation is due **Saturday, March 17th, by 6 pm**

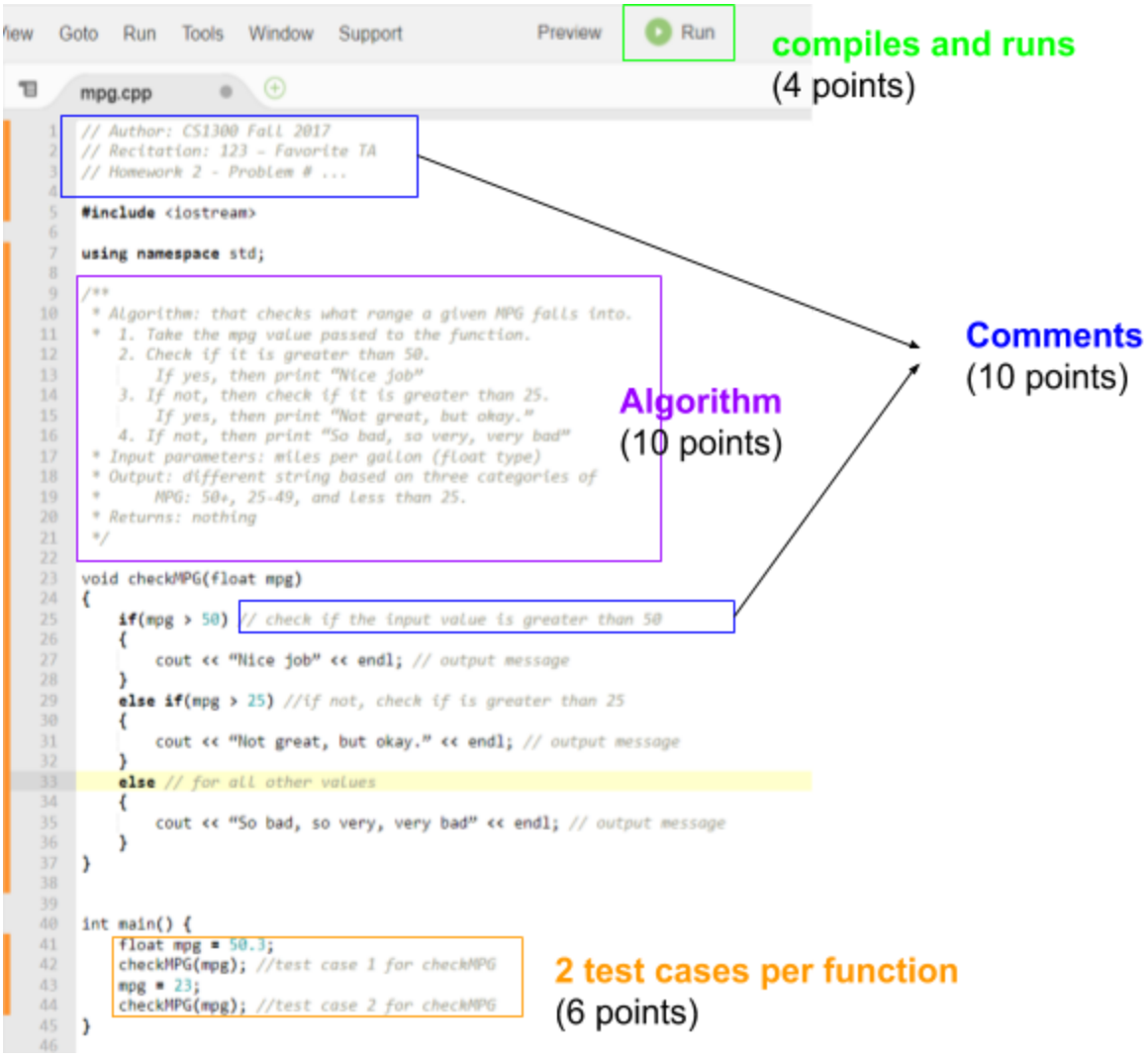
- All components (Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by **Saturday, March 17th 6:00 pm** for your solution to receive points.
- Recitation attendance is required to receive credit.
- **Develop in Cloud9:** For this recitation assignment, write and test 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 *Rec9* 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 9*. 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 9 (File Submission)* on moodle.

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

- 4 points for code compiling and running
- 6 points for two test cases for each function in main. For this assignment, you should create at least two objects from the class you create and call the methods associated with the class on each object.
- 10 points for an algorithm description 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/…
// Recitation 9 - Problem # ...
```

    - comments for functions / test cases



Classes

Please read the information at the following links:

1. C++ Classes and Objects
2. C++ Class Member Functions
3. C++ Class Access Modifiers (skip *The protected members*)
4. C++ Class Constructor and Destructor

Separate Header and Implementation Files

Making use of the ***scope modifier*** :: allows us to place our class in two separate files:

1. a header (.h) file that has our class declaration
2. a class implementation .cpp file with our function definitions.

Typically the header file is the name of the file followed by a *.h,* while the implementation file is the name of the class followed by a *.cpp*. For the class *Box* we would have header file *Box.h* and implementation file *Box.cpp*.

This is the *Box* class from the tutorial linked above split up into two files.

Box.h

```
#ifndef BOX_H
#define BOX_H

#include <iostream>

using namespace std;

class Box {
   Private:                    // data members
      double length;         // Length of a box
      double breadth;        // Breadth of a box
      double height;         // Height of a box
```

```
   public:
      // Member functions declaration
      double getVolume(void);

       void setLength( double );
      void setBreadth( double );
      void setHeight( double );
};
#endif
```

Box.cpp

```
#include "Box.h"


// Member functions definitions
double Box::getVolume(void) {
   return length * breadth * height;
}

void Box::setLength( double len ) {
   length = len;
}
void Box::setBreadth( double bre ) {
   breadth = bre;
}
void Box::setHeight( double hei ) {
   height = hei;
}
```

A program can make use of the *Box* class by adding `#include "Box.h"` .

Benefits of creating separate header and implementation files include:

- Another programmer can use your class without having to know the implementation details.
- The class can be re-used by many different programs.

makeBoxes.cpp

```
#include <iostream>
#include "Box.h"


using namespace std;
```

```
// Main function for the program
int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

Problem Set

In the first part of this recitation, you'll be creating a simple "Robot" class that represents a Robot similar to the one that was described in Homework 1.

The Robot can move around on a grid of square tiles. This Robot has an x position, y position, and a heading (the cardinal direction it is facing). The robot can also move. It can turn left (turn 90 degrees left), turn right (turn 90 degrees right), and move forward one tile.

Question 1

Every Robot has an x position, y position, and a heading. We want to represent these attributes of the Robot by data members in the class.

Create a class "Robot" with the following ***public*** data members:

- an integer x for the robot's x position
- an integer y for the robot's y position
- a character heading for the robot's heading ('N', 'S', 'E', "W')

Then, add a ***public*** constructor that takes

- an int start_x
- an int start_y
- a char start_heading

and sets the robot's attributes.

Question 2

In general it is better practice to make class data members *private* and only allow the user of the class to access them through setter and getter methods. We do this so that we can have more control over what values attributes are set to.

Someone might accidentally do something like this:

```
Robot bb8 = Robot(0, 0, 'N');

bb8.heading = '?';
```

By only allowing the heading attribute to be changed via the setter, for example, we could ensure that heading is always 'N', 'S', 'E' or 'W'.

```
Robot bb8 = Robot(0, 0, 'N');

bb8.setHeading('?');
```

For this question, change the x, y, and heading data members so that they have the *private* modifier.

Then, create the getter and setter methods for the Robot class. In particular, write methods to get and set the values of the x, y, and heading attributes.

Name your functions the following:

- **getX**
- **getY**
- **getHeading**
- **setX**
- **setY**
- **setHeading**

The getters should return the value of the corresponding attribute, and the setters should take in a value and set the attribute accordingly. (e.g., getX should return the x value for the robot, setX should take in an integer and set the x value to that integer).

Question 3

Add the public functions **turnLeft**, **turnRight**, and **moveForward**. The functions should update the robot's x, y, and heading attributes and should not return anything.

turnLeft and **turnRight** should update the robot's heading. If the robot's heading is 'N', turnLeft should result in the new heading being 'W'

moveForward should update the robot's x or y position by 1. If the robot's heading is 'N', **moveForward** should result in the y position increasing by one. If the robot's heading is 'S', **moveForward** should result in the y position decreasing by one.

Question 4

Now we'll write another class *battleShip*.

If you're not familiar with the game, it might helpful to go read up on it before doing this question. https://en.wikipedia.org/wiki/Battleship_(game)

```
class battleShip{
	public:
		battleShip(string);
		~battleShip();

		void setShipName(string);
		string getShipName();

		void setSize(int);
		int getSize();

		void recordHit();   // Increments the hits data member
		bool isSunk();      // returns true if hits is greater than equal to size

	private:
		string name;
		int size;
		int hits;
};
```

For this question, we've provided a complete class definition in the coderunner box. Write the function definitions for all **8** member functions of the *battleShip* class.

The constructor `battleShip(string)` takes the ship name and should set all the data members of the ship to the correct value. The default size of a ship should be -1. What should `hits` be set to for a new ship?

The function `recordHit() should` print `"<name> has been hit <hits> times."`

- For example, if a ship named "Carrier" with size 5 had 4 hits, I would expect the output
  `"Carrier has been hit 4 times."`

The function `isSunk() should` print `"<name> has been sunk."`

- For example, if a ship named "Destroyer" has sunk, I would expect the output
  `"Destroyer has been sunk."`

Question 5

Finally, fill in the code for a *play* function that uses the *battleShip* class and runs a game of *Battleship*.

Keep hitting all three ships until all three have been sunk, and always hit them in the order Destroyer, Carrier, then Cruiser.

```
void play(){
	//TODO: Declare 3 instances/objects of the battleship class: Destroyer Carrier
Cruiser



	//TOD0: Give the ships a size: Destroyer-3 Carrier-5 Cruiser-2
	// you will need to call the appropriate methods



	// Once you have this running for one, expand this while loop to include the
	// other two instances. Have the while loop end when all three ships have been
	// sunk. Make your while condition while ship one is sunk OR ship two is sunk
OR ship three is sunk.
```

```
        while(ship_one.isSunk() == false){
                ship_one.recordHit();
        }
}
```

If your program is running correctly, we expect the following output:

```
Destroyer has been hit 1 times.
Carrier has been hit 1 times.
Cruiser has been hit 1 times.
Destroyer has been hit 2 times.
Carrier has been hit 2 times.
Cruiser has been hit 2 times.
Destroyer has been hit 3 times.
Carrier has been hit 3 times.
Cruiser has been hit 3 times.
Destroyer has been sunk.
Destroyer has been hit 4 times.
Carrier has been hit 4 times.
Cruiser has been hit 4 times.
Destroyer has been sunk.
Destroyer has been hit 5 times.
Carrier has been hit 5 times.
Cruiser has been hit 5 times.
Destroyer has been sunk.
Carrier has been sunk.
Cruiser has been sunk.
```