Programming Assignment #3

This assignment will give you some practice with object-oriented design and coding (classes, objects, constructors, operator overloading, friend functions.) The task is to define and implement a class called Point, which represents a point in a 2-dimensional Cartesian coordinate system. To help you understand the required functionality that the class must implement, a driver program is available that demonstrates how a client will the use objects of the Point class. One of the main tasks is to overload several operators that can be used by the client. They are defined as below. The number in parentheses is the approximate number of lines of code required. You'll notice that, with the exception of the rotation code, they are all trivial.

Operator	Functionality	Description
`%`	Rotation	Rotates a Point about the origin the specified number of degrees. Returns a new Point. This is a member function.(Varies)
`-`	Distance (Binary operator)	Calculates the difference between two Points and returns the distance (a double). This is a member function. (4)
`^`	Midpoint	Calculates the midpoint of two Points. Returns a new Point. This is a member function. (4)
`+=`	Translation	Adds two Points or a Point and a double and returns a reference to the left-hand operand which has been modified. These are both member functions. (3/3)
`-`	Translation	Subtracts a double from a Point and returns a new Point. This is a member function. (2)
`++`	Pre/Post Increment	Adds one to the x/y values of the object. There are two versions, one for pre- and one for post-increment. Pre-increment returns a reference to the incremented Point and post-increment returns a new Point with the value of the "before incremented Point". Both are member functions. (3/3)
`--`	Pre/Post Decrement	Subtracts one from the x/y values of the object. There are two versions, one for pre- and one for post-decrement. Pre-decrement returns a reference to the decremented Point and post-decrement returns a new Point with the value of the "before decremented Point". Both are member functions. (3/3)
`-`	Unary negation	Returns a new Point that has the x/y values of the input Point negated. This is a member function. (1)
`+`	Translation	Adds two Points or a Point and a double and returns a new Point. There are two versions of the double/Point function. One is a member (2), the other is a non-member, non-friend (1). The Point/Point method is the member function. (2)
`*`	Scale	Multiplies a Point by some numeric factor (double) and returns a new Point. There will be two versions of this, one is a member and the other is a non-member, non-friend. (2/1)
`<<`	Output	Outputs a Point in the form of a string: (x, y), where x and y are the coordinates, e.g. (4, 7). This is a **friend** function. **Make sure you format it exactly as shown.** (2)
`>>`	Input	Inputs a Point. Allows 2 numbers (separated by whitespace) to be entered, e.g. 4 7. This is a **friend** function. The inputs are both doubles. (2)

The following syntax will be supported:

```
Point pt1(3, 4);  // pt1 is (3, 4)
Point pt2;        // pt2 is (0, 0)
Point pt3(pt1);   // pt3 is (3, 4) (uses the default copy constructor)
Point pt4 = pt1;  // pt4 is (3, 4) (uses the default copy constructor)
Point pt5;        // pt5 is (0, 0)
pt5 = pt4;        // pt5 is (3, 4) (uses the default copy assignment operator)
```

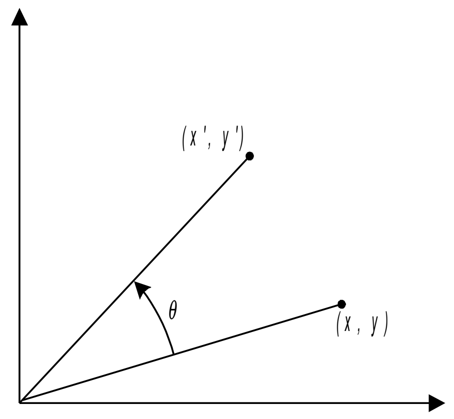
However, do not allow this syntax:

```
Point p6 = 4; // this should not compile
Point p7(4);  // this should not compile
```

If the above code compiles with your implementation, then your implementation is incorrect and must be changed.

The starting point for the interface is in **Point.h** which is provided. You'll need to modify this before implementing the functionality in **Point.cpp**. You will not need to use the **new** keyword in this assignment as we do not require any dynamic memory allocation. If you are not sure of the functionality of an operator, look at the sample driver and output. All of the information you need can be discovered from the sample test program (driver-sample.cpp) that is provided.

Most of the math involved is simple high-school algebra and most of the functions require only a couple of lines of code or so. The exception is `operator%` which does rotation. I've supplied some useful equations in case some of you may have forgotten them.

Descriptions	**Equations**
The midpoint of two points.	$\left(\frac{(x_1+x_2)}{2}, \frac{(y_1+y_2)}{2}\right)$
The distance between two points.	$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
The distance of a point from the origin.	$\sqrt{(x^2+y^2)}$
	You can use the rotation matrix that you learned in MAT140 for counter-clockwise rotation. If you haven't learned it or you've forgotten it, please ask one of the TAs or tutors in the Academic Support Center for assistance. $R(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}$ $\begin{bmatrix}x' \\ y'\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x \\ y\end{bmatrix}$

What to submit

You must submit your header file and implementation file (Point.h and Point.cpp) in a .zip file to the appropriate submission page. Note that you are not submitting any other files.

Source files	**Description**
Point.cpp	The implementation file. All implementation for the functions goes here. You must document the file with a file header comment and functions (function header comments) using the proper Doxygen tags. The only header files you need are already included.
Point.h	The header file. You willl need to modify the one I posted. You must document the file with a file header comment. **No implementation is permitted in this file.**

Usual stuff
Your code must compile cleanly (using the compilers specified) to be accepted. The code must be formatted and printed as per the documentation on the website.

Make sure your name and other info is on all documents.