CS 170 | Spring 2017
Programming Assignment #4

Due Date
- Wednesday, May 24, 2017
- 01:00 am

Remember: The major point of this assignment is for you to learn how to submit your homework correctly. You should strive to follow all of the directions exactly as specified in the handouts. The syllabus that was handed out during the first day of class also contains important information on how to submit your homework. If you are still unsure of how to do something, you should ask for help, either from myself or from a tutor in the lab or from another student.

Information

This assignment will give you some practice with object-oriented design and coding (classes, objects, constructors, operator overloading, and 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 such:

Symbol	Operator	Description
`%`	**Rotation**	Rotates a Point about the origin the specified number of degrees. Returns a new Point. This is a member function. **THIS WILL BE COUNTERCLOCKWISE ROTATION.**
`-`	**Distance (Binary operator)**	Calculates the difference between two Points and returns the distance (a double). This is a member function.
`^`	**Midpoint**	Calculates the midpoint of two Points. Returns a new Point. This is a member function.
`+=`	**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.
`-`	**Translation**	Subtracts a double from a Point and returns a new Point. This is a member function.
`++`	**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.
`--`	**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.
`-`	**Unary negation**	Returns a new Point that has the x/y values of the input Point negated. This is a member function.
`+`	**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, the other is a non-member, non-friend. The Point/Point method is the member function.
`*`	**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.
`<<`	**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.**
`>>`	**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.

:

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 assignment operator)
```

However, do not allow this syntax:

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

You are given a partial interface file and a partial implementation file, as well as a sample driver and output for the driver.

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. If you are unsure what a formula should do use the internet to look it up.

Remember that the rotation function should be counterclockwise rotation.

- Sample command lines for compiling:

GNU:

```
g++ -o Point.exe -Wall -Wextra -O -ansi -pedantic Driver.cpp Point.cpp
```

Microsoft:

```
cl -W4 -EHa -WX -Za -FePoint.exe Driver.cpp Point.cpp
```

Borland:

```
bcc32 -ePoint.exe -w Driver.cpp Point.cpp
```

Remember to use **const** where appropriate. **YOU HAVE BEEN WARNED.** (Hint: Make some test cases that use constant Points in certain places.)

You must submit your header file, implementation file (Point.h, Point.cpp), and Doxygen generated index.chm as well as a makefile for each compiler that will build into a separate folder. Submit these in a .zip file to the appropriate Moodle submission page. You are submitting a total of six files.

Put all 3 files into a zip file called CS170_your.login_4.zip.

Here are some popular Questions and Answers for this assignment.

1. **Question**:
   The table in the handout says that the subtraction operator calculates the difference between two Points, but the driver program also shows subtraction between a Point and a double. Which is the correct one?
   Answer:
   Actually, they are both correct. You need to overload the subtraction operator for both cases. The first case is as described in the handout: the distance between two points is a double. The second case will be a double subtracted from a Point. The result will be a Point. The reason for this seemingly "weird" rule is that subtracting a positive value from a Point should be the same as adding a negative value to a Point:

```
void TranslateTest(void)
{
  Point pt1(6, 8), pt2(4, 1);

  // This should result in a double
  double d = pt1 - pt2;

  // These two should result in the same Point
  Point pt3 = pt1 - 2;
  Point pt4 = pt1 + -2;

  // THIS SHOULD NOT COMPILE
  pt3 = 2 - pt1;
}
```

2. **Question**:
   Can we include other header files in our .cpp file?
   Answer:
   No.

3. **Question**:
   My code compiles and runs all of the tests with only one constructor. Why do I need two?
   Answer:
   Does your code compile and run the tests that are NOT supposed to compile? If so, then you have the wrong constructor. (Hint: You won't have any default parameters in your constructors.)

4. **Question**:
   Are we allowed to create any gettor functions so the non-member, non-friend functions can access the private m_x and m_y members?
   Answer:
   No. You do not need any gettor functions. You are only allowed to create the functions specified. That is, 2 constructors and 18 operators (14 member, 2 friend, and 2 non-member, non-friend.) If you forgot how to implement the non-member, non-friend functions without gettor functions, review the lecture notes.

5. **Question**:
   All of my output seems to be correct except for the rotation. I'm getting `-0.000` instead of just `0.000`. What's the problem and how can I fix it?

Answer:
The problem is with the double data type. When using integers, there is only 0; there is no -0. However, with doubles (or any floating point type), you can have both +0.0 and -0.0. Usually, this isn't a problem, but if you round a very small negative number, you will get -0.000. For example, if you have the value `0.00000000000664` and you print it out rounded to 3 decimal places, you'll get `0.000`. But, if you have the value `-0.00000000000664` and you print it out rounded to 3 decimal places, you'll get `-0.000`.

To fix this, we have to check to see how small the value is, and if it is small enough, we will call it zero. For this assignment, we will say that any value that is between -0.00001 and +0.00001 is zero. This arbitrary value (0.00001) is called an *epsilon*. You'll see that there is a constant in the partial implementation file like this:

```
const double EPSILON = 0.00001;
```

You should have code similar to this in your rotation code (for both x and y):

```
// If value is between -EPSILON and +EPSILON, make it 0.0
if (value > -EPSILON && value < EPSILON)
      value = 0.0;
```

6. **Question**:
Why does Borland's compiler give me an error message when I try to use sin and cos in my code? The error is something like this:

Error E2268 Point.cpp 89: Call to undefined function 'cos' in function ...

Answer:
You are calling the functions incorrectly. All C/C++ library functions are in the **`std`** namespace. You must specify the namespace like this: std::cos(value); Microsoft and GNU allow it without the namespace, which is non-standard.