Lab1

In Lecture 2, we saw the public interface for the class Point.

```
class Point {
  public:
    static Point *makeCartesian(double x, double y);
    static Point *makePolar(double r, double theta);

    double getX();
    double getY();
    double getR();
    double getTheta();
}
```

In this lab, you will implement this interface, as well as add three more public functions to it.

Step One

First, create an empty directory for lab1, then download and unpack this zip file into it. This file contains starter code for the lab:

- main.cpp, which contains the program's main routine and code that will test your implementation,
- Point.h, which contains the public and private parts of the basic API for the point class.
- Point.cpp, which contains an implementation of this basic API that stores information in Cartesian coordinates. Take note of how the static factory methods are implemented using the private constructor.
- run.out, which contains the output that your final program is supposed to generate.

Step Two

Compile main.cpp and Point.cpp together. This is done by giving both filenames as arguments to g++.

```
g++ main.cpp Point.cpp
```

This will create a file 'a.out' which is the executable that you run.

So, run a.out and confirm that the output is the same as the first 5 lines in run.out.

Step Three

Your first coding task is to change the implementation in Point.cpp and Point.h to use polar rather than Cartesian coordinates to store the information. This will involve changing **all** of the member functions in Point.cpp. The constructor should take the arguments r and theta. The private part of Point.h should now look like:

```
private:
    Point(double r, double theta);
    double r;
    double theta;
```

Once you have completed this, test your work by compiling and running the program. The output of your program should again be the first 5 lines of run.out.

Step Four

Add the following three lines in the public section of Point.h.

```
Point* rotate(double alpha);
Point* translate(double dx, double dy);
Point* scale(double scaleFactor);
```

Now, edit Point.cpp and add the implementations of these routines. Each one of them should return a new Point that is related to the old point by the transformation specified.

- `rotate` takes a point (r, theta) and returns the point (r, theta + alpha).
- `translate` takes a point (x, y) and returns the point (x+dx, y+dy).
- `scale` take a point (r, theta) and returns the point (r*scaleFactor, theta).

Once you have completed this, edit main.cpp to remove the comment around the last six lines of code in the body of main. Then compile the program and run it. The output should now match run.out. If it does, then call the TA to see it and give you your marks.

Updated Wed May 22 2019, 23:13 by shermer.