CSCI-15 Inheritance assignment 6, 120 points. Due 11/15/18

Adapt the class Point class from one of my Point-Circle examples. Make sure Point has friend operator << and operator >> functions and Distance() and Slope() methods. Distance() and Slope() may take only one Point as a parameter. Create a Quadrilateral class by aggregation on four Points. Then create an inheritance hierarchy for the classes Quadrilateral (the base class), Trapezoid, Parallelogram, Rectangle and Square. The points in these shapes must be entered in counter-clockwise order, starting at the lower-left-most point. I will explain more about what I mean by this in class. This will greatly simplify the calculations needed for the areas. You may use any reasonable scheme to hold the Points in the Quadrilateral class; for example, an array of four Points or four separate Points, but you may not just keep eight doubles for the coordinates. Regardless of how you represent the Points, they values must be accessible from the client as Points called a, b, c and d in the getX() and setX() methods to return or set values into Points as described below.

- Supply getA(), getB(), etc., and setAll() (taking 4 Points, not 8 doubles), Perimeter() and Area() methods for the Quadrilateral class. The base Quadrilateral operator << function will print "This is a Quadrilateral", go to the next line, then print the points in the order a, b, c, d on one line.
- Supply overridden Area() methods for each derived class. These must calculate using (usually different) formulae appropriate for the current type of figure. These may not call their base class Area() methods.
- When you override the Perimeter() method in any derived class it may simply call down to the parent class perimeter, or you may use a simpler perimeter calculation if one is available.
- Supply a friend operator << function for each derived class to show the type of figure (e.g., something like "This is a parallelogram", with a different message for each type) and (eventually) its four vertex coordinates. Call the immediate base class << function directly from each derived class's operator << to print the prior (base) type of shape and eventually, when it reaches the Quadrilateral class, the points. Thus, you will print the types (one per line) all the way down the inheritance chain by using the base class' friend operator << function. **Do not print the points themselves in any derived class' operator << friend functions.** You should do this by using a print() method and making << a wrapper around print(). This will make the next assignment much simpler. Similarly, the friend operator >> should be a wrapper around a read() method.

For simplicity, you may assume your Area() methods will work correctly only when the base is parallel to the X-axis, and may approximate a value otherwise. This is not an analytic geometry problem, it is an inheritance problem. For this program's Area() method, you may assume the base is parallel to the X-axis and is defined by the first two Points entered. However, other functions (e.g., looking for parallel or perpendicular) must work if the base AB is not parallel to the X axis. You may always assume the segment AB is the base.

The constructors and data entry methods for each type must ensure that the points are entered in the correct order. If not, set the points to some appropriate default values (unique and correct for that type, but not correct for the next derived type) and print an error message. These constructors and data entry/manipulator methods will each eventually use a single protected helper method of the base class to verify that the points are entered in the correct sequence. Thus, each derived class will first validate that the object is a correct parent class object, then add

validation for the new type. These validations must be in a protected method called Validate(), which has no parameters and returns void.

Write a driver client for your classes that declares objects of each type. Put the quadrilateral types and data points into a file to read them into the program. Code the types and points (for entry purposes) as T (x, y) (x, y) (x, y) (x, y), where T is a code for the type of quadrilateral and (x, y) is a Point's data. Pass in the input and output files on the command line. Except as specified above (points entered out of order, etc.), you may assume correctly formatted data.

Then print the appropriate data (type, points, perimeter and area) for each type of object. Test with both legal and illegal values (e.g., shape not of the appropriate type, Points entered in the wrong order).

Definitions (for use ***only*** in this program – some of these do not match the mathematical definitions!), additional instructions, and hints:

A quadrilateral is a closed plane figure with four linear sides and four internal angles. For this assignment, assume simple (no sides crossing) convex (all internal angles < 180º) quadrilaterals, since that simplifies the problem considerably.

To get the area of a quadrilateral, you can divide it into 2 triangles and find the sum of areas of the triangles. Look up Heron's formula for an easy way to do this. You may use another method if you wish, there are several.

A trapezoid is a quadrilateral with exactly ONE pair of parallel sides (either pair of opposite sides). For this program, you may assume the base and the top are the parallel sides. An easy way to get its area is ((base + top) / 2) * height.

A parallelogram is a quadrilateral with both pairs of sides parallel. Area is base * height.

A rectangle is a parallelogram with right angles at all vertices. Area is still base * height.

A square is a rectangle with all sides of equal length. Area is any side squared.

Distance between two points (x_1, y_1) and (x_2, y_2) is given by: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

The slope of a line between two points (x_1, y_1) and (x_2, y_2) is $(y_1 - y_2) / (x_1 - x_2)$, unless the line is vertical, in which case the slope is undefined. Write a Slope() method for Point that takes a Point and calculates and returns the slope of the line between self and the parameter. Use a very large Point member constant like 10^{100} for the undefined slope value.

Two lines are parallel if they have the same slope or if both slopes are undefined. They are perpendicular if the product of the slopes is -1, or if the slope of one of the lines is undefined and the slope of the other is zero.

For this program only, the base of a shape is the side defined by the first two points entered for the shape.

Remember that the floating-point types in C++ are approximations, not exact values. Write an Equals() function that takes 2 doubles and returns true if they are "close enough" to the same value, i.e., within 10^{-9} of each other (there are more accurate ways to test this). Please, don't ever use == to test equality for floating-point types! (**Note: this is NOT a method of any class…**)

This program is NOT hard. There are a lot of TINY methods, and a few more small methods. Your client will probably be less than one page of code. 120 points.

Your next assignment will be to make this inheritance chain polymorphic and test the polymorphism.