CSC 2430 – Data Structures 1
Homework #5 – ADT Triangles
Due: Wednesday, February 17, 2016

A "Triangle" ADT (Abstract Data Type)

Design and implement an ADT that represents a triangle. The data for the ADT must include the three side lengths of the triangle. This data should be in the private section of the Triangle class that implements the ADT.

Include two initialization operations (the class's constructors).

- **Triangle( )**: provide default values for the ADT's data. Use (3, 4, 5) as the default sides.
- **Triangle(const double sideA, const double sideB, const double sideC)**
  uses client-supplied values for the ADT's data. Be sure to check and verify the validity of the triangle data (that is, that the 3 sides really do form a triangle).
- You must design your triangle object so that it always represents a valid triangle. If the user attempts to construct a Triangle with invalid sides, your constructor must output to cerr a brief error message describing the problem, and then finish constructing the object by forcing the sides to the default triangle configuration.

In addition to the two constructors, the Triangle ADT also must include the following operations (methods):

- **access methods** - return the values of each of the ADT's internal data members (each of the sides)
  e.g., Triangle.h: **double getSideA( ) const;** // a "getter" method

  Triangle.cpp:
```
double Triangle::getSideA( ) const
{
    return(SideA);
}
```
- **bool isRightTriangle( ) const**
  Determine whether the triangle is a right triangle (e.g., $c^2 = a^2 + b^2$ or $a^2 = c^2 + b^2$, etc.).
  Note that the triangle side lengths can be specified in any order. Be sure to check all legal variations.
- **bool isEquilateralTriangle ( ) const**
  Determine whether the triangle is equilateral (all sides equal)
- **bool isIsoscelesTriangle ( ) const**
  Determine whether the triangle is isosceles (any two sides equal)
- **double TriangleArea( ) const**
  Compute and return the area of the triangle
  (e.g., it's easiest to use Heron's method, look it up in Wikipedia or another source)

- Notice that, for this lab, there are no methods to modify a triangle's side-length data members once it has been constructed. The only way to establish a triangle with specific side lengths is to **construct** one.
  However, it is possible to assign Triangle objects (e.g., TriA = TriB;), which will create an exact copy.
  You can also create and assign a Triangle object "on the fly" (e.g., TriA = Triangle(3, 4, 5);).

For this exercise, you must create a project with 3 source-code files:

- a **class header declaration file** (e.g., triangle.h)
  Be sure that you carefully document the "interface" specifications for your class.
- a **class implementation file** (e..g, triangle.cpp)
- a **main program for testing** out your class implementation. Create your own test program examples and data. Be certain to include enough details to demonstrate that "triangle" objects work properly. Use the example on the following page as a guideline and be sure to test at least the examples shown.

Turn in your source code listings and example runs that demonstrate the use of triangle objects, and show that your design and implementation work correctly.

```
// Example main() test program OUTLINE - customize to complete lab requirements
#include <iostream>
using namespace std;
#include "triangle.h"

void printTriangleDetails(const Triangle &tri) {  // const parameter by reference
      cout << "Triangle has sides (";
      cout << tri.getSideA() << ", ";
      cout << tri.getSideB() << ", ";
      cout << tri.getSideC() << ")" << endl;

      if(tri.isRightTriangle())
            cout << "  Triangle is a right triangle" << endl;
      else
            cout << "  Triangle is not a right triangle" << endl;

      // etc, etc, etc, for the other details about a triangle object
}

int main( ) {
      Triangle t1;                  // Default triangle
      cout << "t1: [" << t1.getSideA() << ", " << t1.getSideB() << ", "
           << t1.getSideC() << "]" << endl;
      cout << "Default triangle:" << endl;
      printTriangleDetails(t1);

      Triangle t2(5, 4, 3);         // convert constructor
      cout << "t2(5, 4, 3) triangle:" << endl;
      printTriangleDetails(t2);

      Triangle t3(4, 5, 3);
      cout << "t3(4, 5, 3) triangle:" << endl;
      printTriangleDetails(t3);

      Triangle t4(4, 6, 4);
      cout << "t4(4, 6, 4) triangle:" << endl;
      printTriangleDetails(t4);

      Triangle t5(2, 4, 2);         // Invalid side combination
      cout << "t5(2, 4, 2) triangle:" << endl;
      printTriangleDetails(t5);

      Triangle t6(5, 2, 5);
      cout << "t6(5, 2, 5) triangle:" << endl;
      printTriangleDetails(t6);

      // etc, etc, etc
      // Code up several additional "test" examples that
      //    illustrate the correct operation of the Triangle object.

      // Try some other usage patterns
      t1 = t4;                      // Assignment of triangle objects
      printTriangleDetails(t1);

      t5 = Triangle(6, 8, 10);      // create triangle "on the fly"
      printTriangleDetails(t5);
      t5 = Triangle(t5.getSideA(), t5.getSideB(), 9); // change 3rd side of t5
      printTriangleDetails(t5);                       //   if valid result

      return(0);
}
```