**CPSC 1071   
Lab 12 - Inheritance**

**Objectives**

The purpose of this lab is to introduce the notion of class inheritance. Similar to the transition from procedural to object oriented programming we made earlier in the semester, understanding *when* and *how* to apply inheritance will (hopefully) not only influence the way you think about and structure your classes, but will also make them more modular and thus easier to maintain in the long run.

With this in mind, the objectives for this lab are as follows:

* understand the notion of a *base class* and *subclass*
* understand how to think about and partition methods/functionality between a base class and an arbitrary number of subclasses
* understand the **protected** and **virtual** keywords -- and how to use them properly

**Inheritance 101**

To demonstrate and introduce the notion of inheritance, we'll start by defining a simple *Polygon* class.

// Polygon.h

class Polygon

{

public:

Polygon();

Polygon(string type, int sides);

string getType();

int getSides();

virtual double computeArea() = 0;

virtual double computePerimeter() = 0;

virtual void print();

protected:

string type;

int sides;

private:

int id;

};

As specified in the header above, our *Polygon* is defined in terms of a *type*, *sides*, and associated methods for accessing and modifying these variables.

There are several things worth noting here:

1. **generality:** *Polygon*, at this point, doesn't seem like a very descriptive/useful class. For instance, what kind of polygon are we talking about? Is it a triangle, a rectangle, hexagon, etc? In short, the generality we're seeing (in the name, fields, and methods) is indeed an intentional design choice, and we'll soon see how we can use this general notion of a *Polygon* to define more *specific* polygons.
2. **virtual methods:** A new feature of the *Polygon* header is the inclusion of the *virtual* modifier preceding the *computeArea()* and *print*methods. The *virtual* keyword indicates that this is a method we are **not** obligated to implement in *Polygon.cpp*; instead, its implementation is left to one of the *Polygon* subclasses.
3. **protected members:** Instead of declaring all member variables to be private, we use a different keyword, *protected* to ensure that any subclasses extending *Polygon* will be able to see/access the variables without needing methods such as *getType()*.

**Polygon subclasses**

Due to the inclusion of the (*virtual*) methods,

virtual double computeArea() = 0;

virtual double computePerimeter() = 0;

you can think of the *Polygon* class as an **abstract** class -- meaning we cannot instantiate it, since *computeArea()* and *computePerimeter()* will not have an implementation in the corresponding *.cpp* files.

Let's look at the specification of a **subclass** of *Polygon* -- a *Rectangle* class:

// rectangle.h

#include "polygon.h"

class Rectangle : public Polygon

{

public:

Rectangle();

Rectangle(double length, double width);

double computeArea();

double computePerimeter();

void print();

protected:

double length;

double width;

};

you'll notice that instead of just saying

class Rectangle { ... };

as we would normally do, we instead use the following syntax:

class Rectangle : public Polygon { ... };

which indicates that our new class, *Rectangle* **inherits** from the general, abstract *Polygon* we defined earlier. Also note that we have dropped the *virtual* modifier from the *computeArea()* method prototype; however, it is still acceptable to keep the *virtual* keyword. Since *computeArea()*and *computePerimeter()* are not defined in the *Polygon* class, we are obligated to implement the methods as part of rectangle.cpp:

//rectangle.cpp

Rectangle::Rectangle() : Polygon()

{}

Rectangle::Rectangle(double length, double width) : Polygon("Rectangle", 4)

{

this->length = length;

this->width = width;

}

double Rectangle::computeArea()

{

return length \* width;

}

double Rectangle::computePerimeter()

{

return 2 \* (length + width);

}

void Rectangle::print()

{

Polygon::print();

std::cout << "length: " << length

<< " width: " << width << std::endl;

}

**So what does all this buy us?**

The *Polygon* class serves as an organizational tool. It cannot be instantiated; however, it provides all the methods that must be implemented by subclasses of *Polygon*. *Polygon* header.

**Getting the files for this lab**

Create a **"lab12"** directory and copy [lab12.tar](https://www.cs.clemson.edu/course/cpsc1070/labs/lab12/Public/lab12.tar) to the directory using:

lab1071copy 12

Next untar the file, i.e.:

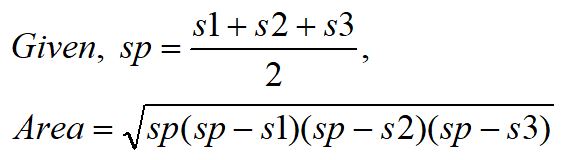
**tar -xvf lab12.tar**

**Task 1: Implement two subclasses of the Polygon class.**

You are to implement a triangle subclass, Triangle, and a regular hexagon subclass, Hexagon. Both subclasses inherit from the *Polygon* class.

**Triangle class**

For a triangle, we need to know the length of the three sides. The specification file for the Triangle class. The Triangle class should implement the getter methods, the *computePerimeter())*, the *computeArea()*, and the *print()* methods for a Triangle object.



**Hexagon class**

A regular hexagon has six equal sides; therefore, we only need to know the length of one side.

For the *Hexagon* class, you are to implement *getSide()*, the *computePerimeter()/*, *computeArea()*, and *print()*methods.

The perimeter is the (number of sides) \* (length of a side). Note that the number of sides is stored in the base class; use it.

The area can be computed using the formula:

https://www.cs.clemson.edu/course/cpsc1070/labs/lab12/hexagon-area.jpg   
Remember to #include *cmath* for the sqrt function.

To help get started, you should refer to the full source code for the *Rectangle* and the modified *Triangle* subclasses.

**Note**: This means you will need to create your own hexagon.h/cpp files and add code there. When you name your class, it should be called*Hexagon*.

**Task 3: modify the Makefile and test your Triangle implementation and your Hexagon implementation**

Complete the following inside *main.cpp*:

1. invoke the *print()* method to print the Triangle object, *t2*, then print out the result of the *computeArea()* and *computePerimeter()* methods for *t2* similar to the examples already provided in *main()*.
2. create a new *Hexagon* object and invoke the *print()* method for the Hexagon object, then print out the result of the *computeArea()* and *computePerimeter()* methods.

**Note**: while doing this, remember to use

#include proper header files.

Use the previous setup as a guide.

modify the *Makefile* so that it also compiles the Triangle class and the Hexagon class.

**Testing**

To compile and test your code type this:

make

./lab12

A sample [output](https://www.cs.clemson.edu/course/cpsc1070/labs/lab12/Public/output.txt) file is provided.

**Handin**

To hand in your project first **make clean**, then use the **tar cvf** command to turn in all of your .cpp and .h files and Makefile, then name the file lab12\_handin.tar.

Use the [web handin](https://handin.cs.clemson.edu/courses/) website to turn in your lab12\_handin.tar.