**CSCI 241 Assignment 9  
100 points (Extra Credit)**

**Purpose**

This short assignment covers inheritance and polymorphism.

**Assignment**

For this project, you will create a series of classes to represent some simple geometric shapes and a small program to test your classes.

**class Shape**

Shape is an *abstract base class*.

*Data Members*

A Shape has the following private data member:

* a name, which is of type string,

*Methods*

The Shape class should have the following public methods:

* a constructor that takes a const string& argument and uses it to initialize the shape's name. Since the Shape class is abstract, this constructor will only be invoked by a derived-class constructor.
* a *virtual* destructor. The method body for this destructor can be empty, since it does not need to delete any dynamic storage. If you don't have a virtual destructor for an abstract base class, the compiler may produce a warning message.
* a *virtual* method called print() that takes no arguments and returns nothing. The method should print the name.
* a *pure virtual method* called getArea() that takes no arguments and returns a double. Since it is pure virtual (or abstract), this method has no definition, only a prototype. It must be defined in any concrete class derived from Shape.

**class Circle**

Circle is derived from Shape using public inheritance.

*Data Members*

A Circle has the following private data member:

* a radius, which is of type int,

*Methods*

The Circle class should have the following methods:

* a "get" method that returns the circle's radius.
* a constructor that takes a string to initialize the circle's name and an int to initialize the circle's radius. The name string should be passed to the Shape constructor.
* an overridden version of getArea() that takes no arguments and returns a double. This method should compute and return the circle's area based on its radius.
* an overridden version of print() that takes no arguments and returns nothing. The method should call the base class print() method to print the name, followed by the circle's radius and area, e.g.:
* green circle, radius 10, area 314.16

**class Cylinder**

Cylinder is derived from Circle using public inheritance.

*Data Members*

A Cylinder has the following private data member:

* a height, which is of type int,

*Methods*

The Cylinder class should have the following methods:

* a constructor that takes a string to initialize the cylinder's name and two ints to initialize the cylinder's radius and height. The name string and the radius should both be passed to the Circleconstructor.
* an overridden version of getArea() that takes no arguments and returns a double. This method should compute and return the cylinder's surface area based on its radius and height.
* a *virtual* method called getVolume() that takes no arguments and returns a double. This method should compute and return the cylinder's volume based on its radius and height.
* an overridden version of print() that takes no arguments and returns nothing. The method should call the print() method for Circle to print the name, radius, and area, then print the cylinder's height and volume, e.g.:
* blue cylinder, radius 8, area 703.72, height 6, volume 1206.37

**class Cone**

Cone is derived from Circle using public inheritance.

*Data Members*

A Cone has the following private data member:

* a height, which is of type int,

*Methods*

The Cone class should have the following methods:

* a constructor that takes a string to initialize the cone's name and two ints to initialize the cone's radius and height. The name string and the radius should both be passed to the Circle constructor.
* an overridden version of getArea() that takes no arguments and returns a double. This method should compute and return the cone's surface area based on its radius and height.
* a *virtual* method called getVolume() that takes no arguments and returns a double. This method should compute and return the cone's volume based on its radius and height.
* an overridden version of print() that takes no arguments and returns nothing. The method should call the print() method for Circle to print the name, radius, and area, then print the cone's height and volume, e.g.:
* yellow cone, radius 4, area 140.88, height 6, volume 100.53

**Main Program**

Write a main program to test your classes. The program should do the following:

1. Create either an array or an STL vector of pointers to Shape objects.
2. Dynamically create some Circles, Cylinders, and Cones (at least two of each). After creating each object, add it to the array or vector.
3. Loop through the array or vector of Shape pointers and call the print() method for each of them.
4. Loop through the array or vector of Shape pointers again and call the print() method for each of the Cylinder objects in the array or vector.
5. Loop through the list of Shape pointers one more time and delete each object.

Output from the program should look something like this:

Printing all shapes...

green circle, radius 10, area 314.16

yellow cone, radius 4, area 140.88, height 6, volume 100.53

blue cylinder, radius 8, area 703.72, height 6, volume 1206.37

purple cone, radius 9, area 576.85, height 7, volume 593.76

red cylinder, radius 3, area 188.50, height 7, volume 197.92

orange circle, radius 5, area 78.54

Printing only cylinders...

blue cylinder, radius 8, area 703.72, height 6, volume 1206.37

red cylinder, radius 3, area 188.50, height 7, volume 197.92

Feel free to use different values for color, radius, and height when you create your shapes.

**Other Points**

* Formulas for area and volume for the various shapes can easily be found online, as can calculators that will let you check your work. The constant M\_PI is available on many C++ compilers in the header file <cmath>.
* Floating-point values such as the area and volume should be printed with two places after the decimal point.
* Your code should enforce const correctness. Methods that do not alter the object should be const, for example.
* Code for each of your classes should be placed in separate files in the usual fashion for non-template C++ classes. That means a header file and a source code file for each class. Do not submit all of the code in one big file just because you're feeling lazy. :-)
* You must have a complete makefile, with a separate rule for compiling a separate object file for the code from each source code file. The linking rule should link all of your object files together to create one executable. The name of your final executable should be assign9.
* Everything inheritance- and polymorphism-related that needs to be done in this assignment is also done in the inheritance example available on the course website.
* Remember, use of the dynamic\_cast operator requires Run Time Type Information (RTTI) to be enabled. It's on by default in g++, but may not be if you're working with a different compiler / IDE.
* Don't forget to get rid of all compiler warnings when the -Wall compilation option is used.
* As always, programs that do not compile on turing / hopper automatically receive 0 points.
* Submit your program using the electronic submission guidelines posted on the course web site and discussed in class.