Polymorphism

Run-Time and Compile-Time Binding

```
#include<iostream>
using namespace std;
void HelloWorld( );
int main( )
{  HelloWorld( );
   return 0;
}

HelloWorld( )
{  cout << "Hello World" << endl;
   }
```

Take any program, such as our Hello World program. The complier binds any call to HelloWorld( ) to the code that implements it. In this case it is only a cout statement. This is *compile-time* binding because the compiler determines what code is to be executed.

Run-Time and Compile-Time Binding

That is, the compiler binds any call of HelloWorld( ) to HelloWorld( )'s entry point.

So far, all of our examples were compile-time binding.
The alternative is *run-time binding*.
That is, the binding of a function's name to an entry point when the program is running and not when it's being compiled.

Polymorphism

- A function is *polymorphic* if its binding occurs at run time instead of compile time.
- In C++, functions may or may not be polymorphic. C++ is a hybrid language.
- In SmallTalk, a pure object oriented language, all functions are polymorphic.

Three Requirements

1. There must be an inheritance hierarchy
2. The classes in the hierarchy must have a virtual method with the same signature
3. There must be either a pointer or a reference to a base class. The pointer or reference is used to invoke a virtual method.

An Example

```
#include<iostream>
using namespace std;
class geometricobject {  //baseclass
public:
    virtual void geo( ) {cout << "Just a geometric object" << endl; }
};
class rectangle : public geometricobject { //derived class
public:
     virtual void geo( ) {cout << "I'm a rectangle" << endl;}  };
class circle : public geometricobject {   // derived class
public:
     virtual void geo( ) {cout << "I'm a circle" << endl; }  };
```

```
int main( )
{ geometricobject *ptr;  //pointer to base class
  int which;

     do  {  cout << "Enter  1 for geometricobject, 2 for rectangle
                                  3 for circle ";
          cin >> which; }   while (which  < 1 || which > 3);

switch (which) {   //set pointer depending on user choice
    case  1: ptr = new geometricobject;  break;
    case 2: ptr = new rectangle;  break;
    case 3: ptr = new circle;  break;
}
ptr -> geo( );    //run-time binding
delete ptr;       //don't leave junk hanging around
return 0;
}
```

Polymorphism Assignment

- Take the polymorphic geometric object class skeleton program and add another geometric object derived class. This class could be a triangle class, or pentagon, or hexagon class, etc.
- Add methods to each derived class (rectangle, circle, and the new derived class you created) to compute the area of each geometric figure. That is, the user will be prompted to enter the proper dimensions and the program will calculate the area.
- Note, you MUST use polymorphism. That is, the classes are virtual.