Polymorphism

- The word ***polymorphism*** means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- C++ polymorphism means that a call to a method will cause a different method to be executed depending on the type of object that invokes the method.
- Consider the above `Shape` example where a base class has been derived by other two classes, `Rectangle` and `Circle`. But now we change the main() function like so,

```
int main()
{
      Shape *pShape;
      Rectangle rect(5, 7, 10, 7);
      Circle circ(2, 3, 4);

      // store the address of rect and print
      pShape = &rect;
      cout << pShape->getArea() << endl;

      // store the address of circ and print
      pShape = &circ;
      cout << pShape->getArea() << endl;

      return 0;
}
```

- Note that this gives the incorrect output, with `getArea()` returning 0.
- The reason for the incorrect output is that the call of the function `getArea`() is being set once by the compiler as the version defined in the base class.
- This is called ***static resolution*** of the function call, or ***static linkage***.
- Make a slight modification to the program and precede the declaration of `getArea`() in the `Shape` class with the keyword `virtual` so that it looks like this,

```
// Note: getArea() in Shape needs to be virtual
virtual float getArea()
{
      cout << "getArea():";
      return 0;
}
```

- This time, the compiler looks at the contents of the pointer instead of it's type. Since addresses of objects of `rect` and `circ` classes are stored in *`pShape` the correct `getArea`() is called.
- As you can see, each of the child classes has a separate implementation for the function getArea(). This is how ***polymorphism*** is generally used. You have different classes with a function of the same name, but with different implementations.
- A virtual function is a function in a base class that is declared using the keyword `virtual`.
- Defining in a ***base*** class a virtual function, with another version in a ***derived*** class, signals to the compiler that we don't want static linkage for this function, instead, the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as ***dynamic linkage***.

- Here is another polymorphic `main()` example using pointer references and new() worth considering.

```
//
// By reference example using new()
//
int main(void)
{
    Shape *spot = new Shape(6, 6);
    Shape *rect = new Rectangle(5, 7, 8, 8);
    Shape *circ = new Circle(2, 3, 4);

    cout << "spot- ("
         << spot->getX() << ","
         << spot->getY() << ") "
         << spot->getArea() << endl;

    cout << "rect- ("
         << rect->getX() << ","
         << rect->getY() << ") "
         << rect->getArea() << endl;

    cout << "circ- ("
         << circ->getX() << ","
         << circ->getY() << ") "
     << circ->getArea() << endl;

    delete spot;
    delete rect;
    delete circ;

    return 0;
}
```