# Lab No.7 Polymorphism and Miscellaneous Topics

## 7.1 Introduction

In object oriented programming, the type compatibility of a derived class pointer to its base pointer is a significant feature. Polymorphism uses this feature and enables object to acquire various forms while referencing various instances of different classes in inheritance hierarchy. To realize polymorphism, virtual function and base class pointer is required. This lab covers polymorphism and virtual function in detail. Also, abstract and concrete class is discussed.

## 7.2 Objectives of the lab:

1. Understand the difference between static and dynamic binding.
2. Demonstrate polymorphism with the help of virtual function and base class pointer.
3. Develop abstract class using pure virtual function.

## 7.3 Pre-Lab

### 7.3.1 Polymorphism

1. A generic term that means 'many shapes'. In C++ the simplest form of Polymorphism is overloading of functions.
2. Ability for objects of different classes to respond differently to the same function call
3. Base-class pointer (or reference) calls a **virtual** function
   - C++ chooses the correct overridden function in object
4. Attained by making a function **virtual** in base class.
5. Keyword **virtual** is used in function declaration.
6. Keyword **virtual** is not necessary to be used in derived classes. The overridden functions in derived classes are virtual automatically.

### 7.3.2 Example: polymorphism.cpp

```
#include <iostream>
using namespace std;

class shape{
    public:
            virtual void draw(){
                    cout<<"Draw shape"<<endl;
            }
};
class circle: public shape{
    public:
            void draw(){      // child function can't be virtual
                    cout<<"Draw circle"<<endl;
            }
```

`

```
};
class rectangle: public shape{
    public:
        void draw(){     // child function can't be virtual
            cout<<"Draw rectangle"<<endl;
        }
};
class triangle: public shape{
    public:
        void draw(){     // child function can't be virtual
            cout<<"Draw triangle"<<endl;
        }
};

int main(){
    shape *sh;                          // special: pointer object
    circle c1; rectangle r1; triangle t1;   // child objects

    sh = &c1;                           // sh is now pointing to circle object
    sh->draw();                         // draw circle

    sh = &r1;                           // sh is now pointing to rectangle object
    sh->draw();                         // draw rectangle

    sh = &t1;                           // sh is now pointing to triangle object
    sh->draw();                         // draw tritangle

    return 0;
}
```

## Output:

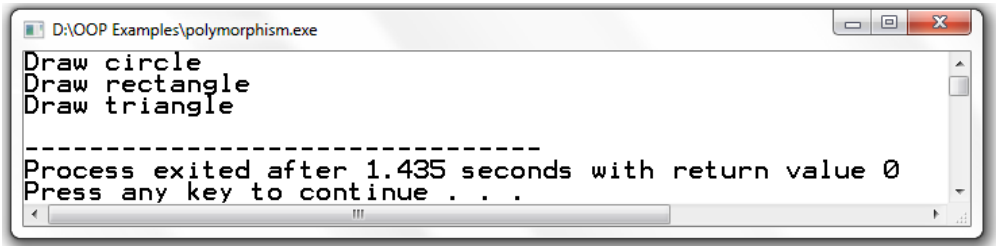


**Figure 7.1: Output polymorphism.cpp**

## Q. What is the effect of the following statement?

**shape s;**

### 7.3.3 Abstract and Concrete Classes

1 Abstract classes

- Sole purpose is to provide a base class for other classes

`

- No objects of an abstract base class can be instantiated
  - Too generic to define real objects, i.e. **TwoDimensionalShape**
- Can have pointers and references

2 Concrete classes
- classes that can instantiate objects

3 Provide specifics to make real objects , i.e. **Square**, **Circle**

4 An instance of abstract class cannot be created

5 A derived class of an abstract base class remains abstract unless the implementation of all the pure virtual functions is not provided.
- Derived class inherits the pure virtual function
- Any class having a pure virtual function is virtual
- Override the pure virtual function in derived class. (Do not provide a 0 in declarator)

### 7.3.4 How to make a class abstract?

1 Making abstract classes

2 Declare one or more virtual functions as "**pure**" by initializing the function to zero **virtual double earnings() const = 0;**

3 A class with no pure virtual function is a concrete class.

## 7.4 Activities

**Perform this activity in C++ only.**

### 7.4.1 Activity

Define an abstract base class **shape** that includes protected data members for area and volume of a shape, public methods for computing area and volume of a shape (make these functions **virtual**), and a display function to display the information about an object. Make this class abstract by making display function pure virtual.

Derive a concrete class **point** from the **shape** class. This **point** class contains two protected data members that hold the position of **point**. Provide no-argument and 2-argument constructors. Override the appropriate functions of base class.

Derive a class **Circle** publicly from the **point** class. This class has a protected data member of radius. Provide a no-argument constructor to initialize the fields to some fixed values. Provide a 3-argument constructor to initialize the data members of **Circle** class to the values sent from outside. Override the methods of base class as required.

Derive another class **Cylinder** from the **Circle** class. Provide a protected data member for height of cylinder. Provide a no-argument constructor for initializing the data members to default values. Provide a 4-argument constructor to initialize x- and y-coordinates, radius, and height of cylinder. Override the methods of base class.

Write a driver program to check the polymorphic behavior of this class.

## 7.5 Testing

`

**Test Cases for Activity 7.4.1**

| Sample Inputs | Sample Outputs |
|---|---|
| Test for inputs of your choice and show results. | |

## 7.6 References:


1. **Class notes**
2. **Object-Oriented Programming in C++ by *Robert Lafore***
3. **How to Program C++ by *Deitel & Deitel***
4. **Programming and Problem Solving with Java by *Nell Dale & Chip Weems***
5. **Murach's Python Programming by *Micheal Urban & Joel Murach***


`