# NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

# OBJECT ORIENTED PROGRAMMING - LAB

Instructors: Muhammad Sudais

Email: muhammad.sudais.v@nu.edu.pk

## LAB 07

## Outline

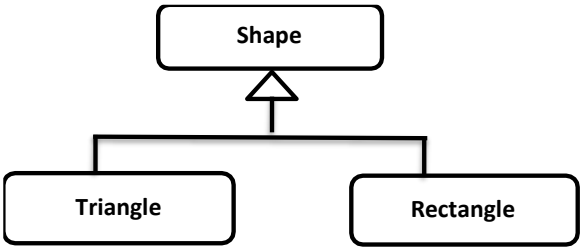
- Types of Inheritance
- Polymorphism
- Function Overloading
- Function Overriding

## TYPES OF INHERITANCE

- SINGLE INHERITANCE
- HIERARICAL
- MULTILEVEL
- MULTIPLE INHERITANCE
- HYBRID INHERITANCE

### HIERARICAL INHERITANCE

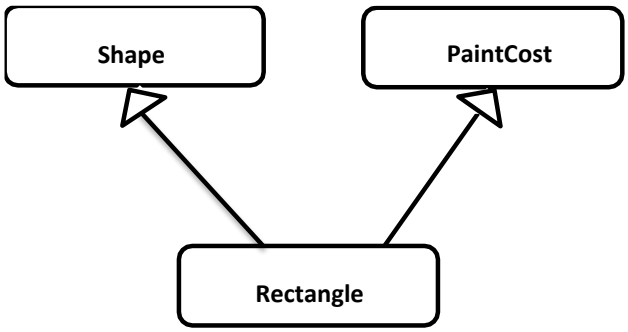
This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.




```
#include <iostream>
using namespace std;
class Shape
{
        protected:
                float width, height;
        public:
                void set_data (float a, float b)
                {
                        width = a;
                        height = b;
                }
};
class Rectangle: public Shape
{
public:
        float area ()
          {
                return (width * height);
          }
};

class Triangle: public Shape  {
public:
        float area ()
        {
                return (width * height / 2);
        }
};
int main ()
{
        Rectangle rect;
        Triangle tri;
        rect.set_data (5,3);
        tri.set_data (2,5);
        cout << rect.area() << endl;
        cout << tri.area() << endl;
        return 0;
}
```

## MULTIPLE INHERITANCE

In multiple inheritance, one class inherits the features of multiple classes simultaneously, hence this type of inheritance is called Multiple Inheritance.

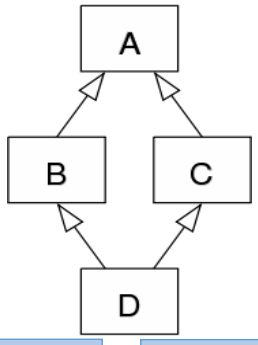



```
#include <iostream>
using namespace std;
class A
{
   protected:
    int a;
   public:
   void get_a(int n)
   {
      a = n;
   }
};

class B
{
   protected:
   int b;
   public:
   void get_b(int n)
   {
      b = n;
   }
};
```

```
class C : public A, public B
{
   public:
    void display()
    {
       std::cout << "The value of a is : " <<a<< std::endl;
       std::cout << "The value of b is : " <<b<< std::endl;
       cout<<"Addition of a and b is : "<<a+b;
    }
};
int main()
{
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();

    return 0;
}
```

## HYBRID INHERITANCE

Hybrid inheritance is a combination of more than one type of inheritance.




```
#include <iostream>
using namespace std;
class A
{
   protected:
   int a;
   public:
   void get_a()
   {
      std::cout << "Enter the value of 'a' : " << std::endl;
      cin>>a;
   }
};

class B : public A
{
   protected:
   int b;
   public:
   void get_b()
   {
       std::cout << "Enter the value of 'b' : " << std::endl;
      cin>>b;
   }
};
class C
{
   protected:
   int c;
   public:
   void get_c()
   {
       std::cout << "Enter the value of c is : " << std::endl;
       cin>>c;
   }
};
```

```
class D : public B, public C
{
   protected:
   int d;
   public:
   void mul()
   {
       get_a();
       get_b();
       get_c();
       std::cout << "Multiplication of a,b,c is : " <<a*b*c<<
std::endl;
   }
};
int main()
{
   D d;
   d.mul();
   return 0;
}
```

# POLYMORPHISM

Polymorphism refers to the ability of a method to be used in different ways, that is, it can take different forms at different times (poly + morphos).

## TYPES OF POLYMORPHISM

There are two types of polymorphism:

- Compile time polymorphism
- Run time polymorphism.

## COMPILE TIME POLYMORPHISM

In C++ you can achieve compile time polymorphism by,

- Constructor Overloading (have discussed in the previous labs)
- Function/Method Overloading
- Operator Overloading

## Function Overloading

Two or more functions can have the same name but different parameters; such functions are called function overloading.

C++ has many features, and one of the most important features is function overloading. It is a code with more than one function with the same name having various types of argument lists. This argument list includes the data type of arguments and the sequence of the arguments.

The function overloading feature is used to improve the readability of the code. It is used so that the programmer does not have to remember various function names. If any class has multiple functions with different parameters having the same name, they are said to be overloaded. If we have to perform a single operation with different numbers or types of arguments, we need to overload the function.

In OOP, function overloading is known as a function of polymorphism. The function can perform various operations best on the argument list. It differs by type or number of arguments they hold. By using a different number of arguments or different types of arguments, the function can be redefined.

### Rules of Function Overloading in C++

Different parameters or three different conditions :

1. These functions have different parameter type
   sum(int a, int b)
   sum(double a, double b)
2. These functions have a different number of parameters
   sum(int a, int b)
   sum(int a, int b, int c)
3. These functions have a different sequence of parameters
   sum(int a, double b)
   sum(double a, int b)

## Function Overriding

If we inherit a class into a Derived class and provide definition of base Class function again inside a derived class, then that function said to be overridden and this mechanism is called function overriding. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the function which is already provided by its base class.

Note: In function overriding, the function in parent class is called the overridden function and function in child class is called overriding function.

### Requirements For Function Overriding:

- Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class.
- Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

### Access Overridden Function in C++

To access the overridden function of the base class, we use the scope resolution operator **::** .

We can also access the overridden function by using a pointer of the base class to point to an object of the derived class and then calling the function from that pointer.

# Examples of Overloading and Overriding:

**Function Overloading without Class**

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

void print(int i) {
  cout << " Here is int " << i << endl;
}
void print(double  f) {
  cout << " Here is float " << f << endl;
}
void print(char const *c) {
  cout << " Here is char* " << c << endl;
}

int main() {
  print(10);
  print(10.10);
  print("ten");
  return 0;
}
```

**Function Overloading in a Class**

```
class Addition
 {
   public:
   int sum(int a,int b)
   {
      return a+b;
   }
   int sum(int a,int b, int c)
   {
     return a+b+c;
   }
 };
int main(void)
{
   Addition obj;
   cout<<obj.sum(20, 15)<<endl;
   cout<<obj.sum(81, 100, 10);
   return 0;
}
```

**Function Overriding**

```
class Base
{
 public:
 void show()
 {
  cout << "Base class\t";
 }
};
class Derived:public Base
{
 public:
 void show()
 {
  cout << "Derived Class";
 }
};
int main()
{
   Base b;      //Base class object
   Derived d;    //Derived class object
   b.show();    //Early Binding Occurs
   d.show();
}
```

# EXERCISE

## Question 1

Implement the inheritance shown in Figure 1.

## Question 2

Implement the inheritance shown in Figure 2.

1. All the classes must have a static data member.
2. All the classes must have constructor.
3. Make an overridden function speak() with different implementation from Animal Class to Bear and Dog Class.
4. In the driver function (main), show the use of static data member, constructor and overriding functions of all the classes.

## Question 3

Implement Hierarchal Inheritance shown in Figure 3. Make sure to print “Overloaded Function” in overloaded functions and “ Overriding Function” in overriding function before its implementation. Also make a driver program to test all the functionalities

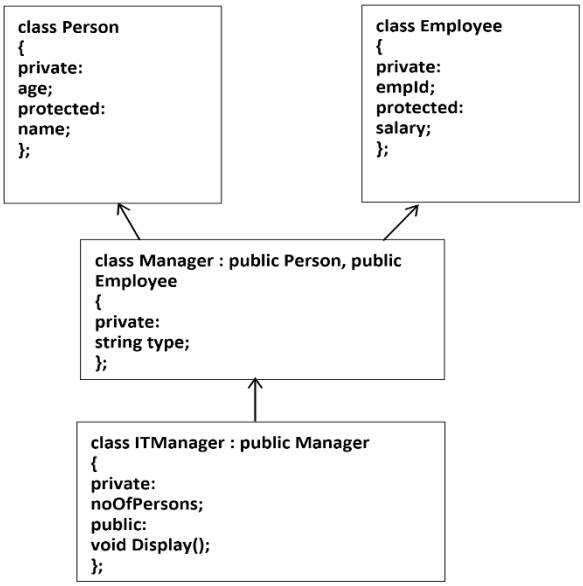
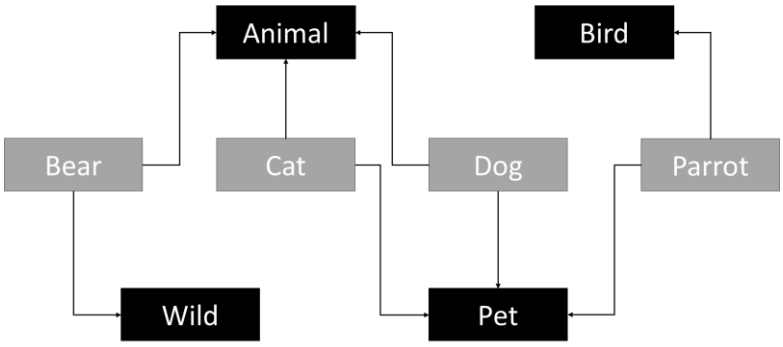



Figure 1

Figure 2



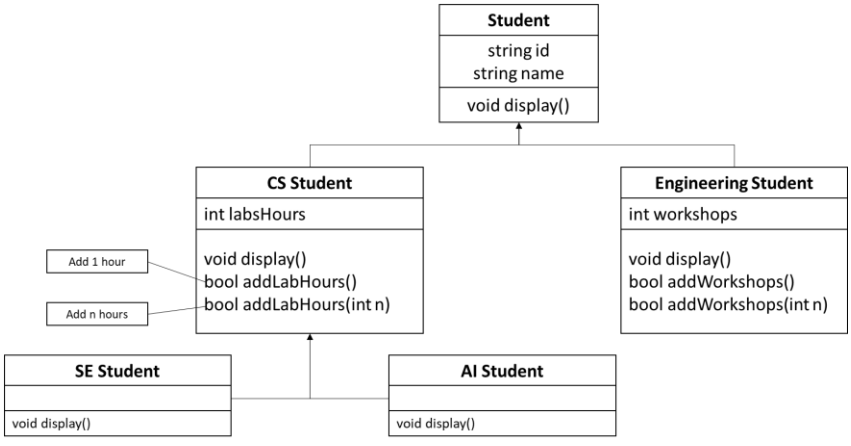




Figure 3