Virtual Function

Object Oriented Programming

C++

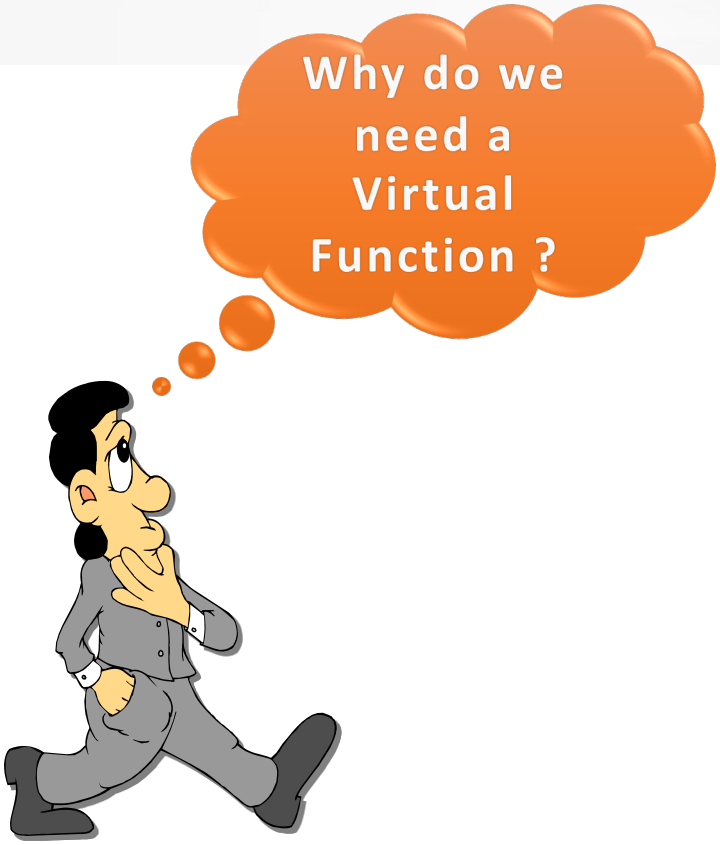
Resource Person:

Asma Naseer

Contents

1. Virtual Function
2. Virtual Class
3. Abstract Class
4. Interface
5. Templates
6. GUI

Introduction



Polymorphism

Polymorphism is a feature of OOPs that allows the object to behave differently in different conditions. In C++ we have two types of polymorphism:

1) Compile time Polymorphism – This is also known as static (or early) binding.
2) Runtime Polymorphism – This is also known as dynamic (or late) binding

Function overloading and Operator overloading are perfect example of Compile time polymorphism

Polymorphism

Virtual Function

Class Animal

```
class Animal {
public:
    void eat() {
            std::cout << "I'm eating generic food."; } };
```

Class Cat

```
class Cat: public Animal {
public:
    void eat() {
            std::cout << "I'm eating cat food."; } };
```

Virtual Function

main

```
void main() {
        Animal *animal = new Animal;
        Cat *cat = new Cat;
        animal->eat();
        cat->eat(); }
```

1. I am eating generic food. I am eating cat food.
2. I am eating cat food. I am eating cat food.
3. I am eating generic food. I am eating generic food.

Virtual Function

func()

```
void func(Animal *xyz) {
        xyz->eat();
}
```

```
void main(){
        Animal *animal = new Animal;
        Cat *cat = new Cat;
        func(animal);
        func(cat); }
```

1. I am eating generic food, I am eating cat food
2. I am eating cat food, I am eating cat food
3. I am eating generic food, I am eating generic food

Virtual Function

Class Animal

```
class Animal {
public:
    virtual void eat() {
        std::cout << "I'm eating generic food.";
} };
```

Class Cat

```
class Cat : public Animal {
public:
    void eat() {
        std::cout << "I'm eating cat food.";
} };
```

Virtual Function

func()

```
void func(Animal *xyz) {
        xyz->eat();
}
```

```
void main(){
        Animal *animal = new Animal;
        Cat *cat = new Cat;
        func(animal);
        func(cat); }
```

1. I am eating generic food, I am eating cat food
2. I am eating cat food, I am eating cat food
3. I am eating generic food, I am eating generic food

Virtual Function

Recap

```
void main() {
        Animal *animal = new Animal;
        Cat *cat = new Cat;
        animal->eat();
        cat->eat(); }
```

1. I am eating generic food, I am eating cat food
2. I am eating cat food, I am eating cat food
3. I am eating generic food, I am eating generic food

Virtual Inheritance (Virtual Base Class)

```
class A
{
   public:
      int i;
};
```

```
class B : virtual public A
{
   public:
      int j;
};
```

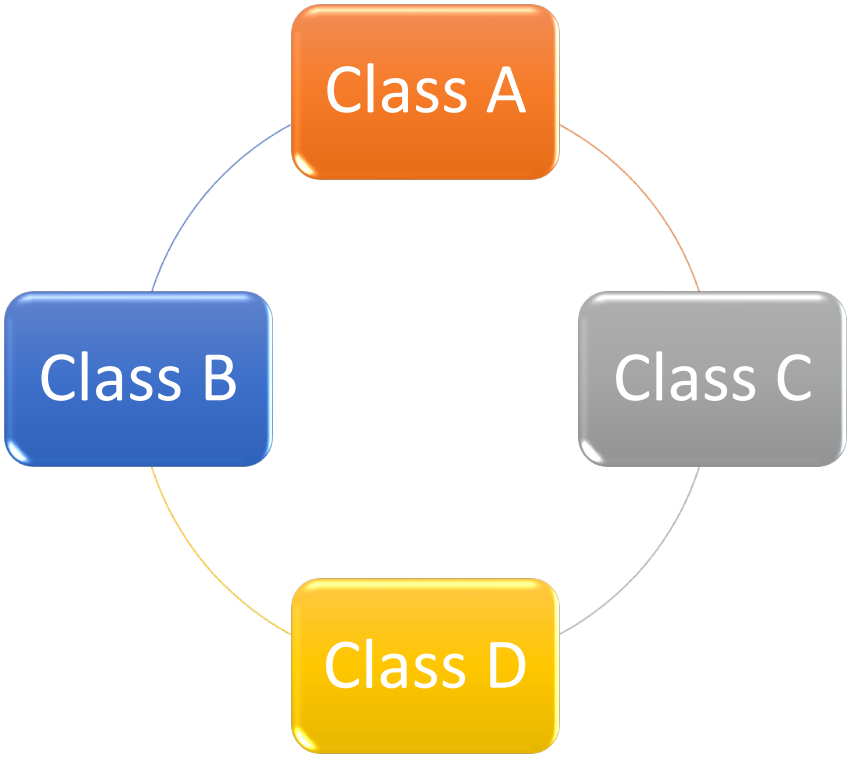
```
class C : virtual public A
{
   public:
      int k;
};
```

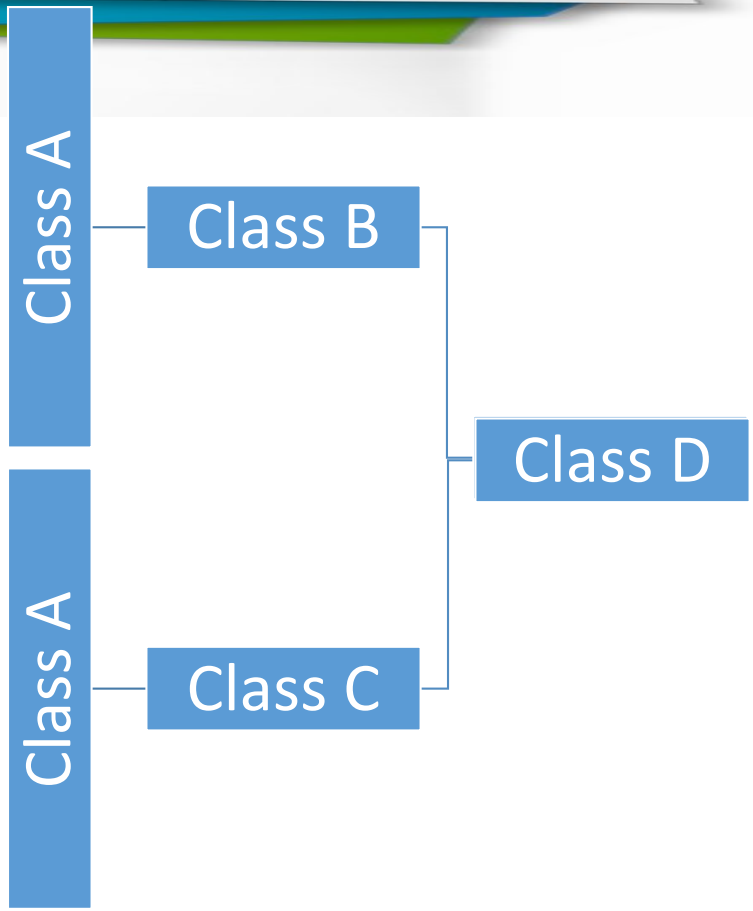
Virtual Class

```
class D: public B, public C
{
   public:
      int sum;
};
```

```
void main() {
   D ob;
   ob.i = 10;
   ob.j = 20;
   ob.k = 30;
   ob.sum = ob.i + ob.j + ob.k;
   cout << “Value of i is : ”<< ob.i<<”\n”;
   cout << “Value of j is : ”<< ob.j<<”\n”;
            cout << “Value of k is :”<<
ob.k<<”\n”;
   cout << “Sum is : ”<< ob.sum <<”\n”;  }
```

Virtual Class





Pure Virtual Function

```
virtual void AbstractMemberFunction ( ) = 0;
```

Abstract Class

```
class A {
public :
        virtual void fun() = 0;    // Pure virtual function makes
                                   // this class Abstract class.
};
```

Abstract class cannot be used as

- **a parameter type,**
- **a function return type**
- **type of an explicit conversion**
- **an object**

It can be used to declare

- **pointers and references to an abstract class.**

Abstract Class

```
class AbstractClass {
public :
        virtual void AbstractMemberFunction() = 0; // Pure virtual function makes
                                                   // this class Abstract class.
        Virtual void NonAbstractMemberFunction1(); // Virtual function.
        void NonAbstractMemberFunction2();
};
```

Abstract Class

```
class Base //Abstract base class
{ public:
      virtual void show() = 0; //Pure Virtual Function
};

class Derived:public Base
{ public:
void show() {
      cout << "Implementation of Virtual Function in Derived class";
} };

void main() {
Base obj; //Compile Time Error
Base *b;
Derived d;
b = &d;
b->show(); }
```