Virtual Functions & Polymorphism

Polymorphism:- It is one of the crucial features of OOP. It simply means "one name, multiple forms". It is implemented through overloaded functions and operators.

Polymorphism

Early Binding

When the information for calling the member functions by matching arguments both type and number, is known to the compiler at compile time. Therefore, compiler is able to select the appropriate function for a particular call at the compile time itself. This is called early binding or static binding or static linking or compile time polymorphism.

This is achieved by :-

- function overloading
- operator overloading

Late Binding

When the member function is selected while the program is running. It is called run-time polymorphism or late-binding or dynamic binding. At run time, when it is known what class objects are under consideration, the appropriate version of the function is invoked. The function is linked with a particular class much later after the compilation this process is called late bind

This is achieved by

↓

Virtual functions.

Virtual functions

When we use same function name in both the base and derived classes, the function in base class is declared as virtual by using the keyword virtual preceding its normal declaration. When a function is made virtual, c++ determines which function to use at run-time based on the type of object pointed to by the base-pointer, rather than the type of pointer.

A virtual function is a function which is defined in base class and is re-defined (over-ridden) in derived class.

Rules for virtual function :-

1. They must be declared in public section of class.
2. They cannot be static & also cannot be friend of another class.
3. They can be accessed using pointer or reference of base class type to achieve run-time polymorphism.
4. The prototype of virtual functions should be same in base as well as derived class.
5. They are always defined in base class & over-ridden in deriv
6. A class may have virtual destructor but not virtual construct
7. They are accessed by using object-pointers.
8. A virtual function in a base class must be defined, even though it may not be used.
9. While a base pointer can point to any type of the derived object, the reverse is not true.

Without Virtual keyword

```
class Base
{
public:
    void show()
    {
    cout << "Base class";
    }
};
class derived : public base
{
public:
    void show()
    {
    Cout << "Derived class";
    }
};
int main()
{
  Base * b; // Base class pointer
  Derived d;
  b = &d;
  b -> show(); //early binding occurs
}
```

o/p

```
Base class
```

Base class pointer will always call base class function without virtual.

Using virtual keyword

```
class Base
{
public:
    virtual void show()
    {
    cout << "Base class";
    }
};
class Derived: public base
{
public:
    void show()
    {
    cout << "Derived class
    }
};
int main()
{
  Base * b; //Base class pointer
  Derived d;
  b = &d;
  b -> show() //Late binding occurs
}
```

o/p

```
Derived class
```

Now Base class pointer points to derived class object ∴ derived class function is called

⇒ Pure Virtual functions

The virtual function inside the base class is seldom used for performing any task. It only serves as a placeholder. Such functions are called "do-nothing" functions. They are defined as follows:

```
Virtual void display () = 0;
```

Such functions are called pure-virtual functions. It is declared in base class & has no definition relative to base class. In such cases, the compiler requires each derived class to either define the function or redeclare it as pure virtual functions.

```
class Base
{
  int x;
  public: virtual void fun()=0;
          int getx() { return x;}
};
class Derived : public Base
 { int y;
public : void fun ()
         {
          cout << "Hello";
         }
};
int main ()
{ Derived d;
  d.fun ();
  return 0;
```

o/p

```
Hello
```

Abstract Class :-

Abstract class is a class which contains atleast one Pure Virtual function in it. Abstract classes are used to provide an interface for its sub classes. Classes inheriting an Abstract class must provide definition to the pure virtual function, otherwise they will also become abstract class.

Characteristics of Abstract Class :—

1. Abstract class cannot be instantiated, but pointers & references of Abstract class type can be created.
2. They can have normal functions and variables along with a pure virtual function. It can have constructor too.
3. Classes inheriting Abstract class must implement all pure virtual functions, otherwise they will become Abstract too.

```
class Base          // Abstract class
{
int x;
public : virtual void fun() = 0; // Pure virtual function
         int getx()
         {return x; }
};
class Derived : public Base
{
  int y;
public : void fun()   // Function defined in derived class
         { cout << "Hello"; }
};
void main()
{ Derived d;
  d.fun();
}
```

Output

Hello