

Polymorphism

- Polymorphism is an ability of a C++ object to take many forms.
- The most important use of polymorphism in C++ OOPs occurs when we want to bind functions to the objects we defined in C++ during compile-time or run-time.
- Polymorphism in C++ can be of two types:

1. Compile time polymorphism:-

- Compiler takes decision of the function to be used at the time of compilation by reading the code.
- It binds the address of function to the object (i.e. which function will be invoked by which object).
- This is also known as early binding or static binding.
- Compile time polymorphism in C++ is achieved using:
 - 1.1 - Function overloading
 - 1.2 - Operator Overloading

2. Run time polymorphism:-

- The binding of object to function call takes place at the time of run time.
- But the decision of using the particular function can be taken by reading the code.
- Here, the binding that occur is late binding
- Run time polymorphism in C++ is achieved using:
 - 2.1 - Virtual functions

Pointer to Derived Class

- A pointer of a base class type can also store the address of an object of the derived class and access the members of class by arrow operator.
- But we can't access all the variables and functions of derived class with that pointer. We can only access those members which are inherited to the derived class by the base class.
- If both base and derived class have a function of same name and a pointer of base class pointing to object of derived class is used to call that function then the function of base class will be invoked.

Virtual Function

- In above mentioned case, we can use that pointer to invoke the function of derived class also by declaring the function of base class as virtual function. This is the principle of run time polymorphism.
- **Syntax for defining virtual function in base class:**

```
virtual <returnDataType> <func.Name>(){  
    ---Function body---}
```

Rules for virtual function:

1. They cannot be static.
2. They are accessed by object pointers.
3. Virtual function can be a friend of another class.

4. A virtual function in the base class might not be used.
5. If a virtual function is defined in a base class, it does not mean that we compulsorily redefine in the derived class. If there is no function in derived class then the virtual function will be invoked for the object of derived class.

Pure Virtual Function:

- A pure virtual function is a virtual function that makes it compulsory for all of its derived class to redefine that function in their body.
- It makes sure that no derived class is created which doesn't redefine it (i.e. function).
- It doesn't perform any operation.
- It is created by a do-nothing function:

```
virtual <returnDataType> <func.Name>() = 0;
```

- It is used to create an abstract base class.

Abstract Base Class:

- A class which contains at least one pure virtual function.
- A class whose objects can't be created. Its sole purpose is to make new derived classes and then we work upon the members of the derived classes.
- The classes which are inheriting from the base class must need to override the virtual function of abstract class otherwise the compiler will throw error.

Operator Overloading

When an operator is overloaded with multiple jobs, it is known as operator overloading.

- Any symbol can be used as function name if it is a valid operator in C language.
- ‘operator’ keyword is preceded.
- Can’t use sizeof and ?: operator.

1). Operator Overloading of Binary Operator (+)

for e.g. `class Complex{`

```
.....  
Complex operator+(Complex c)  
{ .....}  
};  
  
int main(){  
Complex c1,c2,c3;  
  
.....  
c3 = c1 +c2; → Operator Overloading }
```

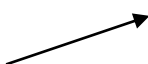

2). Operator Overloading of Unary Operator (-)

for e.g. `class Complex{`

```
.....  
Complex operator-() → no argument for unary operator  
{.....}  
};  
  
void main(){  
Complex c1, c2;  
c2 = -c1; }
```

3). Operator Overloading of Increment/Decrement Operator

for e.g. `class Integer{`

```
.....  
Integer operator++()  pre-increment  
{.....}  
Integer operator--(int)  post -increment  
{.....}  
};  
void main(){  
Integer I1, I2, I3, I4;  
.....  
I2 = ++I1;    → Pre- increment  
I3 = I4++;    → post increment  
}
```

Here, int parameter is set just to make compiler able to differentiate between post increment and pre increment operator. No argument will be passed at the time of calling the function.