POLYMORPHISM

The word "polymorphism" means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Types of Polymorphism

- **Compile-time Polymorphism.**
- **Runtime Polymorphism.**

Runtime Polymorphism

This type of polymorphism is achieved by **Function Overriding**. Late binding and dynamic polymorphism are other names for runtime polymorphism. The function call is resolved at runtime in runtime polymorphism. In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime.

Runtime Polymorphism

This type of polymorphism is achieved by **Function Overriding**. Late binding and dynamic polymorphism are other names for runtime polymorphism. The function call is resolved at runtime in runtime polymorphism. In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime.

A. Function Overriding

Function Overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

```
class Parent
{
public:
        void GeeksforGeeks()
        {
        statements;
        }
};

class Child: public Parent
{
public:
        void GeeksforGeeks()
        {
        Statements;
        }
};

int main()
{
Child Child_Derived;
Child_Derived.GeeksforGeeks();
return 0;
}
```

Virtual Function

A virtual function is a member function that is declared in the base class using the keyword virtual and is re-defined (Overridden) in the derived class.

Some Key Points About Virtual Functions:

- Virtual functions are Dynamic in nature.
- They are defined by inserting the keyword "**virtual**" inside a base class and are always declared with a base class and overridden in a child class
- A virtual function is called during Runtime

Below is the C++ program to demonstrate virtual function:

```
// C++ Program to demonstrate
// the Virtual Function
```

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

// Declaring a Base class
class GFG_Base {

public:
   // virtual function
   virtual void display()
   {
      cout << "Called virtual Base Class function" <<
            "\n\n";
   }

   void print()
   {
      cout << "Called GFG_Base print function" <<
            "\n\n";
   }
};
// Declaring a Child Class
class GFG_Child : public GFG_Base {
public:
   void display()
   {
      cout << "Called GFG_Child Display Function" <<
            "\n\n";
   }
   void print()
   {
      cout << "Called GFG_Child print Function" <<
            "\n\n";
   }
};
// Driver code
int main()
{
   // Create a reference of class bird
   GFG_Base* base;
   GFG_Child child;
   base = &child;
   // This will call the virtual function
   base->GFG_Base::display();
   // this will call the non-virtual function
   base->print();
}
```

Output

```
Called virtual Base Class function
Called GFG_Base print function
```

Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example, a LinkedList class may be allowed to access private members of Node.

A friend class can access both private and protected members of the class in which it has been declared as friend.

```
class Node {
private:
    int key;
    Node* next;
    /* Other members of Node Class */

    // Now class  LinkedList can
    // access private members of Node
    friend class LinkedList;
};
```

```
#include<iostream>
using namespace std;
class A
{
    int x;
          public:
    A()
    {
        x=10; }
    friend class B;    //friend class
};
class B
{  public:
        void display(A &t)
        {
            cout<<endl<<"The value of x="<<t.x;
        }
};

main()
{
    A _a;
    B _b;
    _b.display(_a);
    return 0;
}
```

Output

The value of x=10

In this example, class B is declared as a friend inside the class A. Therefore, B is a friend of class A.

Class B can access the private members of class A.

```
/ Example: Find the largest of two numbers using Friend Function
#include<iostream>
using namespace std;
class Largest
{
    int a,b,m;
    public:
        void set_data();
        friend void find_max(Largest);
};
void Largest::set_data()
{
    cout<<"Enter the First No:";
    cin>>a;
    cout<<"Enter the Second No:";
    cin>>b;
}
void find_max(Largest t)
{
    if(t.a>t.b)
        t.m=t.a;
    else
        t.m=t.b;

        cout<<"Maximum Number is\t"<<t.m;
}
main()
{
    Largest l;
    l.set_data();
    find_max(l);
    return 0;}
```

Output

```
Enter the First No:Enter the Second No:Maximum Number is    0
```

Output:-

```
Enter the First No: 21

Enter the Second No:32

Maximum Number is    32
```