



Polymorphism in C++

What is polymorphism?

Polymorphism means "*many forms*", and it occurs when we have many classes that are related to each other by inheritance.

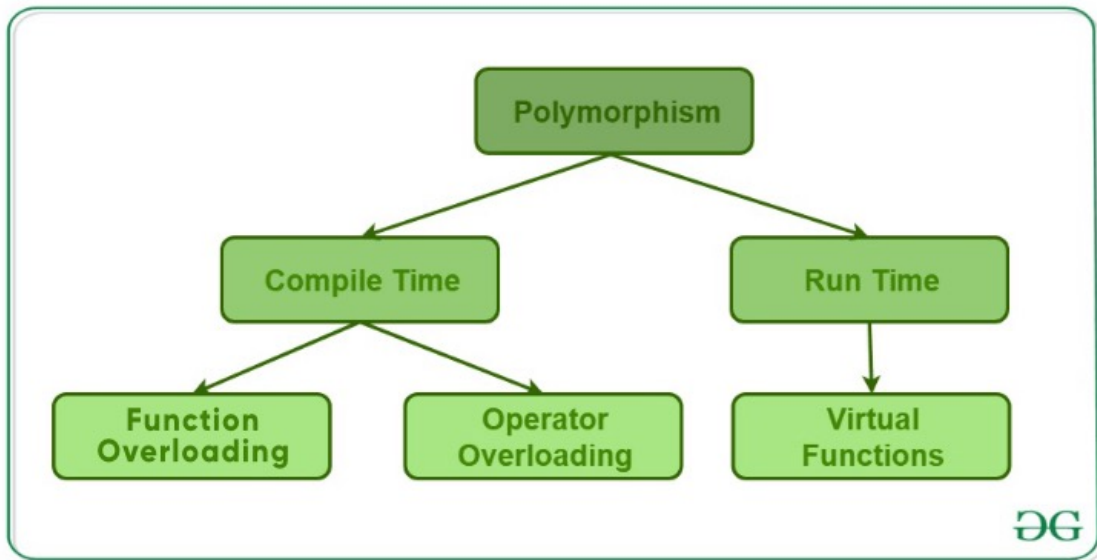
We learned already that **Inheritance** lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform ***different*** tasks. This allows us to perform a single action in different ways.

What is polymorphism?

Polymorphism enables you to "program in the general" rather than "program in the specific."

In particular, polymorphism enables you to write programs that process objects of classes that are part of the same class hierarchy as if they were all objects of the hierarchy's base class.

Types of Polymorphism





Function Overloading

Multiple functions with the same name but different parameters.

Functions can be overloaded by changing the ***number of arguments*** or/and changing the ***type of arguments***.

Function Overloading

```
class Geeks {
```

```
public:
```

```
// Function with 1 int parameter
```

```
void func(int x)
```

```
{  
    cout << "value of x is " << x << endl;  
}
```

```
// Function with same name but
```

```
// 1 double parameter
```

```
void func(double x)
```

```
{  
    cout << "value of x is " << x << endl;  
}
```

```
// Function with same name and
```

```
// 2 int parameters
```

```
void func(int x, int y)
```

```
{  
    cout << "value of x and y is " << x << ", " << y  
        << endl;  
}
```

```
// Function being called depends
```

```
// on the parameters passed
```

```
// func() is called with int value
```


```
obj1.func(7);
```

```
// func() is called with double value
```

```
obj1.func(9.132);
```

```
// func() is called with 2 int values
```

```
obj1.func(85, 64);
```



Operator Overloading

Provide the operators with a special meaning for a data type.

For example, we can make use of the addition operator (+) for string class to concatenate two strings. We know that the task of this operator is to add two operands.

Operator Overloading

public:

```
Complex(int r = 0, int i = 0)
{
    real = r;
    imag = i;
}

// This is automatically called
// when '+' is used with between
// two Complex objects
Complex operator+(Complex const& obj)
{
    Complex res;
    res.real = real + obj.real;
    res.imag = imag + obj.imag;
    return res;
}

void print() { cout << real << " + i" << imag << endl; }
```

```
Complex c1(10, 5), c2(2, 4);
```

```
// An example call to "operator+"
Complex c3 = c1 + c2;
c3.print();
```




Virtual Functions

This type of polymorphism is achieved by **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***.

Virtual Functions

```
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 Functions

```
// C++ program for function overriding
#include <bits/stdc++.h>
using namespace std;

class base {
public:
    virtual void print()
    {
        cout << "print base class" << endl;
    }

    void show() { cout << "show base class" << endl; }
};

class derived : public base {
public:
    // print () is already virtual function in
    // derived class, we could also declared as
    // virtual void print () explicitly
    void print() { cout << "print derived class" << endl; }

    void show() { cout << "show derived class" << endl; }
};

// Driver code
int main()
{
    base* bptr;
    derived d;
    bptr = &d;

    // Virtual function, binded at
    // runtime (Runtime polymorphism)
    bptr->print();

    // Non-virtual function, binded
    // at compile time
    bptr->show();

    return 0;
}
```

Output

print derived class
show base class

Virtual Functions

C++

```
// C++ program for function overriding with data members
#include <bits/stdc++.h>
using namespace std;

// base class declaration.
class Animal {
public:
    string color = "Black";
};

// inheriting Animal class.
class Dog : public Animal {
public:
    string color = "Grey";
};

// Driver code
int main(void)
{
    Animal d = Dog(); // accessing the field by reference
                      // variable which refers to derived
    cout << d.color;
}
```

Virtual Functions

C++

```
// C++ program for function overriding with data members
#include <bits/stdc++.h>
using namespace std;

// base class declaration.
class Animal {
public:
    string color = "Black";
};

// inheriting Animal class.
class Dog : public Animal {
public:
    string color = "Grey";
};

// Driver code
int main(void)
{
    Animal d = Dog(); // accessing the field by reference
                      // variable which refers to derived
    cout << d.color;
}
```

Output

Black