Polymorphism

Md. Sharif Hossen
Assistant Professor, Dept. of ICT, Comilla University,

Polymorphism

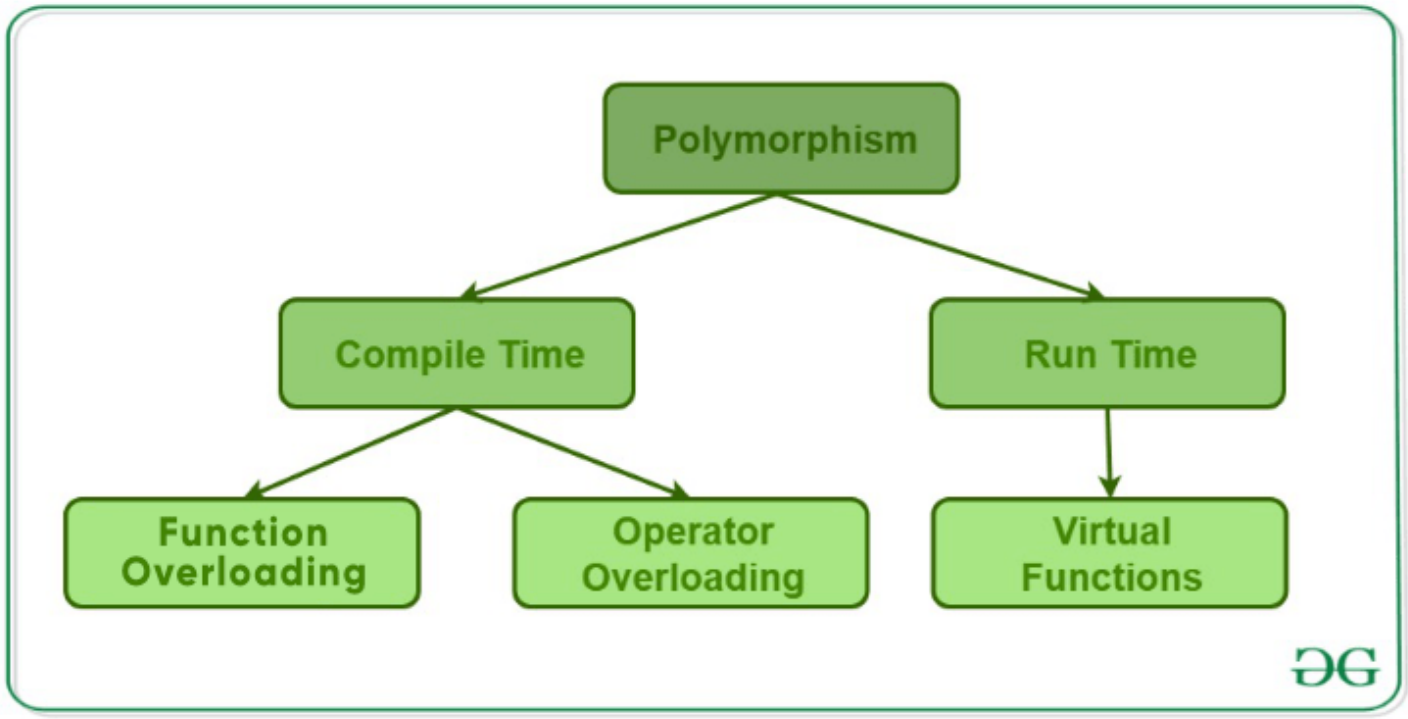
The term "Polymorphism" is the combination of **"poly" + "morphs"** which means many forms. In object-oriented programming, we use 3 main concepts: inheritance, encapsulation, and polymorphism.

Let's consider a real-life example of polymorphism. A lady behaves like a teacher in a classroom, mother or daughter in a home and customer in a market. Here, a single person is behaving differently according to the situations. Like a man at the same time is a **father, a husband, an employee**. So the same person posses **different behavior in different situations**. This is called **polymorphism**.

Polymorphism

In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism



Polymorphism/Compile time

Compile time polymorphism: The overloaded functions are invoked by matching the **type and number of arguments**. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by **function overloading and operator overloading** which is also known as **static binding or early binding**.

Function Overloading: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**. In the following example, a single function named ***func*** acts differently in three different situations which is the property of polymorphism.

Polymorphism/Compile time

Function Overloading

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

Polymorphism/Compile time

Function Overloading

```
    // function with same name and 2 int parameters
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y <<
            endl;
    }
};

int main() {

    Geeks obj1;
    obj1.func(7);
    obj1.func(9.132);
    obj1.func(85,64);
    return 0;
}
```

```
value of x is 7
value of x is 9.132
value of x and y is 85, 64
```

Md. Sharif Hossen
Assistant Professor, Dept. of ICT, Comilla University,

Polymorphism/Compile time

Operator overloading: C++ also provide option to **overload operators**. For example, we can make the **operator ('+') for string class to concatenate two strings**.

Operator overloading is a compile-time polymorphism in which the **operator is overloaded to provide the special meaning to the user-defined data type**. The advantage of Operators overloading is to perform different operations on the same operand.

Operator that cannot be overloaded are as follows:

- Scope operator (::)
- Sizeof
- member selector(.)
- member pointer selector(*)
- ternary operator(?:)

Polymorphism/Compile time

Return_type **operator** operatorname()

{

}

Memory address using &

Variables can be considered as the locations in the computer's memory which can be accessed by their identifier (their name). This way, the program does not need to care about the physical address of the data in memory; it simply uses the identifier whenever it needs to refer to the variable.

The **memory address** of a variable can be obtained using the & operator.

```
int x  = 100;
```

Pointer

A **pointer** is a variable that **stores the memory address as its value**. A pointer variable points to a data type (like int or string) of the same type, and is created with the *** operator**.

```
string food = "Pizza";  // A food variable of type string
string* ptr = &food;    // pointer variable, ptr, that stores the address of food
cout << food << "\n"; // Output the value of food (Pizza)
cout << &food << "\n"; // Output the memory address
cout << ptr << "\n"; // output the memory address
```

There are three ways to declare pointer variables:

```
string* mystring;  // Preferred
string *mystring;
string * mystring;
```

Polymorphism/Runtime

Run time polymorphism is achieved when the **object's method is invoked at the run time** instead of compile time. It is achieved by **method overriding** which is also known as **dynamic binding or late binding**.

Function overriding on the other hand 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.

C++ virtual function

- A C++ virtual function is a member function in the base class that you **redefine in a derived class**. It is declared using the virtual keyword.
- It is used to **tell the compiler to perform dynamic linkage or late binding** on the function.

Md. Sharif Hossen
Assistant Professor, Dept. of ICT, Comilla University,

Polymorphism/Runtime

C++ virtual function

- There is a **necessity to use the single pointer** to refer to all the objects of the different classes. So, we **create the pointer to the base class** that refers to all the derived objects. But, when base class pointer **contains the address of the derived class** object, always **executes the base class function**. This issue can only be resolved by using the 'virtual' function.
- A 'virtual' is a keyword preceding the **normal declaration** of a function. When the function is made **virtual**, C++ determines **which function is to be invoked at the runtime** based on the type of the object pointed by the base class pointer.

Polymorphism/Runtime

In late binding function call is resolved during runtime. Therefore compiler determines the type of object at runtime, and then binds the function call.

Rules of Virtual Function

- Virtual functions must be **members** of some class.
- Virtual functions **cannot be static** members.
- They are **accessed through object pointers**.
- They can be a **friend** of another class.
- A virtual function **must be defined in the base** class, even though it is not used.
- We **cannot have a virtual constructor**, but we can have a virtual destructor
- Consider the situation when we don't use the virtual keyword.

Polymorphism/Runtime

- The **prototypes of a virtual** function of the base class and all the derived classes must be **identical**. If the two functions with the same name but **different prototypes**, C++ will consider them as the **overloaded functions**.

Pure Virtual Function

- A virtual function is not used for performing any task. It only serves as a placeholder.
- When the function has no definition, such function is known as "**do-nothing**" function.
- The "**do-nothing**" function is known as a **pure virtual function**. A pure virtual function is a function declared in the base class that has no definition relative to the base class.

Polymorphism/Runtime

- A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.
- The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.

Polymorphism/Runtime

```
#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:
    void print () //print () is already virtual
    { cout<< "print derived class" <<endl; }

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

Polymorphism/Runtime

Compile time polymorphism	Run time polymorphism
The function to be invoked is known at the compile time.	The function to be invoked is known at the run time.
It is also known as overloading, early binding and static binding.	It is also known as overriding, Dynamic binding and late binding.
Overloading is a compile time polymorphism where more than one method is having the same name but with the different number of parameters or the type of the parameters.	Overriding is a run time polymorphism where more than one method is having the same name, number of parameters and the type of the parameters.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution as it is known at the compile time.	It provides slow execution as it is known at the run time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as all the things execute at the run time.