Lecture 16

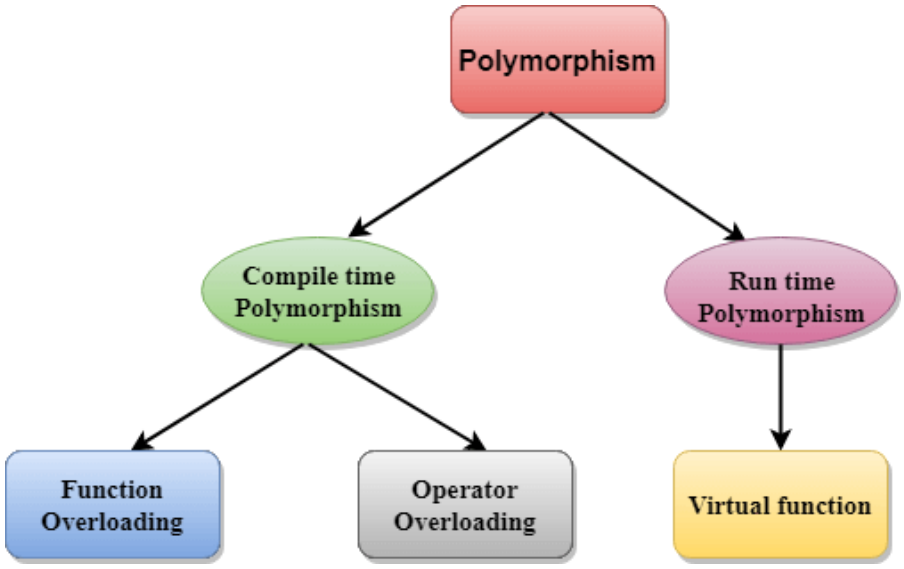
C++ Polymorphism

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

Real Life Example of 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.

There are two types of polymorphism in C++:



- **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. Now, let's consider the case where function name and prototype is same.

- **Run time polymorphism**: 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.

Differences b/w compile time and run time polymorphism.

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.

C++ Compile time Polymorphism Example

```
// C++ program to illustrate the concept of static binding
#include <iostream>
using namespace std;

class ComputeSum
{
   public:

   int sum(int x, int y) {
      return x + y;
```

```
    }

    int sum(int x, int y, int z) {
        return x + y + z;
    }
};

int main()
{
    ComputeSum obj;
    cout << "Sum is " << obj.sum(10, 20) << endl;
    cout << "Sum is " << obj.sum(10, 20, 30) << endl;

    return 0;
}
```

C++ Runtime Polymorphism Example

Let's see a simple example of run time polymorphism in C++.

// an example without the virtual keyword.

```
#include <iostream>
using namespace std;
class Animal {
   public:
  void eat(){
   cout<<"Eating...";
   }
};
class Dog: public Animal
{
 public:
 void eat()
   {        cout<<"Eating bread...";
   }
};
```

```
int main(void) {
  Dog d = Dog();
  d.eat();
  return 0;
}
```

Output:

```
Eating bread...
```