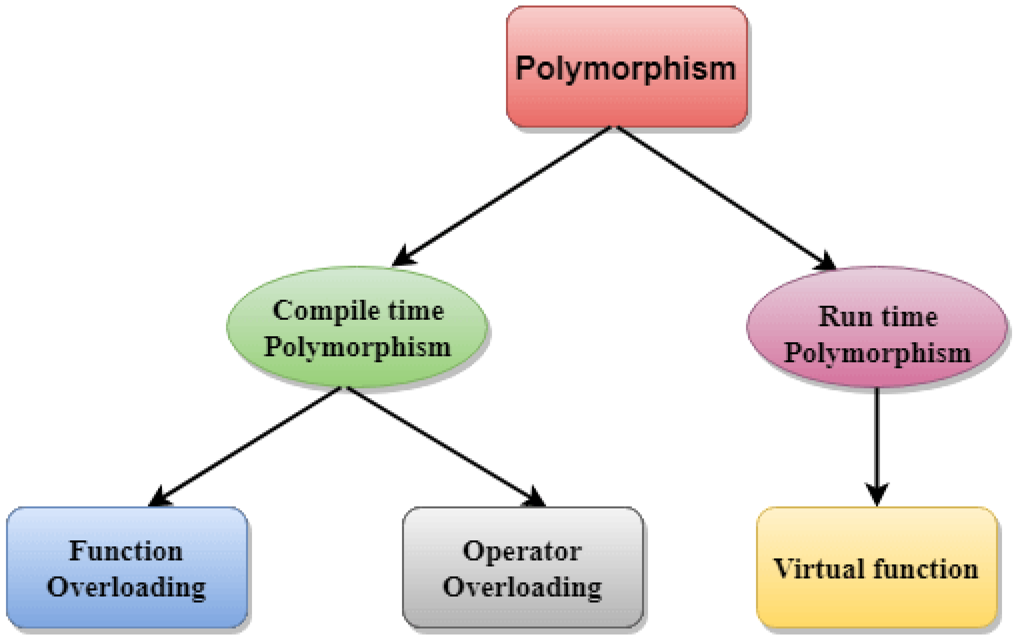
C++ Polymorphism

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.

```
 class A                           //  base class declaration.
{
        int a;
        public:
        void display()
        {
             cout<< "Class A ";
         }
    };
class B : public A                    //  derived class declaration.
{
    int b;
    public:
   void display()
   {
        cout<<"Class B";
   }
};
```

In the above case, the prototype of display() function is the same in both the **base and derived class**. Therefore, the static binding cannot be applied. It would be great if the appropriate function is selected at the run time. This is known as **run time polymorphism**.

- **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
It is also known as overloading, early binding and static binding.	It is also known as overriding, Dynan 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 polymorphis than one method is having the same n parameters and the type of the parame
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and p
It provides fast execution as it is known at the compile time.	It provides slow execution as it is kn time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as all the things exe time.

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...
```

C++ Run time Polymorphism Example: By using two derived class

Let's see another example of run time polymorphism in C++ where we are having two derived classes.

// an example with virtual keyword.

```
#include <iostream>
using namespace std;
class Shape {                               //  base class
    public:
virtual void draw(){                        // virtual function
cout<<"drawing..."<<endl;
    }
};
class Rectangle: public Shape                  //  inheriting Shape class.
{
 public:
 void draw()
   {
       cout<<"drawing rectangle..."<<endl;
    }
};
class Circle: public Shape                   //  inheriting Shape class.

{
 public:
```

```
 void draw()
  {
     cout<<"drawing circle..."<<endl;
  }
};
int main(void) {
    Shape *s;                          //  base class pointer.
    Shape sh;                          // base class object.
       Rectangle rec;
        Circle cir;
      s=&sh;
     s->draw();
        s=&rec;
     s->draw();
    s=?
    s->draw();
}
```

Output:

```
drawing...
drawing rectangle...
drawing circle...
```

Runtime Polymorphism with Data Members

Runtime Polymorphism can be achieved by data members in C++. Let's see an example where we are accessing the field by reference variable which refers to the instance of derived class.

```
#include <iostream>
using namespace std;
class Animal {                                    //  base class declaration.
    public:
    string color = "Black";
};
class Dog: public Animal                         // inheriting Animal class.
```

```
{
 public:
   string color = "Grey";
};
int main(void) {
    Animal d= Dog();
   cout<<d.color;
}
```

Output:

```
Black
```