OOPS

Polymorphism

The word **polymorphism** means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Consider the following example where a base class has been derived by other two classes.

```
#include <bits/stdc++.h>
using namespace std;

class Base {
  protected:
   int a, b;

  public:
   Base(int _a, int _b) {
      a = _a;
      b = _b;
   }
   void func() { cout << "Inside Parent Class" << endl; }
};

class Derived1 : public Base {
  public:
   Derived1(int a, int b) : Base(a, b) {}

   void func() { cout << "Inside Derived1 Class" << endl; }
};

class Derived2 : public Base {
  public:
   Derived2(int a, int b) : Base(a, b) {}

   void func() { cout << "Inside Derived2 Class" << endl; }
};

// Main function for the program
int main() {
   Base *b;
   Derived1 d1(10, 7);
   Derived2 d2(10, 5);

   // Store the address of Derived1
   b = &d1;

   // Call Derived1 Func.
   b->func();

   // Store the address of Derived2
   b = &d2;

   // Call Derived2 Func.
   b->func();
```

```
   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Inside Parent Class
Inside Parent Class
```

The reason for the incorrect output is that the call of the function ***func()*** is being set once by the compiler as the version defined in the base class.

This is called ***static resolution*** of the function call, or ***static linkage*** - the function call is fixed before the program is executed. This is also sometimes called ***early binding*** because the ***func()*** function is set during the compilation of the program.

But now, let's make a slight modification in our program and precede the declaration of ***func()*** in the Base class with the keyword **virtual** so that it looks like below

```
class Base {
  protected:
   int a, b;

  public:
   Base(int _a, int _b) {
      a = _a;
      b = _b;
   }
   virtual void func() { cout << "Inside Parent Class" << endl; }
};
```

After this slight modification, when the previous example code is compiled and executed, it produces the following result –

```
Inside Derived1 Class
Inside Derived2 Class
```

This time, the compiler looks at the contents of the pointer instead of it's type. Hence, since addresses of objects of d1 and d2 classes are stored in *b the respective func() function is called.

As you can see, each of the child classes has a separate implementation for the function func(). This is how **polymorphism** is generally used.

You have different classes with a function of the same name, and even the same parameters, but with different implementations.

Virtual Functions

A ***virtual*** function is a function in a base class that is declared using the keyword **virtual**. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.

What we do want is the selection of the function to be called at any given point in the program to be ***based on the kind of object for which it is called***. This sort of operation is referred to as ***dynamic linkage***, or ***late binding***.

Pure Virtual Functions

It is possible that you want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.

We can change the virtual function ***func()*** in the base class to the following

```
class Base {
  protected:
   int a, b;

  public:
   Base(int _a, int _b) {
      a = _a;
      b = _b;
   }
   virtual void func() = 0;
};
```

The = 0 tells the compiler that the function has no body and above virtual function will be called ***pure virtual function***.

NO NOTES TO SHOW