VIRTUAL FUNCTION AND VIRTUAL ATTRIBUTE IN CPP

Calling a Virtual Function Through a Base Class Reference

In the preceding example, a virtual function was called through a base-class pointer, but the polymorphic nature of a virtual function is also available when called through a base-class reference. As explained in Chapter 13, a reference is an implicit pointer. Thus, a base-class reference can be used to refer to an object of the base class or any object derived from that base. When a virtual function is called through a base-class reference, the version of the function executed is determined by the object being referred to at the time of the call.The most common situation in which a virtual function is invoked through a base class reference is when the reference is a function parameter.

For example, consider the following variation on the preceding program.

```
/* Here, a base class reference is used to access
a virtual function. */
#include   <iostream>
using namespace std;
class base {
public:
virtual void vfunc() {
cout << "This is base's vfunc().\n";
  }
  };
  class derived1 : public base {
  public:
  void vfunc() {
  cout << "This is derived1's vfunc().\n";
  }
  };
  class derived2 : public base {
  public:
  void vfunc() {
```

```
cout << "This is derived2's vfunc().\n";
}
};
// Use a base class reference parameter.
void f(base &r) {
r.vfunc();
}
int main()
{
base b;
derived1 d1;
derived2 d2;
f(b); // pass a base object to f()
f(d1); // pass a derived1 object to f()
f(d2); // pass a derived2 object to f()
return 0;
}
```

This program produces the same output as its preceding version. In this example, the function **f( )** defines a reference parameter of type **base**. Inside **main( )**, the function is called using objects of type **base**, **derived1**, and **derived2**. Inside **f( )**, the specific version of **vfunc( )** that is called is determined by the type of object being referenced when the function is called. For the sake of simplicity, the rest of the examples in this chapter will call virtual functions through base-class pointers, but the effects are same for base-class references..

The Virtual Attribute Is Inherited

When a virtual function is inherited, its virtual nature is also inherited. This means that when a derived class that has inherited a virtual function is itself used as a base class for another derived class, the virtual function can still be overridden. Put differently, no matter how many times a virtual function is inherited, it remains virtual.

For example, consider this program:

```
#include   <iostream>
using  namespace  std;
class base {
public:
virtual void vfunc() {
cout << "This is base's vfunc().\n";
}
};
class derived1 : public base {
public:
void vfunc() {
cout << "This is derived1's vfunc().\n";
}
};
/* derived2 inherits virtual function vfunc()
from derived1. */
class derived2 : public derived1 {
public:
// vfunc() is still virtual
void vfunc() {
 cout << "This is derived2's vfunc().\n";
 }
 };
 int main()
 {
 base *p, b;
 derived1 d1;
 derived2 d2;
 // point to base
 p = &b;
 p->vfunc(); // access base's vfunc()
```

```
// point to derived1
p = &d1;
p->vfunc(); // access derived1's vfunc()
// point to derived2
p = &d2;
p->vfunc(); // access derived2's vfunc()
return 0;
}
```

As expected, the preceding program displays this output:

```
This is base's vfunc().
This is derived1's vfunc().
This is derived2's vfunc().
```

In this case, **derived2** inherits **derived1** rather than **base**, but **vfunc( )** is still virtual.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iii-object-oriented-programming-with-c-10cs36-notes.pdf