VIRTUAL FUNCTIONS ARE HIERARCHICAL

Virtual Functions Are Hierarchical

As explained, when a function is declared as **virtual** by a base class, it may be overridden by a derived class. However, the function does not have to be overridden. When a derived class fails to override a virtual function, then when an object of that derived class accesses that function, the function defined by the base class is used.

For example, consider this program in which **derived2** does not override **vfunc( )**:

```
#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:
// vfunc() not overridden by derived2, base's is used
};
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(); // use base's vfunc()
 return 0;
 }
```

The program produces this output:

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

Because **derived2** does not override **vfunc( )**, the function defined by **base** is used when **vfunc( )** is referenced relative to objects of type **derived2**. The preceding program illustrates a special case of a more general rule. Because inheritance is hierarchical in C++, it makes sense that virtual functions are also hierarchical. This means that when a derived class fails to override a virtual function, the first redefinition found in reverse order of derivation is used.

For example, in the following program, **derived2** is derived from **derived1**, which is derived from **base**.However, **derived2** does not override **vfunc( )**. This means that, relative to **derived2**, the closest version of **vfunc( )** is in **derived1**. Therefore, it is **derived1::vfunc( )** that is used when an object of **derived2** attempts to call **vfunc( )**.

```
#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 derived1 {
  public:
  /* vfunc() not overridden by derived2.
  In this case, since derived2 is derived from
  derived1, derived1's vfunc() is used.
  */
  };
  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(); // use derived1's vfunc()
  return 0;
  }
```

The program displays the following:

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