PURE VIRTUAL FUNCTIONS

Pure Virtual Functions

As the examples in the preceding section illustrate, when a virtual function is not redefined by a derived class, the version defined in the base class will be used. However, in many situations there can be no meaningful definition of a virtual function within a base class. For example, a base class may not be able to define an object sufficiently to allow a base-class virtual function to be created. Further, in some situations you will want to ensure that all derived classes override a virtual function. To handle these two cases, C++ supports the pure virtual function.

A *pure virtual function* is a virtual function that has no definition within the base class.

To declare a pure virtual function, use this general form:

virtual *type func-name(parameter-list)* = 0;

When a virtual function is made pure, any derived class must provide its own definition. If the derived class fails to override the pure virtual function, a compile-time error will result.

The following program contains a simple example of a pure virtual function. The base class, **number**, contains an integer called **val**, the function **setval( )**, and the pure virtual function **show( )**. The derived classes **hextype**, **dectype**, and **octtype** inherit **number** and redefine **show( )** so that it outputs the value of **val** in each respective number base (that is, hexadecimal, decimal, or octal).

```
#include <iostream>
using namespace std;
class number {
protected:
int val;
public:
void setval(int i) { val = i; }
// show() is a pure virtual function
virtual void show() = 0;
};
class hextype : public number {
public:
```

```
void show() {
cout << hex << val << "\n";
}
};
class dectype : public number {
public:
void show() {
cout << val << "\n";
}
};
class octtype : public number {
public:
void show() {
cout << oct << val << "\n";
}
};
int main()
{
dectype d;
hextype h;
octtype o;
d.setval(20);
d.show(); // displays 20 - decimal
h.setval(20);
h.show(); // displays 14 – hexadecimal
o.setval(20);
o.show(); // displays 24 - octal
return 0;
}
```

Although this example is quite simple, it illustrates how a base class may not be able to meaningfully define a virtual function. In this case, **number** simply provides the common

interface for the derived types to use. There is no reason to define **show( )** inside **number** since the base of the number is undefined. Of course, you can always create a placeholder definition of a virtual function. However, making **show( )** pure also ensures that all derived classes will indeed redefine it to meet their own needs. Keep in mind that when a virtual function is declared as pure, all derived classes must override it. If a derived class fails to do this, a compile-time error will result.

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