RETURNING OBJECTS AND ARRAYS OF OBJECTS IN CPP

Returning Objects

A function may return an object to the caller. For example, this is a valid C++ program:

```
// Returning objects from a function.
#include   <iostream>
using  namespace  std;
class myclass {
int i;
public:
void set_i(int n) { i=n; }
int get_i() { return i; }
};
myclass f(); // return object of type myclass
int main()
{
myclass o;
o = f();
cout << o.get_i() << "\n";
return 0;
}
myclass f()
{
myclass  x;
x.set_i(1);
return x;
}
```

When an object is returned by a function, a temporary object is automatically created that holds the return value. It is this object that is actually returned by the function. After the value has been returned, this object is destroyed. The destruction of this temporary object may cause

unexpected side effects in some situations. For example, if the object returned by the function has a destructor that frees dynamically allocated memory, that memory will be freed even though the object that is receiving the return value is still using it. There are ways to overcome this problem that involve overloading the assignment operator (see Chapter 15) and defining a copy constructor.

Arrays of Objects

In C++, it is possible to have arrays of objects. The syntax for declaring and using an object array is exactly the same as it is for any other type of array. For example, this program uses a three-element array of objects:

```
#include <iostream>
using namespace std;
class cl {
int i;
public:
void set_i(int j) { i=j; }
int get_i() { return i; }
};
int main()
{
cl ob[3];
int i;
for(i=0; i<3; i++) ob[i].set_i(i+1);
for(i=0; i<3; i++)
cout << ob[i].get_i() << "\n";
return 0;
}
```

This program displays the numbers **1**, **2**, and **3** on the screen.

If a class defines a parameterized constructor, you may initialize each object in an array by specifying an initialization list, just like you do for other types of arrays. However, the exact form of the initialization list will be decided by the number of parameters required by the object's constructors. For objects whose constructors have only one parameter, you can simply specify a list of initial values, using the normal array-initialization syntax. As each element in the array is created, a value from the list is passed to the constructor's parameter.

For example, here is a slightly different version of the preceding program that uses an initialization:

```
#include <iostream>
using namespace std;
class cl {
int i;
public:
cl(int j) { i=j; } // constructor
int get_i() { return i; }
};
int main()
{
cl ob[3] = {1, 2, 3}; // initializers
int i;
for(i=0; i<3; i++)
cout << ob[i].get_i() << "\n";
return 0;
}
```

As before, this program displays the numbers **1**, **2**, and **3** on the screen.

Actually, the initialization syntax shown in the preceding program is shorthand for this longer form: cl ob[3] = { cl(1), cl(2), cl(3) };

Here, the constructor for **cl** is invoked explicitly. Of course, the short form used in the program is more common. The short form works because of the automatic conversion that applies to constructors taking only one argument. Thus, the short form can only be used to initialize object arrays whose constructors only require one argument. If an object's constructor requires two or more arguments, you will have to use the longer initialization form.

For example,

```
#include <iostream>
using namespace std;
class cl {
int h;
```

```
int i;
public:
cl(int j, int k) { h=j; i=k; } // constructor with 2 parameters
int get_i() {return i;}
int get_h() {return h;}
};
int main()
{
cl ob[3] = {
cl(1, 2), // initialize
cl(3, 4),
cl(5, 6)
};
int i;
for(i=0; i<3; i++) {
cout << ob[i].get_h();
cout << ", ";
cout << ob[i].get_i() << "\n";
}
return 0;
}
```

Here, **cl**'s constructor has two parameters and, therefore, requires two arguments. This means that the shorthand initialization format cannot be used and the long form, shown in the example, must be employed.

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