CONSTRUCTORS AND DESTRUCTORS IN CPP

Constructors

It is very common for some part of an object to require initialization before it can be used. For example, think back to the **stack** class developed earlier in this chapter. Before the stack could be used, **tos** had to be set to zero. This was performed by using the function **init( ).** Because the requirement for initialization is so common, C++ allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor function.

A *constructor* is a special function that is a member of a class and has the same name as that class. For example, here is how the **stack** class looks when converted to use a constructor for initialization:

```
// This creates the class stack.
class stack {
int stck[SIZE];
int tos;
public:
stack(); // constructor
void push(int i);
int pop();
};
```

Notice that the constructor **stack( )** has no return type specified. In C++, constructors cannot return values and, thus, have no return type.

The **stack( )** constructor is coded like this:

```
// stack's constructor
stack::stack()
{
```

```
tos = 0;
cout << "Stack Initialized\n";
}
```

Keep in mind that the message **Stack Initialized** is output as a way to illustrate the constructor. In actual practice, most constructors will not output or input anything. They will simply perform various initializations. An object's constructor is automatically called when the object is created. This means that it is called when the object's declaration is executed.

Destructors

Destructors

The complement of the constructor is the *destructor*. In many circumstances, an object will need to perform some action or actions when it is destroyed. Local objects are created when their block is entered, and destroyed when the block is left. Global objects are destroyed when the program terminates. When an object is destroyed, its destructor (if it has one) is automatically called. There are many reasons why a destructor may be needed. For example, an object may need to deallocate memory that it had previously allocated or it may need to close a file that it had opened. In C++, it is the destructor that handles deactivation events. The destructor has the same name as the constructor, but it is preceded by a ~. For example, here is the **stack** class and its constructor and destructor. (Keep in mind that the **stack** class does not require a destructor; the one shown here is just for illustration.)

```
// This creates the class stack.
class stack {
int stck[SIZE];
int tos;
public:
stack(); // constructor
~stack(); // destructor
void push(int i);
int pop();
};
// stack's constructor
```

```
stack::stack()
{
tos = 0;
cout << "Stack Initialized\n";
}
// stack's destructor
stack::~stack()
{
cout << "Stack Destroyed\n";
}
```

Notice that, like constructors, destructors do not have return values. To see how constructors and destructors work, here is a new version of the **stack** program examined earlier in this chapter. Observe that **init( )** is no longer needed.

```
// Using a constructor and destructor.
#include <iostream>
using namespace std;
#define SIZE 100
// This creates the class stack.
class stack {
int stck[SIZE];
int tos;
public:
stack(); // constructor
~stack(); // destructor
void push(int i);
int pop();
};
// stack's constructor
stack::stack()
{
tos = 0;
```

```
cout << "Stack Initialized\n";
}
// stack's destructor
stack::~stack()
{
cout << "Stack Destroyed\n";
}
void stack::push(int i)
{
if(tos==SIZE) {
cout << "Stack is full.\n";
return;
}
stck[tos] = i;
tos++;
}
int stack::pop()
{
if(tos==0) {
cout << "Stack underflow.\n";
return 0;
}
tos--;
return stck[tos];
}
int main()
{
stack a, b; // create two stack objects
a.push(1);
b.push(2);
a.push(3);
```

```
b.push(4);
cout << a.pop() << " ";
cout << a.pop() << " ";
cout << b.pop() << " ";
cout << b.pop() << "\n";
return 0;
}
```

This program displays the following:

```
Stack Initialized
Stack Initialized
3 1 4 2
Stack Destroyed
Stack Destroyed
```

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