Some compilers might dispose of the temporary object later, delaying the destructor call.

Finally, at the end, the program displays this:

Done
Bye, NanoSmart!
Bye, Nifty Foods!

When the main() terminates, its local variables (stock1 and stock2) pass from our plane of existence. Because such automatic variables go on the stack, the last object created is the first deleted, and the first created is the last deleted. (Recall that "NanoSmart" originally was in stock1 but later was transferred to stock2, and stock1 was reset to "Nifty Foods".)

The output points out that there is a fundamental difference between the following two statements:

```cpp
Stock stock2 = Stock("Boffo Objects", 2, 2.0);
stock1 = Stock("Nifty Foods", 10, 50.0); // temporary object
```

The first statement is initialization; it creates an object with the indicated value; it may or may not create a temporary object. The second statement is assignment. It always creates a temporary object and then copies it to an existing object.

**Tip**

If you can set object values either by initialization or by assignment, choose initialization. It usually is more efficient.

**const Member Functions**

Consider the following code snippets:

```cpp
const Stock land = Stock("Kludgehorn Properties");
land.show();
```

With current C++, the compiler should object to the second line. Why? Because the code