This statement creates a `Stock` object, initializes it to the values provided by the arguments, and assigns the address of the object to the `pstock` pointer. In this case, the object doesn't have a name, but you can use the pointer to manage the object. We put off further discussion of pointers to objects until Chapter 11.

Constructors are used differently from the other class methods. Normally, you use an object to invoke a method:

```
stock1.show(); // stock1 object invokes show() method
```

However, you can't use an object to invoke a constructor because, until the constructor finishes its work of making the object, there is no object. Rather than being invoked by an object, the constructor is used to create the object.

**The Default Constructor**

The *default constructor* is the constructor used to create an object when you don't provide explicit initialization values. That is, it's the constructor used for declarations like this:

```
Stock stock1; // uses the default constructor
```

Hey, Listing 10.3 already did that! The reason this statement works is that if you fail to provide any constructors, C++ automatically supplies a default constructor. It's a default version of a default constructor, and it does nothing. For the `Stock` class, it would look like this:

```
Stock::Stock() {}
```

The net result is that the `stock1` object is created with its members uninitialized, just as

```
int x;
```

creates `x` without providing it a value. The fact that the default constructor has no arguments reflects the fact that no values appear in the declaration.

A curious fact about the default constructor is that the compiler provides one only if you don't define any constructors. Once you define any constructor for a class, the responsibility for providing a default constructor for that class passes from the compiler to