you. If you provide a nondefault constructor, such as `Stock(const char * co, int n, double pr)` and don't provide your own version of a default constructor, then a declaration like

```
Stock stock1; // not possible with current constructor
```

becomes an error. The reason for this behavior is that you might want to make it impossible to create uninitialized objects. If, however, you wish to create objects without explicit initialization, you must define your own default constructor. This is a constructor that takes no arguments. You can define a default constructor two ways. One is to provide default values for all the arguments to the existing constructor:

```
Stock(const char * co = "Error", int n = 0, double pr = 0.0);
```

The second is to use function overloading to define a second constructor, one that has no arguments:

```
Stock();
```

You can have only one default constructor, so be sure that you don't do both. (With early versions of C++, you could use only the second method for creating a default constructor.)

Actually, you usually should initialize objects in order to ensure that all members begin with known, reasonable values. Thus, the default constructor typically provides implicit initialization for all member values. Here, for example, is how you might define one for the `Stock` class:

```
Stock::Stock()
{
    strcpy(company, "no name");
    shares = 0;
    share_val = 0.0;
    total_val = 0.0;
}
```

**Tip**

When you design a class, you usually should provide a default constructor that implicitly initializes all class