bool pop(Item & item); // pop top into item
}
#endif

Compatibility Note

If your system hasn't implemented the bool type, you can use int, 0, and 1 rather than bool, false, and true. Alternatively, your system might support an earlier, non-standard form, such as boolean or Boolean.

In this example, the private section shows that the stack is implemented by using an array, but the public section doesn't reveal that fact. Thus, you can replace the array with, say, a dynamic array without changing the class interface. That means changing the stack implementation doesn't require that you recode programs using the stack. You just recompile the stack code and link it with existing program code.

The interface is redundant in that pop() and push() return information about the stack status (full or empty) instead of being type void. This provides the programmer with a couple of options as to how to handle exceeding the stack limit or emptying the stack. He or she can use isempty() and isfull() to check before attempting to modify the stack, or else use the return value of push() and pop() to determine if the operation is successful.

Rather than define the stack in terms of some particular type, the class describes it in terms of a general Item type. In this case, the header file uses typedef to make Item the same as unsigned long. If you want, say, a stack of double or of a structure type, you can change the typedef and leave the class declaration and method definitions unaltered. Class templates (see Chapter 14, "Reusing Code in C++") provide a more powerful method for isolating the type of data stored from the class design.

Next, let's implement the class methods. Listing 10.11 shows one possibility.

Listing 10.11 stack.cpp

// stack.cpp -- Stack member functions
#include "stack.h"