• You can check to see if the stack is full.

• You can check to see if the stack is empty.

You can match this description with a class declaration in which the public member functions provide an interface that represents the stack operations. The private data members take care of storing the stack data. The class concept is a nice match to the ADT approach.

The private section has to commit itself to how to hold the data. For example, you can use an ordinary array, a dynamically allocated array, or some more advanced data structure, such as a linked list. The public interface, however, should hide the exact representation. Instead, it should be expressed in general terms, such as creating a stack, pushing an item, and so on. **Listing 10.10** shows one approach. It assumes that the `bool` type has been implemented. If it hasn't been on your system, you can use `int`, 0, and 1 rather than `bool`, `false`, and `true`.

**Listing 10.10 stack.h**

```cpp
// stack.h -- class definition for the stack ADT
#ifndef STACK_H_
#define STACK_H_
typedef unsigned long Item;

class Stack
{
private:
    enum {MAX = 10};    // constant specific to class
    Item items[MAX];    // holds stack items
    int top;            // index for top stack item
public:
    Stack();
    bool isempty() const;
    bool isfull() const;
    // push() returns false if stack already is full, true otherwise
    bool push(const Item & item);   // add item to stack
    // pop() returns false if stack already is empty, true otherwise
```