Generic ADTs

Midway through Section 19.4, we improved the stack ADT by making it easier to change the type of items stored in a stack—all we had to do was modify the definition of the `Item` type. It's still somewhat of a nuisance to do so; it would be nicer if a stack could accommodate items of any type, without the need to modify the `stack.h` file. Also note that our stack ADT suffers from a serious flaw: a program can't create two stacks whose items have different types. It's easy to create multiple stacks, but those stacks must have items with identical types. To allow stacks with different item types, we'd have to make copies of the stack ADT's header file and source file and modify one set of files so that the `Stack` type and its associated functions have different names.

What we'd like to have is a single "generic" stack type from which we could create a stack of integers, a stack of strings, or any other stack that we might need. There are various ways to create such a type in C, but none are completely satisfactory. The most common approach uses `void *` as the item type, which allows arbitrary pointers to be pushed and popped. With this technique, the `stack-ADT.h` file would be similar to our original version; however, the prototypes of the `push` and `pop` functions would have the following appearance:

```
void push(Stack s, void *p);
void *pop(Stack s);
```

`pop` returns a pointer to the item popped from the stack; if the stack is empty, it returns a null pointer.

There are two disadvantages to using `void *` as the item type. One is that this approach doesn't work for data that can't be represented in pointer form. Items could be strings (which are represented by a pointer to the first character in the string) or dynamically allocated structures but not basic types such as `int` and `double`. The other disadvantage is that error checking is no longer possible. A stack that stores `void *` items will happily allow a mixture of pointers of different types: there's no way to detect an error caused by pushing a pointer of the wrong type.

ADTs in Newer Languages

The problems that we've just discussed are dealt with much more cleanly in newer C-based languages, such as C++, Java, and C#. Name clashes are prevented by defining function names within a *class*. A stack ADT would be represented by a `Stack` class; the stack functions would belong to this class, and would only be recognized by the compiler when applied to a `Stack` object. These languages have a feature known as ***exception handling*** that allows functions such as `push` and `pop` to "throw" an exception when they detect an error condition. Code in the client can then deal with the error by "catching" the exception. C++, Java, and C# also provide special features for defining generic ADTs. In C++, for example, we would define a stack ***template***, leaving the item type unspecified.