```
void push(Stack s, Item i);
Item pop(Stack s);

#endif
```

The changes to the file are shown in **bold.** Besides the addition of the `Item` type, the `push` and `pop` functions have been modified. `push` now has a parameter of type `Item`, and `pop` returns a value of type `Item`. We'll use this version of `stackADT.h` from now on; it replaces the earlier version.

The `stackADT.c` file will need to be modified to match the new `stackADT.h`. The changes are minimal, however. The `stack_type` structure will now contain an array whose elements have type `Item` instead of `int`:

```
struct stack_type {
  Item contents[STACK_SIZE];
  int top;
};
```

The only other changes are to `push` (the second parameter now has type `Item`) and `pop` (which returns a value of type `Item`). The bodies of `push` and `pop` are unchanged.

The `stackclient.c` file can be used to test the new `stackADT.h` and `stackADT.c` to verify that the `Stack` type still works (it does!). Now we can change the item type any time we want by simply modifying the definition of the `Item` type in `stackADT.h`. (Although we won't have to change the `stackADT.c` file, we'll still need to recompile it.)

Implementing the Stack ADT Using a Dynamic Array

Another problem with the stack ADT as it currently stands is that each stack has a fixed maximum size, which is currently set at 100 items. This limit can be increased to any number we wish, of course, but all stacks created using the `Stack` type will have the same limit. There's no way to have stacks with different capacities or to set the stack size as the program is running.

There are two solutions to this problem. One is to implement the stack as a linked list, in which case there's no fixed limit on its size. We'll investigate this solution in a moment. First, though, let's try the other approach, which involves storing stack items in a dynamically allocated array.

dynamically allocated arrays ➤ 17.3

The crux of the latter approach is to modify the `stack_type` structure so that the `contents` member is a *pointer* to the array in which the items are stored, not the array itself:

```
struct stack_type {
  Item *contents;
  int top;
  int size;
};
```