```
int pop(void)
{
  if (is_empty())
    terminate("Error in pop: stack is empty.");
  return contents[--top];
}
```

The variables that make up the stack (`contents` and `top`) are both declared `static`, since there's no reason for the rest of the program to access them directly. The `terminate` function is also declared `static`. This function isn't part of the module's interface; instead, it's designed for use solely within the implementation of the module.

As a matter of style, some programmers use macros to indicate which functions and variables are "public" (accessible elsewhere in the program) and which are "private" (limited to a single file):

```
#define PUBLIC  /* empty */
#define PRIVATE static
```

The reason for writing `PRIVATE` instead of `static` is that the latter has more than one use in C; `PRIVATE` makes it clear that we're using it to enforce information hiding. Here's what the stack implementation would look like if we were to use `PUBLIC` and `PRIVATE`:

```
PRIVATE int contents[STACK_SIZE];
PRIVATE int top = 0;

PRIVATE void terminate(const char *message) { ... }

PUBLIC void make_empty(void) { ... }

PUBLIC bool is_empty(void) { ... }

PUBLIC bool is_full(void) { ... }

PUBLIC void push(int i) { ... }

PUBLIC int pop(void) { ... }
```

Now we'll switch to a linked-list implementation of the stack module:

stack2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

struct node {
  int data;
  struct node *next;
};

static struct node *top = NULL;
```