Now try this error!

Don't dispense with the ERROR.C file just yet. Don't close the window, and don't zap the project. (If you did, use your editor to load the ERROR.C file and prepare to reedit.)

Change Line 6 in the ERROR.C source code file to read this way:

```
retrun(0);
```

In case you don't see it, the word `return` has been mangled to read `retrun`; the second *r* and the *u* are transposed. Otherwise, the zero in the parentheses and the semicolon are unchanged.

The way C works is that it just assumes that `retrun` is something you're serious about and not a typo. The compiler couldn't care less. But the *linker* goes nuts over it. That's because it's the linker that glues program files together. It catches the error when it doesn't find the word `retrun` in any of its libraries. And, like any frazzled librarian, the linker spews forth an error message.

Save the modified ERROR.C file to disk. Then recompile. Brace yourself for an error message along the lines of

```
temporary_filename.o: In function 'main':
temporary_filename.o: undefined reference to 'retrun'
```

Or, the message may look like this:

```
temporary_filename.o(blah-blah):error.c: undefined reference
           to 'retrun'
```

It's harder to tell where the error took place here; unlike compiler errors, linker errors tend to be vague. In this case, the linker is explaining that the error is in reference to the word `retrun`. So, rather than use a line-number reference, you can always just search for the bogus text.

To fix the error, reedit the source code and change `retrun` back to `return`. Save. Recompile. The linker should be pleased.

- As I mention elsewhere in this book, the GCC compiler both compiles and links.
- If the linker is run as a separate program, it obviously produces its own error messages.
- A temporary file is created by the compiler, an *object code* file that ends in .O — which you can see in the error message output. This object code file is deleted by GCC.