No need to fill your head with this

C programming has two degrees of errors: warnings and errors. Some compilers call the errors *critical* errors. (Sounds like a mistake Roger Ebert would make.) Other times, they're *fatal* errors, like opening a creepy closet in one of those *Scream* movies.

The warning error means "Ooo, this doesn't look tasty, but I'll serve it to you anyway." Chances are, your program runs, but it may not do what you intend. Or, it may just be that the compiler is being touchy. With most C compilers, you can switch off some of the more persnickety warning error messages.

The critical error means "Dear Lordy, you tried to do something so criminal that I cannot morally complete this program." Okay, maybe it's not that severe. But the compiler cannot complete its task because it just doesn't understand your instructions.

You can fix the ERROR.C program by adding a semicolon. Edit Line 5 and add a semicolon to the end of the line. Also correct the sentence displayed on the screen so that it reads as follows:

```
printf("This program will no longer err.\n");
```

Other than changing Line 5, everything else in the program remains untouched.

Save ERROR.C back to disk. Recompile the program and then run it:

```
This program will no longer err.
```

Indeed, it does not!

- I can't always tell you where to fix your programs. ERROR.C is the only program listed in this book that contains an on-purpose error. When you get an error message, you should check it to see where the error is in your source code. Then cross-check your source code with what's listed in this book. That way, you find what's wrong. But when you venture out on your own and experiment, you have only the error message to go by when you're hunting down your own errors.
- Pull two *R*s out of ERRORS and you have Eros, the Greek god of love. The Roman god of love was Cupid. Replace the *C* in Cupid with *St* and you have Stupid. Stupid errors — how lovely!