

- The best time for using the TRUE or FALSE shortcut words is when you create a while loop. Refer to Chapters 17 and 18.
- \checkmark In the C language, the value 1 is taken to be TRUE, and 0 is FALSE.
- Some C programmers prefer to define TRUE and FALSE this way:

```
#define FALSE 0
#define TRUE(!FALSE)
```

Whatever. This book defines TRUE as 1 and FALSE as "not true," which works out to 0. If you see the preceding #defines in a program, everything still works as advertised. Just don't let it throw you.

- ✓ Some compilers may issue a warning message when no comparison is in the if statement's parentheses. That's just the compiler being smart again; it recognizes the certainty of the if keyword's "comparison" and tells you whether it is always TRUE or is always FALSE.
- Some compilers may also point out that the second if statements are never executed because the if condition is always false. Big deal. Run the program anyway.

Avoiding the Topic of Macros



Another # thing used in the C language is the *macro*. Like #include and #define, macros are instructions to the compiler that tell it how to handle conditions and whether specific parts of the source code are to be used or skipped. Weird, but handy.

Though there's no reason to describe in detail what macros are and because I'm not crazy enough to waste time with an example, you may see even more pound-sign goobers playing the macro game in someone else's C source code. Pick any from this list:

#if
#else
#endif
#ifdef
#ifndef

These are also instructions to the C compiler, designed primarily to indicate whether a group of statements should be compiled or included with the final program.