the primary compiler for many UNIX-based operating systems, including Linux, BSD, and Mac OS X, and it's used extensively for commercial software development. For more information about GCC, visit *gcc.gnu.org*.

Q: How good is GCC at finding errors in programs?

A: GCC has various command-line options that control how thoroughly it checks programs. When these options are used, GCC is quite good at finding potential trouble spots in a program. Here are some of the more popular options:

`-Wall`	Causes the compiler to produce warning messages when it detects possible errors. (`-W` can be followed by codes for specific warnings; `-Wall` means "all `-W` options.") Should be used in conjunction with `-O` for maximum effect.
`-W`	Issues additional warning messages beyond those produced by `-Wall`.
`-pedantic`	Issues all warnings required by the C standard. Causes programs that use nonstandard features to be rejected.
`-ansi`	Turns off features of GCC that aren't standard C and enables a few standard features that are normally disabled.
`-std=c89`	
`-std=c99`	Specifies which version of C the compiler should use to check the program.

These options are often used in combination:

```
% gcc -O -Wall -W -pedantic -ansi -std=c99 -o pun pun.c
```

Q: Why is C so terse? It seems as though programs would be more readable if C used `begin` and `end` instead of `{` and `}`, `integer` instead of `int`, and so forth. [p. 12]

A: Legend has it that the brevity of C programs is due to the environment that existed in Bell Labs at the time the language was developed. The first C compiler ran on a DEC PDP-11 (an early minicomputer); programmers used a teletype—essentially a typewriter connected to a computer—to enter programs and print listings. Because teletypes were very slow (they could print only 10 characters per second), minimizing the number of characters in a program was clearly advantageous.

Q: In some C books, the `main` function ends with `exit(0)` instead of `return 0`. Are these the same? [p. 14]

A: When they appear inside `main`, these statements are indeed equivalent: both terminate the program, returning the value 0 to the operating system. Which one to use is mostly a matter of taste.

Q: What happens if a program reaches the end of the `main` function without executing a `return` statement? [p. 14]

A: The `return` statement isn't mandatory; if it's missing, the program will still ter-