file uses a feature that didn't exist prior to the original C89 standard. To prevent the header file from being used with older, nonstandard compilers, it could contain an `#ifndef` directive that tests for the existence of the `__STDC__` macro:

__STDC__ macro ➤ 14.3

```
#ifndef __STDC__
#error This header requires a Standard C compiler
#endif
```

15.3 Dividing a Program into Files

Let's now use what we know about header files and source files to develop a simple technique for dividing a program into files. We'll concentrate on functions, but the same principles apply to external variables as well. We'll assume that the program has already been designed; that is, we've decided what functions the program will need and how to arrange the functions into logically related groups. (We'll discuss program design in Chapter 19.)

Here's how we'll proceed. Each set of functions will go into a separate source file (let's use the name `foo.c` for one such file). In addition, we'll create a header file with the same name as the source file, but with the extension `.h` (`foo.h`, in our case). Into `foo.h`, we'll put prototypes for the functions defined in `foo.c`. (Functions that are designed for use only within `foo.c` need not—and should not—be declared in `foo.h`. The `read_char` function in our next program is an example.) We'll include `foo.h` in each source file that needs to call a function defined in `foo.c`. Moreover, we'll include `foo.h` in `foo.c` so that the compiler can check that the function prototypes in `foo.h` are consistent with the definitions in `foo.c`.

The `main` function will go in a file whose name matches the name of the program—if we want the program to be known as `bar`, then `main` should be in the file `bar.c`. It's possible that there are other functions in the same file as `main`, so long as they're not called from other files in the program.

PROGRAM Text Formatting

To illustrate the technique that we've just discussed, let's apply it to a small text-formatting program named `justify`. As sample input to `justify`, we'll use a file named `quote` that contains the following (poorly formatted) quotation from "The development of the C programming language" by Dennis M. Ritchie (in *History of Programming Languages II*, edited by T. J. Bergin, Jr., and R. G. Gibson, Jr., Addison-Wesley, Reading, Mass., 1996, pages 671–687):

```
   C      is quirky,   flawed,     and  an
enormous    success.        Although accidents of    history
 surely  helped,    it evidently    satisfied    a    need

    for  a   system  implementation    language    efficient
```