Q & A

Q: You don't have any examples that use the `#include` directive to include a source file. What would happen if we were to do this?

A: That's not a good practice, although it's not illegal. Here's an example of the kind of trouble you can get into. Suppose that `foo.c` defines a function `f` that we'll need in `bar.c` and `baz.c`, so we put the directive

```
#include "foo.c"
```

in both `bar.c` and `baz.c`. Each of these files will compile nicely. The problem comes later, when the linker discovers two copies of the object code for `f`. Of course, we would have gotten away with including `foo.c` if only `bar.c` had included it, not `baz.c` as well. To avoid problems, it's best to use `#include` only with header files, not source files.

Q: What are the exact search rules for the `#include` directive? [p. 351]

A: That depends on your compiler. The C standard is deliberately vague in its description of `#include`. If the file name is enclosed in *brackets*, the preprocessor looks in a "sequence of implementation-defined places," as the standard obliquely puts it. If the file name is enclosed in *quotation marks*, the file "is searched for in an implementation-defined manner" and, if not found, then searched as if its name had been enclosed in brackets. The reason for this waffling is simple: not all operating systems have hierarchical (tree-like) file systems.

To make matters even more interesting, the standard doesn't require that names enclosed in brackets be file names at all, leaving open the possibility that `#include` directives using `<>` are handled entirely within the compiler.

Q: I don't understand why each source file needs its own header file. Why not have one big header file containing macro definitions, type definitions, and function prototypes? By including this file, each source file would have access to all the shared information it needs. [p. 354]

A: The "one big header file" approach certainly works; a number of programmers use it. And it does have an advantage: with only one header file, there are fewer files to manage. For large programs, however, the disadvantages of this approach tend to outweigh its advantages.

Using a single header file provides no useful information to someone reading the program later. With multiple header files, the reader can quickly see what other parts of the program are used by a particular source file.

But that's not all. Since each source file depends on the big header file, changing it will cause all source files to be recompiled—a significant drawback in a large program. To make matters worse, the header file will probably change frequently because of the large amount of information it contains.