14 The Preprocessor

There will always be things we wish to say in our programs that in all known languages can only be said poorly.

In previous chapters, I've used the `#define` and `#include` directives without going into detail about what they do. These directives—and others that we haven't yet covered—are handled by the ***preprocessor***, a piece of software that edits C programs just prior to compilation. Its reliance on a preprocessor makes C (along with C++) unique among major programming languages.

The preprocessor is a powerful tool, but it also can be a source of hard-to-find bugs. Moreover, the preprocessor can easily be misused to create programs that are almost impossible to understand. Although some C programmers depend heavily on the preprocessor, I recommend that it—like so many other things in life—be used in moderation.

This chapter begins by describing how the preprocessor works (Section 14.1) and giving some general rules that affect all preprocessing directives (Section 14.2). Sections 14.3 and 14.4 cover two of the preprocessor's major capabilities: macro definition and conditional compilation. (I'll defer detailed coverage of file inclusion, the other major capability, until Chapter 15.) Section 14.5 discusses the preprocessor's lesser-used directives: `#error`, `#line`, and `#pragma`.

14.1 How the Preprocessor Works

The behavior of the preprocessor is controlled by ***preprocessing directives:*** commands that begin with a # character. We've encountered two of these directives, `#define` and `#include`, in previous chapters.

The `#define` directive defines a ***macro***—a name that represents something else, such as a constant or frequently used expression. The preprocessor responds to a `#define` directive by storing the name of the macro together with its definition.