Sharing Function Prototypes

Suppose that a source file contains a call of a function `f` that's defined in another file, `foo.c`. Calling `f` without declaring it first is risky. Without a prototype to rely on, the compiler is forced to assume that `f`'s return type is `int` and that the number of parameters matches the number of arguments in the call of `f`. The arguments themselves are converted automatically to a kind of "standard form" by the default argument promotions. The compiler's assumptions may well be wrong, but it has no way to check them, since it compiles only one file at a time. If the assumptions are incorrect, the program probably won't work, and there won't be any clues as to why it doesn't. (For this reason, C99 prohibits calling a function for which the compiler has not yet seen a declaration or definition.)

default argument promotions ➤93

⚠ When calling a function `f` that's defined in another file, always make sure that the compiler has seen a prototype for `f` prior to the call.

Our first impulse is to declare `f` in the file where it's called. That solves the problem but can create a maintenance nightmare. Suppose that the function is called in fifty different source files. How can we ensure that `f`'s prototypes are the same in all the files? How can we guarantee that they match the definition of `f` in `foo.c`? If `f` should change later, how can we find all the files where it's used?

The solution is obvious: put `f`'s prototype in a header file, then include the header file in all the places where `f` is called. Since `f` is defined in `foo.c`, let's name the header file `foo.h`. In addition to including `foo.h` in the source files where `f` is called, we'll need to include it in `foo.c`, enabling the compiler to check that `f`'s prototype in `foo.h` matches its definition in `foo.c`.

Q&A

⚠ Always include the header file declaring a function `f` in the source file that contains `f`'s definition. Failure to do so can cause hard-to-find bugs, since calls of `f` elsewhere in the program may not match `f`'s definition.

If `foo.c` contains other functions, most of them should be declared in the same header file as `f`. After all, the other functions in `foo.c` are presumably related to `f`; any file that contains a call of `f` probably needs some of the other functions in `foo.c`. Functions that are intended for use only within `foo.c` shouldn't be declared in a header file, however; to do so would be misleading.

To illustrate the use of function prototypes in header files, let's return to the RPN calculator of Section 15.1. The `stack.c` file will contain definitions of the `make_empty`, `is_empty`, `is_full`, `push`, and `pop` functions. The following prototypes for these functions should go in the `stack.h` header file:

```
void make_empty(void);
int is_empty(void);
```