Functions Hidden by Macros

Q&A

It's common for C programmers to replace small functions by parameterized macros. This practice occurs even in the standard library. The C standard allows headers to define macros that have the same names as library functions, but protects the programmer by requiring that a true function be available as well. As a result, it's not unusual for a library header to declare a function *and* define a macro with the same name.

We've already seen an example of a macro duplicating a library function. `getchar` is a library function declared in the `<stdio.h>` header. It has the following prototype:

```
int getchar(void);
```

`<stdio.h>` usually defines `getchar` as a macro as well:

```
#define getchar() getc(stdin)
```

By default, a call of `getchar` will be treated as a macro invocation (since macro names are replaced during preprocessing).

Most of the time, we're happy using a macro instead of a true function, because it will probably make our program run faster. Occasionally, though, we want a genuine function, perhaps to minimize the size of the executable code.

If the need arises, we can remove a macro definition (thus gaining access to the true function) by using the `#undef` directive. For example, we could undefine the `getchar` macro after including `<stdio.h>`:

`#undef` directive ➤ *14.3*

```
#include <stdio.h>
#undef getchar
```

If `getchar` *isn't* a macro, no harm has been done; `#undef` has no effect when given a name that's not defined as a macro.

As an alternative, we can disable individual uses of a macro by putting parentheses around its name:

```
ch = (getchar)();    /* instead of ch = getchar(); */
```

The preprocessor can't spot a parameterized macro unless its name is followed by a left parenthesis. The compiler isn't so easily fooled, however; it can still recognize `getchar` as a function.

21.2 C89 Library Overview

We'll now take a quick look at the headers in the C89 standard library. This section can serve as a "road map" to help you determine which part of the library you need. Each header is described in detail later in this chapter or in a subsequent chapter.