Macros for Format Specifiers

The types declared in the `<stdint.h>` header can be used to make programs more portable, but they create new headaches for the programmer. Consider the problem of displaying the value of the variable `i`, where `i` has type `int_least32_t`. The statement

```
printf("i = %d\n", i);
```

may not work, because `i` doesn't necessarily have `int` type. If `int_least32_t` is another name for the `long int` type, then the correct conversion specification is `%ld`, not `%d`. In order to use the ...`printf` and ...`scanf` functions in a portable manner, we need a way to write conversion specifications that correspond to each of the types declared in `<stdint.h>`. That's where the `<inttypes.h>` header comes in. For each `<stdint.h>` type, `<inttypes.h>` provides a macro that expands into a string literal containing the proper conversion specifier for that type.

Each macro name has three parts:

- The name begins with either `PRI` or `SCN`, depending on whether the macro will be used in a call of a ...`printf` function or a ...`scanf` function.
- Next comes a one-letter conversion specifier (`d` or `i` for a signed type; `o`, `u`, `x`, or `X` for an unsigned type).
- The last part of the name indicates which `<stdint.h>` type is involved. For example, the name of a macro that corresponds to the `int_least`*N*`_t` type would end with `LEAST`*N*.

Let's return to our previous example, which involved displaying an integer of type `int_least32_t`. Instead of using `d` as the conversion specifier, we'll switch to the `PRIdLEAST32` macro. To use the macro, we'll split the `printf` format string into three pieces and replace the `d` in `%d` by `PRIdLEAST32`:

```
printf("i = %" PRIdLEAST32 "\n", i);
```

The value of `PRIdLEAST32` is probably either `"d"` (if `int_least32_t` is the same as the `int` type) or `"ld"` (if `int_least32_t` is the same as `long int`). Let's assume that it's `"ld"` for the sake of discussion. After macro replacement, the statement becomes

```
printf("i = %" "ld" "\n", i);
```

Once the compiler joins the three string literals into one (which it will do automatically), the statement will have the following appearance:

```
printf("i = %ld\n", i);
```

Note that we can still include flags, a field width, and other options in our conversion specification; `PRIdLEAST32` supplies only the conversion specifier and possibly a length modifier, such as the letter `l`.

Table 27.3 lists the `<inttypes.h>` macros.