course, so we can't count on them to be available with other compilers. These headers often provide functions that are specific on a particular computer or operating system (which explains why they're not standard). They may provide functions that allow more control over the screen and keyboard. Headers that support graphics or a window-based user interface are also common.

The standard headers consist primarily of function prototypes, type definitions, and macro definitions. If one of our files contains a call of a function declared in a header or uses one of the types or macros defined there, we'll need to include the header at the beginning of the file. When a file includes several standard headers, the order of `#include` directives doesn't matter. It's also legal to include a standard header more than once.

Restrictions on Names Used in the Library

Any file that includes a standard header must obey a couple of rules. First, it can't use the names of macros defined in that header for any other purpose. If a file includes `<stdio.h>`, for example, it can't reuse `NULL`, since a macro by that name is already defined in `<stdio.h>`. Second, library names with file scope (`typedef` names, in particular) can't be redefined at the file level. Thus, if a file includes `<stdio.h>`, it can't define `size_t` as a identifier with file scope, since `<stdio.h>` defines `size_t` to be a `typedef` name.

Although these restrictions are pretty obvious, C has other restrictions that you might not expect:

- ***Identifiers that begin with an underscore followed by an upper-case letter or a second underscore*** are reserved for use within the library; programs should never use names of this form for any purpose.
- ***Identifiers that begin with an underscore*** are reserved for use as identifiers and tags with file scope. You should never use such a name for your own purposes unless it's declared inside a function.
- ***Every identifier with external linkage in the standard library*** is reserved for use as an identifier with external linkage. In particular, the names of all standard library functions are reserved. Thus, even if a file *doesn't* include `<stdio.h>`, it shouldn't define an external function named `printf`, since there's already a function with this name in the library.

These rules apply to *every* file in a program, regardless of which headers the file includes. Although these rules aren't always enforced, failing to obey them can lead to a program that's not portable.

The rules listed above apply not just to names that are currently used in the library, but also to names that are set aside for future use. The complete description of which names are reserved is rather lengthy; you'll find it in the C standard under "future library directions." As an example, C reserves identifiers that begin with `str` followed by a lower-case letter, so that functions with such names can be added to the `<string.h>` header.