Q: My version of `<errno.h>` defines other macros besides `EDOM` and `ERANGE`. Is this practice legal? [p. 630]

A: Yes. The C standard allows macros that represent other error conditions, provided that their names begin with the letter `E` followed by a digit or an upper-case letter. UNIX implementations typically define a huge number of such macros.

Q: Some of the macros that represent signals have cryptic names, like `SIGFPE` and `SIGSEGV`. Where do these names come from? [p. 631]

A: The names of these signals date back to the early C compilers, which ran on a DEC PDP-11. The PDP-11 hardware could detect errors with names like "Floating Point Exception" and "Segmentation Violation."

Q: OK, I'm curious. Unless it's invoked by `abort` or `raise`, a signal handler shouldn't call a standard library function, but you said there were exceptions to this rule. What are they? [p. 632]

A: A signal handler is allowed to call the `signal` function, provided that the first argument is the signal that it's handling at the moment. This proviso is important, because it allows a signal handler to reinstall itself. In C99, a signal handler may also call the `abort` function or the `_Exit` function.

C99

`_Exit` function ➤*26.2*

***Q: Following up on the previous question, a signal handler normally isn't supposed to access variables with static storage duration. What's the exception to this rule?**

A: That one's a bit harder. The answer involves a type named `sig_atomic_t` that's declared in the `<signal.h>` header. `sig_atomic_t` is an integer type that can be accessed "as an atomic entity," according to the C standard. In other words, the CPU can fetch a `sig_atomic_t` value from memory or store one in memory with a single machine instruction, rather than using two or more machine instructions. `sig_atomic_t` is often defined to be `int`, since most CPUs can load or store an `int` value in one instruction.

That brings us to the exception to the rule that a signal-handling function isn't supposed to access static variables. The C standard allows a signal handler to store a value in a `sig_atomic_t` variable—even one with static storage duration—provided that it's declared `volatile`. To see the reason for this arcane rule, consider what might happen if a signal handler were to modify a static variable that's of a type that's wider than `sig_atomic_t`. If the program had fetched part of the variable from memory just before the signal occurred, then completed the fetch after the signal is handled, it could end up with a garbage value. `sig_atomic_t` variables can be fetched in a single step, so this problem doesn't occur. Declaring the variable to be `volatile` warns the compiler that the variable's value may change at any time. (A signal could suddenly be raised, invoking a signal handler that modifies the variable.)

`volatile` type qualifier ➤*20.3*

Q: The `tsignal.c` program calls `printf` from inside a signal handler. Isn't that illegal?