assert has one disadvantage: it slightly increases the running time of a program because of the extra check it performs. Using assert once in a while probably won't have any great effect on a program's speed, but even this small time penalty may be unacceptable in critical applications. As a result, many programmers use assert during testing, then disable it when the program is finished. Disabling assert is easy: we need only define the macro NDEBUG prior to including the <assert.h> header:

```
#define NDEBUG
#include <assert.h>
```

The value of NDEBUG doesn't matter, just the fact that it's defined. If the program should fail later, we can reactivate assert by removing NDEBUG's definition.

⚠ Avoid putting an expression that has a side effect—including a function call—inside an assert; if assert is disabled at a later date, the expression won't be evaluated. Consider the following example:

```
assert((p = malloc(n)) != NULL);
```

If NDEBUG is defined, assert will be ignored and malloc won't be called.

24.2 The <errno.h> Header: Errors

Some functions in the standard library indicate failure by storing an error code (a positive integer) in errno, an int variable declared in <errno.h>. (errno may actually be a macro. If so, the C standard requires that it represent an lvalue, allowing us to use it like a variable.) Most of the functions that rely on errno belong to <math.h>, but there are a few in other parts of the library.

lvalues ➤4.2

Let's say that we need to use a library function that signals an error by storing a value in errno. After calling the function, we can check whether the value of errno is nonzero; if so, an error occurred during the function call. For example, suppose that we want to check whether a call of the sqrt (square root) function has failed. Here's what the code would look like:

sqrt function ➤23.3

```
errno = 0;
y = sqrt(x);
if (errno != 0) {
  fprintf(stderr, "sqrt error; program terminated.\n");
  exit(EXIT_FAILURE);
}
```

Q&A When errno is used to detect an error in a call of a library function, it's important to store zero in errno before calling the function. Although errno is zero at the beginning of program execution, it could have been altered by a later function call. Library functions never clear errno; that's the program's responsibility.