24 Error Handling

There are two ways to write error-free programs; only the third one works.

Although student programs often fail when subjected to unexpected input, commercial programs need to be "bulletproof"—able to recover gracefully from errors instead of crashing. Making programs bulletproof requires that we anticipate errors that might arise during the execution of the program, include a check for each one, and provide a suitable action for the program to perform if the error should occur.

This chapter describes two ways for programs to check for errors: by using the `assert` macro and by testing the `errno` variable. Section 24.1 covers the `<assert.h>` header, where `assert` is defined. Section 24.2 discusses the `<errno.h>` header, to which the `errno` variable belongs. This section also includes coverage of the `perror` and `strerror` functions. These functions, which come from `<stdio.h>` and `<string.h>`, respectively, are closely related to the `errno` variable.

Section 24.3 explains how programs can detect and handle conditions known as signals, some of which represent errors. The functions that deal with signals are declared in the `<signal.h>` header.

Finally, Section 24.4 explores the `setjmp`/`longjmp` mechanism, which is often used for responding to errors. Both `setjmp` and `longjmp` belong to the `<setjmp.h>` header.

Error detection and handling aren't among C's strengths. C indicates run-time errors in a variety of ways rather than in a single, uniform way. Furthermore, it's the programmer's responsibility to include code to test for errors. It's easy to overlook potential errors; if one of these should actually occur, the program often continues running, albeit not very well. Newer languages such as C++, Java, and C# have an "exception handling" feature that makes it easier to detect and respond to errors.