A: That's usually true, but not if the program calls `abort` to terminate. Even when `abort` isn't used, though, there are still good reasons to call `fclose`. First, it reduces the number of open files. Operating systems limit the number of files that a program may have open at the same time; large programs may bump into this limit. (The macro `FOPEN_MAX`, defined in `<stdio.h>`, specifies the minimum number of files that the implementation guarantees can be open simultaneously.) Second, the program becomes easier to understand and modify; by looking for the call of `fclose`, it's easier for the reader to determine the point at which a file is no longer in use. Third, there's the issue of safety. Closing a file ensures that its contents and directory entry are updated properly; if the program should crash later, at least the file will be intact.

abort function ➤26.2

Q: I'm writing a program that will prompt the user to enter a file name. How long should I make the character array that will store the file name? [p. 546]

A: That depends on your operating system. Fortunately, you can use the macro `FILENAME_MAX` (defined in `<stdio.h>`) to specify the size of the array. `FILENAME_MAX` is the length of a string that will hold the longest file name that the implementation guarantees can be opened.

Q: Can `fflush` flush a stream that was opened for both reading and writing? [p. 549]

A: According to the C standard, the effect of calling `fflush` is defined for a stream that (a) was opened for output, or (b) was opened for updating and whose last operation was not a read. In all other cases, the effect of calling `fflush` is undefined. When `fflush` is passed a null pointer, it flushes all streams that satisfy either (a) or (b).

Q: Can the format string in a call of `...printf` or `...scanf` be a variable?

A: Sure; it can be any expression of type `char *`. This property makes the `...printf` and `...scanf` functions even more versatile than we've had reason to suspect. Consider the following classic example from Kernighan and Ritchie's *The C Programming Language*, which prints a program's command-line arguments, separated by spaces:

```
while (--argc > 0)
  printf((argc > 1) ? "%s " : "%s", *++argv);
```

The format string is the expression `(argc > 1) ? "%s " : "%s"`, which evaluates to `"%s "` for all command-line arguments but the last.

Q: Which library functions other than `clearerr` clear a stream's error and end-of-file indicators? [p. 565]

A: Calling `rewind` clears both indicators, as does opening or reopening the stream. Calling `ungetc`, `fseek`, or `fsetpos` clears just the end-of-file indicator.

Q: I can't get `feof` to work; it seems to return zero even at end-of-file. What am I doing wrong? [p. 565]