Stream I/O

26.10.2022 • מאמר

These functions process data in different sizes and formats, from single characters to large data structures. They also provide buffering, which can improve performance. The default size of a stream buffer is 4K. These routines affect only buffers created by the run-time library routines, and have no effect on buffers created by the operating system.

Stream I/O routines

הרחב טבלה

Routine	Use
clearerr, clearerr_s	Clear error indicator for stream
fclose	Close stream
_fcloseall	Close all open streams except `stdin`, `stdout`, and `stderr`
_fdopen, wfdopen	Associate stream with file descriptor of open file
feof	Test for end of file on stream
ferror	Test for error on stream
fflush	Flush stream to buffer or storage device
fgetc, fgetwc	Read character from stream (function versions of `getc` and `getwc`)
_fgetchar, _fgetwchar	Read character from `stdin` (function versions of `getchar` and `getwchar`)
fgetpos	Get position indicator of stream
fgets, fgetws	Read string from stream
_fileno	Get file descriptor associated with stream
_flushall	Flush all streams to buffer or storage device
fopen, _wfopen, fopen_s, _wfopen_s	Open stream

Routine	Use
fprintf, _fprintf_l, fwprintf, _fwprintf_l, fprintf_s, _fprintf_s_l, fwprintf_s, _fwprintf_s_l	Write formatted data to stream
fputc, fputwc	Write a character to a stream (function versions of **`putc`** and **`putwc`**)
_fputchar, _fputwchar	Write character to **`stdout`** (function versions of **`putchar`** and **`putwchar`**)
fputs, fputws	Write string to stream
fread	Read unformatted data from stream
freopen, _wfreopen, freopen_s, _wfreopen_s	Reassign `FILE` stream pointer to new file or device
fscanf, fwscanf, fscanf_s, _fscanf_s_l, fwscanf_s, _fwscanf_s_l	Read formatted data from stream
fseek, _fseeki64	Move file position to given location
fsetpos	Set position indicator of stream
_fsopen, _wfsopen	Open stream with file sharing
ftell, _ftelli64	Get current file position
fwrite	Write unformatted data items to stream
getc, getwc	Read character from stream (macro versions of **`fgetc`** and **`fgetwc`**)
getchar, getwchar	Read character from **`stdin`** (macro versions of **`fgetchar`** and **`fgetwchar`**)
_getmaxstdio	Returns the number of simultaneously open files permitted at the stream I/O level.
gets_s, _getws_s	Read line from **`stdin`**
_getw	Read binary **`int`** from stream
printf, _printf_l, wprintf, _wprintf_l,printf_s, _printf_s_l, wprintf_s, _wprintf_s_l	Write formatted data to **`stdout`**
putc, putwc	Write character to a stream (macro versions of **`fputc`** and **`fputwc`**)
putchar, putwchar	Write character to **`stdout`** (macro versions of **`fputchar`** and **`fputwchar`**)

Routine	Use
puts, _putws	Write line to stream
_putw	Write binary `int` to stream
rewind	Move file position to beginning of stream
_rmtmp	Remove temporary files created by `tmpfile`
scanf, _scanf_l, wscanf, _wscanf_l,scanf_s, _scanf_s_l, wscanf_s, _wscanf_s_l	Read formatted data from `stdin`
setbuf	Control stream buffering
_setmaxstdio	Set a maximum for the number of simultaneously open files at the stream I/O level.
setvbuf	Control stream buffering and buffer size
_snprintf, _snwprintf, _snprintf_s, _snprintf_s_l, _snwprintf_s, _snwprintf_s_l	Write formatted data of specified length to string
_snscanf, _snwscanf, _snscanf_s, _snscanf_s_l, _snwscanf_s, _snwscanf_s_l	Read formatted data of a specified length from the standard input stream.
sprintf, swprintf, sprintf_s, _sprintf_s_l, swprintf_s, _swprintf_s_l	Write formatted data to string
sscanf, swscanf, sscanf_s, _sscanf_s_l, swscanf_s, _swscanf_s_l	Read formatted data from string
_tempnam, _wtempnam	Generate temporary filename in given directory
tmpfile, tmpfile_s	Create temporary file
tmpnam, _wtmpnam, tmpnam_s, _wtmpnam_s	Generate temporary filename
ungetc, ungetwc	Push character back onto stream
_vcprintf, _vcwprintf, _vcprintf_s, _vcprintf_s_l, _vcwprintf_s, _vcwprintf_s_l	Write formatted data to the console.
vfprintf, vfwprintf, vfprintf_s, _vfprintf_s_l, vfwprintf_s, _vfwprintf_s_l	Write formatted data to stream
vprintf, vwprintf, vprintf_s, _vprintf_s_l, vwprintf_s, _vwprintf_s_l	Write formatted data to `stdout`

Routine	Use
_vsnprintf, _vsnwprintf, vsnprintf_s, _vsnprintf_s, _vsnprintf_s_l, _vsnwprintf_s, _vsnwprintf_s_l	Write formatted data of specified length to buffer
vsprintf, vswprintf, vsprintf_s, _vsprintf_s_l, vswprintf_s, _vswprintf_s_l	Write formatted data to buffer

When a program begins execution, the startup code automatically opens several streams: standard input (pointed to by `stdin`), standard output (pointed to by `stdout`), and standard error (pointed to by `stderr`). These streams are directed to the console (keyboard and screen) by default. Use `freopen` to redirect `stdin`, `stdout`, or `stderr` to a disk file or a device.

Files opened using the stream routines are buffered by default. The `stdout` and `stderr` functions are flushed whenever they're full or, if you're writing to a character device, after each library call. If a program terminates abnormally, output buffers may not be flushed, resulting in loss of data. Use `fflush` or `_flushall` to ensure that the buffer associated with a specified file is flushed to the operating system, or all open buffers are flushed. The operating system can cache data before writing it to disk. The commit-to-disk feature ensures that the flushed buffer contents aren't lost if there's a system failure.

There are two ways to commit buffer contents to disk:

- Link with the file COMMODE.OBJ to set a global commit flag. The default setting of the global flag is `n`, for "no-commit."
- Set the mode flag to `c` with `fopen` or `_fdopen`.

Any file specifically opened with either the `c` or the `n` flag behaves according to the flag, regardless of the state of the global commit/no-commit flag.

If your program doesn't explicitly close a stream, the stream is automatically closed when the program terminates. However, you should close a stream when your program finishes with it, as the number of streams that can be open at one time is limited. See _setmaxstdio for information on this limit.

Input can follow output directly only with an intervening call to `fflush` or to a file-positioning function (`fseek`, `fsetpos`, or `rewind`). Input can be followed by output without an intervening call to a file-positioning function, if the input operation encounters the end of the file.

See also

Input and output

Universal C runtime routines by category
