Next: Unreading, Previous: Character Input, Up: I/O on Streams [Contents][Index]

12.9 Line-Oriented Input

Since many programs interpret input on the basis of lines, it is convenient to have functions to read a line of text from a stream.

Standard C has functions to do this, but they aren't very safe: null characters and even (for `gets`) long lines can confuse them. So the GNU C Library provides the nonstandard `getline` function that makes it easy to read lines reliably.

Another GNU extension, `getdelim`, generalizes `getline`. It reads a delimited record, defined as everything through the next occurrence of a specified delimiter character.

All these functions are declared in `stdio.h`.

Function: *ssize_t* **getline** *(char **lineptr, size_t *n, FILE *stream)*

> Preliminary: | MT-Safe | AS-Unsafe corrupt heap | AC-Unsafe lock corrupt mem | See POSIX Safety Concepts.
>
> This function reads an entire line from *stream*, storing the text (including the newline and a terminating null character) in a buffer and storing the buffer address in `*lineptr`.
>
> Before calling `getline`, you should place in `*lineptr` the address of a buffer `*n` bytes long, allocated with `malloc`. If this buffer is long enough to hold the line, `getline` stores the line in this buffer. Otherwise, `getline` makes the buffer bigger using `realloc`, storing the new buffer address back in `*lineptr` and the increased size back in `*n`. See Unconstrained Allocation.
>
> If you set `*lineptr` to a null pointer, and `*n` to zero, before the call, then `getline` allocates the initial buffer for you by calling `malloc`. This buffer remains allocated even if `getline` encounters errors and is unable to read any bytes.
>
> In either case, when `getline` returns, `*lineptr` is a `char *` which points to the text of the line.
>
> When `getline` is successful, it returns the number of characters read (including the newline, but not including the terminating null). This value enables you to distinguish null characters that are part of the line from the null character inserted as a terminator.
>
> This function is a GNU extension, but it is the recommended way to read lines from a stream. The alternative standard functions are unreliable.
>
> If an error occurs or end of file is reached without any bytes read, `getline` returns `-1`.

Function: *ssize_t* **getdelim** *(char **lineptr, size_t *n, int delimiter, FILE *stream)*

> Preliminary: | MT-Safe | AS-Unsafe corrupt heap | AC-Unsafe lock corrupt mem | See POSIX Safety Concepts.
>
> This function is like `getline` except that the character which tells it to stop reading is not necessarily newline. The argument *delimiter* specifies the delimiter character; `getdelim` keeps reading until it sees that character (or end of file).
>
> The text is stored in *lineptr*, including the delimiter character and a terminating null. Like `getline`, `getdelim` makes *lineptr* bigger if it isn't big enough.
>
> `getline` is in fact implemented in terms of `getdelim`, just like this:

```
ssize_t
getline (char **lineptr, size_t *n, FILE *stream)
{
  return getdelim (lineptr, n, '\n', stream);
}
```

Function: *char ** **fgets** *(char *s, int count, FILE *stream)*

Preliminary: | MT-Safe | AS-Unsafe corrupt | AC-Unsafe lock corrupt | See POSIX Safety Concepts.

The `fgets` function reads characters from the stream *stream* up to and including a newline character and stores them in the string *s*, adding a null character to mark the end of the string. You must supply *count* characters worth of space in *s*, but the number of characters read is at most *count* - 1. The extra character space is used to hold the null character at the end of the string.

If the system is already at end of file when you call `fgets`, then the contents of the array *s* are unchanged and a null pointer is returned. A null pointer is also returned if a read error occurs. Otherwise, the return value is the pointer *s*.

Warning: If the input data has a null character, you can't tell. So don't use `fgets` unless you know the data cannot contain a null. Don't use it to read files edited by the user because, if the user inserts a null character, you should either handle it properly or print a clear error message. We recommend using `getline` instead of `fgets`.

Function: *wchar_t ** **fgetws** *(wchar_t *ws, int count, FILE *stream)*

Preliminary: | MT-Safe | AS-Unsafe corrupt | AC-Unsafe lock corrupt | See POSIX Safety Concepts.

The `fgetws` function reads wide characters from the stream *stream* up to and including a newline character and stores them in the string *ws*, adding a null wide character to mark the end of the string. You must supply *count* wide characters worth of space in *ws*, but the number of characters read is at most *count* - 1. The extra character space is used to hold the null wide character at the end of the string.

If the system is already at end of file when you call `fgetws`, then the contents of the array *ws* are unchanged and a null pointer is returned. A null pointer is also returned if a read error occurs. Otherwise, the return value is the pointer *ws*.

Warning: If the input data has a null wide character (which are null bytes in the input stream), you can't tell. So don't use `fgetws` unless you know the data cannot contain a null. Don't use it to read files edited by the user because, if the user inserts a null character, you should either handle it properly or print a clear error message.

Function: *char ** **fgets_unlocked** *(char *s, int count, FILE *stream)*

Preliminary: | MT-Safe race:stream | AS-Unsafe corrupt | AC-Unsafe corrupt | See POSIX Safety Concepts.

The `fgets_unlocked` function is equivalent to the `fgets` function except that it does not implicitly lock the stream.

This function is a GNU extension.

Function: *wchar_t ** **fgetws_unlocked** *(wchar_t *ws, int count, FILE *stream)*

Preliminary: | MT-Safe race:stream | AS-Unsafe corrupt | AC-Unsafe corrupt | See POSIX Safety Concepts.

The `fgetws_unlocked` function is equivalent to the `fgetws` function except that it does not implicitly lock the stream.

This function is a GNU extension.

Deprecated function: *char* * **gets** *(char *s)*

Preliminary: | MT-Safe | AS-Unsafe corrupt | AC-Unsafe lock corrupt | See POSIX Safety Concepts.

The function `gets` reads characters from the stream `stdin` up to the next newline character, and stores them in the string *s*. The newline character is discarded (note that this differs from the behavior of `fgets`, which copies the newline character into the string). If `gets` encounters a read error or end-of-file, it returns a null pointer; otherwise it returns *s*.

Warning: The `gets` function is **very dangerous** because it provides no protection against overflowing the string *s*. The GNU C Library includes it for compatibility only. You should **always** use `fgets` or `getline` instead. To remind you of this, the linker (if using GNU `ld`) will issue a warning whenever you use `gets`.

Next: Unreading, Previous: Character Input, Up: I/O on Streams [Contents][Index]