vscanf, vfscanf, vsscanf, vscanf_s, vfscanf_s, vsscanf_s

Defined in header <stdio.h>

`int vscanf( const char *restrict format, va_list vlist );`	(1)	(since C99)
`int vfscanf( FILE *restrict stream, const char *restrict format, va_list vlist );`	(2)	(since C99)
`int vsscanf( const char *restrict buffer, const char *restrict format, va_list vlist );`	(3)	(since C99)
`int vscanf_s(const char *restrict format, va_list vlist);`	(4)	(since C11)
`int vfscanf_s( FILE *restrict stream, const char *restrict format, va_list vlist);`	(5)	(since C11)
`int vsscanf_s( const char *restrict buffer, const char *restrict format, va_list vlist);`	(6)	(since C11)

Reads data from the a variety of sources, interprets it according to format and stores the results into locations defined by vlist.

1) Reads the data from stdin

2) Reads the data from file stream stream

3) Reads the data from null-terminated character string buffer. Reaching the end of the string is equivalent to reaching the end-of-file condition for fscanf

4-6) Same as (1-3), except that `%c`, `%s`, and `%[` conversion specifiers each expect two arguments (the usual pointer and a value of type rsize_t indicating the size of the receiving array, which may be 1 when reading with a %c into a single char) and except that the following errors are detected at runtime and call the currently installed constraint handler function:

- any of the arguments of pointer type is a null pointer
- format, stream, or buffer is a null pointer
- the number of characters that would be written by %c, %s, or %[, plus the terminating null character, would exceed the second (rsize_t) argument provided for each of those conversion specifiers
- optionally, any other detectable error, such as unknown conversion specifier

As with all bounds-checked functions, vscanf_s , vfscanf_s, and vsscanf_s are only guaranteed to be available if `__STDC_LIB_EXT1__` is defined by the implementation and if the user defines `__STDC_WANT_LIB_EXT1__` to the integer constant 1 before including stdio.h.

Parameters

stream	-	input file stream to read from
buffer	-	pointer to a null-terminated character string to read from
format	-	pointer to a null-terminated character string specifying how to read the input
vlist	-	variable argument list containing the receiving arguments.

The **format** string consists of

- non-whitespace multibyte characters except %: each such character in the format string consumes exactly one identical character from the input stream, or causes the function to fail if the next character on the stream does not compare equal.
- whitespace characters: any single whitespace character in the format string consumes all available consecutive whitespace characters from the input (determined as if by calling isspace in a loop). Note that there is no difference between `"\n"`, `" "`, `"\t\t"`, or other whitespace in the format string.
- conversion specifications. Each conversion specification has the following format:
  - introductory % character
  - (optional) assignment-suppressing character *. If this option is present, the function does not assign the result of the conversion to any receiving argument.
  - (optional) integer number (greater than zero) that specifies *maximum field width*, that is, the maximum number of characters that the function is allowed to consume when doing the conversion specified by the current conversion specification. Note that %s and %[may lead to buffer overflow if the width is not provided.
  - (optional) *length modifier* that specifies the size of the receiving argument, that is, the actual destination type. This affects the conversion accuracy and overflow rules. The default destination

type is different for each conversion type (see table below).

- conversion format specifier

The following format specifiers are available:

Conversion specifier	Explanation	Argument type								
	Length modifier →	hh (C99)	h	(none)	l	ll (C99)	j (C99)	z (C99)	t (C99)	L
%	matches literal %	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
c	matches a **character** or a sequence of **characters** If a width specifier is used, matches exactly *width* characters (the argument must be a pointer to an array with sufficient room). Unlike %s and %[, does not append the null character to the array.	N/A	N/A	char*	wchar_t*	N/A	N/A	N/A	N/A	N/A
s	matches a sequence of non-whitespace characters (a **string**) If width specifier is used, matches up to *width* or until the first whitespace character, whichever appears first. Always stores a null character in addition to the characters matched (so the argument array must have room for at least *width+1* characters)	N/A	N/A	char*	wchar_t*	N/A	N/A	N/A	N/A	N/A
[*set*]	matches a non-empty sequence of character from *set* of characters. If the first character of the set is ^, then all characters not in the set are matched. If the set begins with] or ^] then the] character is also included into the set. It is implementation-defined whether the character - in the non-initial position in the scanset may be indicating a range, as in [0-9]. If width specifier is used, matches only up to *width*. Always stores a null character in addition to the characters matched (so the argument array must have room for at least *width+1* characters)	N/A	N/A	char*	wchar_t*	N/A	N/A	N/A	N/A	N/A
d	matches a **decimal integer**. The format of the number is the same as expected by strtol() with the value 10 for the base argument	signed char* or unsigned char*	signed short* or unsigned short*	signed int* or unsigned int*	signed long* or unsigned long*	signed long long* or unsigned long long*	intmax_t* or uintmax_t*	size_t*	ptrdiff_t*	N/A
i	matches an **integer**. The format of the number is the same as expected by strtol() with the value 0 for the base argument (base is determined by the first characters parsed)	signed char* or unsigned char*	signed short* or unsigned short*	signed int* or unsigned int*	signed long* or unsigned long*	signed long long* or unsigned long long*	intmax_t* or uintmax_t*	size_t*	ptrdiff_t*	N/A
u	matches an unsigned **decimal integer**. The format of the number is the same as expected by strtoul() with the value 10 for the base argument.	signed char* or unsigned char*	signed short* or unsigned short*	signed int* or unsigned int*	signed long* or unsigned long*	signed long long* or unsigned long long*	intmax_t* or uintmax_t*	size_t*	ptrdiff_t*	N/A
o	matches an unsigned **octal integer**. The format of the number is the same as expected by strtoul() with the value 8 for the base argument	signed char* or unsigned char*	signed short* or unsigned short*	signed int* or unsigned int*	signed long* or unsigned long*	signed long long* or unsigned long long*	intmax_t* or uintmax_t*	size_t*	ptrdiff_t*	N/A
x, X	matches an unsigned **hexadecimal integer**. The format of the number is the same as expected by strtoul() with the value 16 for the base argument	signed char* or unsigned char*	signed short* or unsigned short*	signed int* or unsigned int*	signed long* or unsigned long*	signed long long* or unsigned long long*	intmax_t* or uintmax_t*	size_t*	ptrdiff_t*	N/A
n	returns the **number of characters read so far**. No input is consumed. Does not increment the assignment count. If the specifier has assignment-suppressing operator defined, the behavior is undefined	signed char* or unsigned char*	signed short* or unsigned short*	signed int* or unsigned int*	signed long* or unsigned long*	signed long long* or unsigned long long*	intmax_t* or uintmax_t*	size_t*	ptrdiff_t*	N/A
a, A(C99) **e, E** **f, F** **g, G**	matches a **floating-point number**. The format of the number is the same as expected by strtof()	N/A	N/A	float*	double*	N/A	N/A	N/A	N/A	long double*
p	matches implementation defined character sequence defining a **pointer**. printf family of functions should produce the same sequence using **%p** format specifier	N/A	N/A	void**	N/A	N/A	N/A	N/A	N/A	N/A

For every conversion specifier other than n, the longest sequence of input characters which does not exceed any specified field width and which either is exactly what the conversion specifier expects or is a prefix of a sequence it would expect, is what's consumed from the stream. The first character, if any, after this consumed sequence remains unread. If the consumed sequence has length zero or if the consumed sequence cannot be converted as specified above, the matching failure occurs unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

All conversion specifiers other than [, c, and n consume and discard all leading whitespace characters (determined as if by calling isspace) before attempting to parse the input. These consumed characters do not count towards the specified maximum field width.

The conversion specifiers lc, ls, and l[perform multibyte-to-wide character conversion as if by calling mbrtowc() with an mbstate_t object initialized to zero before the first character is converted.

The conversion specifiers s and [always store the null terminator in addition to the matched characters. The size of the destination array must be at least one greater than the specified field width. The use of `%s` or `%[`, without specifying the destination array size, is as unsafe as gets

The correct conversion specifications for the fixed-width integer types (int8_t, etc) are defined in the header <inttypes.h> (although SCNdMAX, SCNuMAX, etc is synonymous with %jd, %ju, etc).

There is a sequence point after the action of each conversion specifier; this permits storing multiple fields in the same "sink" variable.

When parsing an incomplete floating-point value that ends in the exponent with no digits, such as parsing `"100er"` with the conversion specifier `%f`, the sequence `"100e"` (the longest prefix of a possibly valid floating-point number) is consumed, resulting in a matching error (the consumed sequence cannot be converted to a floating-point number), with `"r"` remaining. Some existing implementations do not follow this rule and roll back to consume only `"100"`, leaving `"er"`, e.g. glibc bug 1765 (https://sourceware.org/bugzilla/show_bug.cgi?id=1765)

Return value

1-3) Number of receiving arguments successfully assigned, or EOF if read failure occurs before the first receiving argument was assigned.

4-6) Same as (1-3), except that EOF is also returned if there is a runtime constraint violation.

Notes

All these functions invoke va_arg at least once, the value of arg is indeterminate after the return. These functions to not invoke va_end, and it must be done by the caller.

Example

Run this code

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdarg.h>

bool checked_sscanf(int count, const char* buf, const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    int rc = vsscanf(buf, fmt, ap);
    va_end(ap);
    return rc == count;
}

int main(void)
{
    int n, m;

    printf("Parsing '1 2'...");
    if(checked_sscanf(2, "1 2", "%d %d", &n, &m))
        puts("success");
    else
        puts("failure");

    printf("Parsing '1 a'...");
    if(checked_sscanf(2, "1 a", "%d %d", &n, &m))
        puts("success");
    else
```

```
        puts("failure");
}
```

Output:

```
Parsing '1 2'...success
Parsing '1 a'...failure
```

References

- C11 standard (ISO/IEC 9899:2011):
  - 7.21.6.9 The vfscanf function (p: 327)
  - 7.21.6.11 The vscanf function (p: 328)
  - 7.21.6.14 The vsscanf function (p: 330)
  - K.3.5.3.9 The vfscanf_s function (p: 597-598)
  - K.3.5.3.11 The vscanf_s function (p: 599)
  - K.3.5.3.14 The vsscanf_s function (p: 602)
- C99 standard (ISO/IEC 9899:1999):
  - 7.19.6.9 The vfscanf function (p: 293)
  - 7.19.6.11 The vscanf function (p: 294)
  - 7.19.6.14 The vsscanf function (p: 295)

See also

scanf **fscanf** **sscanf** **scanf_s** (C11) **fscanf_s** (C11) **sscanf_s** (C11)	reads formatted input from stdin, a file stream or a buffer (function)
vprintf **vfprintf** **vsprintf** **vsnprintf** (C99) **vprintf_s** (C11) **vfprintf_s** (C11) **vsprintf_s** (C11) **vsnprintf_s** (C11)	prints formatted output to stdout, a file stream or a buffer using variable argument list (function)
C++ documentation for **vscanf**, **vfscanf**, **vsscanf**	

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=c/io/vfscanf&oldid=125689"