std::scanf, std::fscanf, std::sscanf

Defined in header `<cstdio>`

`int scanf( const char* format, ... );`	(1)
`int fscanf( std::FILE* stream, const char* format, ... );`	(2)
`int sscanf( const char* buffer, const char* format, ... );`	(3)

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

1) Reads the data from `stdin`

2) Reads the data from file stream `stream`

3) Reads the data from null-terminated character string `buffer`

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
...	-	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** (C++11)	**h**	(none)	**l**	**ll** (C++11)	**j** (C++11)	**z** (C++11)	**t** (C++11)	**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(C++11) **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 std::gets

The correct conversion specifications for the fixed-width integer types (int8_t, etc) are defined in the header <cinttypes> (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

Number of receiving arguments successfully assigned (which may be zero in case a matching failure occurred before the first receiving argument was assigned), or EOF if input failure occurs before the first receiving argument was assigned.

Complexity

Not guaranteed. Notably, some implementations of sscanf are $O(N)$, where `N = std::strlen(buffer)` [1] (https://sourceware.org/bugzilla/show_bug.cgi?id=17577) . For performant string parsing, see std::from_chars.

Notes

Because most conversion specifiers first consume all consecutive whitespace, code such as

```
std::scanf("%d", &a);
std::scanf("%d", &b);
```

will read two integers that are entered on different lines (second %d will consume the newline left over by the first) or on the same line, separated by spaces or tabs (second %d will consume the spaces or tabs).

The conversion specifiers that do not consume leading whitespace, such as %c, can be made to do so by using a whitespace character in the format string:

```
std::scanf("%d", &a);
std::scanf(" %c", &c); // ignore the endline after %d, then read a char
```

Note that some implementations of sscanf involve a call to strlen, which makes their runtime linear on the length of the entire string. This means that if sscanf is called in a loop to repeatedly parse values from the front of a string, your code might run in quadratic time (example (https://nee.lv/2021/02/28/How-I-cut-GTA-Online-loading-times-by-70/#Problem-one-It%E2%80%99s%E2%80%A6-strlen)).

Example

Run this code

```
#include <iostream>
#include <clocale>
#include <cstdio>

int main()
```

```
{
    int i, j;
    float x, y;
    char str1[10], str2[4];
    wchar_t warr[2];
    std::setlocale(LC_ALL, "en_US.utf8");

    char input[] = "25 54.32E-1 Thompson 56789 0123 56ß水";
    // parse as follows:
    // %d: an integer
    // %f: a floating-point value
    // %9s: a string of at most 9 non-whitespace characters
    // %2d: two-digit integer (digits 5 and 6)
    // %f: a floating-point value (digits 7, 8, 9)
    // %*d an integer which isn't stored anywhere
    // ' ': all consecutive whitespace
    // %3[0-9]: a string of at most 3 digits (digits 5 and 6)
    // %2lc: two wide characters, using multibyte to wide conversion
    int ret = std::sscanf(input, "%d%f%9s%2d%f%*d %3[0-9]%2lc",
                          &i, &x, str1, &j, &y, str2, warr);

    std::cout << "Converted " << ret << " fields:\n"
              << "i = " << i << "\nx = " << x << '\n'
              << "str1 = " << str1 << "\nj = " << j << '\n'
              << "y = " << y << "\nstr2 = " << str2 << '\n'
              << std::hex << "warr[0] = U+" << (int)warr[0]
              << " warr[1] = U+" << (int)warr[1] << '\n';
}
```

Output:

```
Converted 7 fields:
i = 25
x = 5.432
str1 = Thompson
j = 56
y = 789
str2 = 56
warr[0] = U+df warr[1] = U+6c34
```

See also

vscanf (C++11) **vfscanf** (C++11) **vsscanf** (C++11)	reads formatted input from `stdin`, a file stream or a buffer using variable argument list (function)
fgets	gets a character string from a file stream (function)
printf **fprintf** **sprintf** **snprintf** (C++11)	prints formatted output to `stdout`, a file stream or a buffer (function)
from_chars (C++17)	converts a character sequence to an integer or floating-point value (function)

C documentation for **scanf, fscanf, sscanf**

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/io/c/fscanf&oldid=127357"