NAME

scanf, **fscanf**, **sscanf**, **vscanf**, **vsscanf**, **vfscanf** - input format conversion

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <stdio.h>

int
scanf(*const char * restrict format, ...*);

int
fscanf(*FILE * restrict stream, const char * restrict format, ...*);

int
sscanf(*const char * restrict str, const char * restrict format, ...*);

#include <stdarg.h>

int
vscanf(*const char * restrict format, va_list ap*);

int
vsscanf(*const char * restrict str, const char * restrict format, va_list ap*);

int
vfscanf(*FILE * restrict stream, const char * restrict format, va_list ap*);

DESCRIPTION

The **scanf**() family of functions scans input according to a *format* as described below. This format may contain *conversion specifiers*; the results from such conversions, if any, are stored through the *pointer* arguments. The **scanf**() function reads input from the standard input stream stdin, **fscanf**() reads input from the stream pointer *stream*, and **sscanf**() reads its input from the character string pointed to by *str*. The **vfscanf**() function is analogous to vfprintf(3) and reads input from the stream pointer *stream* using a variable argument list of pointers (see stdarg(3)). The **vscanf**() function scans a variable argument list from the standard input and the **vsscanf**() function scans it from a string; these are analogous to the **vprintf**() and **vsprintf**() functions respectively. Each successive *pointer* argument must correspond properly with each successive conversion specifier (but see the ***** conversion below). All conversions are introduced by the **%** (percent sign) character. The *format* string may also contain other characters.

White space (such as blanks, tabs, or newlines) in the *format* string match any amount of white space, including none, in the input. Everything else matches only itself. Scanning stops when an input character does not match such a format character. Scanning also stops when an input conversion cannot be made (see below).

CONVERSIONS

Following the **%** character introducing a conversion there may be a number of *flag* characters, as follows:

***** Suppresses assignment. The conversion that follows occurs as usual, but no pointer is used; the result of the conversion is simply discarded.

hh Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *char* (rather than *int*).

h Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *short int* (rather than *int*).

l (ell) Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *long int* (rather than *int*), that the conversion will be one of **a**, **e**, **f**, or **g** and the next pointer is a pointer to *double* (rather than *float*), or that the conversion will be one of **c**, **s** or **[** and the next pointer is a pointer to an array of *wchar_t* (rather than *char*).

ll (ell ell)
Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *long long int* (rather than *int*).

L Indicates that the conversion will be one of **a**, **e**, **f**, or **g** and the next pointer is a pointer to *long double*.

j Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *intmax_t* (rather than *int*).

t Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *ptrdiff_t* (rather than *int*).

w*N* (where *N* is 8, 16, 32, or 64) Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *intN_t* (rather than *int*).

wf*N* (where *N* is 8, 16, 32, or 64) Indicates that the conversion will be one of **bdioux** or **n** and the

next pointer is a pointer to a *int_fastN_t* (rather than *int*).

z Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *size_t* (rather than *int*).

q (deprecated.) Indicates that the conversion will be one of **bdioux** or **n** and the next pointer is a pointer to a *long long int* (rather than *int*).

In addition to these flags, there may be an optional maximum field width, expressed as a decimal integer, between the **%** and the conversion. If no width is given, a default of "infinity" is used (with one exception, below); otherwise at most this many bytes are scanned in processing the conversion. In the case of the **lc**, **ls** and **l[** conversions, the field width specifies the maximum number of multibyte characters that will be scanned. Before conversion begins, most conversions skip white space; this white space is not counted against the field width.

The following conversions are available:

% Matches a literal ‘%’. That is, "%%" in the format string matches a single input ‘%’ character. No conversion is done, and assignment does not occur.

b, **B** Matches an optionally signed binary integer; the next pointer must be a pointer to *unsigned int*.

d Matches an optionally signed decimal integer; the next pointer must be a pointer to *int*.

i Matches an optionally signed integer; the next pointer must be a pointer to *int*. The integer is read in base 2 if it begins with ‘0b’ or ‘0B’, in base 16 if it begins with ‘0x’ or ‘0X’, in base 8 if it begins with ‘0’, and in base 10 otherwise. Only characters that correspond to the base are used.

o Matches an octal integer; the next pointer must be a pointer to *unsigned int*.

u Matches an optionally signed decimal integer; the next pointer must be a pointer to *unsigned int*.

x, **X** Matches an optionally signed hexadecimal integer; the next pointer must be a pointer to *unsigned int*.

a, **A**, **e**, **E**, **f**, **F**, **g**, **G**
Matches a floating-point number in the style of strtod(3). The next pointer must be a pointer to *float* (unless **l** or **L** is specified.)

s Matches a sequence of non-white-space characters; the next pointer must be a pointer to *char*, and the array must be large enough to accept all the sequence and the terminating NUL character. The input string stops at white space or at the maximum field width, whichever occurs first.

If an **l** qualifier is present, the next pointer must be a pointer to *wchar_t*, into which the input will be placed after conversion by mbrtowc(3).

S The same as **ls**.

c Matches a sequence of *width* count characters (default 1); the next pointer must be a pointer to *char*, and there must be enough room for all the characters (no terminating NUL is added). The usual skip of leading white space is suppressed. To skip white space first, use an explicit space in the format.

If an **l** qualifier is present, the next pointer must be a pointer to *wchar_t*, into which the input will be placed after conversion by mbrtowc(3).

C The same as **lc**.

[Matches a nonempty sequence of characters from the specified set of accepted characters; the next pointer must be a pointer to *char*, and there must be enough room for all the characters in the string, plus a terminating NUL character. The usual skip of leading white space is suppressed. The string is to be made up of characters in (or not in) a particular set; the set is defined by the characters between the open bracket **[** character and a close bracket **]** character. The set *excludes* those characters if the first character after the open bracket is a circumflex ^. To include a close bracket in the set, make it the first character after the open bracket or the circumflex; any other position will end the set. The hyphen character **-** is also special; when placed between two other characters, it adds all intervening characters to the set. To include a hyphen, make it the last character before the final close bracket. For instance, ‘[^]0-9-]’ means the set "everything except close bracket, zero through nine, and hyphen". The string ends with the appearance of a character not in the (or, with a circumflex, in) set or when the field width runs out.

If an **l** qualifier is present, the next pointer must be a pointer to *wchar_t*, into which the input will be placed after conversion by mbrtowc(3).

p Matches a pointer value (as printed by ‘%p’ in printf(3)); the next pointer must be a pointer to *void*.

n Nothing is expected; instead, the number of characters consumed thus far from the input is stored through the next pointer, which must be a pointer to *int*. This is *not* a conversion, although it can be suppressed with the ***** flag.

The decimal point character is defined in the program's locale (category LC_NUMERIC).

For backwards compatibility, a "conversion" of '%\0' causes an immediate return of EOF.

RETURN VALUES

These functions return the number of input items assigned, which can be fewer than provided for, or even zero, in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned; typically this is due to an invalid input character, such as an alphabetic character for a '%d' conversion. The value EOF is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions which were successfully completed is returned.

SEE ALSO

getc(3), mbrtowc(3), printf(3), strtod(3), strtol(3), strtoul(3), wscanf(3)

STANDARDS

The functions **fscanf**(), **scanf**(), **sscanf**(), **vfscanf**(), **vscanf**() and **vsscanf**() conform to ISO/IEC 9899:1999 ("ISO C99").

HISTORY

The functions **scanf**(), **fscanf**(), and **sscanf**() first appeared in Version 7 AT&T UNIX, and **vscanf**(), **vsscanf**(), and **vfscanf**() in 4.3BSD-Reno.

BUGS

Earlier implementations of **scanf** treated **%D**, **%E**, **%F**, **%O** and **%X** as their lowercase equivalents with an **l** modifier. In addition, **scanf** treated an unknown conversion character as **%d** or **%D**, depending on its case. This functionality has been removed.

Numerical strings are truncated to 512 characters; for example, **%f** and **%d** are implicitly **%512f** and **%512d**.

The **%n$** modifiers for positional arguments are not implemented.

The **scanf** family of functions do not correctly handle multibyte characters in the *format* argument.