NAME

scanf, fscanf, sscanf – convert formatted input

SYNOPSIS

#include <stdio.h>

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

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

int sscanf(const char **s***, const char** **format***, . . .);**

MT-LEVEL

MT-Safe

DESCRIPTION

scanf( ) reads from the standard input stream, **stdin**.

fscanf( ) reads from the stream *strm*.

sscanf( ) reads from the character string *s*.

Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string, *format*, described below and a set of pointer arguments indicating where the converted input should be stored. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, new-lines, or form-feeds) that, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %) that must match the next character of the input stream.
3. Conversion specifications consisting of the character **%** or the character sequence **%***digits***$**, an optional assignment suppression character *****, a decimal digit string that specifies an optional numerical maximum field width, an optional letter **l** (ell), **L**, or **h** indicating the size of the receiving object, and a conversion code:
   % or *digit*, ***, *decimal digit string*, **h** or **l** or **L**, *conversion code*

   The following defines which size indicators can used with which conversion codes, and the size they indicate.

Conversion Code	**Size Indicator**	**Size**
d, i, n	none	int
	h	short int
	l	long int
o, u, x	none	unsigned int
	h	unsigned short int
	l	unsigned long int
e, f, g	none	float
	l	double
	L	long double

The **h**, **l**, or **L** modifier is ignored with any other conversion codes.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument unless assignment suppression was indicated by the character ∗. The suppression of assignment provides a way of describing an input field that is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the maximum field width, if one is specified, is exhausted. For all descriptors except the character **[** and the character **c**, white space leading an input field is ignored.

Conversions can be applied to the *nth* argument in the argument list, rather than to the next unused argument. In this case, the conversion character % (see above) is replaced by the sequence %*digits*$ where *digits* is a decimal integer *n*, giving the position of the argument in the argument list. The first such argument, **%1$**, immediately follows *format*. The control string can contain either form of a conversion specification, that is, % or %*digits*$, although the two forms cannot be mixed within a single control string.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion codes are valid:

% A single % is expected in the input at this point; no assignment is done.

d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the **strtol( )** function with the value 10 for the *base* argument. The corresponding argument should be a pointer to integer.

u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the **strtoul( )** function (see **strtol**(3C)) with the value 10 for the *base* argument. The corresponding argument should be a pointer to unsigned integer.

o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the **strtoul( )** function with the value 8 for the *base* argument. The corresponding argument should be a pointer to unsigned integer.

x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the **strtoul( )** function with the value 16 for the *base* argument. The corresponding argument should be a pointer to unsigned integer.

i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the **strtol( )** function with the value 0 for the *base* argument. The corresponding argument should be a pointer to integer.

n No input is consumed. The corresponding argument should be a pointer to integer into which is to be written the number of characters read from the input stream so far by the call to the function. Execution of a **%n** directive does not increment the assignment count returned at the completion of execution of the function.

e,**f**,**g** Matches an optionally signed floating point number, whose format is the same as expected for the subject string of the **strtod** function. The corresponding argument should be a pointer to floating.

s A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white-space character.

ws A wide character string is expected; the corresponding argument should be a wide character pointer pointing to an array of wide characters large enough to accept the wide character string and a terminating **\0**, which will be added automatically. The input field is terminated by a white-space character.

c Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added. The normal skip over white space is suppressed.

wc Matches a sequence of wide characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added. The normal skip over white space is suppressed.

[Matches a nonempty sequence of characters from a set of expected characters (the *scanset*). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically. The conversion specifier includes all subsequent characters in the *format* string, up to and including the matching right bracket (**]**). The characters between the brackets (the *scanlist*) comprise the scanset, unless the character after the left bracket is a circumflex (**^**), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with **[]** or **[^]**, the right bracket character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.

A range of characters in the scanset may be represented by the construct *first* – *last*; thus **[0123456789]** may be expressed **[0–9]**. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The character – will also stand for itself whenever it is the first or the last character in the scanlist. To include the right bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanlist and in this case it will not be syntactically interpreted as the closing bracket. At least one character must match for this conversion to be considered successful.

p Matches the set of implementation-defined sequences produced as output by the **%p** conversion of the **printf**(3S) function. The corresponding argument should be a pointer to **void**. If the input item is a value converted earlier during the same program execution, the pointer that results compares equal to that value; otherwise, the behavior of the **%p** conversion is undefined.

If an invalid conversion character follows the **%**, the results of the operation may not be predictable.

The conversion specifiers **E**, **G**, and **X** are also valid and, under the **-Xa** and **-Xc** compilation modes (see **cc**(1B)), behave the same as **e**, **g**, and **x**, respectively. Under the **-Xt** compilation mode, **E**, **G**, and **X** behave the same as **le**, **lg**, and **lx**, respectively.

Each function allows for detection of a language-dependent decimal point character in the input string. The decimal point character is defined by the program's locale (category **LC_NUMERIC**). In the **"C"** locale, or in a locale where the decimal point character is not defined, the decimal point character defaults to a period (**.**).

The **scanf( )** conversion terminates at end of file, at the end of the control string, or when an input character conflicts with the control string.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the **%n** directive.

LC_NUMERIC
Determines how numeric formats are handled. In the "C" locale, numeric

handling follows the U.S. rules.

RETURN VALUES

These routines return the number of successfully matched and assigned input items; this number can be 0 in the event of an early matching failure between an input character and the control string. If the input ends before the first matching failure or conversion, **EOF** is returned.

EXAMPLES

The call to the function **scanf( )**:

```
int i, n; float x; char name[50];
n = scanf ("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E–1 thompson
```

will assign to **n** the value **3**, to **i** the value **25**, to **x** the value **5.432**, and **name** will contain **thompson\0**.

The call to the function **scanf( )**:

```
int i; float x; char name[50];
(void) scanf ("%2d%f%*d %[0–9]", &i, &x, name);
```

with the input line:

```
56789 0123 56a72
```

will assign **56** to **i**, **789.0** to **x**, skip **0123**, and place the characters **56\0** in **name**. The next character read from **stdin** will be **a**.

FILES

/usr/lib/locale/*locale*/LC_NUMERIC/numeric
LC_NUMERIC database for *locale*

SEE ALSO

cc(1B), **strtod**(3C), **strtol**(3C), **printf**(3S)