NAME

scanf, fscanf, sscanf − formatted input conversion

SYNOPSIS

#include <stdio.h>

scanf (format [, pointer] . . .)
char *format;

fscanf (stream, format [, pointer] . . .)
FILE *stream;
char *format;

sscanf (s, format [, pointer] . . .)
char *s, *format;

DESCRIPTION

Scanf reads from the standard input stream *stdin*. *Fscanf* reads from the named input *stream*. *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.

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

1. Blanks, tabs, or new-lines, which cause input to be read up to the next non-white-space character.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

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 *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

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

d a decimal integer is expected; the corresponding argument should be an integer pointer.

o an octal integer is expected; the corresponding argument should be an integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

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. The input field is terminated by a space character or a new-line.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

e
f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits possibly containing a

decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

[indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (ˆ), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ˆ, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceded by **l** to indicate that a pointer to **long** rather than **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be capitalized or preceded by **l** to indicate that a pointer to **double** rather than **float** is in the argument list. The character **h** will function similarly in the future to indicate **short** data items.

Scanf conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

Scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

EXAMPLES

The call:

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

with the input line

```
25   54.32E−1   thompson
```

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

```
int i; float x; char name[50];
scanf ("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to *getchar* will return **a**.

SEE ALSO

atof(3C), getc(3S), printf(3S)

NOTE

Trailing white space (including a new-line) is left unread unless matched in the control string.

DIAGNOSTICS

The *scanf* functions return EOF on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.