NAME

printf, fprintf, sprintf, vprintf, vfprintf, vsprintf – formatted output conversion

SYNOPSIS

/usr/ucb/cc [*flag* ...] *file* ...

#include <stdio.h>

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

int fprintf(*stream, format, va_list***)**
FILE **stream*;
char **format*;
va_dcl;

char *sprintf(*s, format, va_list***)**
char **s*, **format*;
va_dcl;

int vprintf(*format, ap***)**
char **format*;
va_list *ap*;

int vfprintf(*stream, format, ap***)**
FILE **stream*;
char **format*;
va_list *ap*;

char *vsprintf(*s, format, ap***)**
char **s,*format*;
va_list *ap*;

DESCRIPTION

printf( ) places output on the standard output stream **stdout**. **fprintf( )** places output on the named output *stream*. **sprintf( )** places "output," followed by the **NULL** character (**0**), in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough storage is available.

vprintf( ), **vfprintf( )**, and **vsprintf( )** are the same as **printf( )**, **fprintf( )**, and **sprintf( )** respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **varargs**(5).

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of zero or more *arg*s. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag '–', described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (**.**) followed by a decimal digit string; a **NULL** digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*. An **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (∗) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted. A negative field width argument is taken as a '–' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

– The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (**+** or –).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and **+** flags both appear, the blank flag will be ignored.

This flag specifies that the value is to be converted to an "alternate form."For **c**, **d**, **i**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** or **X** conversion, a non-zero result will have **0x** or **0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,**i**,**o**,**u**,**x**,**X** The integer *arg* is converted to signed decimal (**d** or **i**), unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a **NULL** string.

f The float or double *arg* is converted to decimal notation in the style [–]*ddd***.***ddd* where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e,**E** The float or double *arg* is converted in the style [–]*d***.***ddd***e**±*ddd*, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

g,**G** The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** or **E** will be used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

The **e**, **E** **f**, **g,** and **G** formats print IEEE indeterminate values (infinity or not-a-number) as “Infinity” or “NaN” respectively.

c The character *arg* is printed.

s The *arg* is taken to be a string (character pointer) and characters from the string are printed until a **NULL** character (**\0**) is encountered or until the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first **NULL** character are printed. A **NULL** value for *arg* will yield undefined results.

% Print a **%**; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Padding takes place only if the specified field width exceeds the actual width. Characters generated by **printf( )** and **fprintf( )** are printed as if **putc**(3S) had been called.

RETURN VALUES

Upon success, **printf( )** and **fprintf( )** return the number of characters transmitted, excluding the null character. **vprintf( )** and **vfprintf( )** return the number of characters transmitted. **sprintf( )** and **vsprintf( )** always return *s*. If an output error is encountered, **printf( )**, **fprint( )**, **vprintf( )**, and **vfprintf( )** return EOF.

EXAMPLES

To print a date and time in the form “Sunday, July 3, 10:02,” where *weekday* and *month* are pointers to **NULL**-terminated strings:

printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);

To print π to 5 decimal places:

printf("pi = %.5f", 4 * atan(1. 0));

SEE ALSO

econvert(3), **putc**(3S), **scanf**(3S), **vprintf**(3S), **varargs**(5)

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

Very wide fields (>128 characters) fail.