NAME

wprintf, **fwprintf**, **swprintf**, **vwprintf**, **vfwprintf**, **vswprintf** - formatted wide character output conversion

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <stdio.h>
#include <wchar.h>

int
fwprintf(*FILE * restrict stream, const wchar_t * restrict format, ...*);

int
swprintf(*wchar_t * restrict ws, size_t n, const wchar_t * restrict format, ...*);

int
wprintf(*const wchar_t * restrict format, ...*);

#include <stdarg.h>

int
vfwprintf(*FILE * restrict stream, const wchar_t * restrict, va_list ap*);

int
vswprintf(*wchar_t * restrict ws, size_t n, const wchar_t *restrict format, va_list ap*);

int
vwprintf(*const wchar_t * restrict format, va_list ap*);

DESCRIPTION

The **wprintf**() family of functions produces output according to a *format* as described below. The **wprintf**() and **vwprintf**() functions write output to stdout, the standard output stream; **fwprintf**() and **vfwprintf**() write output to the given output *stream*; **swprintf**() and **vswprintf**() write to the wide character string *ws*.

These functions write the output under the control of a *format* string that specifies how subsequent arguments (or arguments accessed via the variable-length argument facilities of stdarg(3)) are converted for output.

These functions return the number of characters printed (not including the trailing '\0' used to end output to strings).

The **swprintf**() and **vswprintf**() functions will fail if *n* or more wide characters were requested to be written,

The format string is composed of zero or more directives: ordinary characters (not **%**), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the **%** character. The arguments must correspond properly (after type promotion) with the conversion specifier. After the **%**, the following appear in sequence:

- An optional field, consisting of a decimal digit string followed by a **$**, specifying the next argument to access. If this field is not provided, the argument following the last argument accessed will be used. Arguments are numbered starting at **1**. If unaccessed arguments in the format string are interspersed with ones that are accessed the results will be indeterminate.

- Zero or more of the following flags:

  '#' The value should be converted to an "alternate form". For **c**, **d**, **i**, **n**, **p**, **s**, and **u** conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero (except if a zero value is printed with an explicit precision of zero). For **x** and **X** conversions, a non-zero result has the string '0x' (or '0X' for **X** conversions) prepended to it. For **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow it (normally, a decimal point appears in the results of those conversions only if a digit follows). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.

  '**0**' (zero) Zero padding. For all conversions except **n**, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion (**d**, **i**, **o**, **u**, **i**, **x**, and **X**), the **0** flag is ignored.

  '**-**' A negative field width flag; the converted value is to be left adjusted on the field boundary. Except for **n** conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A **-** overrides a **0** if both are given.

  ' ' (space) A blank should be left before a positive number produced by a signed conversion (**a**, **A**, **d**, **e**, **E**, **f**, **F**, **g**, **G**, or **i**).

‘+’ A sign must always be placed before a number produced by a signed conversion. A **+** overrides a space if both are used.

‘’’ Decimal conversions (**d**, **u**, or **i**) or the integral portion of a floating point conversion (**f** or **F**) should be grouped and separated by thousands using the non-monetary separator returned by localeconv(3).

- 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 with spaces on the left (or right, if the left-adjustment flag has been given) to fill out the field width.

- An optional precision, in the form of a period **.** followed by an optional digit string. If the digit string is omitted, the precision is taken as zero. This gives the minimum number of digits to appear for **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal-point for **a**, **A**, **e**, **E**, **f**, and **F** conversions, the maximum number of significant digits for **g** and **G** conversions, or the maximum number of characters to be printed from a string for **s** conversions.

- An optional length modifier, that specifies the size of the argument. The following length modifiers are valid for the **d**, **i**, **n**, **o**, **u**, **x**, or **X** conversion:

Modifier	**d**, **i**	**o**, **u**, **x**, **X**	**n**
hh	*signed char*	*unsigned char*	*signed char **
h	*short*	*unsigned short*	*short **
l (ell)	*long*	*unsigned long*	*long **
ll (ell ell)	*long long*	*unsigned long long*	*long long **
j	*intmax_t*	*uintmax_t*	*intmax_t **
t	*ptrdiff_t*	(see note)	*ptrdiff_t **
z	(see note)	*size_t*	(see note)
q *(deprecated)*	*quad_t*	*u_quad_t*	*quad_t **

Note: the **t** modifier, when applied to a **o**, **u**, **x**, or **X** conversion, indicates that the argument is of an unsigned type equivalent in size to a *ptrdiff_t*. The **z** modifier, when applied to a **d** or **i** conversion, indicates that the argument is of a signed type equivalent in size to a *size_t*. Similarly, when applied to an **n** conversion, it indicates that the argument is a pointer to a signed type equivalent in size to a *size_t*.

The following length modifier is valid for the **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion:

Modifier	**a**, **A**, **e**, **E**, **f**, **F**, **g**, **G**
L	*long double*

The following length modifier is valid for the **c** or **s** conversion:

Modifier	**c**	**s**
l (ell)	*wint_t*	*wchar_t* *

- A character that specifies the type of conversion to be applied.

A field width or precision, or both, may be indicated by an asterisk '*' or an asterisk followed by one or more decimal digits and a '$' instead of a digit string. In this case, an *int* argument supplies the field width or precision. A negative field width is treated as a left adjustment flag followed by a positive field width; a negative precision is treated as though it were missing. If a single format directive mixes positional (nn$) and non-positional arguments, the results are undefined.

The conversion specifiers and their meanings are:

diouxX The *int* (or appropriate variant) argument is converted to signed decimal (**d** and **i**), unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal (**x** and **X**) notation. The letters "abcdef" are used for **x** conversions; the letters "ABCDEF" are used for **X** conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.

DOU The *long int* argument is converted to signed decimal, unsigned octal, or unsigned decimal, as if the format had been **ld**, **lo**, or **lu** respectively. These conversion characters are deprecated, and will eventually disappear.

eE The *double* argument is rounded and converted in the style [-]*d*.*ddd*e+-*dd* where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An **E** conversion uses the letter 'E' (rather than 'e') to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00.

For **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversions, positive and negative infinity are represented as inf and -inf respectively when using the lowercase conversion character, and INF and -INF respectively when using the uppercase conversion character. Similarly, NaN is represented as nan when using the lowercase conversion, and NAN when using the uppercase conversion.

fF The *double* argument is rounded and converted to decimal notation in the style [-]*ddd*.*ddd*, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no

decimal-point character appears. If a decimal point appears, at least one digit appears before it.

gG The *double* argument is converted in style **f** or **e** (or **F** or **E** for **G** conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style **e** is used if the exponent from its conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.

aA The *double* argument is converted to hexadecimal notation in the style [-]0x*h.hhhp*[+-]*d*, where the number of digits after the hexadecimal-point character is equal to the precision specification. If the precision is missing, it is taken as enough to exactly represent the floating-point number; if the precision is explicitly zero, no hexadecimal-point character appears. This is an exact conversion of the mantissa+exponent internal floating point representation; the [-]0x*h.hhh* portion represents exactly the mantissa; only denormalized mantissas have a zero value to the left of the hexadecimal point. The **p** is a literal character ‘p’; the exponent is preceded by a positive or negative sign and is represented in decimal, using only enough characters to represent the exponent. The **A** conversion uses the prefix "0X" (rather than "0x"), the letters "ABCDEF" (rather than "abcdef") to represent the hex digits, and the letter ‘P’ (rather than ‘p’) to separate the mantissa and exponent.

C Treated as **c** with the **l** (ell) modifier.

c The *int* argument is converted to an *unsigned char*, then to a *wchar_t* as if by btowc(3), and the resulting character is written.

If the **l** (ell) modifier is used, the *wint_t* argument is converted to a *wchar_t* and written.

S Treated as **s** with the **l** (ell) modifier.

s The *char* * argument is expected to be a pointer to an array of character type (pointer to a string) containing a multibyte sequence. Characters from the array are converted to wide characters and written up to (but not including) a terminating NUL character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating NUL character.

If the **l** (ell) modifier is used, the *wchar_t* * argument is expected to be a pointer to an array of wide characters (pointer to a wide string). Each wide character in the string is written. Wide characters from the array are written up to (but not including) a terminating wide NUL character; if a precision is specified, no more than the number specified are written (including

shift sequences). If a precision is given, no null character need be present; if the precision is not specified, or is greater than the number of characters in the string, the array must contain a terminating wide NUL character.

p The *void* * pointer argument is printed in hexadecimal (as if by '%#x' or '%#lx').

n The number of characters written so far is stored into the integer indicated by the *int* * (or variant) pointer argument. No argument is converted.

% A '%' is written. No argument is converted. The complete conversion specification is '%%'.

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

In no case does a non-existent or small field width cause truncation of a numeric field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

SEE ALSO

btowc(3), fputws(3), printf(3), putwc(3), setlocale(3), wcsrtombs(3), wscanf(3)

STANDARDS

Subject to the caveats noted in the *BUGS* section of printf(3), the **wprintf**(), **fwprintf**(), **swprintf**(), **vwprintf**(), **vfwprintf**() and **vswprintf**() functions conform to ISO/IEC 9899:1999 ("ISO C99").

SECURITY CONSIDERATIONS

Refer to printf(3).