7.21.6 Formatted input/output functions

1 The formatted input/output functions shall behave as if there is a sequence point after the actions associated with each specifier.[274)]

7.21.6.1 The `fprintf` function

Synopsis

1
```
#include <stdio.h>
int fprintf(FILE * restrict stream,
     const char * restrict format, ...);
```

Description

2 The **`fprintf`** function writes output to the stream pointed to by **`stream`**, under control of the string pointed to by **`format`** that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored. The **`fprintf`** function returns when the end of the format string is encountered.

3 The format shall be a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: ordinary multibyte characters (not `%`), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments, converting them, if applicable, according to the corresponding conversion specifier, and then writing the result to the output stream.

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

— Zero or more *flags* (in any order) that modify the meaning of the conversion specification.

— An optional minimum *field width*. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk **`*`** (described later) or a nonnegative decimal integer.[275)]

— An optional *precision* that gives the minimum number of digits to appear for the **`d`**, **`i`**, **`o`**, **`u`**, **`x`**, and **`X`** conversions, the number of digits to appear after the decimal-point character for **`a`**, **`A`**, **`e`**, **`E`**, **`f`**, and **`F`** conversions, the maximum number of significant digits for the **`g`** and **`G`** conversions, or the maximum number of bytes to be written for

274) The **`fprintf`** functions perform writes to memory for the **`%n`** specifier.

275) Note that `0` is taken as a flag, not as the beginning of a field width.

s conversions. The precision takes the form of a period (**.**) followed either by an asterisk ***** (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.

— An optional *length modifier* that specifies the size of the argument.

— A *conversion specifier* character that specifies the type of conversion to be applied.

5 As noted above, a field width, or precision, or both, may be indicated by an asterisk. In this case, an **int** argument supplies the field width or precision. The arguments specifying field width, or precision, or both, shall appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a **-** flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

6 The flag characters and their meanings are:

- The result of the conversion is left-justified within the field. (It is right-justified if this flag is not specified.)

+ The result of a signed conversion always begins with a plus or minus sign. (It begins with a sign only when a negative value is converted if this flag is not specified.)[276)]

space If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space is prefixed to the result. If the *space* and **+** flags both appear, the *space* flag is ignored.

The result is converted to an ''alternative form''. For **o** conversion, it increases the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both 0, a single 0 is printed). For **x** (or **X**) conversion, a nonzero result has **0x** (or **0X**) prefixed to it. For **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversions, the result of converting a floating-point number always contains a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if a digit follows it.) For **g** and **G** conversions, trailing zeros are *not* removed from the result. For other conversions, the behavior is undefined.

0 For **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversions, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the **0** and **-** flags both appear, the **0** flag is ignored. For **d**, **i**, **o**, **u**, **x**, and **X**

276) The results of all floating conversions of a negative zero, and of negative values that round to zero, include a minus sign.

conversions, if a precision is specified, the **0** flag is ignored. For other conversions, the behavior is undefined.

7 The length modifiers and their meanings are:

hh Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **signed char** or **unsigned char** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **signed char** or **unsigned char** before printing); or that a following **n** conversion specifier applies to a pointer to a **signed char** argument.

h Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **short int** or **unsigned short int** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **short int** or **unsigned short int** before printing); or that a following **n** conversion specifier applies to a pointer to a **short int** argument.

l (ell) Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long int** or **unsigned long int** argument; that a following **n** conversion specifier applies to a pointer to a **long int** argument; that a following **c** conversion specifier applies to a **wint_t** argument; that a following **s** conversion specifier applies to a pointer to a **wchar_t** argument; or has no effect on a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier.

ll (ell-ell) Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long long int** or **unsigned long long int** argument; or that a following **n** conversion specifier applies to a pointer to a **long long int** argument.

j Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to an **intmax_t** or **uintmax_t** argument; or that a following **n** conversion specifier applies to a pointer to an **intmax_t** argument.

z Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **size_t** or the corresponding signed integer type argument; or that a following **n** conversion specifier applies to a pointer to a signed integer type corresponding to **size_t** argument.

t Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **ptrdiff_t** or the corresponding unsigned integer type argument; or that a following **n** conversion specifier applies to a pointer to a **ptrdiff_t** argument.

L Specifies that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier applies to a **long double** argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

8 The conversion specifiers and their meanings are:

d,i The **int** argument is converted to signed decimal in the style *[-]dddd*. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

o,u,x,X The **unsigned int** argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** or **X**) in the style *dddd*; 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 is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

f,F A **double** argument representing a floating-point number is 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 zero and the **#** flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

A **double** argument representing an infinity is converted in one of the styles *[-]***inf** or *[-]***infinity** — which style is implementation-defined. A **double** argument representing a NaN is converted in one of the styles *[-]***nan** or *[-]***nan(***n-char-sequence***)** — which style, and the meaning of any *n-char-sequence*, is implementation-defined. The **F** conversion specifier produces **INF**, **INFINITY**, or **NAN** instead of **inf**, **infinity**, or **nan**, respectively.[277)]

e,E A **double** argument representing a floating-point number is converted in the style *[-]d***.***ddd* **e**±*dd*, where there is one digit (which is nonzero if the argument is nonzero) 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

277) When applied to infinite and NaN values, the **-**, **+**, and *space* flag characters have their usual meaning; the **#** and **0** flag characters have no effect.

6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The **E** conversion specifier produces a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits, and only as many more digits as necessary to represent the exponent. If the value is zero, the exponent is zero.

A **double** argument representing an infinity or NaN is converted in the style of an **f** or **F** conversion specifier.

g,G A **double** argument representing a floating-point number is converted in style **f** or **e** (or in style **F** or **E** in the case of a **G** conversion specifier), depending on the value converted and the precision. Let P equal the precision if nonzero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style **E** would have an exponent of X:

— if $P > X \geq -4$, the conversion is with style **f** (or **F**) and precision $P - (X + 1)$.

— otherwise, the conversion is with style **e** (or **E**) and precision $P - 1$.

Finally, unless the **#** flag is used, any trailing zeros are removed from the fractional portion of the result and the decimal-point character is removed if there is no fractional portion remaining.

A **double** argument representing an infinity or NaN is converted in the style of an **f** or **F** conversion specifier.

a,A A **double** argument representing a floating-point number is converted in the style *[-]***0x***h.hhhh***p±***d*, where there is one hexadecimal digit (which is nonzero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point character[278)] and the number of hexadecimal digits after it is equal to the precision; if the precision is missing and **FLT_RADIX** is a power of 2, then the precision is sufficient for an exact representation of the value; if the precision is missing and **FLT_RADIX** is not a power of 2, then the precision is sufficient to

278) Binary implementations can choose the hexadecimal digit to the left of the decimal-point character so that subsequent digits align to nibble (4-bit) boundaries.

distinguish[279] values of type **double**, except that trailing zeros may be omitted; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. The letters **abcdef** are used for **a** conversion and the letters **ABCDEF** for **A** conversion. The **A** conversion specifier produces a number with **X** and **P** instead of **x** and **p**. The exponent always contains at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent is zero.

A **double** argument representing an infinity or NaN is converted in the style of an **f** or **F** conversion specifier.

c If no **l** length modifier is present, the **int** argument is converted to an **unsigned char**, and the resulting character is written.

If an **l** length modifier is present, the **wint_t** argument is converted as if by an **ls** conversion specification with no precision and an argument that points to the initial element of a two-element array of **wchar_t**, the first element containing the **wint_t** argument to the **lc** conversion specification and the second a null wide character.

s If no **l** length modifier is present, the argument shall be a pointer to the initial element of an array of character type.[280] Characters from the array are written up to (but not including) the terminating null character. If the precision is specified, no more than that many bytes are written. If the precision is not specified or is greater than the size of the array, the array shall contain a null character.

If an **l** length modifier is present, the argument shall be a pointer to the initial element of an array of **wchar_t** type. Wide characters from the array are converted to multibyte characters (each as if by a call to the **wcrtomb** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting multibyte characters are written up to (but not including) the terminating null character (byte). If no precision is specified, the array shall contain a null wide character. If a precision is specified, no more than that many bytes are written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the multibyte character sequence length given by

---

279) The precision p is sufficient to distinguish values of the source type if $16^{p-1} > b^n$ where b is **FLT_RADIX** and n is the number of base-b digits in the significand of the source type. A smaller p might suffice depending on the implementation's scheme for determining the digit to the left of the decimal-point character.

280) No special provisions are made for multibyte characters.

the precision, the function would need to access a wide character one past the end of the array. In no case is a partial multibyte character written.[281)]

p The argument shall be a pointer to **void**. The value of the pointer is converted to a sequence of printing characters, in an implementation-defined manner.

n The argument shall be a pointer to signed integer into which is *written* the number of characters written to the output stream so far by this call to **fprintf**. No argument is converted, but one is consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.

% A **%** character is written. No argument is converted. The complete conversion specification shall be **%%**.

9 If a conversion specification is invalid, the behavior is undefined.[282)] If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.

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

11 For **a** and **A** conversions, if **FLT_RADIX** is a power of 2, the value is correctly rounded to a hexadecimal floating number with the given precision.

Recommended practice

12 For **a** and **A** conversions, if **FLT_RADIX** is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

13 For **e**, **E**, **f**, **F**, **g**, and **G** conversions, if the number of significant decimal digits is at most **DECIMAL_DIG**, then the result should be correctly rounded.[283)] If the number of significant decimal digits is more than **DECIMAL_DIG** but the source value is exactly representable with **DECIMAL_DIG** digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having **DECIMAL_DIG** significant digits; the value

---

281) Redundant shift sequences may result if multibyte characters have a state-dependent encoding.

282) See ''future library directions'' (7.31.11).

283) For binary-to-decimal conversion, the result format's values are the numbers representable with the given format specifier. The number of significant digits is determined by the format specifier, and in the case of fixed-point conversion by the source value as well.

of the resultant decimal string D should satisfy $L \le D \le U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

Returns

14 The **fprintf** function returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

Environmental limits

15 The number of characters that can be produced by any single conversion shall be at least 4095.

16 EXAMPLE 1 To print a date and time in the form "Sunday, July 3, 10:02" followed by π to five decimal places:

```
#include <math.h>
#include <stdio.h>
/* ... */
char *weekday, *month;      // pointers to strings
int day, hour, min;
fprintf(stdout, "%s, %s %d, %.2d:%.2d\n",
       weekday, month, day, hour, min);
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```

17 EXAMPLE 2 In this example, multibyte characters do not have a state-dependent encoding, and the members of the extended character set that consist of more than one byte each consist of exactly two bytes, the first of which is denoted here by a □ and the second by an uppercase letter.

18 Given the following wide string with length seven,

```
static wchar_t wstr[] = L"□X□Yabc□Z□W";
```

the seven calls

```
fprintf(stdout, "|1234567890123|\n");
fprintf(stdout, "|%13ls|\n", wstr);
fprintf(stdout, "|%-13.9ls|\n", wstr);
fprintf(stdout, "|%13.10ls|\n", wstr);
fprintf(stdout, "|%13.11ls|\n", wstr);
fprintf(stdout, "|%13.15ls|\n", &wstr[2]);
fprintf(stdout, "|%13lc|\n", (wint_t) wstr[5]);
```

will print the following seven lines:

```
|1234567890123|
|  □X□Yabc□Z□W|
|□X□Yabc□Z    |
|    □X□Yabc□Z|
|  □X□Yabc□Z□W|
|      abc□Z□W|
|           □Z|
```

Forward references: conversion state (7.29.6), the **wcrtomb** function (7.29.6.3.3).