Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'fprintf.3p' command

$ man fprintf.3p

FPRINTF(3P) POSIX Programmer's Manual FPRINTF(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dprintf, fprintf, printf, snprintf, sprintf ? print formatted output

SYNOPSIS

```
#include <stdio.h>
int dprintf(int fildes, const char *restrict format, ...);
int fprintf(FILE *restrict stream, const char *restrict format, ...);
int printf(const char *restrict format, ...);
int snprintf(char *restrict s, size_t n,
    const char *restrict format, ...);
int sprintf(char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

Excluding dprintf(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the require? ments described here and the ISO C standard is unintentional. This vol? ume of POSIX.1?2017 defers to the ISO C standard.

The fprintf() function shall place output on the named output stream.

The printf() function shall place output on the standard output stream

stdout. The sprintf() function shall place output followed by the null byte, '\0', in consecutive bytes starting at *s; it is the user's re? sponsibility to ensure that enough space is available.

The dprintf() function shall be equivalent to the fprintf() function, except that dprintf() shall write output to the file associated with the file descriptor specified by the fildes argument rather than place output on a stream.

The snprintf() function shall be equivalent to sprintf(), with the ad? dition of the n argument which states the size of the buffer referred to by s. If n is zero, nothing shall be written and s may be a null pointer. Otherwise, output bytes beyond the n?1st shall be discarded instead of being written to the array, and a null byte is written at the end of the bytes actually written into the array.

If copying takes place between objects that overlap as a result of a call to sprintf() or snprintf(), the results are undefined.

Each of these functions converts, formats, and prints its arguments un? der control of the format. The format is a character string, beginning and ending in its initial shift state, if any. The format is composed of zero or more directives: ordinary characters, which are simply copied to the output stream, and conversion specifications, each of which shall result in the fetching of zero or more arguments. The re? sults are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but are otherwise ignored.

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion specifier character % (see below) is replaced by the se? quence "%n$", where n is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The format can contain either numbered argument conversion specifica?

tions (that is, "%n$" and "*m$"), or unnumbered argument conversion specifications (that is, % and *), but not both. The only exception to this is that %% can be mixed with the "%n$" form. The results of mixing numbered and unnumbered argument specifications in a format string are undefined. When numbered argument specifications are used, specifying the Nth argument requires that all the leading arguments, from the first to the (N-1)th, are specified in the format string.

In format strings containing the "%n$" form of conversion specifica? tion, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the % form of conversion specification, each conversion specification uses the first unused argument in the ar? gument list.

All forms of the fprintf() functions allow for the insertion of a lan? guage-dependent radix character in the output string. The radix charac? ter is defined in the current locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('.').

Each conversion specification is introduced by the '%' character or by the character sequence "%n$", after which the following appear in se? quence:

* Zero or more flags (in any order), which modify the meaning of the conversion specification.
* An optional minimum field width. If the converted value has fewer bytes than the field width, it shall be padded with <space> charac? ters by default on the left; it shall be padded on the right if the left-adjustment flag ('-'), described below, is given to the field width. The field width takes the form of an <asterisk> ('*'), de? scribed below, or a decimal integer.
* An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the num? ber of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of signifi?

cant digits for the g and G conversion specifiers; or the maximum number of bytes to be printed from a string in the s and S conver? sion specifiers. The precision takes the form of a <period> ('.') followed either by an <asterisk> ('*'), described below, or an op? tional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion speci? fier, the behavior is undefined.

* An optional length modifier that specifies the size of the argu? ment.

* A conversion specifier character that indicates the type of conver? sion to be applied.

A field width, or precision, or both, may be indicated by an <asterisk> ('*'). In this case an argument of type int supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the "%n$" form of a conversion specification, a field width or precision may be indicated by the sequence "*m$", where m is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);

The flag characters and their meanings are:

' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For other con? versions the behavior is undefined. The non-monetary grouping character is used.

- The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.

+ The result of a signed conversion shall always begin with a sign ('+' or '-'). The conversion shall begin with a sign only when a negative value is converted if this flag is not speci? fied.

<space> If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

Specifies that the value is to be converted to an alternative form. For o conversion, it shall increase 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 specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G con? version specifiers, trailing zeros shall not be removed from the result as they normally are. For other conversion speci? fiers, the behavior is undefined.

0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, 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 conversion specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and <apostrophe> flags both appear, the grouping characters are in? serted before zero padding. For other conversions, the behavior is undefined.

The length modifiers and their meanings are:

hh Specifies that a following d, i, o, u, x, or X conversion spec?

ifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer pro? motions, but its value shall be converted to signed char or un? signed 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 spec? ifier applies to a short or unsigned short argument (the argu? ment will have been promoted according to the integer promo? tions, but its value shall be converted to short or unsigned short before printing); or that a following n conversion speci? fier applies to a pointer to a short argument.

l (ell) Specifies that a following d, i, o, u, x, or X conversion spec? ifier applies to a long or unsigned long argument; that a fol? lowing n conversion specifier applies to a pointer to a long argument; that a following c conversion specifier applies to a wint_t argument; that a following s conversion specifier ap? plies 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 spec? ifier applies to a long long or unsigned long long argument; or that a following n conversion specifier applies to a pointer to a long long argument.

j Specifies that a following d, i, o, u, x, or X conversion spec? ifier 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 spec? ifier applies to a size_t or the corresponding signed integer type argument; or that a following n conversion specifier ap? plies to a pointer to a signed integer type corresponding to a size_t argument.

t Specifies that a following d, i, o, u, x, or X conversion spec?

ifier applies to a ptrdiff_t or the corresponding unsigned 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.

The conversion specifiers and their meanings are:

d, i The int argument shall be converted to a signed decimal in the style "[-]dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be repre? sented in fewer digits, it shall be expanded with leading ze? ros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

o The unsigned argument shall be converted to unsigned octal for? mat 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 shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

u The unsigned argument shall be converted to unsigned decimal format 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 shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

x The unsigned argument shall be converted to unsigned hexadeci? mal format in the style "dddd"; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer dig? its, it shall be expanded with leading zeros. The default pre? cision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

X Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".

f, F The double argument shall be converted to decimal notation in the style "[-]ddd.ddd", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix charac? ter shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.

A double argument representing an infinity shall be converted in one of the styles "[-]inf" or "[-]infinity"; which style is implementation-defined. A double argument representing a NaN shall be converted in one of the styles "[-]nan(n-char-se? quence)" or "[-]nan"; 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.

e, E The double argument shall be converted in the style "[-]d.ddde?dd", where there is one digit before the radix char? acter (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall appear. The low-order digit shall be rounded in an implementa? tion-defined manner. The E conversion specifier shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.

A double argument representing an infinity or NaN shall be con? verted in the style of an f or F conversion specifier.

g, G The double argument representing a floating-point number shall be converted in the style f or e (or in the style F or E in the

case of a G conversion specifier), depending on the value con?
verted and the precision. Let P equal the precision if non-
zero, 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?-4, the conversion shall be with style f (or F) and
precision P-(X+1).

-- Otherwise, the conversion shall be with style e (or E) and
precision P-1.

Finally, unless the '#' flag is used, any trailing zeros shall
be removed from the fractional portion of the result and the
decimal-point character shall be removed if there is no frac?
tional portion remaining.

A double argument representing an infinity or NaN shall be con?
verted in the style of an f or F conversion specifier.

a, A A double argument representing a floating-point number shall be
converted in the style "[-]0xh.hhhhp?d", where there is one
hexadecimal digit (which shall be non-zero if the argument is a
normalized floating-point number and is otherwise unspecified)
before the decimal-point character and the number of hexadeci?
mal digits after it is equal to the precision; if the precision
is missing and FLT_RADIX is a power of 2, then the precision
shall be 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 shall be sufficient to distinguish values of
type double, except that trailing zeros may be omitted; if the
precision is zero and the '#' flag is not specified, no deci?
mal-point character shall appear. The letters "abcdef" shall be
used for a conversion and the letters "ABCDEF" for A conver?
sion. The A conversion specifier produces a number with 'X' and
'P' instead of 'x' and 'p'. The exponent shall always contain
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 shall be zero.

A double argument representing an infinity or NaN shall be con? verted in the style of an f or F conversion specifier.

c The int argument shall be converted to an unsigned char, and the resulting byte shall be written.

If an l (ell) qualifier is present, the wint_t argument shall be converted as if by an ls conversion specification with no precision and an argument that points to a two-element array of type wchar_t, the first element of which contains the wint_t argument to the ls conversion specification and the second ele? ment contains a null wide character.

s The argument shall be a pointer to an array of char. Bytes from the array shall be written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the appli? cation shall ensure that the array contains a null byte.

If an l (ell) qualifier is present, the argument shall be a pointer to an array of type wchar_t. Wide characters from the array shall be converted to 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 characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written.

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

n The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the fprintf() functions. No argument is converted.

C Equivalent to lc.

S Equivalent to ls.

% Print a '%' character; no argument is converted. The complete conversion specification shall be %%.

If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined. In no case shall 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 shall be expanded to contain the conversion result. Charac? ters generated by fprintf() and printf() are printed as if fputc() had been called.

For the a and A conversion specifiers, if FLT_RADIX is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.

For a and A conversions, if FLT_RADIX is not a power of 2 and the re? sult 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.

For the e, E, f, F, g, and G conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal dig? its is more than DECIMAL_DIG but the source value is exactly repre? sentable 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 of the resultant decimal string D should satisfy L <= D <= U, with the extra stipulation that the error should have a correct sign for the current rounding direction.

The last data modification and last file status change timestamps of the file shall be marked for update:

1. Between the call to a successful execution of fprintf() or printf() and the next successful completion of a call to fflush() or fclose() on the same stream or a call to exit() or abort()
2. Upon successful completion of a call to dprintf()

RETURN VALUE

Upon successful completion, the dprintf(), fprintf(), and printf() functions shall return the number of bytes transmitted.

Upon successful completion, the sprintf() function shall return the number of bytes written to s, excluding the terminating null byte.

Upon successful completion, the snprintf() function shall return the number of bytes that would be written to s had n been sufficiently large excluding the terminating null byte.

If an output error was encountered, these functions shall return a neg? ative value and set errno to indicate the error.

If the value of n is zero on a call to snprintf(), nothing shall be written, the number of bytes that would have been written had n been sufficiently large excluding the terminating null shall be returned, and s may be a null pointer.

ERRORS

For the conditions under which dprintf(), fprintf(), and printf() fail and may fail, refer to fputc() or fputwc().

In addition, all forms of fprintf() shall fail if:

EILSEQ A wide-character code that does not correspond to a valid char? acter has been detected.

EOVERFLOW

The value to be returned is greater than {INT_MAX}.

The dprintf() function may fail if:

EBADF The fildes argument is not a valid file descriptor.

The dprintf(), fprintf(), and printf() functions may fail if:

ENOMEM Insufficient storage space is available.

The snprintf() function shall fail if:

EOVERFLOW

The value of n is greater than {INT_MAX}.

The following sections are informative.

EXAMPLES

Printing Language-Independent Date and Time

The following statement can be used to print date and time using a lan?

guage-independent format:

```
printf(format, weekday, month, day, hour, min);
```

For American usage, format could be a pointer to the following string:

```
"%s, %s %d, %d:%.2d\n"
```

This example would produce the following message:

```
Sunday, July 3, 10:02
```

For German usage, format could be a pointer to the following string:

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

This definition of format would produce the following message:

```
Sonntag, 3. Juli, 10:02
```

Printing File Information

The following example prints information about the type, permissions, and number of links of a specific file in a directory.

The first two calls to printf() use data decoded from a previous stat() call. The user-defined strperm() function shall return a string similar to the one at the beginning of the output for the following command:

```
ls -l
```

The next call to printf() outputs the owner's name if it is found using getpwuid(); the getpwuid() function shall return a passwd structure from which the name of the user is extracted. If the user name is not found, the program instead prints out the numeric value of the user ID.

The next call prints out the group name if it is found using get?

grgid(); getgrgid() is very similar to getpwuid() except that it shall

return group information based on the group number. Once again, if the group is not found, the program prints the numeric value of the group for the entry.

The final call to printf() prints the size of the file.

```
#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>
char *strperm (mode_t);
...
struct stat statbuf;
struct passwd *pwd;
struct group *grp;
...
printf("%10.10s", strperm (statbuf.st_mode));
printf("%4d", statbuf.st_nlink);
if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
    printf(" %-8.8s", pwd->pw_name);
else
    printf(" %-8ld", (long) statbuf.st_uid);
if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8ld", (long) statbuf.st_gid);
printf("%9jd", (intmax_t) statbuf.st_size);
...
```

Printing a Localized Date String

The following example gets a localized date string. The nl_langinfo() function shall return the localized date string, which specifies the order and layout of the date. The strftime() function takes this infor? mation and, using the tm structure for values, places the date and time information into datestring. The printf() function then outputs dat? estring and the name of the entry.

```
#include <stdio.h>
#include <time.h>
#include <langinfo.h>
...
struct dirent *dp;
struct tm *tm;
char datestring[256];
...
strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
printf(" %s %s\n", datestring, dp->d_name);
...
```

Printing Error Information

The following example uses fprintf() to write error information to standard error.

In the first group of calls, the program tries to open the password lock file named LOCKFILE. If the file already exists, this is an er? ror, as indicated by the O_EXCL flag on the open() function. If the call fails, the program assumes that someone else is updating the pass? word file, and the program exits.

The next group of calls saves a new password file as the current pass? word file by creating a link between LOCKFILE and the new password file PASSWDFILE.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
...
```

```
int pfd;
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
if (link(LOCKFILE,PASSWDFILE) == -1) {
    fprintf(stderr, "Link error: %s\n", strerror(errno));
    exit(1);
}
...
```

Printing Usage Information

The following example checks to make sure the program has the necessary arguments, and uses fprintf() to print usage information if the expected number of arguments is not present.

```
#include <stdio.h>
#include <stdlib.h>
...
char *Options = "hdbtl";
...
if (argc < 2) {
    fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
}
...
```

Formatting a Decimal String

The following example prints a key and data pair on stdout. Note use of the <asterisk> ('*') in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

```
#include <stdio.h>
```

```
...
long i;
char *keystr;
int elementlen, len;
...
while (len < elementlen) {
...
    printf("%s Element%0*ld\n", keystr, elementlen, i);
...
}
```

Creating a Pathname

The following example creates a pathname using information from a previous getpwnam() function that returned the password database entry of the user.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
...
char *pathname;
struct passwd *pw;
size_t len;
...
// digits required for pid_t is number of bits times
// log2(10) = approx 10/33
len = strlen(pw->pw_dir) + 1 + 1+(sizeof(pid_t)*80+32)/33 +
    sizeof ".out";
pathname = malloc(len);
if (pathname != NULL)
{
    snprintf(pathname, len, "%s/%jd.out", pw->pw_dir,
```

```
            (intmax_t)getpid());
        ...
    }
```

Reporting an Event

The following example loops until an event has timed out. The pause() function waits forever unless it receives a signal. The fprintf() statement should never occur due to the possible return values of pause().

```
    #include <stdio.h>
    #include <unistd.h>
    #include <string.h>
    #include <errno.h>
    ...
    while (!event_complete) {
    ...
        if (pause() != -1 || errno != EINTR)
            fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
    }
    ...
```

Printing Monetary Information

The following example uses strfmon() to convert a number and store it as a formatted monetary string named convbuf. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

```
    #include <monetary.h>
    #include <stdio.h>
    ...
    struct tblfmt {
        char *format;
        char *description;
    };
    struct tblfmt table[] = {
        { "%n", "default formatting" },
```

```
        { "%11n", "right align within an 11 character field" },
        { "%#5n", "aligned columns for values up to 99999" },
        { "%=*#5n", "specify a fill character" },
        { "%=0#5n", "fill characters do not use grouping" },
        { "%^#5n", "disable the grouping separator" },
        { "%^#5.0n", "round off to whole units" },
        { "%^#5.4n", "increase the precision" },
        { "%(#5n", "use an alternative pos/neg style" },
        { "%!(#5n", "disable the currency symbol" },
    };
    ...
    float input[3];
    int i, j;
    char convbuf[100];
    ...
    strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
    if (j == 0) {
        printf("%s%s%s\n", table[i].format,
            convbuf, table[i].description);
    }
    else {
        printf("%s\n", convbuf);
    }
    ...
```

Printing Wide Characters

The following example prints a series of wide characters. Suppose that "L`@`" expands to three bytes:

```
    wchar_t wz [3] = L"@@";       // Zero-terminated
    wchar_t wn [3] = L"@@@";      // Unterminated
    fprintf (stdout,"%ls", wz);   // Outputs 6 bytes
    fprintf (stdout,"%ls", wn);   // Undefined because wn has no terminator
    fprintf (stdout,"%4ls", wz);  // Outputs 3 bytes
    fprintf (stdout,"%4ls", wn);  // Outputs 3 bytes; no terminator needed
```

```
fprintf (stdout,"%9ls", wz);  // Outputs 6 bytes
fprintf (stdout,"%9ls", wn);  // Outputs 9 bytes; no terminator needed
fprintf (stdout,"%10ls", wz); // Outputs 6 bytes
fprintf (stdout,"%10ls", wn); // Undefined because wn has no terminator
```

In the last line of the example, after processing three characters, nine bytes have been output. The fourth character must then be examined to determine whether it converts to one byte or more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth character in the array, the behavior is undefined.

APPLICATION USAGE

If the application calling fprintf() has any objects of type wint_t or wchar_t, it must also include the <wchar.h> header to have these ob? jects defined.

RATIONALE

If an implementation detects that there are insufficient arguments for the format, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, Standard I/O Streams, fputc(), fscanf(), setlocale(), strfmon(), wcrtomb()

The Base Definitions volume of POSIX.1?2017, Chapter 7, Locale, <int? types.h>, <stdio.h>, <wchar.h>

at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .