IEEE P1003.2 Draft 11.2 – September 1991

Unresolved Objections

These subclauses were removed from Draft 10. The Unresolved Objections are maintained in a separate list and do not meet ISO editing requirements for an informative annex.

2.11.10.1 Rationale Rationale. *(This subclause is not a part of P1003.2)*

The Rationale subclauses will be moved to Annex E in the final POSIX.2. Some of the subheadings may be collapsed in that document; in these drafts the working group has not always been very rigorous about what is a description of usage versus a history of decisions made, for example. The final rationale will de-emphasize the chronological aspects of working group decisions.

2.12 File Format Notation

The Standard Input, Standard Output, Standard Error, Input Files, and Output Files subclauses of the utility descriptions, when provided, use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the C language *printf* () function, as described in this clause. When used in Standard Input or Input Files subclauses of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the C language *scanf* () function to read the input file.

The description of an individual record is as follows:

"*<format >*", [*<arg1>*, *<arg2>*, . . . , *<argn>*]

The *format* is a character string that contains three types of objects defined below:

characters Characters that are not *escape sequences* or *conversion specifications* , as described below, shall be copied to the output.

escape sequences
Represent nongraphic characters.

conversion specifications
Specifies the output format of each argument. (See below.)

The following characters have the following special meaning in the format string:

" " (An empty character position.) One or more `<blank>` characters.

Δ Exactly one `<space>` character.

The escape-sequences in Table 2-15 depict the associated action on display devices capable of the action.

Each conversion specification shall be introduced by the percent-sign character (%). After the character %, the following shall appear in sequence:

Table 2-15 – Escape Sequences

Escape Sequence	Represents Character	Terminal Action
`\\`	backslash	None.
`\a`	`<alert>`	Attempts to alert the user through audible or visible notification.
`\b`	`<backspace>`	Moves the printing position to one column before the current position, unless the current position is the start of a line.
`\f`	`<form-feed>`	Moves the printing position to the initial printing position of the next logical page.
`\n`	`<newline>`	Moves the printing position to the start of the next line.
`\r`	`<carriage-return>`	Moves the printing position to the start of the current line.
`\t`	`<tab>`	Moves the printing position to the next tab position on the current line. If there are no more tab positions left on the line, the behavior is undefined.
`\v`	`<vertical tab>`	Moves the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

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

field width An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left [or right, if the left-adjustment flag (−), described below, has been given] to the field width.

precision Gives the minimum number of digits to appear for the `d`, `o`, `i`, `u`, `x`, or `X` conversions (the field shall be padded with leading zeros), the number of digits to appear after the radix character for the `e` and `f` conversions, the maximum number of significant digits for the `g` conversion; or the maximum number of bytes to be written from a string in `s` conversion. The precision shall take the form of a period (`.`) followed by a decimal digit string; a null digit string shall be treated as zero.

conversion characters
A conversion character (see below) that indicates the type of conversion to be applied.

The *flag* characters and their meanings are:

− The result of the conversion shall be left-justified within the field.

\+ The result of a signed conversion always shall begin with a sign (+ or −).

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and + flags both appear, the <space> flag shall be ignored.

\# The value is to be converted to an ‘‘alternate form.’’ For c, d, i, u, and s conversions, the behavior is undefined. For o conversion, it shall increase the precision to force the first digit of the result to be a zero. For x or X conversion, a nonzero result shall have 0x or 0X prefixed to it, respectively. For e, E, f, g and G conversions, the result shall always contain a radix character, even if no digits follow the radix character. For g and G conversions, trailing zeroes shall not be removed from the result as they usually are.

0 For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeroes (following any indication of sign or base) shall be used to pad to the field width; no space padding shall be performed. If the 0 and – flags both appear, the 0 flag shall be ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag shall be ignored. For other conversions, the behavior is undefined.

Each conversion character shall result in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be ignored.

The *conversion characters* and their meanings are:

d,i,o,u,x,X The integer argument shall be written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers shall convert to signed decimal in the style [–]*dddd* . The x conversion shall use the numbers and letters 0123456789abcdef and the X conversion shall use the numbers and letters 0123456789ABCDEF. The *precision* component of the argument shall specify the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it shall be expanded with leading zeroes. The default precision shall be 1. The result of converting a zero value with a precision of 0 shall be no characters. If both the field width and precision are omitted, the implementation may precede and/or follow numeric arguments of types d, i, and u with <blank>s; arguments of type o (octal) may be preceded with leading zeroes.

f The floating point number argument shall be written in decimal notation in the style "[–]*ddd* .*ddd* ", where the number of digits after the radix character (shown here as a decimal point) shall be equal to the *precision* specification. The LC_NUMERIC locale category shall determine the radix character to use in this format. If the *precision* is omitted from the argument, six digits shall be written after the radix character; if the *precision* is explicitly 0, no radix character shall appear.

e,E — The floating point number argument shall be written in the style "[−]*d*.*ddd* e±*dd*" (the symbol ± indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The LC_NUMERIC locale category shall determine the radix character to use in this format. When the precision is missing, six digits shall be written after the radix character; if the precision is 0, no radix character shall appear. The E conversion character shall produce a number with E instead of e introducing the exponent. The exponent always shall contain at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits shall be written as necessary.

g,G — The floating point number argument shall be written in style f or e (or in style E in the case of a G conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted: style e shall be used only if the exponent resulting from the conversion is less than −4 or greater than or equal to the precision. Trailing zeroes shall be removed from the result. A radix character shall appear only if it is followed by a digit.

c — The integer argument shall be converted to an *unsigned char* and the resulting byte shall be written.

s — The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the *precision* specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.

% — Write a % character; no argument shall be converted.

In no case does a nonexistent or insufficient *field width* cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be simply expanded to contain the conversion result.

2.12.1 File Format Notation Rationale. *(This subclause is not a part of P1003.2)*

This clause was originally derived from the description of *printf* () in the *SVID* , but it has been updated following the publication of the C Standard {7}. It is not identical to the C Standard's {7}*printf* (), as it deals with integers as being essentially one type, disregarding possible internal differences between *int* , *short* , and *long* . It has also had some of the internal C language dependencies removed (such as the requirement for null-terminated strings).

This standard provides a rigorous description of the format of utility input and output files. It is the intention of this standard that these descriptions be adequate sources of information so that portable applications can use other utilities such as `lex` or `awk` to reliably parse the output of these utilities as their input in, say a pipeline.

The notation for spaces allows some flexibility for application output. Note that an empty character position in *format* represents one or more <blank> characters on the output (not *white space*, which can include <newline>s). Therefore, another utility that reads that output as its input must be prepared to parse the data using *scanf*(), `awk`, etc. The Δ character is used when exactly one <space> is output.

The treatment of integers and spaces is different from the real *printf*(), in that they can be surrounded with <blank>s. This was done so that, given a format such as:

"%d\n", *<foo>*

the implementation could use a real *printf*() such as

```
printf("%6d\n", foo);
```

and still conform. It would have been possible for the standard to use "%6d\n", but it would have been difficult to pick a number that would have pleased everyone. This notation is thus somewhat like *scanf*() in addition to *printf*().

The *printf*() function was chosen as a model as most of the working group was familiar with it and it was thought that many of the readers would be as well.

One difference from the C function *printf*() is that the `l` and `h` conversion characters are not used. As expressed by this standard, there is no differentiation between decimal values for *int* s versus *long* s versus *short* s. The specifications `%d` or `%i` should be interpreted as an arbitrary length sequence of digits. Also, no distinction is made between single precision and double precision numbers (*float/double* in C). These are simply referred to as floating point numbers.

Many of the output descriptions in this standard use the term *line*, such as:

"%s", *<input line>*

Since the definition of *line* includes the trailing <newline> character already, there is no need to include a "\n" in the format; a double <newline> would otherwise result.

In the language at the end of the clause:

> ‘‘In no case does a nonexistent or insufficient *field width* cause truncation of a field; . . . ’’

the term ‘‘field width’’ should not be confused with the term ‘‘precision’’ used in the description of `%s`.

Examples:

To represent the output of a program that prints a date and time in the form `Sunday, July 3, 10:02`, where *<weekday>* and *<month>* are strings:

`"%s,△%s△%d,△%d:%.2d\n"`, *<weekday>* , *<month>* , *<day>*, *<hour>*, *<min>*

To show π written to 5 decimal places:

`"pi△=△%.5f\n"`, *<value of π>*

To show an input file format consisting of five colon-separated fields:

`"%s:%s:%s:%s:%s\n"`, *<arg1>*, *<arg2>*, *<arg3>*, *<arg4>*, *<arg5>*

2.13 Configuration Values

2.13.1 Symbolic Limits

This clause lists magnitude limitations imposed by a specific implementation. The braces notation, {LIMIT}, is used in this standard to indicate these values, but the braces are not part of the name.

Table 2-16 – Utility Limit Minimum Values

Name	Description	Value	
{POSIX2_BC_BASE_MAX}	The maximum *obase* value allowed by the `bc` utility.	99	
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the `bc` utility.	2048	
{POSIX2_BC_SCALE_MAX}	The maximum *scale* value allowed by the `bc` utility.	99	
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the `bc` utility.	1000	
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the LC_COLLATE `order` keyword in the locale definition file; see 2.5.2.2.3.	2	
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the `expr` utility.	32	
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing `<newline>`.	2048	
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a regular expression permitted when using the interval notation `\{`*m*`,` *n*`\}`; see 2.8.3.3.	255	
{POSIX2_VERSION}	This value indicates the version of the utilities in this standard that are provided by the implementation. It will change with each published version of this standard.	199???	1