**Document Number: N3184**

Submitter: Aaron Peter Bachmann

Submission Date: 2023-11-24

# Format Specifiers for Floating-point Numbers

## Summary

Traditionally C has provided format specifiers for all scalar types in C.

Presently a portable format specifier for `_Float128` is missing, as well as for other floating-point types such as `_Float16`, `_Float32`, `_Float32x`, `_Float64`, `_Float64x` and the decimal floating-point types `_DecimalNx`. This paper proposes adding `%wN` format length modifiers for floating-point types modeled after the `%wN` length modifiers for integer types. The paper also proposes to add `%wxN` for `_FloatNx and %wxDN for _DecimalNx`. This shall apply for the printf family as well as the scanf familiy of functions.

## Relevant developments in floating point

There is clearly a trend towards higher precision (and range) and more strictly defined floating-point arithmetic. Floating point is to a great extend converging towards IEEE-754[1].

William Kahan "For now the 10-byte Extended format is a tolerable compromise between the value of extra-precise arithmetic and the price of implementing it to run fast; very soon two more bytes of precision will become tolerable, and ultimately a 16-byte format ... That kind of gradual evolution towards wider precision was already in view when IEEE Standard 754 for Floating-Point Arithmetic was framed."

POWER9, PA-RISC, SPARC, RISC-V have definitions for binary128 instructions. In POWER9 these instructions are actually implemented in real hardware.

IEEE-754 2008 defines binary floating-point formats (also as exchange formats) (and operations on these) for 16, 32, 64, 128 bit and in addition for `N` bit, where `(N%32==0) && (N>128)`.

Several software libraries for binary128 are widely available e. g. John Hausers SoftFloat [1], GNU libquadmath [2], Fabrice Bellards SoftFP library [3] . Matlab [4] supports it, Fortran [5] does. Boost [6] comes with `__float128` and `_Quad`. These are just examples – not intended to be exhaustive.

Hardware support for `_Float16` and `_Bfloat16` is becoming available in newer (as of 2023) general-purpose-hardware and graphics hardware.

In the recent years user accessible extended word-sizes (e. g. 40 bit floats or 80 bit floats or 82 bit floats) for floating point seem to have fallen out of favor (68000 and ITANIUM are mostly gone but x86 is still here) in main-stream computing but are still common in DSPs.

[1] IEEE754-2019 is equivalent to ISO/IEC 60559:2020. As C is an ISO-standard it seems appropriate to refer to ISO/IEC 60559 when it is used in suggestions for normative wording. Elsewhere the paper keeps IEEE 754 since it predates ISO/IEC 60559.

Limited hardware support for binaray128 is possible even without full floating-point support in hardware via floating-point-assists [7].

We can expect more direct hardware support for binary128 in the years to come.

## Shortcomings in C23 and earlier

Converting textual representations of floating-point numbers from/to their binary representation correctly and efficiently – and providing a user-friendly programming interface – is non-trivial. There are quite a number of relatively recent publications covering the topic. It is clearly the purpose of a std-library to provide the functionality desired.

C provides `strfromf()`, `strfromd()`, `strfromencfN()`, `strfromfNx()`, `strtofN()`, `strtofNx()`.

Especially the `strfrom…()` functions are awkward to use for following reasons:

1. The user must compute a conservative but not overly wasteful estimate of the space required. This estimate depends on the format string, the range of the floating-point type used as input and/or the actual range of values possible in the context.
   For a simple `_Float128` this can approach 5 kBytes even when field-with and precision are not given in the format string!
2. Manual resource-management of the memory allocated including error handling is required. It is laborious to write test-code for the error path.
3. Using separate conversion-functions badly interacts with a common extension to the printf family, `%$N`. This is a very common extension. POSIX has it. It helps keeping together what belongs together and it is very useful for internationalization.

Note: using `%e`, `%g` or `%a` but not `%f` simplifies the situation considerably.

## Prior work

- As `long double` was introduced `L` has been introduced as length modifier for `%f`, `%e`, `%g` (note: `%a` came later, but `L` as length modifier exists for `%a` as well)
- As `long long` and `unsigned long long` were introduced `ll` has been introduced as length modifier for `%d`, `%i`, `%u`, `%o` and `%x`.
- As decimal floating-point types were introduced `H`, `D`, `DD` were introduced.
- As ISO/IEC TR 18037 [8] [9] introduced `_Fract` and `_Accum` the new format specifiers `%r`, `%R`, `%k`, `%K` where added.
- GNU libquadmath uses `Q` in `quadmath_snprintf()` and registers `Q` as length-modifier for the printf family. Calling `fabsq(0.0q)` is sufficient to do the registration.
- Platforms using ieee754 binary128 as format for long double support printing binary128 either via `%Ld` or another format specifier, often `%Qf` (e. g. Itanium)
- Wg14 was wise enough to limit the required portable range of `_BitInt(N)`, so that these are easily printed after casting them to `[unsigned] long long`.
- Wg14 has already introduced `%wN` and `%wfN` as length modifiers for integer-types [10].

**Intentionally not covered**

- _Bfloat16 and other formats not part of IEEE 754 are not covered[2]. This also applies to other less common floating-point formats. _Bfloat16 and others can easily be printed after casting to _Float32 without any loss of information. A relative recent C++ draft [11] mentions _Bfloat16. When reading textual representations of _Bfloat16 as _Float64 and converting it to _Bfloat16 afterwards an error due to double rounding is possible, but only if the input has an unreasonable amount of excess precision and the error will be extremely small, less than 1e-12 ULP of the _Bfloat16!
  The paper also ignores double double. It is not defined in C. It is or will be superseded by _Float128.
- Neither q nor Q as length modifier are proposed, consuming one character after the other is not sustainable. We are running out of characters available. We have already taken H, D, R, K and to some extend B from the upper-case characters. Lower case letters are limited in number as well.
- Nothing special for float_t and/or double_t is needed. For FLT_EVAL_METHOD 0, 1 and 2 these are just typedefs for float, double or long double. The mapping is unambiguous. float_t, double_t and FLT_EVAL_METHOD are found in the same header.
- Nothing special for _Decimal32_t and/or _Decimal64_t is needed. For DEC_EVAL_METHOD 0, 1 and 2 these are just typedefs for _Decimal32 or _Decimal64. The mapping is unambiguous. _Decimal32_t, _Decimal64_t and DEC_EVAL_METHOD are found in the same header.

**The `scanf` family**

For the scanf family the same format specifiers are reserved. They can be useful in their own right. It is also desirable to have them for reasons of symmetry. They can be helpful to avoid double rounding.

**Words in defense of `printf()` and friends**

printf() is not all that bad. It will be with us for a long time, even if it is possible that alternatives will arise. Thus, it is worth extending it.

1. Quite often printf() is condemned because its use is not type-safe. Luckily, this is only partly true. Many compilers have a lot of built-in knowledge on printf() and other std-functions with variable argument lists. There are a few only in Standard C, but much more in POSIX. It is common that the format-argument of printf() is a constant or can be tacked back to one or several possible constant strings. With proper annotations in system-header files this even works (a bit) when internationalization-facilities are used. So, in practice very many uses of printf() are type-safe.
2. Interpreting the format string at runtime offers additional flexibility. This is useful for internationalization.

[2] %wb16 can be introduced later, once C supports _Bfloat16.

3. Undeniably, there are buffer-overflow bugs associated with the use of `printf()`. But `printf()` is not hard to use correctly, just use the optional precision with `%s` and be careful with `%...f` and ensure type specifiers and arguments match.
4. `printf()` allows for small code-size.
5. `printf()` is a very well-known widely used function.
6. `printf()` is easy to use. It is not a coincidence that there is GNU autosprintf [12]. This allows simple formatting to `std::cout`, `std::cerr`, `std::string` for C++ users.

## Converting binary floating-point numbers from/to decimal representation is surprisingly difficult

Writing conversion routines perfectly suitable for the early 1970s is easy. But we have more and higher expectations now:

- round-trip should be an identity operation
- high accuracy, preferable correctly rounded honoring the rounding mode
- fast with low variance of run-time, i. e. mostly independent of the values converted
- space efficient, i. e. small tables only
- no dynamic memory allocation[3]

Numerous papers written in decades – some of them quite recent - are an indication that converting binary floating-point numbers from/to decimal representation is indeed difficult [13][14][15][16][17][18][19][20].

## Deviation from prior practice

Traditionally C has used different length modifiers for integer types and floating-point types, e. g. `h` for integer types and `D` for decimal floating-point types. A single `h` often denotes 16 bit a single `H` is 32 bit always. In contrast `%w32u` und `%w32f` are 32 bit both. So reusing `w` seems reasonable.

## Annex K

Nothing has to be done for Annex K as the functions in Annex K reference other sections of the standard when format specifiers are used.

## Wording changes:

The changes given here are relative to N3054 [21] for the printf family:

In 7.23.6.1 p7 after

| | |
|---|---|
| wN | Specifies that a following b, d, i, o, u, x, or X conversion specifier applies to an integer argument with a specific width where N is a positive decimal integer with no leading zeros (the argument will have been promoted according to the integer promotions, but its value shall be converted to the unpromoted type); or that a following n conversion specifier applies to a pointer to an integer type argument with a width of N bits. All |

[3] Desirable, but still not widely seen.

minimum-width integer types (7.22.1.2) and exact-width integer types (7.22.1.1) defined in the header <stdint.h> shall be supported. Other supported values of N are implementation-defined.

insert

wN Specifies that a following a, f, g or e conversion specifier applies to a _FloatN argument with a specific width where N is a positive decimal integer with no leading zeros; or that a following n conversion specifier applies to a pointer to an integer type argument with a width of N bits. All _FloatN types provided by the implementation shall be supported. Other values of N are reserved.

wxN Specifies that a following a, f, g or e conversion specifier applies to a _FloatNx argument with a specific width where N is a positive decimal integer with no leading zeros; or that a following n conversion specifier applies to a pointer to an integer type argument with a width of N bits. All _FloatNx types provided by the implementation shall be supported. Other values of N are reserved.

and after

DD Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **_Decimal128** argument.

insert

wND Specifies that a following a, f, g or e conversion specifier applies to an _DecimalN argument with a specific width where N is a positive decimal integer with no leading zeros; or that a following n conversion specifier applies to a pointer to an integer type argument with a width of N bits. All _DecimalN types provided by the implementation shall be supported. Other values of N are reserved.

wxND Specifies that a following a, f, g or e conversion specifier applies to an _DecimalNx argument with a specific width where N is a positive decimal integer with no leading zeros; or that a following N conversion specifier applies to a pointer to an integer type argument with a width of N bits. All _DecimalNx types provided by the implementation shall be supported. Other values of N are reserved.

Apply the same changes to 7.31.2.1 p7.

And after 7.6.3.1 after

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

insert:

If _Float64x, _Float128, _Decimal64x or _Decimal128 are supported the number of characters that can be produced by any single conversion shall be at least 8191. This also applies when _FloatN, _FloatNx, _DecimalN or _DecimalNx for N>128 are supported.

In addition, the number of characters that can be produced by any single conversion shall exceed the number of characters produced by a %f conversion without specifying field with or precision by at least 100.

And in …

after

In 7.33.16 p1 after

1 Lowercase letters may be added to the conversion specifiers and length modifiers in fprintf and fscanf. Other characters may be used in extensions.

insert

All integer values of N for %wN for _FloatN not specified in 7.23.6.1 and 7.33.16 are reserved.

All integer values of N %wxN for _FloatNx not specified in in 7.23.6.1 and 7.33.16 are reserved.

All integer values of N for %wND for _DecomalN not specified in 7.23.6.1 and 7.33.16 are reserved.

All integer values of N %wxND for _DecimalNx not specified in in 7.23.6.1 and 7.33.16 are reserved.

**scanf family:**

The changes given here are relative to N3054 [21] for the scanf family:

In 7.23.6.2 p11 after

| | |
|---|---|
| wN | Specifies that a following b, d, i, o, u, x, or X, or n conversion specifier applies to an argument which is a pointer to an integer with a specific width where N is a positive decimal integer with no leading zeros. All minimum-width integer types (7.22.1.2) and exact-width integer types (7.22.1.1) defined in the header <stdint.h> shall be supported. Other supported values of N are implementation-defined. |

insert

| | |
|---|---|
| wN | Specifies that a following a, A, e, E, f, F, g or G conversion specifier applies to an argument which is a pointer to _FloatN with a specific width where N is a positive decimal integer with no leading zeros. All _FloatN types provided by the implementation shall be supported. Other values of N are reserved. |
| wNx | Specifies that a following a, A, e, E, f, F, g or G conversion specifier applies to an |

argument which is a pointer to a _FloatNx with a specific width where N is a positive decimal integer with no leading zeros. All _FloatNx types provided by the implementation shall be supported. Other values of N are reserved.

and after

| | |
|---|---|
| wfN | Specifies that a following b, d, i, o, u, x, or X, or n conversion specifier applies to an argument which is a pointer to a fastest minimum-width integer with a specific width where N is a positive decimal integer with no leading zeros. All fastest minimum-width integer types (7.22.1.3) defined in the header <stdint.h> shall be supported. Other supported values of N are implementation-defined |

insert

| | |
|---|---|
| wND | Specifies that a following a, A, e, E, f, F, g or G conversion specifier applies to an argument which is a pointer to _DecimalN with a specific width where N is a positive decimal integer with no leading zeros. All _DecomalN types provided by the implementation shall be supported. Other values of N are reserved. |
| wxND | Specifies that a following a, A, e, E, f, F, g or G conversion specifier applies to an argument which is a pointer to _DecimalNx with a specific width where N is a positive decimal integer with no leading zeros. All _DecimalNx types provided by the implementation shall be supported. Other values of N are reserved. |

Apply the same changes to 7.31.2.1 p11.

**Possible straw polls:**

Depending on the discussion:

Decision poll: Does the Committee wish to adopt the changes proposed in N3184

1. All changes proposed.
2. The changes proposed for the printf family for `_FloatN`
3. The changes proposed for the printf family for `_FloatNx`
4. The changes proposed for the scanf family for `_FloatN`
5. The changes proposed for the scanf family for `_FloatNx`
6. The changes proposed for the printf family for `_DecimalN`
7. The changes proposed for the printf family for `_DecimalNx`
8. The changes proposed for the scanf family for `_DecimalN`
9. The changes proposed for the scanf family for `_DecimalNx`

If not, Opinion poll: Does the Committee want to see something along the lines of N3184 in a future paper?

**Acknowledgements:**

I want to thank Thomas Kemmer and Reinhard Kopka for helpful comments.

**References:**

[1] http://www.jhauser.us/arithmetic/SoftFloat.html

[2] https://gcc.gnu.org/onlinedocs/libquadmath/

[3] https://bellard.org/softfp/

[4] https://www.mathworks.com/products/matlab.html

[5] Fortran 2018 (ISO/IEC 1539-1:2018)

[6] https://www.boost.org/

[7] Peter Markstein, IA-64 and Elementary Functions: Speed and Precision (Hewlett-Packard Professional Books)

[8] ISO/IEC TR 18037:2008 Programming languages — C — Extensions to support embedded processors

[9] N1275 2007/10/01 Wakker, TR 18037 (for SC 22 review)

[10] N2680 2021/03/09 Seacord, Specific-width length modifier (updates N 2623)

[11] ISO-IEC JTC 1-SC 22-WG 21_N4928_Working Draft, Standard for Programming Language C++

[12] https://www.gnu.org/software/gettext/libasprintf/manual/autosprintf.html

[13] Matula, David W. “In-and-Out Conversions.” Communications of the ACM 11, 1 (January 1968), 47-50.

[14] Matula, David W. “A Formalization of Floating-Point Numeric Base Conversion.” IEEE Transactions on Computers C-19, 8 (August 1970), 681-692.

[15] Guy L. Steele Jr., Jon L. White, How to Print Floating-Point Numbers Accurately

[16] David M. Gay. Correctly rounded binary-decimal and decimal-binary conversions. Numerical Analysis Manuscript 90-10, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, November 1990.

[17] Clinger, William D. How to read floating point numbers accurately. Proc. ACM SIGPLAN ‘90 Conference on Programming Language Design and Implementation (White Plains, New York, June 1990)

[18] Florian Loitsch, Printing Floating-Point Numbers Quickly and Accurately with Integers

[19] Ulf Adams. 2018. Ryu: Fast Float-to-String Conversion. In Proceedings of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’18). ACM, New York, NY, USA,

[20] Raffaello Giulietti, The Schubfach way to render doubles

[21] https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3054.pdf