Proposal for C23
WG14 N2806

Title:	5.2.4.2.2 cleanup (N2672 update)
Author, affiliation:	C FP group
Date:	2021-08-21
Proposal category:	Editorial
Reference:	N2596, N2672

This paper is a minor update to N2672. Vincent Lefevre sent email to CFP pointing out an error in the definition of *normalized float-point number* (shown below), namely that "all possible *k* digits" isn't meaningful since *k* is just the index in the model formula. The suggested changes below include a simpler definition that fixes the error. Otherwise, what follows is unchanged from N2672.

5.2.4.2.2 begins with:

> [1] The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.21)

The second "that" refers back to "characteristics", which isn't clear from the sentence structure. The suggested changes below offer a clarification.

5.2.4.2.2 also has:

> [4] Floating types shall be able to represent zero (all f_k == 0) and all normalized floating-point numbers ($f_1 > 0$ and all possible *k* digits and *e* exponents result in values representable in the type). In addition, floating types may be able to contain other kinds of floating-point numbers,22) such as negative zero, subnormal floating-point numbers ($x \neq 0$, $e = e_{min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{min}$, $f_1 = 0$), and values that are not floating-point numbers, such as infinities and NaNs. …
>
> [5] An implementation may give zero and values that are not floating-point numbers (such as infinities and NaNs) a sign or may leave them unsigned. Wherever such values are unsigned, any requirement in this document to retrieve the sign shall produce an unspecified sign, and any requirement to set the sign shall be ignored.

In [4] it would be more straightforward to just say zeros may be signed or unsigned.

Since [4] already allows negative zero, [5]'s saying zeros may have a sign is redundant.

At this level of abstraction (values, not bit representations), it would be better to avoid mentioning signed NaNs. The IEC 60559 levels of abstraction (see 3.2) do not have signed NaNs at Level 3 which most closely matches the abstraction of the C model in 5.2.4.2.2. IEC 60559 does not interpret the sign bit of NaNs at the bit representation level. The footnote in the suggested changes is intended to help with ambiguities about levels of abstraction (values vs bit representations).

The second sentence of [5] is not consistent with other specification in C. For example, regarding **`signbit`**, footnote 253 says "... If zero is unsigned, it is treated as positive." (Wording for **`signbit`** is problematic, which is addressed in a separate proposal.)

In [5] is the only use of "retrieve" in the N2596. The term "get" is used in the specification of payload functions (F.10.13) and "get" pairs nicely with "set" which appears later in the same sentence in [5].

In [4], "=" should be used instead of "==".

The suggested changes below address these issues.

Suggested changes:

Changes for 5.2.4.2.2:

> [1] The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and allows other values. The characteristics ~~that~~ provide information about an implementation's floating-point arithmetic.21)
> ...
>
> [4] Floating types shall be able to represent signed zeros or an unsigned zero (all f_k =~~=~~ 0) and all normalized floating-point numbers (all x with $f_1 > 0$ ~~and all possible k digits and e exponents result in values representable in the type~~). In addition, floating types may be able to contain other kinds of floating-point numbers,22) such as ~~negative zero,~~ subnormal floating-point numbers ($x \neq 0$, $e = e_{min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{min}$, $f_1 = 0$), and values that are not floating-point numbers, such as ~~infinities and NaNs~~ NaNs and (signed or unsigned) infinities. ...
>
> [5] ~~An implementation may give zero and values that are not floating-point numbers (such as infinities and NaNs) a sign or may leave them unsigned.~~ Wherever ~~such~~ values are unsigned, any requirement in this document to ~~retrieve~~ get the sign shall produce an unspecified sign, and any requirement to set the sign shall be ignored, unless specified otherwise.*)

*) Bit representations of floating-point values might include a sign bit, even if the value can be regarded as unsigned. IEC 60559 NaNs are such values.