Proposal for C26
WG14 Nnnnn

Title:	**DRAFT** Annex G and F updates
Author:	Damian McGuckin
Date:	2024-08-08
Proposal Category:	Editorial
Reference:	N3220

The changes listed here fall into two categories. Some make details of the various paragraphs and special cases more consistent. Others enhance readability or clarity, and in a very few cases, fill in gaps. Both Annex F and Annex G should now be consistent both internally and with each other.

The removal of imaginary numbers as per WG14 proposal **n3274.pdf** is now a done deal as per the post June 2024 meeting report **n3302.pdf** and so anything to do with imaginary numbers no longer appears here.

Comments within this document including **ASIDE:** comments, and various editorial annotations used to highlight changes, will disappear from this document before submission.

Outstanding Issues for CFP

- Changes noted in CFP 3155 to 6.2.5 are assumed accepted.
- Double check that the additional formulae added to 6.6.1 Paragraph 9 are valid!
- Double check the proof on Page 14 for the G.6.3.5 changes!
- Does more need to be said about NaN signs and sign bits, in both the context of Annex F and Annex G?
- Is a reference to Thomas & Kahan for the optimal handling of `csqrt` needed?
- Are any other references about the evaluation of functions of complex arguments needed?

Conventions

A changed dash-list item is labelled as **-N-**, where **N** is the item number within that list.

A new dash-list item labelled as **ΔN**, appears **after** an existing item number **N**.

A new dash-list item labelled as **∇N**, appears **before** an existing item number

Anything marked with **CHANGED** or **NOTE** needs to be checked for correctness.

The use of commas, terms, and sentence structure, and is more consistent.

Some of the inconsistencies can be traced to different authors at different points in time. For example, the special cases for the arguments $+0 + i\,\infty$ and$+0 + i$ NaN to **`ctanh`** did not e ven exist in the 2005 draft. Their later addition did not exploit the clarity the plus sign provides in the (already included) special case for $+0 + i\,0$. Hopefully all such inconsistencies have now been found!

The special cases are sometimes reordered (by argument) for consistency amongst functions. The order in which special cases appear within a given function is now the same across the various functions.

Where the domain for which a special case applies is qualified, Annex F uses inequalities. Annex G did not. It now does. Domains are now qualified by mathematical inequalities across the board. The avoids the confusion that the appearance of variations of terms for the same concept like *non-zero finite x* and *finite non-zero x*, or worse still, *finite positive signed y* and *positive signed finite y*.

The mandated syntax of special cases spells out the function, the argument associated with the special case, the result for this case, any domain qualification for which the special cases if that is the case, and finally, any exceptions that are raised. This syntax makes it easier to compare domains between special cases.

Because complex number phraseology is not always the same across the world, some of the simple nomenclature used herein is spelt out at the start to reduce such misunderstanding.

Annex F consistency fixes

Many inconsistencies exist in the use of commas, terms, ways of expressing the same sort of sentence in two different places, and ways of describing a domain like *finite* $x > 0$, and expressing a domain with **if** and **even if** as well as the mathematically correct **for**.

When n appears within a domain specification currently, the word integer will qualify n within an inequality or equality or assertion for clarity, consistency and readability, even if the function prototype may already mandate that some n has an integer type such as **long long int**.

F.10 Mathematics <math.h> and <tgmath.h> - Paragraph 1 -

This clause contains specifications for `<math.h>` and `<tgmath.h>` functions that are particularly suited to ISO/IEC 60559 implementations.

F.10.1.1 The `acos` functions

-2- **`acos`** (x) returns a NaN for $|x| > 1$ and raises the "invalid" floating-point exception.

F.10.1.1 The `asin` functions

-2- **`asin`** (x) returns a NaN for $|x| > 1$ and raises the "invalid" floating-point exception.

F.10.1.4 The `atan2` functions

-7- **`atan2`** $(\pm y, -\infty)$ returns $\pm\pi$ for $0 < y < \infty$.

-8- **`atan2`** $(\pm y, +\infty)$ returns ± 0 for $0 < y < \infty$.

-9- **`atan2`** $(\pm\infty, x)$ returns $\pm\frac{\pi}{2}$ for $|x| < \infty$.

F.10.1.8 The `acospi` functions

-2- **`acospi`** (x) returns a NaN for $|x| > 1$ and raises the "invalid" floating-point exception.

F.10.1.9 The `asinpi` functions

-2- **`asinpi`** (x) returns a NaN for $|x| > 1$ and raises the "invalid" floating-point exception.

F.10.1.11 The `atan2pi` functions

-7- **`atan2pi`** $(\pm y, -\infty)$ returns ± 1 for $0 < y < \infty$.

-8- **`atan2pi`** $(\pm y, +\infty)$ returns ± 0 for $0 < y < \infty$.

-9- **`atan2pi`** $(\pm\infty, x)$ returns $\pm\frac{1}{2}$ for $|x| < \infty$.

F.10.1.12 The `cospi` functions

-2- **`cospi`** $(n + \frac{1}{2})$ returns +0 for all integers n.

F.10.1.13 The `sinpi` functions

-2- **`sinpi`** $(\pm n)$ returns ± 0 for integers $n > 0$.

F.10.1.14 The `tanpi` functions

-2- **`tanpi`** (n) returns +0 for even integers $n > 0$ and odd integers $n < 0$.

-3- **`tanpi`** (n) returns -0 for odd integers $n > 0$ and even integers $n < 0$.

-4- **`tanpi`** $(n + \frac{1}{2})$ returns $+\infty$ for even integers n and raises the "divide-by-zero" floating-point exception.

-5- **`tanpi`** $(n + \frac{1}{2})$ returns $-\infty$ for odd integers n and raises the "divide-by-zero" floating-point exception.

F.10.2.1 The `acosh` functions

-2- **`acosh`** (x) returns a NaN for $x < 1$ and raises the "invalid" floating-point exception.

F.10.2.3 The `atanh` functions

-2- **`atanh`** $(\pm x)$ returns $\pm\infty$ for $x = 1$ and raises the "di vide-by-zero" floating-point exception.

-3- **`atanh`** (x) returns a NaN for $|x| > 1$ and raises the "invalid" floating-point exception.

F.10.3.7 The `frexp` functions

-1- **`frexp`** $(\pm 0, p)$ stores 0 in the object pointed to by p and returns ± 0.

-2- **`frexp`** $(\pm\infty, p)$ stores an unspecified value in the object pointed to by p and returns $\pm\infty$.

ASIDE: The order of the *stores*... and*r eturns*... has been swapped (and a comma disappears) to be consistent with the phrasing seen in the third (unchanged) entry which cannot be swapped for other reasons.

F.10.3.8 The `ilogb` functions - Paragraph 3

`ilogb` (x) returns the value specified in 7.12.6.8 for x is a NaN, x is ± 0 or $x = \pm\infty$ and raises the "invalid" floating-point exception.

F.10.3.11 The `log` functions

-3- **`log`** (x) returns a NaN for $x < 0$ and raises the "invalid" floating-point exception.

F.10.3.12 The `log10` functions - Renumber as F.10.3.13

-3- **`log10`** (x) returns a NaN for $x < 0$ and raises the "invalid" floating-point exception.

F.10.3.13 The `log10p1` functions - Renumber as F.10.3.14

-3- **`log10p1`** (x) returns a NaN for $x < -1$ and raises the "invalid" floating-point exception.

F.10.3.14 The `log1p` and `logp1` functions - Renumber as F.10.3.12

-3- **`logp1`** (x) returns a NaN for $x < -1$ and raises the "invalid" floating-point exception.

F.10.3.15 The `log2` functions

-3- **`log2`** (x) returns a NaN for $x < 0$ and raises the "invalid" floating-point exception.

F.10.3.16 The `log2p1` functions

-3- **`log2p1`** (x) returns a NaN for $x < -1$ and raises the "invalid" floating-point exception.

F.10.4.2 The `compoundn` functions

-1- **`compoundn`** $(x, 0)$ returns 1 for $-1 \le x \le \infty$.

-Δ1- **`compoundn`** (NaN, 0) returns 1.

-2- **`compoundn`** (x, n) returns a NaN for $x < -1$ and raises the "invalid" floating-point exception.

-3- **`compoundn`** $(-1, n)$ returns $+\infty$ for integers $n < 0$ and raises the "divide-by-zero" floating-point exception.

-4- **`compoundn`** $(-1, n)$ returns $+0$ for integers $n > 0$.

-5- **`compoundn`** $(+\infty, n)$ returns $+0$ for integers $n < 0$.

-6- **`compoundn`** $(+\infty, n)$ returns $+\infty$ for integers $n > 0$.

The order of the less-than-zero and greater-than-zero cases of item 5 and 6 now match that of items 3 and 4.

F.10.4.4 The `hypot` functions

-2- **`hypot`** $(x, \pm 0)$ returns the absolute value of x for any x not a NaN.

-3- **`hypot`** $(\pm\infty, y)$ returns $+\infty$ for $|y| \leq \infty$.

-4- **`hypot`** (x, NaN) returns a NaN for $|x| \neq \infty$.

-Δ4- **`hypot`** $(\pm\infty, \text{NaN})$ returns $+\infty$.

F.10.4.5 The `pow` functions

-1- **`pow`** $(\pm 0, y)$ returns $\pm\infty$ for $-\infty <$ odd−integral $y < 0$ and raises the "divide-by-zero" floating-point exception. **CHANGED domain because $y = -\infty$ is not odd−integral**.

-2- **`pow`** $(\pm 0, y)$ returns $+\infty$ for $-\infty <$ not odd−integral $y < 0$ and raises the "divide-by-zero" floating-point exception.

-4- **`pow`** $(\pm 0, y)$ returns ± 0 for $0 <$ odd−integral $y < \infty$.

-5- **`pow`** $(\pm 0, y)$ returns $+0$ for $0 <$ not odd−integral $y < \infty$. **NOTE:** see next item

-Δ5- **`pow`** $(\pm 0, \infty)$ returns $+0$. **NOTE:** Handle case of infinity separately for clarity

— ...

-9- **`pow`** (x, y) returns a NaN for $-\infty < x < 0$ and $0 <$ | non− integral $y| < \infty$ and raises the "invalid" floating-point exception.

— ...

-14- **`pow`** $(-\infty, y)$ returns -0 for $-\infty <$ odd−integral $y < 0$.

-15- **`pow`** $(-\infty, y)$ returns $+0$ for $-\infty <$ not odd−integral $y < 0$. **NOTE:** see next item

-Δ15- **`pow`** $(-\infty, -\infty)$ returns $+0$. **NOTE:** Handle case of infinity separately for clarity

-16- **`pow`** $(-\infty, y)$ returns $-\infty$ for $0 <$ odd−integral $y < \infty$.

-17- **`pow`** $(-\infty, y)$ returns $+\infty$ for $0 <$ not odd−integral $y < \infty$. **NOTE:** see next item

-Δ17- **`pow`** $(-\infty, +\infty)$ returns $+\infty$. **NOTE:** Handle case of infinity separately for clarity

F.10.4.6 The `pown` functions - 4th+5th reordered to match 2nd+3rd and simplified 6th

-2- **`pown`** $(\pm 0, n)$ returns $\pm\infty$ for odd integers $n < 0$ and raises the "divide-by-zero" floating-point exception.

-3- **`pown`** $(\pm 0, n)$ returns $+\infty$ for even integers $n < 0$ and raises the "divide-by-zero" floating-point exception.

-4- **`pown`** $(\pm 0, n)$ returns ± 0 for odd integers $n > 0$.

-5- **`pown`** $(\pm 0, n)$ returns $+0$ for even integers $n > 0$.

-6- **`pown`** $(\pm\infty, n)$ returns **`pown`** $(\pm 0, -n)$ for integers $n < 0$.

-7- **`pown`** $(\pm\infty, n)$ returns $\pm\infty$ for odd integers $n > 0$.

-8- **`pown`** $(\pm\infty, n)$ returns $+\infty$ for even integers $n > 0$.

NOTE: The order of the integer domain qualification of -2- through -5- is now odd, even, odd, even.

F.10.4.7 The `powr` functions

-1- **`powr`** $(x, \pm 0)$ returns 1 for $0 < x < \infty$.

-2- **`powr`** $(\pm 0, y)$ returns $+\infty$ for $-\infty < y < 0$ and raises the "divide-by-zero" floating-point exception.

-5- **`powr`** $(+1, y)$ returns 1 for $|y| < \infty$.

-7- **`powr`** (x, y) returns a NaN for $x < 0$ and raises the "invalid" floating-point exception.

F.10.4.8 The `rootn` functions

-1- **`rootn`** $(\pm 0, n)$ returns $\pm\infty$ for odd integers $n < 0$ and raises the "divide-by-zero" floating-point exception.

-2- **`rootn`** $(\pm 0, n)$ returns $+\infty$ for even integers $n < 0$ and raises the "divide-by-zero" floating-point exception.

-3- **`rootn`** $(\pm 0, n)$ returns +0 for even integers $n > 0$.

-4- **`rootn`** $(\pm 0, n)$ returns +0 for odd integers $n > 0$.

-5- **`rootn`** $(+\infty, n)$ returns $+\infty$ for integers $n > 0$.

-6- **`rootn`** $(-\infty, n)$ returns $+\infty$ for odd integers $n > 0$.

-7- **`rootn`** $(-\infty, n)$ returns a NaN for even integers $n > 0$ and raises the "invalid" floating-point exception.

-8- **`rootn`** $(+\infty, n)$ returns +0 for integers $n < 0$.

-9- **`rootn`** $(-\infty, n)$ returns +0 for odd integers $n < 0$.

-10- **`rootn`** $(-\infty, n)$ returns a NaN for even integers $n < 0$ and raises the "invalid" floating-point exception.

-11- **`rootn`** $(x, 0)$ returns a NaN for $|x| \le \infty$ and raises the "invalid" floating-point exception.

-Δ11- **`rootn`** (NaN, 0) returns a NaN and raises the "invalid" floating-point exception.

-12- **`rootn`** (x, n) returns a NaN for $x < 0$ and even integers $n > 0$ and raises the "invalid" floating-point exception.

F.10.4.9 The `rsqrt` functions

-2- **`sqrt`** $(+\infty)$ returns +0.

-3- **`rsqrt`** (x) returns a NaN for $x < 0$ and raises the "invalid" floating-point exception.

F.10.4.10 The `sqrt` functions

-3- **`sqrt`** (x) returns a NaN for $x < 0$ and raises the "invalid" floating-point exception.

F.10.5.3 The `lgamma` functions

-3- **`lgamma`** (x) returns $+\infty$ for integral $x \le 0$ and raises the "divide-by-zero" floating-point exception.

F.10.5.4 The `tgamma` functions

-2- **`tgamma`** (x) returns a NaN for integral $x < 0$ and raises the "invalid" floating-point exception.

F.10.7.1 The `fmod` functions - split domain of -2- for clarity

-1- **`fmod`** $(\pm 0, y)$ returns ± 0 for $y \neq 0$.

-2- **`fmod`** $(x, \pm 0)$ returns a NaN for $|x| \leq \infty$ and raises the "invalid" floating-point exception.

-3- **`fmod`** $(x, \pm\infty)$ returns x for $|x| < \infty$.

-4- **`fmod`** $(\pm\infty, y)$ returns a NaN for $|y| \leq \infty$ and raises the "invalid" floating-point exception.

F.10.7.2 The `remainder` functions - split domain of -2- for clarity

-1- **`remainder`** $(\pm 0, y)$ returns ± 0 for $y \neq 0$.

-2- **`remainder`** $(x, \pm 0)$ returns a NaN for $|x| \leq \infty$ and raises the "invalid" floating-point exception.

-3- **`remainder`** $(x, \pm\infty)$ returns x for $|x| < \infty$.

-4- **`remainder`** $(\pm\infty, y)$ returns a NaN for $|y| \leq \infty$ and raises the "invalid" floating-point exception.

F.10.8.3 The `nextafter` functions

-1- **`nextafter`**(x, y) raises the "overflow" and "inexact" floating-point exceptions when the function value returned is infinite and $|x| < \infty$.

-2- **`nextafter`**(x, y) raises the "underflow" and "inexact" floating-point exceptions when the function value returned is subnormal or zero and $x \neq y$.

Annex G consistency fixes

G.6.1 Paragraph 1

DELETED due to the removal of Imaginary types.

G.6.1 Paragraph 2 - Now Paragraph 1 - Consistent Between F+G

This clause contains specifications for `<complex.h>` functions that are particularly suited to ISO/IEC 60559 implementations. Each such function f generally takes a single complex number argument z, which can be written in terms of its real ($x = \mathbf{Re}(z)$) and imaginary ($y = \mathbf{Im}(z)$) components as $x + i\,y$ (6.2.5#17). Where f returns a complex number result, it too is generally written in terms of its real and imaginary components. Unless otherwise specified, where the $\pm$ symbol occurs within both a function argument and its result, it implies two cases: one with each $\pm$ symbol replaced with a plus (+) symbol in both argument and result, and the other with each $\pm$ symbol replaced with a minus (–) symbol in both argument and result. Where the $\pm$ symbol occurs only within an argument and not in the function result, the result is independent of that sign. Where the $\pm$ symbol occurs only within a function result and not in an argument, the sign of that part of the result is unspecified. Where specifications appear for a function f which is a member function of a family of functions, the specification apply to all member functions of that family. even though only the (principal) member function is seen.

G.6.1. Paragraph 2 - Replacement - If Necessary Cross References CFP 3155

In this and the following subclauses, $\bar{z}$ or **`conj`**(z) is the complex conjugate (7.3.9.4) of z, or mathematically, if z is defined as $z = x + i\,y$ in terms of its real and imaginary components then, $\bar{z} =$ **`conj`**$(z) = x - i\,y$.

G.6.1. Paragraph 5 - Add this sentence

Where a NaN appears in a special case's argument(s) or result, even though its representation might have a sign bit, that bit has no interpretation.

G.6.1 Paragraph 7 - Augment table text for consistency

ASIDE: *The table here is purely for reference - it is unchanged. Only the text prior to the table is changed. The verb* **covered** *in the original text in parentheses referring to Annex F is replaced with* **specified** *for consistency with Paragraph 8.*

Each of the functions **`cabs`** and **`carg`** (7.3) is specified by a formula in terms of a real function (whose special cases were specified earlier in Annex F):

`cabs(`$x + i\,y$**`)`**	$=$ **`hypot(`**x, y**`)`**
`carg(`$x + i\,y$**`)`**	$=$ **`atan2(`**y, x**`)`**

G.6.1 Paragraph 8 - Make consistent with Paragraph 7

ASIDE: *The table here is purely for reference - it is unchanged. Only the text prior to the table is changed for consistency and clarity. The original text said special cases were* **specified below** *which is wrong.*

Each of the functions **`casin`**, **`catan`**, **`ccos`**, **`csin`**, and **`ctan`** (7.3) is specified by a formula in terms of some other complex function (with their special cases then specified later in this annex):

`casin(`z**`)`**	$=$ **`-`**i **`casinh(`**$i\,z$**`)`**
`catan(`z**`)`**	$=$ **`-`**i **`catanh(`**$i\,z$**`)`**
`ccos(`z**`)`**	$=$ **`ccosh(`**$i\,z$**`)`**
`csin(`z**`)`**	$=$ **`-`**i **`csinh(`**$i\,z$**`)`**
`ctan(`z**`)`**	$=$ **`-`**i **`ctanh(`**$i\,z$**`)`**

G.6.1 Paragraph 9 - Reword for clarity and append missing formula

ASIDE: Having the identities for formulae in Paragraph 7 and 8 of G.6.1 makes it very easy to read the document in the context of the functions to which they refer. Conversely, a lack of mathematical identities within this document for lots of other functions makes it extremely difficult (if not confusing) to read the document when those same functions are being mentioned. Even though this lack of specifications for half of the functions was by design, to ensure clarity and readability of the special cases which appear in the rest of Annex G, specifications are needed for functions noted in this annex, and these should importantly be in the style and terminology used throughout this standards document.

Each of **`cacos`**, **`cacosh`**, **`casinh`**, **`catanh`**, **`ccosh`**, **`csinh`**, **`ctanh`**, **`cexp`**, **`clog`**, and **`csqrt`** (7.3) is an instance of a complex function f of a comple x argument z which satisfies

$$\operatorname{conj}(f(z)) = f(\operatorname{conj}(z))$$

i.e. the conjugate of the evaluation of f for that ar gument z is identical to the value of that same function evaluated for $\bar{z}$, the conjugate of that same z. Given some complex number $z = x + i\,y$ in the upper complex half-plane, i.e. $|x| < \infty$ with $0 < y < \infty$ or y is a +0, (or a $\bar{z} = z = x - i\,y$ in the lower complex half-plane), every one of these ten functions is specified by a formula in terms of other real or complex functions:

$$\begin{aligned}
&\mathbf{cacos}(x \pm i\,y) &&= \tfrac{\pi}{2} - \mathbf{casin}(x \pm i\,y)\\
&\mathbf{cacosh}(x \pm i\,y) &&= \pm i\ \mathbf{cacos}(x \pm i\,y)\\
&\mathbf{casinh}(x \pm i\,y) &&= -i\ \mathbf{casin}(\,i \times (x \pm i\,y))\\
&\mathbf{catanh}(x \pm i\,y) &&= -i\ \mathbf{catan}(\,i \times (x \pm i\,y))\\
&\mathbf{ccosh}(x \pm i\,y) &&= \mathbf{cosh}(x)\mathbf{cos}(\pm y)\ +\ i\ \mathbf{sinh}(x)\mathbf{sin}(\pm y)\\
&\mathbf{csinh}(x \pm i\,y) &&= \mathbf{sinh}(x)\mathbf{cos}(\pm y)\ +\ i\ \mathbf{cosh}(x)\mathbf{sin}(\pm y)\\
&\mathbf{ctanh}(x \pm i\,y) &&= \frac{\mathbf{csinh}(x \pm i\,y)}{\mathbf{ccosh}(x \pm i\,y)}\\
&\mathbf{cexp}(x \pm i\,y) &&= \mathbf{exp}(x) \times (\,\mathbf{cos}(\pm y)\ +\ i\ \mathbf{sin}(\pm y))\\
&\mathbf{clog}(x \pm i\,y) &&= \mathbf{log}(\mathbf{cabs}(x \pm i\,y))\ +\ i\ \mathbf{carg}(x \pm i\,y)\\
&\mathbf{csqrt}(x \pm i\,y) &&= \mathbf{sqrt}\left(\frac{\mathbf{cabs}(x \pm i\,y) + x}{\mathbf{2}}\right) \pm\ i\ \mathbf{sqrt}\left(\frac{\mathbf{cabs}(x \pm i\,y) - x}{\mathbf{2}}\right)
\end{aligned}$$

These specifications are included to assist in reading the special cases of these function which follow shortly. While mathematically correct, their algebra is suboptimal for use in performant, accurate, robust numerical implementations and relevant references should be consulted for such implementations.

ASIDE: The introduction to the above specification needs to be improved.

G.6.1 Paragraph 10 - Moved from Paragraph 9 - Please CHECK for correctness

For the special case of a single argument $\text{NaN} + i\,\text{NaN}$ whose real and imaginary components are both a NaN, every one of these ten functions just returns the result $\text{NaN} + i\,\text{NaN}$ and raises no floating-point exception. For any other special cases of these functions, their behavior, including their treatment of any "invalid" and "divide-by-zero" floating-point exception are specified in the subclauses which follow. Within these subclauses, any specifications for the upper complex half-plane (i.e. where **Im***(z)* is always positively signed) imply specifications for the lower complex half-plane (i.e. where **Im***(z)* is always negatively signed). Also, if one such function f is either even or odd, i.e. it respectively satisfies $f(-z) = f(z)$ or $f(-z) = -f(z)$, then specifications for the first quadrant imply specifications for the other three quadrants. As noted previously, for a family of functions, where the specifications for the principal function is seen, they apply to all member functions within that family.

DELETE the last item of G.6.2.1, G.6.3.1-6, G.6.4.1-2, and G.6.5.2 because the first sentence of the previous paragraph covers such special cases already.

G.6.2.1 The **cacos** functions

-3- **cacos** $(\pm 0 + i\,\text{NaN})$ returns $\frac{\pi}{2} + i\,\text{NaN}$. **NO CHANGE - for reference only**

-4- **cacos** $(x + i\,\infty)$ returns $\frac{\pi}{2} - i\,\infty$ for $|x| < \infty$.

-5- **cacos** $(x + i\,\text{NaN})$ returns $\text{NaN} + i\,\text{NaN}$ for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇6- **cacos** $(-\infty + i\,0)$ returns $\pi - i\,\infty$.

-6- **cacos** $(-\infty + i\,y)$ returns $\pi - i\,\infty$ for $0 < y < \infty$.

-∇7- **cacos** $(+\infty + i\,0)$ returns $+0 - i\,\infty$.

-7- **cacos** $(+\infty + i\,y)$ returns $+0 - i\,\infty$ for $0 < y < \infty$.

-8- **cacos** $(-\infty + i\,\infty)$ returns $\frac{3\pi}{4} - i\,\infty$.

-9- **cacos** $(+\infty + i\,\infty)$ returns $\frac{\pi}{4} - i\,\infty$.

-10- **cacos** $(-\infty + i\,\text{NaN})$ returns $\text{NaN} \pm i\,\infty$(where the sign of the imaginary part of the result is unspecified). **CHANGED split for correctness of unspecified sign**.

-Δ10- **cacos** $(+\infty + i\,\text{NaN})$ returns $\text{NaN} \pm i\,\infty$(where the sign of the imaginary part of the result is unspecified). **CHANGED split for correctness of unspecified sign**.

-11- **cacos** $(\text{NaN} + i\,y)$ returns $\text{NaN} + i\,\text{NaN}$ for $y < \infty$ and optionally raises the "invalid" floating-point exception.

NOTE: Item -3- above is used to deduce **cacosh**$(\pm 0 + i\,\text{NaN}) = i$ **cacos**$(\pm 0 + i\,\text{NaN})$ in clause G.6.3.1.

G.6.3.1 The **cacosh** functions - 3rd+4th Reordered

-2- **cacosh** $(\pm 0 + i\,0)$ returns $+0 + i\,\frac{\pi}{2}$.

-3- **cacosh** $(+0 + i\,\text{NaN})$ returns $\text{NaN} \pm i\,\frac{\pi}{2}$, where the (where the sign of the imaginary part of the result is unspecified). **CHANGED ar gument - real part is +0 - see NOTE at the end of the previous clause**.

-Δ3- **cacosh** $(-0 + i\,\text{NaN})$ returns $\text{NaN} \pm i\,\frac{\pi}{2}$, where the (where the sign of the imaginary part of the result is unspecified). **CHANGED ar gument - real part is -0 - see NOTE at the end of the previous clause**.

-4!! **cacosh** $(x + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{2}$ for $|x| < \infty$.

-5- **cacosh** $(x + i\,\text{NaN})$ returns $\text{NaN} + i\,\text{NaN}$ for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇6- **cacosh** $(-\infty + i\,0)$ returns $+\infty + i\,\pi$.

-6- **cacosh** $(-\infty + i\,y)$ returns $+\infty + i\,\pi$ for $0 < y < \infty$.

-∇7- **cacosh** $(+\infty + i\,0)$ returns $+\infty + i\,0$.

-7- **cacosh** $(+\infty + i\,y)$ returns $+\infty + i\,0$ for $0 < y < \infty$.

-8- **cacosh** $(-\infty + i\,\infty)$ returns $+\infty + i\,\frac{3\pi}{4}$. **NO CHANGE - for reference only**

-9- **cacosh** $(+\infty + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{4}$.

— ...

-11- **cacosh** $(\text{NaN} + i\,y)$ returns $\text{NaN} + i\,\text{NaN}$ for $y < \infty$ and optionally raises the "invalid" floating-point exception.

G.6.3.2 The `casinh` functions

— **`casinh(conj(`*z*`)) = conj(casinh(`*z*`))`** and **`casinh(`*z*`)`** is odd.

— ...

-∇3- **`casinh`** $(+0 + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{2}$.

-3- **`casinh`** $(x + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{2}$ for $0 < x < \infty$.

-4- **`casinh`** $(x + i\,\text{NaN})$ returns $\text{NaN} + i\,\text{NaN}$ for $|x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇5- **`casinh`** $(+\infty + i\,0)$ returns $+\infty + i\,0$.

-5- **`casinh`** $(+\infty + i\,y)$ returns $+\infty + i\,0$ for $0 < y < \infty$.

-6- **`casinh`** $(+\infty + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{4}$.

— ...

-9- **`casinh`** $(\text{NaN} + i\,y)$ returns $\text{NaN} + i\,\text{NaN}$ for $0 < y < \infty$ and optionally raises the "invalid" floating-point exception.

G.6.3.3 The `catanh` functions

— **`catanh(conj(`*z*`)) = conj(catanh(`*z*`))`** and **`catanh(`*z*`)`** is odd.

— ...

-Δ2- **`catanh`** $(+0 + i\,\infty)$ returns $+0 + i\,\frac{\pi}{2}$. **CHANGED domain to agree with other odd functions like `csinh`**.

— ...

-5- **`catanh`** $(x + i\,\infty)$ returns $+0 + i\,\frac{\pi}{2}$ for $0 < x < \infty$.

-6- **`catanh`** $(x + i\,\text{NaN})$ returns $\text{NaN} + i\,\text{NaN}$ for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇7- **`catanh`** $(+\infty + i\,0)$ returns $+0 + i\,\frac{\pi}{2}$.

-7- **`catanh`** $(+\infty + i\,y)$ returns $+0 + i\,\frac{\pi}{2}$ for $0 < y < \infty$.

-8- **`catanh`** $(+\infty + i\,\infty)$ returns $+0 + i\,\frac{\pi}{2}$.

— ...

-10- **`catanh`** $(\text{NaN} + i\,y)$ returns $\text{NaN} + i\,\text{NaN}$ for $y < \infty$ and optionally raises the "invalid" floating-point exception.

-11- **`catanh`** $(\text{NaN} + i\,\infty)$ returns $\pm 0 + i\,\frac{\pi}{2}$(where the sign of the real part of the result is unspecified).

G.6.3.4 The `ccosh` functions

-5- **`ccosh`** $(x + i\infty)$ returns NaN + i NaN for $0 < |x| < \infty$ and raises the "invalid" floating-point exception.

-6- **`ccosh`** $(x + i\,\text{NaN})$ returns NaN + i NaN for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

— ...

-8- **`ccosh`** $(+\infty + i\,y)$ returns $+\infty\,(\cos(y) + i\sin(y))$ for $0 < y < \infty$.

— ...

-12- **`ccosh`** $(\text{NaN} + i\,y)$ returns NaN + i NaN for $0 < y \le \infty$ and optionally raises the "invalid" floating-point exception.

G.6.3.5 The `csinh` functions

— **`csinh(conj(`*z*`)) = conj(csinh(`*z*`))`** and **`csinh(`*z*`)`** is odd.

— ...

-5- **`csinh`** $(x + i\infty)$ returns NaN + i NaN for $0 < |x| < \infty$ and raises the "invalid" floating-point exception. **CHANGED domain for consistency - See Proof G63 on Page 14**.

-6- **`csinh`** $(x + i\,\text{NaN})$ returns NaN + i NaN for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

— ...

-8- **`csinh`** $(+\infty + i\,y)$ returns $+\infty\,(\cos(y) + i\sin(y))$ for $0 < y < \infty$.

— ...

-12- **`csinh`** $(\text{NaN} + i\,y)$ returns NaN + i NaN for $0 < y \le \infty$ and optionally raises the "invalid" floating-point exception.

G.6.3.6 The `ctanh` functions - 4th+5th Reordered - Changed 0 to +0 for clarity"

-3!! **`ctanh`** $(+0 + i\,\infty)$ returns $+0 + i$ NaN and raises the "in valid" floating-point exception.

-4!! **`ctanh`** $(+0 + i\,\text{NaN})$ returns $+0 + i$ NaN.

-5- **`ctanh`** $(x + i\infty)$ returns NaN + i NaN for $0 < |x| < \infty$ and raises the "invalid" floating-point exception.

-6- **`ctanh`** $(x + i\,\text{NaN})$ returns NaN + i NaN for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇7- **`ctanh`** $(+\infty + i\,0)$ returns $1 + i\,0$.

-7- **`ctanh`** $(+\infty + i\,y)$ returns $1 + i\,0$ **`sin`** $(2y)$ for $0 < y < \infty$.

— ...

-11- **`ctanh`** $(\text{NaN} + i\,y)$ returns NaN + i NaN for $0 < y \le \infty$ and optionally raises the "invalid" floating-point exception.

ASIDE: Each **0** in special cases -3- and -4- is prefixed by a plus (+) symbol for consistency with all other odd (and even) functions which have a plus (+) symbol in front of the **0** for those same cases.

G.6.4.1 The `cexp` functions

-3- **`cexp`** $(x + i\,\infty)$ returns NaN $+ i$ NaN for $|x| < \infty$ and raises the "invalid" floating-point exception.

-4- **`cexp`** $(x + i\,$NaN$)$ returns NaN $+ i$ NaN for $|x| < \infty$ and optionally raises the "invalid" floating-point exception.

— ...

-5- **DELETE:** Well, actually moved to ∇7

-∇6- **`cexp`** $(-\infty + i\,0)$ returns $+0 + i\,0$.

-6- **`cexp`** $(-\infty + i\,y)$ returns $+0\,(\cos(y) + i\sin(y))$ for $0 < y < \infty$.

-∇7- **`cexp`** $(+\infty + i\,0)$ returns $+\infty + i\,0$.

-7- **`cexp`** $(+\infty + i\,y)$ returns $+\infty\,(\cos(y) + i\sin(y))$ for $0 < y < \infty$.

— ...

-9- **`cexp`** $(+\infty + i\,\infty)$ returns $\pm\infty + i$ NaN (where the sign of the real part of the result is unspecified) and raises the "invalid" floating-point exception.

— ...

-13- **`cexp`** (NaN $+ i\,y$) returns NaN $+ i$ NaN for $0 < y \le \infty$ and optionally raises the "invalid" floating-point exception.

G.6.4.2 The `clog` functions

-4- **`clog`** $(x + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{2}$ for $|x| < \infty$.

-5- **`clog`** $(x + i\,$NaN$)$ returns NaN $+ i$ NaN for $|x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇6- **`clog`** $(-\infty + i\,0)$ returns $+\infty + i\,\pi$.

-6- **`clog`** $(-\infty + i\,y)$ returns $+\infty + i\,\pi$ for $0 < y < \infty$.

-∇7- **`clog`** $(+\infty + i\,0)$ returns $+\infty + i\,0$.

-7- **`clog`** $(+\infty + i\,y)$ returns $+\infty + i\,0$ for $0 < y < \infty$.

— ...

-9- **`clog`** $(+\infty + i\,\infty)$ returns $+\infty + i\,\frac{\pi}{4}$.

— ...

-11- **`clog`** (NaN $+ i\,y$) returns NaN $+ i$ NaN for $y < \infty$ and optionally raises the "invalid" floating-point exception.

G.6.5.2 The `csqrt` functions - Consistency fixes

-3- **`csqrt`** $(x + i\,\infty)$ returns $+\infty + i\,\infty$ for $|x| \leq \infty$. **NOTE:** case $\mathrm{NaN} + i\,\infty$ moved in -Δ9- to match ordering

-4- **`csqrt`** $(x + i\,\mathrm{NaN})$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $|x| < \infty$ and optionally raises the "invalid" floating-point exception.

-∇5- **`csqrt`** $(-\infty + i\,0)$ returns $+0 + i\,\infty$.

-5- **`csqrt`** $(-\infty + i\,y)$ returns $+0 + i\,\infty$ for $0 < y < \infty$.

-∇6- **`csqrt`** $(+\infty + i\,0)$ returns $+\infty + i\,0$.

-6- **`csqrt`** $(+\infty + i\,y)$ returns $+\infty + i\,0$ for $0 < y < \infty$.

— ...

-9- **`csqrt`** $(\mathrm{NaN} + i\,y)$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $y < \infty$ and optionally raises the "invalid" floating-point exception.

-Δ9- **`csqrt`** $(\mathrm{NaN} + i\,\infty)$ returns $+\infty + i\,\infty$. **NOTE:** this case was originally in -3- above"

ASIDE: The removal of "*y* is a NaN" from the domain qualification of special case -3- makes it consistent with how it is handled in the **`cacosh`**, **`casinh`**, **`catanh`**, and **`clog`** routines.

Consider the following (strict syntax / almost consistent) Annex G special cases in the draft C2X document.

G.6.3.4 The `ccosh` functions

-5- **`ccosh`**$(x + i\infty)$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $0 < |x| < \infty$ and raises the "invalid" floating-point exception.
-6- **`ccosh`**$(x + i\,\mathrm{NaN})$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

G.6.3.5 The `csinh` functions

-5- **`csinh`**$(x + i\infty)$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $0 < x < \infty$ and raises the "invalid" floating-point exception.
-6- **`csinh`**$(x + i\,\mathrm{NaN})$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

G.6.3.6 The `ctanh` functions

-5- **`ctanh`**$(x + i\infty)$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $0 < |x| < \infty$ and raises the "invalid" floating-point exception.
-6- **`ctanh`**$(x + i\,\mathrm{NaN})$ returns $\mathrm{NaN} + i\,\mathrm{NaN}$ for $0 < |x| < \infty$ and optionally raises the "invalid" floating-point exception.

Why is the domain qualification of **`csinh`**'s **-5-** so different to the rest. **OR IS IT?** The case says:

$$\mathbf{csinh}(x + i\infty) = \mathrm{NaN} + i\,\mathrm{NaN} \quad \cdots\; 0 < x < \infty \tag{1}$$

But **`csinh`** is odd, so

$$\mathbf{csinh}(x + i\infty) = -\mathbf{csinh}(-(x + i\infty)) = -\mathbf{csinh}(-x - i\infty)$$

But with **`conj`** $(-x + i\infty) = -x - i\infty$, this can be written as

$$\mathbf{csinh}(x + i\infty) = -\mathbf{csinh}(\mathbf{conj}(-x + i\infty)) \tag{2}$$

For this function, it is known that for any complex number z

$$\mathbf{csinh}(\mathbf{conj}(z)) = \mathbf{conj}(\mathbf{csinh}(z)) \tag{3}$$

Applying this to $-x + i\infty$ yields

$$\mathbf{csinh}(\mathbf{conj}(-x + i\infty)) = \mathbf{conj}(\mathbf{csinh}(-x + i\infty)) \tag{4}$$

Plugging Eq(4) into Eq(2) yields

$$\mathbf{csinh}(x + i\infty) = -\mathbf{conj}(\mathbf{csinh}(-x + i\infty)) \tag{5}$$

Negating and conjugating both sides,

$$\mathbf{conj}(-\mathbf{csinh}(x + i\infty)) = \mathbf{csinh}(-x + i\infty) \tag{6}$$

Using Eq(1) in the LHS of Eq(6) and remembering that the sign of NaN has no interpretation,

$$\begin{aligned} \mathbf{conj}(-(\mathrm{NaN} + i\,\mathrm{NaN})) &= \mathbf{csinh}(-x + i\infty) \\ \mathbf{conj}(\mathrm{NaN} + i\,\mathrm{NaN}) &= \mathbf{csinh}(-x + i\infty) \\ \mathrm{NaN} + i\,\mathrm{NaN} &= \mathbf{csinh}(-x + i\infty) \end{aligned} \tag{7}$$

Rewriting Eq(7) in the style of Eq(1) produces

$$\mathbf{csinh}(-x + i\infty) = \mathrm{NaN} + i\,\mathrm{NaN} \quad \cdots\; 0 < x < \infty \tag{8}$$

The combination of Eq(1) and Eq(8) can be written more concisely as a one-liner

$$\mathbf{csinh}(x + i\infty) = \mathrm{NaN} + i\,\mathrm{NaN} \quad \cdots\; 0 < |x| < \infty \tag{9}$$

This is the same domain qualification as the other five cases!! As this is more consistent and less confusing, it makes more sense to use this domain qualification going forward.