Following is C26-cmplx-mods-aug24-body-67.pdf converted to Word, with inline comments in red.

- Jim Thomas 20240915

Proposal for C26
WG14 Nnnnn

Title:	**DRAFT** changes to 6.X and 7.X
Author:	Damian McGuckin
Date:	2024-08-08
Proposal Category:	Editorial
Reference:	N3301

None of the following terms is never defined

complex floating type	7.2.27#10
complex value	6.3.2.7#2, (post CFP 3155) 6.2.5#17, 7.3.9.3#3, & Annex G
complex number	6.2.5#17

The last of these does also appear in the contents and index.

If these terms were intentional, they need to be defined. Otherwise, their occurrences need to be fixed.

6.2.5 #15 defines complex types and says "The real floating and complex types are collectively called the *floating types*."

"complex value" is a value in a complex type

"complex number" is a number in complex type or a mathematical complex number, depending on context

Overleaf, plus a new clause for **7.3.1#5** are suggested fixes for all of these issues except those in Annex G.

A further change in **7.3.1** arises from the recent removal of imaginary types from the standard. A line has recently been added the implementation may define a macro imaginary but a program may un-**`define`** and perhaps then re-**`define`** that identifier

This was submitted by Jens and accepted by WG14 between the May 2024 CFP meetings and the June 2024 CFP meeting so that CFP never got a chance to review it. While the removal of imaginary types makes this clause confusing, I am sure there is a good reason why Jens inserted it. But for now, just to trigger debate, it has been deleted. Feedback please?

Maybe address this by asking for rationale via WG14 email? Or maybe Rajan or Fred knows.

While **7.3.9.3#3** wants to return an expression whose value is the mathematical equivalent of $x + i\,y$, i has not been mentioned anywhere in the body of the standard to that point. That said, $i = \sqrt{\square\square} - 1$ or the unit imaginary value or the imaginary unit as seen in the existing footnote [227)] referenced in **7.3.1#3**.

See proposal in [3262]

After CFP 3155, **6.2.5#17** uses both *complex number* and *complex values*.

> Each complex type has the same representation and alignment requirements as an array type containing exactly two elements of the corresponding real type; the first element is equal to the real part, and the second element to the imaginary part, of the complex number.

"complex value" would be better than "complex number" here.

In this document complex values are sometimes written in the form $x + i\,y$ where x and y are the values of the real and imaginary parts*).

*) The symbol i denotes the mathematical imaginary unit which has the property i^2 = -1

The form $x - i\,y$ represents $x + i(-y)$.

I did not notice this earlier. Some of those words are out of place. The definition of *i* does not appear until 150 pages later, and even then, in a footnote.

Delete that footnote in favor of the one above.

And the term *complex number* does does appear in that form again until Annex G apart from one isolated appearance in **7.3.9.3#3**.

If

`creal` *functions return the real part value*

why do the **`cimag`** *functions return the imaginary part value (as a real)*

Why the need for the parenthesised text?

It's there to help avoid misinterpreting it to mean the imaginary value *i y*.

It is meaningless as the standard does not even define a *real*. It does define a *real type* which is either an integer type of a real floating type which really is not what is wanted here.

I don't think use of "(real)" here is confusing, but it is too casual for a standard.

Consider deleting it, since returning an imaginary value isn't possible now.

In **6.2.5#17,** it was noted that the type of the imaginary part of an object of complex type was already of *corresponding real type*.

See the subtle change and offer something better!

7.3.1#4 - Introduction - Forward References

There is an unnecessary forward reference which can be simplified by tidying up the English in that sentence. As it already has a major flaw, i.e. implying that parameters are **`float`** or **`long double`** rather than **`float complex`** or **`long double complex`**,

Yes. Needs to be fixed.

and is far too long even before that flaw is corrected, the sentence was split, corrected and cleaned up.

7.3.9.5#2 - cproj - missing Forward Reference

This makes reference to *complex infinities* without a forward reference to **G.2** where a *complex infinity* is defined. That is a jump of 350+ pages which not only a big stretch but is not mentioned. A little bit naughty! Should we move **G.2** to become the new **7.3.1#6**?

Better to void mentioning complex infinities outside of Annex G. See [3267].

7.3.9.5#2 - cproj - description

Here are some definitions for **The Extended Complex Plane**.

The extended complex plane represents the extended complex numbers, that is, the set of complex numbers plus a value $\tilde{\infty}$ for infinity. The Riemann sphere is a model for this extended complex plane.

The set of extended complex numbers is the set of complex numbers $\mathbb{C}$, with the complex infinity $\tilde{\infty}$ adjoined as an element of the set. The extended complex plane is strictly the *Alexandroff extension* of $\mathbb{C}$. A single point, called *the infinity* ($\tilde{\infty}$) is added to $\mathbb{C}$. There is only one infinity in this set, so if one is on the complex plane, and one sets out along any line going out to infinity, all the lines meet at the same infinity. In practice this means that

$$\overline{\mathbb{C}} = \mathbb{C} \cup \tilde{\infty}$$

This $\overline{\mathbb{C}}$ can be thought of as $\mathbb{C}$ with *the infinity*. *The infinity* collects all the points having infinite distance from the origin of $\mathbb{C}$, i.e. $\pm 0 \pm i0$. The best way to visualize the extended complex numbers is through one of its representations, the Riemann sphere. In this representation, one can think of the north and south poles of the sphere as an infinity and a zero, respectively. (The *one-point Alexandroff compactification* of $\tilde{\infty}$ onto $\mathbb{C}$ is a term that is appears in the mix).

ASIDE: Note that *the infinity* (above) is very different to *an infinity* of complex type as defined in **G.2**.

ASIDE: The above says there is only one such *the infinity* and in IEEE 754 there are two, $\infty \pm i0$.
Infinitely many: +/-inf + *i y* and *x* +/- *i* inf, for all *x* and *y*.

Oops.
I realised the definition for **`cproj`**() was less than succinct when somebody (whose only exposure to this concept is through the C library and not complex analysis), referred to it as the **Riemann projection**. Riemann never called it that, nor has any of his peers called it that. If one talks topology, it is technically the

Alexandroff extension of $\overline{\mathbb{C}}$ but the idea of a new term is unthinkable as it buys us nothing.

Currently, we have

> The **`cproj`** functions compute a projection of *z* onto the Riemann sphere where *z* projects to *z* except that all complex infinities, even those with one infinite part and one NaN part, project to positive infinity on the real axis.

To be more precise, I would suggest

> The **`cproj`** functions compute a projection of *z* onto the extended complex plane (as modelled by the Riemann sphere) where *z* projects to *z* except that all complex infinities, even those with one infinite part and one NaN part, project to positive infinity on the real axis.

Sadly, Wolfram considers the Riemann sphere as a commonly used alias for the extended complex plane which is less than helpful.

As the extended complex plane is also a manifold, one day we should sit down with a topology expert over a beer or two and try and understand this stuff.

Yes, language regarding Riemann sphere is problematic. See below.

See the not so subtle change and offer something better!

6.2.5#17 Types - REWORD - the old definition was slightly loose English?

Each complex type has the same representation and alignment requirements as an array type containing exactly two elements of the corresponding real type. What is mathematically the *real part* of an object of complex would be the first element of that array type, while what is mathematically the *imaginary part* of an object of complex type would be the second element of that array type.

Italicizing *real part* and *imaginary part* as you suggest might be helpful.

Use of "is equal to" in the current version is not quite right, because of signed zeros and NaNs. Suggest changing to "represents".

Otherwise I don't see any need for change.

6.3.2#7 Conversions - Real and complex - eliminate *complex values*

When a value of complex type is converted to a real type other than **`bool`**,[52)] the imaginary part of the object of complex type is discarded and the value of the real part is converted according to the conversion rules for the corresponding real type.

I think the current text is ok/better. Conversion is about values, not objects.

7.3.1#3 Introduction - CHANGE - delete mention of imaginary

The macro **`complex`** expands to **`_Complex`**; the macro **`_Complex_I`** expands to an arithmetic constant expression of type **`float _Complex`** with the value of the imaginary unit;[227)] the macro **`I`** expands to **`_Complex_I`** or to an arithmetic constant expression of another implementation-defined type with the same value and precision. Notwithstanding the provisions of 7.1.3, a program may un-define and perhaps then subsequently re-define the macros **`complex`** and **`I`**.

It looks like the current specification is intended to allow an implementation to continue to support imaginary types (as previously specified) as an extension. We should find out if that is the intention before proposing any change.

7.3.1#4 Introduction - CHANGE - correct last sentence and fix grammar

Each synopsis specifies a family of functions (or macros). A synopsis consist of a principal function (or macro) with one or more **`doublecomplex`** parameters and either a **`doublecomplex`** or **`double`** return value, and other functions (or macros), each of whose name is the principal's name albeit suffixed with an **`f`** (or **`l`**). These other functions are functionally identical to the principal function (or macro) but differ in having one or more **`float`** (or **`longdouble`**) **`complex`** parameters and a **`float`** (or **`longdouble`**) **`complex`** or **`float`** (or **`longdouble`**) return value.

Yes, it would be good to include the **CMPLX** macros here.

The synopsis for the **CX_LIMITED_RANGE** pragma should be excluded.

How about:

Each synopsis, other than for the **CX_LIMITED_RANGE** pragma, specifies a family of functions or macros. A family consists of: a principal function (macro) for which the corresponding real type of the parameters and return value is **double**; and analogous functions (macros), with the same name but with **f** (**F**) and **l** (**L**) suffixes, for which the corresponding real type of the parameters and return value are **float** and **long double**.

7.3.1#5 Introduction - NEW

An object of complex type is often referenced in this document as z. If x is the real part of z (mathematically x = **Re**(z)) and y is the imaginary part of z (mathematically y = **Im**(z)), then $z = x + i\,y$ holds where i is the imaginary unit.[227] That expanded form $x + i\,y$ will used henceforth as needed instead of just a bare z. The mathematical conjugate of any such $z = x + i\,y$, commonly abbreviated as $\bar{z}$, can likewise be written in terms of its parts as most commonly $\bar{z} = x - i\,y$, or less often, as $\bar{z} = x + i(-y)$.

Forward references: ISO/IEC 60559-compatible complex arithmetic (Annex G).

Prefer introducing $x + i\,y$ in 6.2.5 #17.

Might want to consider adding a footnote in **conj** description:

*) The complex conjugate of $x + i\,y$ is $x - i\,y$. In this document, conj(z) denotes the complex conjugate of z.

7.2.27#10 Type-generic math <tgmath.h>

Change *complex floating type* to *complex type*. There is currently no such concept. **Should there be**?

See comment above.

Better as is. Note gcc has complex integer types.

7.3.9.2#3 The cimag functions - REVISE - questionable

The **`cimag`** functions return the imaginary part value.

Or "… the value of the imaginary part".

See comment above.

7.3.9.3#3 The CMPLX macros - CHANGE - avoid redundant words

The **`CMPLX`** macros return **`x`**+ *i***`y`**.

Since they are macros, it seems better to be as explicit as possible.

7.3.9.5#2 The cproj functions - REVISE - questionable

The **`cproj`** functions compute a projection of z onto the extended complex plane (as modelled by the Riemann sphere) where z projects to z except for the case where z is a complex infinity (G.2), even one with one infinite part and one NaN part, it projects to positive infinity on the real axis. If z has an infinite part, then **`cproj`**(z) is equivalent to

It's not our "extended complex plane".

The sign of the 0 imaginary part isn't specified.

Maybe it would be better to just say what the functions do:

The **cproj** function project all z with a an infinite real or imaginary part (even if the other part is a quiet NaN) to **CMPLX(INFINITY, +0.0)**, and otherwise project z to z.

and

The **`cproj`** functions return the value of the projection.

A footnote could explain the intended application.

7.3.9.5#3 The cproj functions - REVISE - questionable

The **`cproj`** functions return the value of the projection onto the extended complex plane.

Suggest a separate proposal just for **cproj**.