Remove imaginary types

Jens Gustedt, INRIA and ICube, France

2024-04-14

---

target

integration into IS ISO/IEC 9899:202y

document history

document number	date	comment
n3240 (https://open-std.org/JTC1/SC22/WG14/www/docs/n3240.pdf)	**202404**	original proposal

license

1 Problem description

Optional imaginary types, indicated by the keyword **`_Imaginary`**, have no current implementation that would be known to WG14, and we are not aware of any plans for implementing them. The document n3206 (https://open-std.org/JTC1/SC22/WG14/www/docs/n3206.htm) described a number of problems with these types. It was discussed in the January 2024 meeting in Strasbourg and as a result there was consensus to remove this option from the C standard.

2 Optional removal of `__STDC_559_COMPLEX__`

It seems that no recent implementation has correctly implemented this feature test macro which has been declared obsolete in C23. Therefor we propose to remove it together with the other changes. Such a removal is direct, so we do not explicitly propose wording below.

3 Questions

1. Does WG14 want to remove optional support for imaginary types as proposed in n3240 (https://open-std.org/JTC1/SC22/WG14/www/docs/n3240.pdf) from C2y?

2. Does WG14 want to remove the specification of `__STDC_559_COMPLEX__` from C2y?

3. Does WG14 want to apply the editorial changes as proposed in n3240 (https://open-std.org/JTC1/SC22/WG14/www/docs/n3240.pdf) for C2y?

4 Wording

Removals are in ~~stroke-out red~~, additions in <u>underlined green</u>.

4.1 Changes to Clause 6

- Remove the footnote from 6.2.5 (Types) p15

- 6.3.1.8 change the first indented item as follows:

  If one operand has decimal floating type, the other operand shall not have standard floating~~, complex or imaginary~~ <u>or complex</u> type.

- Remove **`_Imaginary`** from the list of keywords in 6.4.1 p1

- Remove the following sentence from 6.4.1 p2

~~*The keyword **_Imaginary** is reserved for specifying imaginary types.*~~

- Remove the footnote from 6.4.1 p2
- Remove the following phrase from 6.7.6 (Storage-class specifiers) p6

  ~~*If the object declared has imaginary type, the initializer shall have imaginary type.*~~

- Remove the following code snippet from 6.7.6 p21 (Example 5)

```
#ifdef __STDC_IEC_60559_COMPLEX__
constexpr double d5 = (double _Imaginary)0.0; // constraint violation
constexpr double d6 = (double _Imaginary)0.0; // constraint violation
constexpr double _Imaginary di1 = 0.0*I;      // ok
constexpr double _Imaginary di2 = 0.0;        // constraint violation
#endif
```

- Remove the mention of imaginary types from 6.5 (Expressions), 5 places.

4.2 Changes to Clause 7.3 (Complex arithmetic)

- Remove p6
- Change p7

  The macro

```
I
```

  expands to ~~either _Imaginary_I or~~ _Complex_I .~~If _Imaginary_I is not defined, I shall expand to _Complex_I.~~

- Change p8

  Notwithstanding the provisions of 7.1.3, a program may undefine and perhaps then redefine the macros complex~~, imaginary,~~ and I .

- Remove the following sentence from the footnotes to 7.3.9.2 (The `cimag` functions) p2 and 7.3.9.6 (The `creal` functions) p2

  ~~*If imaginary types are supported, z and creal(z)+cimag(z)*I are equivalent expressions.*~~

- In 7.3.9.3 (The `CMPLX` macros) change the initial sentence of p4 as follows, and then replace the occurrences of ~~`_Imaginary_I`~~ by `_Complex_I`.

  These macros act as if ~~the implementation supported imaginary types and~~ the definitions were:

4.3 Optional editorial change

With the above changes, the introductory text to 7.3 becomes a bit disorganized. We propose to reorganize it a bit with the following editorial change.

- Remove current paragraphs p6, p7 and p8.
- Include a new paragraph before current paragraph p4.

  *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;[239)] the macro I expands to _Complex_I. Notwithstanding the provisions of 7.1.13, a program may undefine and perhaps then redefine the macros **complex** and I.*

And keep footnote [239)]

> [239)] *The imaginary unit is a number i such that $i^2 = -1$.*

4.4 Changes to Annex G

There are a lot of removals from Annex G, such that a diff is merely unreadable. An appendix to this document has the proposed replacement, supposing that `__STDC_559_COMPLEX__` is also removed from C2y.

5 Changes to the LaTeX source – notes to reviewers and editors

The branch “imaginary” has the four patches

- *ae7dceb* remove optional support of imaginary types from clauses 6 and 7
- *b782aa5* remove support for imaginary types from annexes
- *82dbdc7* remove `__STDC_559_COMPLEX__`
- *ea8f238* editorial: reorganize the start of complex.h a bit

The first three implement the changes as indicated. The fourth is an editorial change to improve readability of the changed document.

6 Appendix: replacement of Annex G

Annex G
(normative)
ISO/IEC 60559-compatible complex arithmetic

G.1 Introduction

1 This annex supplements Annex F to specify complex arithmetic for compatibility with ISO/IEC 60559 real floating-point arithmetic. An implementation that defines `__STDC_IEC_60559_COMPLEX__` shall conform to the specifications in this annex.[449]

G.2 Conventions

1 A complex value with at least one infinite part is regarded as an *infinity* (even if its other part is a quiet NaN). A complex value is a *finite number* if each of its parts is a finite number (neither infinite nor NaN). A complex value is a *zero* if each of its parts is a zero.

G.3 Complex arithmetic <complex.h>

G.3.1 General

1 This subclause contains specifications for the `<complex.h>` functions that are particularly suited to ISO/IEC 60559 implementations. For families of functions, the specifications apply to all of the functions even though only the principal function is shown. Unless otherwise specified, where the symbol "±" occurs in both an argument and the result, the result has the same sign as the argument.

2 The functions are continuous onto both sides of their branch cuts, taking into account the sign of zero. For example, **csqrt**$(-2 \pm i0) = \pm i\sqrt{2}$.

3 Since complex values are composed of real values, each function may be regarded as computing real values from real values. Except as noted, the functions treat real infinities, NaNs, signed zeros, subnormals, and the floating-point exception flags in a manner consistent with the specifications for real functions in F.10.[450]

4 In subsequent subclauses in G.3 "NaN" refers to a quiet NaN. The behavior of signaling NaNs in this annex is implementation-defined.

5 The functions **cimag**, **conj**, **cproj**, and **creal** are fully specified for all implementations, including ISO/IEC 60559 ones, in 7.3.9. These functions raise no floating-point exceptions.

6 Each of the functions **cabs** and **carg** is specified by a formula in terms of a real function (whose special cases are covered in Annex F):

```
cabs(x + iy) = hypot(x, y)
carg(x + iy) = atan2(y, x)
```

7 Each of the functions **casin**, **catan**, **ccos**, **csin**, and **ctan** is specified implicitly by a formula in terms of other complex functions (whose special cases are specified below):

```
casin(z) = −i casinh(iz)
catan(z) = −i catanh(iz)
ccos(z)  = ccosh(iz)
csin(z)  = −i csinh(iz)
ctan(z)  = −i ctanh(iz)
```

8 For the other functions, the following subclauses specify behavior for special cases, including treatment of the "invalid" and "divide-by-zero" floating-point exceptions. For families of functions, the specifications apply to all of the functions even though only the principal function is shown. For a function f satisfying $f(\mathrm{conj}(z)) = \mathrm{conj}(f(z))$, the specifications for the upper half-plane imply the specifications for the lower half-plane; if the function f is also either even, $f(-z) = f(z)$, or odd,

[449] Implementations that do not define `__STDC_IEC_60559_COMPLEX__` are not required to conform to these specifications.
[450] As noted in G.2, a complex value with at least one infinite part is regarded as an infinity even if its other part is a quiet NaN.

$f(-z) = -f(z)$, then the specifications for the first quadrant imply the specifications for the other three quadrants.

9 In the following subclauses, $\mathrm{cis}(y)$ is defined as $\cos(y) + i\sin(y)$.

G.3.2 Trigonometric functions

G.3.2.1 The `cacos` functions

1
- `cacos`(`conj`(z)) = `conj`(`cacos`(z)).
- `cacos`$(\pm 0 + i0)$ returns $\frac{\pi}{2} - i0$.
- `cacos`$(\pm 0 + i\mathrm{NaN})$ returns $\frac{\pi}{2} + i\mathrm{NaN}$.
- `cacos`$(x + i\infty)$ returns $\frac{\pi}{2} - i\infty$, for finite x.
- `cacos`$(x + i\mathrm{NaN})$ returns $\mathrm{NaN} + i\mathrm{NaN}$ and optionally raises the "invalid" floating-point exception, for nonzero finite x.
- `cacos`$(-\infty + iy)$ returns $\pi - i\infty$, for positive-signed finite y.
- `cacos`$(+\infty + iy)$ returns $+0 - i\infty$, for positive-signed finite y.
- `cacos`$(-\infty + i\infty)$ returns $3\frac{\pi}{4} - i\infty$.
- `cacos`$(+\infty + i\infty)$ returns $\frac{\pi}{4} - i\infty$.
- `cacos`$(\pm\infty + i\mathrm{NaN})$ returns $\mathrm{NaN} \pm i\infty$ (where the sign of the imaginary part of the result is unspecified).
- `cacos`$(\mathrm{NaN} + iy)$ returns $\mathrm{NaN} + i\mathrm{NaN}$ and optionally raises the "invalid" floating-point exception, for finite y.
- `cacos`$(\mathrm{NaN} + i\infty)$ returns $\mathrm{NaN} - i\infty$.
- `cacos`$(\mathrm{NaN} + i\mathrm{NaN})$ returns $\mathrm{NaN} + i\mathrm{NaN}$.

G.3.3 Hyperbolic functions

G.3.3.1 The `cacosh` functions

1
- `cacosh`(`conj`(z)) = `conj`(`cacosh`(z)).
- `cacosh`$(\pm 0 + i0)$ returns $+0 + \frac{i\pi}{2}$.
- `cacosh`$(x + i\infty)$ returns $+\infty + \frac{i\pi}{2}$, for finite x.
- `cacosh`$(0 + i\mathrm{NaN})$ returns $\mathrm{NaN} \pm \frac{i\pi}{2}$ (where the sign of the imaginary part of the result is unspecified).
- `cacosh`$(x + i\mathrm{NaN})$ returns $\mathrm{NaN} + i\mathrm{NaN}$ and optionally raises the "invalid" floating-point exception, for finite nonzero x.
- `cacosh`$(-\infty + iy)$ returns $+\infty + i\pi$, for positive-signed finite y.
- `cacosh`$(+\infty + iy)$ returns $+\infty + i0$, for positive-signed finite y.
- `cacosh`$(-\infty + i\infty)$ returns $+\infty + i\frac{3\pi}{4}$.
- `cacosh`$(+\infty + i\infty)$ returns $+\infty + \frac{i\pi}{4}$.
- `cacosh`$(\pm\infty + i\mathrm{NaN})$ returns $+\infty + i\mathrm{NaN}$.
- `cacosh`$(\mathrm{NaN} + iy)$ returns $\mathrm{NaN} + i\mathrm{NaN}$ and optionally raises the "invalid" floating-point exception, for finite y.
- `cacosh`$(\mathrm{NaN} + i\infty)$ returns $+\infty + i\mathrm{NaN}$.
- `cacosh`$(\mathrm{NaN} + i\mathrm{NaN})$ returns $\mathrm{NaN} + i\mathrm{NaN}$.

G.3.3.2 The `casinh` functions

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

— `casinh`$(+0 + i0)$ returns $0 + i0$.

— `casinh`$(x + i\infty)$ returns $+\infty + \frac{i\pi}{2}$ for positive-signed finite x.

— `casinh`$(x + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite x.

— `casinh`$(+\infty + iy)$ returns $+\infty + i0$ for positive-signed finite y.

— `casinh`$(+\infty + i\infty)$ returns $+\infty + \frac{i\pi}{4}$.

— `casinh`$(+\infty + i\text{NaN})$ returns $+\infty + i\text{NaN}$.

— `casinh`$(\text{NaN} + i0)$ returns $\text{NaN} + i0$.

— `casinh`$(\text{NaN} + iy)$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite nonzero y.

— `casinh`$(\text{NaN} + i\infty)$ returns $\pm\infty + i\text{NaN}$ (where the sign of the real part of the result is unspecified).

— `casinh`$(\text{NaN} + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$.

G.3.3.3 The `catanh` functions

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

— `catanh`$(+0 + i0)$ returns $+0 + i0$.

— `catanh`$(+0 + i\text{NaN})$ returns $+0 + i\text{NaN}$.

— `catanh`$(+1 + i0)$ returns $+\infty + i0$ and raises the "divide-by-zero" floating-point exception.

— `catanh`$(x + i\infty)$ returns $+0 + \frac{i\pi}{2}$, for finite positive-signed x.

— `catanh`$(x + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for nonzero finite x.

— `catanh`$(+\infty + iy)$ returns $+0 + \frac{i\pi}{2}$, for finite positive-signed y.

— `catanh`$(+\infty + i\infty)$ returns $+0 + \frac{i\pi}{2}$.

— `catanh`$(+\infty + i\text{NaN})$ returns $+0 + i\text{NaN}$.

— `catanh`$(\text{NaN} + iy)$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite y.

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

— `catanh`$(\text{NaN} + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$.

G.3.3.4 The `ccosh` functions

1 — `ccosh`(`conj`(z)) = `conj`(`ccosh`(z)) and `ccosh` is even.

— `ccosh`$(+0 + i0)$ returns $1 + i0$.

— `ccosh`$(+0 + i\infty)$ returns $\text{NaN} \pm i0$ (where the sign of the imaginary part of the result is unspecified) and raises the "invalid" floating-point exception.

— `ccosh`$(+0 + i\text{NaN})$ returns $\text{NaN} \pm i0$ (where the sign of the imaginary part of the result is unspecified).

— `ccosh`$(x + i\infty)$ returns NaN $+ i$NaN and raises the "invalid" floating-point exception, for finite nonzero x.

— `ccosh`$(x + i\text{NaN})$ returns NaN $+ i$NaN and optionally raises the "invalid" floating-point exception, for finite nonzero x.

— `ccosh`$(+\infty + i0)$ returns $+\infty + i0$.

— `ccosh`$(+\infty + iy)$ returns $+\infty\ \operatorname{cis}(y)$, for finite nonzero y.

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

— `ccosh`$(+\infty + i\text{NaN})$ returns $+\infty + i$NaN.

— `ccosh`$(\text{NaN} + i0)$ returns NaN$\pm i0$ (where the sign of the imaginary part of the result is unspecified).

— `ccosh`$(\text{NaN} + iy)$ returns NaN $+ i$NaN and optionally raises the "invalid" floating-point exception, for all nonzero numbers y.

— `ccosh`$(\text{NaN} + i\text{NaN})$ returns NaN $+ i$NaN.

G.3.3.5 The `csinh` functions

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

— `csinh`$(+0 + i0)$ returns $+0 + i0$.

— `csinh`$(+0 + i\infty)$ returns $\pm 0 + i$NaN (where the sign of the real part of the result is unspecified) and raises the "invalid" floating-point exception.

— `csinh`$(+0 + i\text{NaN})$ returns $\pm 0 + i$NaN (where the sign of the real part of the result is unspecified).

— `csinh`$(x + i\infty)$ returns NaN $+ i$NaN and raises the "invalid" floating-point exception, for positive finite x.

— `csinh`$(x + i\text{NaN})$ returns NaN $+ i$NaN and optionally raises the "invalid" floating-point exception, for finite nonzero x.

— `csinh`$(+\infty + i0)$ returns $+\infty + i0$.

— `csinh`$(+\infty + iy)$ returns $+\infty\ \operatorname{cis}(y)$, for positive finite y.

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

— `csinh`$(+\infty + i\text{NaN})$ returns $\pm\infty + i$NaN (where the sign of the real part of the result is unspecified).

— `csinh`$(\text{NaN} + i0)$ returns NaN $+ i0$.

— `csinh`$(\text{NaN} + iy)$ returns NaN $+ i$NaN and optionally raises the "invalid" floating-point exception, for all nonzero numbers y.

— `csinh`$(\text{NaN} + i\text{NaN})$ returns NaN $+ i$NaN.

G.3.3.6 The `ctanh` functions

1 — `ctanh`(`conj`(z)) = `conj`(`ctanh`(z)) and `ctanh` is odd.

— `ctanh`$(+0 + i0)$ returns $+0 + i0$.

— `ctanh`$(0 + i\infty)$ returns $0 + i$NaN and raises the "invalid" floating-point exception.

— `ctanh`$(x + i\infty)$ returns NaN + iNaN and raises the "invalid" floating-point exception, for finite nonzero x.

— `ctanh`$(0 + i\text{NaN})$ returns $0 + i$NaN.

— `ctanh`$(x + i\text{NaN})$ returns NaN + iNaN and optionally raises the "invalid" floating-point exception, for finite nonzero x.

— `ctanh`$(+\infty + iy)$ returns $1 + i0\sin(2y)$, for positive-signed finite y.

— `ctanh`$(+\infty + i\infty)$ returns $1 \pm i0$ (where the sign of the imaginary part of the result is unspecified).

— `ctanh`$(+\infty + i\text{NaN})$ returns $1 \pm i0$ (where the sign of the imaginary part of the result is unspecified).

— `ctanh`$(\text{NaN} + i0)$ returns NaN + $i0$.

— `ctanh`$(\text{NaN} + iy)$ returns NaN + iNaN and optionally raises the "invalid" floating-point exception, for all nonzero numbers y.

— `ctanh`$(\text{NaN} + i\text{NaN})$ returns NaN + iNaN.

G.3.4 Exponential and logarithmic functions

G.3.4.1 The `cexp` functions

1 — `cexp`(`conj`(z)) = `conj`(`cexp`(z)).

— `cexp`$(\pm 0 + i0)$ returns $1 + i0$.

— `cexp`$(x + i\infty)$ returns NaN + iNaN and raises the "invalid" floating-point exception, for finite x.

— `cexp`$(x + i\text{NaN})$ returns NaN + iNaN and optionally raises the "invalid" floating-point exception, for finite x.

— `cexp`$(+\infty + i0)$ returns $+\infty + i0$.

— `cexp`$(-\infty + iy)$ returns $+0 \ \text{cis}(y)$, for finite y.

— `cexp`$(+\infty + iy)$ returns $+\infty \ \text{cis}(y)$, for finite nonzero y.

— `cexp`$(-\infty + i\infty)$ returns $\pm 0 \pm i0$ (where the signs of the real and imaginary parts of the result are unspecified).

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

— `cexp`$(-\infty + i\text{NaN})$ returns $\pm 0 \pm i0$ (where the signs of the real and imaginary parts of the result are unspecified).

— `cexp`$(+\infty + i\text{NaN})$ returns $\pm\infty + i$NaN (where the sign of the real part of the result is unspecified).

— `cexp`$(\text{NaN} + i0)$ returns NaN + $i0$.

— `cexp`$(\text{NaN} + iy)$ returns NaN + iNaN and optionally raises the "invalid" floating-point exception, for all nonzero numbers y.

— `cexp`$(\text{NaN} + i\text{NaN})$ returns NaN + iNaN.

G.3.4.2 The `clog` functions

1
- `clog`$(\texttt{conj}(z)) = \texttt{conj}(\texttt{clog}(z))$.
- `clog`$(-0 + i0)$ returns $-\infty + i\pi$ and raises the "divide-by-zero" floating-point exception.
- `clog`$(+0 + i0)$ returns $-\infty + i0$ and raises the "divide-by-zero" floating-point exception.
- `clog`$(x + i\infty)$ returns $+\infty + \frac{i\pi}{2}$, for finite x.
- `clog`$(x + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite x.
- `clog`$(-\infty + iy)$ returns $+\infty + i\pi$, for finite positive-signed y.
- `clog`$(+\infty + iy)$ returns $+\infty + i0$, for finite positive-signed y.
- `clog`$(-\infty + i\infty)$ returns $+\infty + i\frac{3\pi}{4}$.
- `clog`$(+\infty + i\infty)$ returns $+\infty + \frac{i\pi}{4}$.
- `clog`$(\pm\infty + i\text{NaN})$ returns $+\infty + i\text{NaN}$.
- `clog`$(\text{NaN} + iy)$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite y.
- `clog`$(\text{NaN} + i\infty)$ returns $+\infty + i\text{NaN}$.
- `clog`$(\text{NaN} + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$.

G.3.5 Power and absolute-value functions

G.3.5.1 The `cpow` functions

1 The `cpow` functions raise floating-point exceptions if appropriate for the calculation of the parts of the result, and may also raise spurious floating-point exceptions.[451)]

G.3.5.2 The `csqrt` functions

1
- `csqrt`$(\texttt{conj}(z)) = \texttt{conj}(\texttt{csqrt}(z))$.
- `csqrt`$(\pm 0 + i0)$ returns $+0 + i0$.
- `csqrt`$(x + i\infty)$ returns $+\infty + i\infty$, for all x (including NaN).
- `csqrt`$(x + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite x.
- `csqrt`$(-\infty + iy)$ returns $+0 + i\infty$, for finite positive-signed y.
- `csqrt`$(+\infty + iy)$ returns $+\infty + i0$, for finite positive-signed y.
- `csqrt`$(-\infty + i\text{NaN})$ returns $\text{NaN}\pm i\infty$ (where the sign of the imaginary part of the result is unspecified).
- `csqrt`$(+\infty + i\text{NaN})$ returns $+\infty + i\text{NaN}$.
- `csqrt`$(\text{NaN} + iy)$ returns $\text{NaN} + i\text{NaN}$ and optionally raises the "invalid" floating-point exception, for finite y.
- `csqrt`$(\text{NaN} + i\text{NaN})$ returns $\text{NaN} + i\text{NaN}$.

[451)]This allows `cpow`(z, c) to be implemented as `cexp`$(c\,\texttt{clog}(z))$ without precluding implementations that treat special cases more carefully.

