Complex Arithmetic <complex.h>

WG14/N517 X3J11/95-118 (Draft 12/21/95)

Jim Thomas
Taligent, Inc.
10201 N. DeAnza Blvd.
Cupertino, CA 95014-2233
`jim_thomas@taligent.com`

This is a proposal for adding the CCE <complex.h> header to C9X.

7.x Complex arithmetic <complex.h>

The header `<complex.h>` defines macros and declares functions that support complex arithmetic.

The macro

```
_Imaginary_I
```

expands to an expression with the const-qualified imaginary type that corresponds to the narrowest real floating type used for expression evaluation[1] and with the value of the imaginary unit.

The macro

```
I
```

is defined to be `_Imaginary_I`.

[Positing such a constant is a natural analog to the mathematical notion of augmenting the reals with the imaginary unit. It allows writing imaginary and complex expressions in common mathematical style, for example `x + y*I`. Note that the multiplication here affects translated code, but does not necessitate an actual floating-point multiply, nor does the addition necessitate a floating-point add.

The choice of `I` instead of `i` concedes to the widespread use of the identifier `i` for other purposes.

The scheme for defining `I` and `_Imaginary_I` establishes a degree of uniformity in the designation of the imaginary unit as `I`, but offers flexibility to use a different identifier for the imaginary unit and to use `I` for other purposes. For example, one might follow the inclusion of `<complex.h>` with

```
#define j _Imaginary_I
#undef I
```

An `i` suffix to designate imaginary constants is not required. Multiplication by `I` provides a sufficiently convenient and more generally useful notation for imaginary terms.

[1] If `FLT_EVAL_METHOD` equals 0, 1, or 2, then `_Imaginary_I` has type `float imaginary`, `double imaginary`, or `long double imaginary`, respectively.

/61

Lacking an imaginary type, other proposals required macros in order to create certain special values. For example, an "imaginary" infinity could be created by `CMPLX(0.0,INFINITY)`. With the imaginary type, imaginary infinity is simply the value of `INFINITY*I`. And, in general, the values of *y*I* and *x + y*I*, where *x* and *y* are real floating values, cover all values of the imaginary and complex types, hence eliminating this need for the complex macros.]

7.x.1 Overloading

Subsequent subclauses specify overloading macros which accept real, imaginary, or complex arguments. A floating type is characterized by its kind (real, imaginary, complex) and its corresponding real type (`float`, `double`, `long double`). Use of a macro invokes a function whose type for each overloading parameter has the same kind as the corresponding argument and has the corresponding real type that is the wider of

— the corresponding real types for all floating arguments to the overloading parameters
— the narrowest real floating format used for expression evaluation

For the returned value, the kind is determined by the function and the argument kind(s) as specified below, and the corresponding real type matches the corresponding real type of the overloading parameters.

Examples

1. Subclause 7.x.2 specifies that the overloading macro for the `cos` function accepts a real, imaginary, or complex argument, that `cos` of a complex is complex, and that `cos` of an imaginary is real. The following table shows the result type, given an expression evaluation method (characterized by `FLT_EVAL_METHOD`) and an argument type.

`FLT_EVAL_METHOD`	argument type	result type
0	`float imaginary`	`float`
0	`double imaginary`	`double`
0	`long double imaginary`	`long double`
0	`float complex`	`float complex`
0	`double complex`	`double complex`
0	`long double complex`	`long double complex`
1	`float imaginary`	`double`
1	`double imaginary`	`double`
1	`long double imaginary`	`long double`
1	`float complex`	`double complex`
1	`double complex`	`double complex`
1	`long double complex`	`long double complex`
2	`float imaginary`	`long double`
2	`double imaginary`	`long double`
2	`long double imaginary`	`long double`
2	`float complex`	`long double complex`
2	`double complex`	`long double complex`
2	`long double complex`	`long double complex`

2. For the `pow` function

```
double pow(double x, double y);
```

162

both `x` and `y` are overloading parameters. Subclause 7.x.2 specifies that the overloading macro for `pow` accepts real, imaginary, or complex arguments and that `pow` is complex if either of its arguments is imaginary or complex. With `FLT_EVAL_METHOD` equal 1 and

```
#include <complex.h>
float complex a;
double complex r;
long b;
/*...*/
r = pow(a, b);
```

the call to `pow` behaves like an invocation of a function such as

```
double complex _Pow(double imaginary x, double y);
```

The implementation doesn't actually require different internal functions for all the combinations of parameter kinds, as

```
double complex _Pow(double complex x, double complex y);
```

would suffice whenever the corresponding real type is `double`. However, more overloads might allow more efficient implementation.

7.x.2 `<math.h>` functions

When the header `<complex.h>` is included, the overloading macros for these `<math.h>` functions accept imaginary and complex (as well as real) arguments:

```
acos    cos    acosh    cosh    exp    fabs
asin    sin    asinh    sinh    log
atan    tan    atanh    tanh    sqrt
```

The return type for `fabs` is always real. Otherwise, for each overloading macro, if the argument is complex then so is the return type. If the argument is imaginary then the return type is real, imaginary, or complex, as appropriate for the particular function: if the argument is imaginary, then the return types of `cos` and `cosh` are real; the return types of `sin`, `tan`, `sinh`, `tanh`, `asin`, and `atanh` are imaginary; and the return types of the others are complex.

7.x.2.1 Branch cuts

Some of the functions below have branch cuts, across which the function is discontinuous. For implementations with a signed zero (including all IEEE implementations), the sign of zero distinguishes one side of a cut from another so the function is continuous (except for format limitations) as the cut is approached from either side. For example, for the square root function, which has a branch cut along the negative real axis, the top of the cut, with imaginary part +0, maps to the positive imaginary axis, and the bottom of the cut, with imaginary part -0, maps to the negative imaginary axis.

Implementations with an unsigned zero cannot distinguish the sides of branch cuts. They map a cut so the function is continuous as the cut is approached coming around the finite endpoint of the cut in a counter clockwise direction. (Branch cuts for the functions

163

specified here have just one finite endpoint.) For example, for the square root function, coming counter clockwise around the finite endpoint of the cut along the negative real axis approaches the cut from above, so the cut maps to the positive imaginary axis.

7.x.2.2 The `acos` macro

For an imaginary or complex argument `z`, the `acos` macro computes the arc cosine of `z`, with branch cuts outside the interval [-1, 1] along the real axis; it returns the arc cosine of `z`, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [0, π] along the real axis.

7.x.2.3 The `asin` macro

For an imaginary or complex argument `z`, the `asin` macro computes the arc sine of `z`, with branch cuts outside the interval [-1, 1] along the real axis; it returns the arc sine of `z`, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [-π/2, π/2] along the real axis.

7.x.2.4 The atan macro

For an imaginary or complex argument `z`, the `atan` macro computes the arc tangent of `z`, with branch cuts outside the interval [-*i*, *i*] along the imaginary axis; it returns the arc tangent of `z`, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [-π/2, π/2] along the real axis.

7.x.2.5 The cos macro

For an imaginary or complex argument `z`, the `cos` macro computes and returns the cosine of `z`.

7.x.2.6 The sin macro

For an imaginary or complex argument `z`, the `sin` macro computes and returns the complex sine of `z`.

7.x.2.7 The tan macro

For an imaginary or complex argument `z`, the `tan` macro computes and returns the tangent of `z`.

7.x.2.8 The acosh macro

For an imaginary or complex argument `z`, the `acosh` macro computes the arc hyperbolic cosine of `z`, with a branch cut at values less than 1 along the real axis; it returns the arc hyperbolic cosine of `z`, in the range of a half-strip of non-negative values along the real axis and in the interval [-*i*π, *i*π] along the imaginary axis.

7.x.2.9 The asinh macro

For an imaginary or complex argument `z`, the `asinh` macro computes the arc hyperbolic sine of `z`, with branch cuts outside the interval [-*i*, *i*] along the imaginary axis; it returns the complex arc hyperbolic sine of `z`, in the range of a strip mathematically unbounded along the real axis and in the interval [-*i*π/2, *i*π/2] along the imaginary axis.

7.x.2.10 The atanh macro

For an imaginary or complex argument `z`, the `atanh` macro computes the arc hyperbolic tangent of `z`, with branch cuts outside the interval [-1, 1] along the real axis; it returns the complex arc hyperbolic tangent of `z`, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, i\pi/2]$ along the imaginary axis.

7.x.2.11 The cosh macro

For an imaginary or complex argument `z`, the `cosh` macro computes and returns the hyperbolic cosine of `z`.

7.x.2.12 The sinh macro

For an imaginary or complex argument `z`, the `sinh` macro computes and returns the hyperbolic sine of `z`.

7.x.2.13 The tanh macro

For an imaginary or complex argument `z`, the `tanh` macro computes and returns the hyperbolic tangent of `z`.

7.x.2.14 The exp macro

For an imaginary or complex argument `z`, the `exp` macro computes and returns the complex base-e exponential of `z`.

7.x.2.15 The log macro

For an imaginary or complex argument `z`, the `log` macro computes the natural (base-e) logarithm of `z`, with a branch cut along the negative real axis; it returns the complex natural logarithm of `z`, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi, i\pi]$ along the imaginary axis.

7.x.2.16 The sqrt macro

For an imaginary or complex argument `z`, the `sqrt` macro computes the square root of `z`, with a branch cut along the negative real axis; it returns the square root of `z`, in the range of the right half-plane.

7.x.2.17 The fabs macro

For an imaginary or complex argument `z`, the `fabs` macro computes and returns the absolute value (also called norm, modulus, or magnitude) of `z`.

7.x.2.18 The pow macro

If either the first argument `x` or the second argument `y` is imaginary or complex, the `pow` macro computes the power function x^y, with a branch cut for the first parameter along the negative real axis; it returns the complex power function x^y.

7.x.3 Complex-specific functions

The header `<complex.h>` declares the following functions which pertain specifically to complex arithmetic. Each function has an overloading macro that accepts an argument of real, imaginary, or complex type. Determination of the type for an overloading parameter is as described in 7.x.2.1. Suppressing the overloading macro makes available a function with the prototype in the synopsis.

7.x.3.1 The arg function

Synopsis

```
#include <complex.h>
double arg(double complex z);
```

Type determination by the overloading macro follows the same pattern as for `fabs`.

Description

The `arg` function computes the argument or phase angle of `z`, with a branch cut along the negative real axis.

Returns

The `arg` function returns the argument or phase angle of `z`, in the range [$-\pi$, π].

7.x.3.2 The conj function

Synopsis

```
#include <complex.h>
double complex conj(double complex z);
```

The kind (real, imaginary, or complex) for the type of the function invoked by the overloading macro matches the kind of the argument.

Description

The `conj` function computes the complex conjugate of `z`, by reversing the sign of its imaginary part, if any.

[Note that `conj(3.0)` yields 3.0, not 3.0 - 0.0.]

Returns

The `conj` function returns the complex conjugate of `z`.

7.x.3.3 The imag function

Synopsis

```
#include <complex.h>
double imag(double complex z);
```

166

Type determination by the overloading macro follows the same pattern as for **`fabs`**.

Description

The **`imag`** function computes the imaginary part of **`z`**.

Returns

The **`imag`** function returns the imaginary part of **`z`**.

7.x.3.4 The proj function

Synopsis

```
#include <complex.h>
double complex proj(double complex z);
```

The return type is real if the argument is real; the return type is complex if the argument is complex or imaginary.

Description

The `proj` function computes a projection of `z` onto the Riemann sphere: `z` projects to `z` except that all infinities (even ones with one infinite part and one NaN part) project to positive infinity on the real axis.

Returns

The `proj` function returns a projection of `z` onto the Riemann sphere.

[Two topologies are commonly used in complex mathematics: the complex plane with its continuum of infinities and the Riemann sphere with its single infinity. The complex plane is better suited for transcendental functions, the Riemann sphere for algebraic functions. The complex types with their multiplicity of infinities provide a useful (though imperfect) model for the complex plane. The `proj` function helps model the Riemann sphere by mapping all infinities to one, and should be used just before any operation, especially comparisons, that might give spurious results for any of the other infinities.

Note that a complex value with one infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is infinite, the complex value is infinite independent of the value of the other part. For the same reason, **`fabs`** returns an infinity if its argument has an infinite part and a NaN part.]

7.x.3.5 The real function

Synopsis

```
#include <complex.h>
double real(double complex z);
```

Type determination by the overloading macro follows the same pattern as for **`imag`** (7.x.3.3).

Description

The **`real`** function computes the real part of **`z`**.

Returns

The **`real`** function returns the real part of **`z`**.

168