INTERNATIONAL STANDARD

ISO/IEC 10967-3

First edition
2006-05-01

Information technology — Language independent arithmetic —

Part 3: Complex integer and floating point arithmetic and complex elementary numerical functions

Technologies de l'information — Arithmétique indépendante des langages —

Partie 3: Arithmétique des nombres complexes entiers et en virgule flottante et fonctions numériques élémentaires complexes

Reference number
ISO/IEC 10967-3:2006(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

IISO/IEC 10967-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

ISO/IEC 10967 consists of the following parts, under the general title *Information technology — Language independent arithmetic*:

— *Part 1: Integer and floating point arithmetic*

— *Part 2: Elementary numerical functions*

— *Part 3: Complex integer and floating point arithmetic and complex elementary numerical functions*

Introduction

The aims

Portability is a key issue for scientific and numerical software in today's heterogeneous computing environment. Such software may be required to run on systems ranging from personal computers to high performance pipelined vector processors and massively parallel systems, and the source code may be ported between several programming languages.

Part 1 of ISO/IEC 10967 specifies the basic properties of integer and floating point types that can be relied upon in writing portable software.

Part 2 of ISO/IEC 10967 specifies a number of additional operations for integer and floating point types, in particular specifications for numerical approximations to elementary functions on reals.

The content

The content of this document is based on part 1 and part 2, and extends part 1's and part 2's specifications to also cover operations approximating imaginary-integer and complex-integer arithmetic, imaginary-real and complex-real arithmetic, as well as imaginary-real and complex-real elementary functions.

The numerical functions covered by this document are computer approximations to mathematical functions of one or more imaginary or complex arguments. Accuracy versus performance requirements often vary with the application at hand. This is recognised by recommending that implementors support more than one library of these numerical functions. Various documentation and (program available) parameters requirements are specified to assist programmers in the selection of the library best suited to the application at hand.

The benefits

Adoption and proper use of this document can lead to the following benefits.

For programming language standards it will be possible to define their arithmetic semantics more precisely without preventing the efficient implementation of the language on a wide range of machine architectures.

Programmers of numeric software will be able to assess the portability of their programs in advance. Programmers will be able to trade off program design requirements for portability in the resulting program. Programs will be able to determine (at run time) the crucial numeric properties of the implementation. They will be able to reject unsuitable implementations, and (possibly) to correctly characterize the accuracy of their own results. Programs will be able to detect (and possibly correct for) exceptions in arithmetic processing.

Procurers of numerical programs will find it easier to determine whether a (properly documented) application program is likely to execute satisfactorily on the platform used. This can be done by comparing the documented requirements of the program against the documented properties of the platform.

Finally, end users of numeric application packages will be able to rely on the correct execution of those packages. That is, for correctly programmed algorithms, the results are reliable if and only if there is no notification.

Information technology — Language independent arithmetic —

Part 3: Complex integer and floating point arithmetic and complex elementary numerical functions

1 Scope

This part of ISO/IEC 10967 specifies the properties of numerical approximations for complex arithmetic operations and many of the complex elementary numerical functions available in a variety of programming languages in common use for mathematical and numerical applications.

An implementor may choose any combination of hardware and software support to meet the specifications of this document. It is the computing environment, as seen by the programmer/user, that does or does not conform to the specifications.

The term *implementation* (of this part) denotes the total computing environment pertinent to this part, including hardware, language processors, subroutine libraries, exception handling facilities, other software, and documentation.

1.1 Inclusions

The specifications of part 1 and part 2 are included by reference in this part.

This document provides specifications for properties of complex and imaginary integer datatypes and floating point datatypes, basic operations on values of these datatypes as well as for some numerical functions for which operand or result values are of imaginary or complex integer datatypes or imaginary or complex floating point datatypes constructed from integer and floating point datatypes satisfying the requirements of part 1 (ISO/IEC 10967-1). Boundaries for the occurrence of exceptions and the maximum error allowed are prescribed for each specified operation. Also the result produced by giving a special value operand, such as an infinity, or a NaN (not-a-number), is prescribed for each specified floating point operation.

This document provides specifications for:

a) Basic imaginary integer and complex integer operations.

b) Non-transcendental imaginary floating point and Cartesian complex floating point operations.

c) Exponentiation, logarithm, radian trigonometric, and hyperbolic operations for imaginary floating point and Cartesian complex floating point.

This document also provides specifications for:

d) The results produced by an included floating point operation when one or more operand values include IEC 60559 special values.

e) Program-visible parameters that characterise certain aspects of the operations.

1.2 Exclusions

This document provides no specifications for:

a) Datatypes and operations for polar complex floating point. This part neither requires nor excludes the presence of such polar complex datatypes and operations.

b) Numerical functions whose operands are of more than one datatype, except certain imaginary/complex combinations. This part neither requires nor excludes the presence of such "mixed operand" operations.

c) A complex interval datatype, or the operations on such datatypes. This part neither requires nor excludes such datatypes or operations.

d) A complex fixed point datatype, or the operations on such datatypes. This part neither requires nor excludes such datatypes or operations.

e) A complex rational datatype, or the operations on such datatypes. This part neither requires nor excludes such datatypes or operations.

f) Matrix, statistical, or symbolic operations (on suitable datatypes). This part neither requires nor excludes such operations or datatypes.

g) The properties of complex arithmetic datatypes that are not related to the numerical process, such as the representation of values on physical media.

h) The properties of integer and floating point datatypes that properly belong in programming language standards or other specifications. Examples include:

   1) the syntax of numerals and expressions in the programming language,
   2) the syntax used for parsed (input) or generated (output) character string forms for numerals by any specific programming language or library,
   3) the precedence of operators in the programming language,
   4) the rules for assignment, parameter passing, and returning value,
   5) the presence or absence of automatic datatype coercions,
   6) the consequences of applying an operation to values of improper datatype, or to uninitialised data.

Furthermore, this part does not provide specifications for how the operations should be implemented or which algorithms are to be used for the various operations.

2 Conformity

It is expected that the provisions of this document will be incorporated by reference and further defined in other International Standards; specifically in programming language standards and in binding standards.

A binding standard specifies the correspondence between one or more of the arithmetic datatypes, parameters, and operations specified in this document and the concrete language syntax of some programming language. More generally, a binding standard specifies the correspondence between certain datatypes, parameters, and operations and the elements of some arbitrary computing entity. A language standard that explicitly provides such binding information can serve as a binding standard.

When a binding standard for a language exists, an implementation shall be said to conform to this document if and only if it conforms to the binding standard. In case of conflict between a binding standard and this document, the specifications of the binding standard take precedence.

When a binding standard requires only a subset of the imaginary or complex integer or imaginary or complex floating point datatypes specified in this document, an implementation remains free to conform to this document with respect to other datatypes independently of that binding standard.

When a binding standard requires only a subset of the operations specified in this document, an implementation remains free to conform to this document with respect to other operations, independently of that binding standard.

When no binding standard between a language and some datatypes or operations specified in this document exists, an implementation conforms to this document if and only if it provides one or more datatypes and one or more operations that together satisfy all the requirements of clauses 5 through 8 that are relevant to those datatypes and operations. The implementation shall then document the binding.

Conformity to this document is always with respect to a specified set of datatypes and set of operations. Conformity to this document implies conformity to part 1 and part 2 of ISO/IEC 10967 for the integer and floating point datatypes and operations used. Under certain circumstances, conformity to IEC 60559 is implied by conformity to part 1 of ISO/IEC 10967.

An implementation is free to provide arithmetic datatypes and arithmetic operations that do not conform to this document, or that are beyond the scope of this document. The implementation shall not claim or imply conformity to this document for such datatypes or operations.

An implementation is permitted to have modes of operation that do not conform to this document. A conforming implementation shall specify how to select the modes of operation that ensure conformity.

NOTES

1 Language bindings are essential. Clause 8 requires an implementation to document a binding if no binding standard exists. See Annex **??** for an example of a conformity statement, and Annex C for suggested language bindings.

2 A complete binding for this document will include (explicitly or by reference) a binding for part 2 and part 1 as well, which in turn may include (explicitly or by reference) a binding for IEC 60559 as well.

3 This document does not require a particular set of operations to be provided. It is not possible to conform to this document without specifying to which datatypes and set of operations (and modes of operation) conformity is claimed.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems.*

ISO/IEC 10967-1, *Information technology – Language independent arithmetic – Part 1: Integer and floating point arithmetic.*

ISO/IEC 10967-2, *Information technology – Language independent arithmetic – Part 2: Elementary numerical functions.*

4 Symbols and definitions

4.1 Symbols

4.1.1 Sets and intervals

In this document, $\mathcal{Z}$ denotes the set of mathematical integers, $\mathcal{G}$ denotes the set of complex integers. $\mathcal{R}$ denotes the set of classical real numbers, and $\mathcal{C}$ denotes the set of complex numbers over $\mathcal{R}$. Note that $\mathcal{Z} \subset \mathcal{R} \subset \mathcal{C}$, and $\mathcal{Z} \subset \mathcal{G} \subset \mathcal{C}$.

The conventional notation for set definition and manipulation is used.

The following notation for intervals is used:

$[x, z]$ designates the interval $\{y \in \mathcal{R} \mid x \leqslant y \leqslant z\}$,
$]x, z]$ designates the interval $\{y \in \mathcal{R} \mid x < y \leqslant z\}$,
$[x, z[$ designates the interval $\{y \in \mathcal{R} \mid x \leqslant y < z\}$, and
$]x, z[$ designates the interval $\{y \in \mathcal{R} \mid x < y < z\}$.

NOTE – The notation using a round bracket for an open end of an interval is not used, for the risk of confusion with the notation for pairs.

4.1.2 Operators and relations

All prefix and infix operators have their conventional (exact) mathematical meaning. The conventional notation for set definition and manipulation is also used. In particular:

$\Rightarrow$ and $\Leftrightarrow$ for logical implication and equivalence
$+$, $-$, $/$, $|x|$, conj, $\lfloor x \rfloor$, $\lceil x \rceil$, and round(x) on complex values
$\cdot$ for multiplication on complex values
$<$, $\leqslant$, $\geqslant$, and $>$ between real values
$=$ and $\neq$ between real, complex, as well as special values

$\cup$, $\cap$, $\in$, $\notin$, $\subset$, $\subseteq$, $\nsubseteq$, $\neq$, and $=$ with sets
$\times$ for the Cartesian product of sets
$\rightarrow$ for a mapping between sets
$|$ for the divides relation between complex integer values (in $\mathcal{G}$)
$\tilde{\imath}$ as the imaginary unit ($\tilde{\imath}^2 = -1$)
$\mathfrak{Re}$ to extract the real part of a complex value (in $\mathcal{C}$)
$\mathfrak{Im}$ to extract the imaginary part of a complex value (in $\mathcal{C}$)

NOTE 1 – $\approx$ is used informally, in notes and the rationale.

For $x \in \mathcal{C}$, the notation $\lfloor x \rfloor$ designates the component-wise largest complex integer where each part is not greater than the corresponding part of x:

$$\lfloor x \rfloor \in \mathcal{G} \text{ and } \mathfrak{Re}(x) - 1 < \mathfrak{Re}(\lfloor x \rfloor) \leqslant \mathfrak{Re}(x) \text{ and } \mathfrak{Im}(x) - 1 < \mathfrak{Im}(\lfloor x \rfloor) \leqslant \mathfrak{Im}(x)$$

the notation $\lceil x \rceil$ designates the component-wise smallest complex integer where each part is not less than the corresponding part of x:

$$\lceil x \rceil \in \mathcal{G} \text{ and } \mathfrak{Re}(x) \leqslant \mathfrak{Re}(\lceil x \rceil) < \mathfrak{Re}(x) + 1 \text{ and } \mathfrak{Im}(x) \leqslant \mathfrak{Im}(\lceil x \rceil) < \mathfrak{Im}(x) + 1$$

and the notation $\text{round}(x)$ designates the complex integer closest to x:

$$\text{round}(x) \in \mathcal{G} \text{ and}$$
$$\mathfrak{Re}(x) - 0.5 \leqslant \mathfrak{Re}(\text{round}(x)) \leqslant \mathfrak{Re}(x) + 0.5 \text{ and } \mathfrak{Im}(x) - 0.5 \leqslant \mathfrak{Im}(\text{round}(x)) \leqslant \mathfrak{Im}(x) + 0.5$$

where in case $\mathfrak{Re}(x)$ or $\mathfrak{Im}(x)$ is exactly half-way between two integers, the even integer is the result component.

The *divides* relation ($|$) on complex integers tests whether a complex integer i divides a complex integer j exactly:

$$i|j \quad \Leftrightarrow \quad (i \neq 0 \text{ and } i \cdot n = j \text{ for some } n \in \mathcal{G})$$

NOTE 2 – $i|j$ is true exactly when j/i is defined and $j/i \in \mathcal{G}$).

4.1.3 Mathematical functions

This document specifies properties for a number of operations numerically approximating some of the elementary functions. The following ideal mathematical functions are defined in chapter 4 of the *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* [11] (e is the Napierian base):

e^x, x^y, $\sqrt{x}$, $|x|$, ln, $\log_b$,
sin, cos, tan, cot, sec, csc, arcsin, arccos, arctan, arccot, arcsec, arccsc,
sinh, cosh, tanh, coth, sech, csch, arcsinh, arccosh, arctanh, arccoth, arcsech, arccsch.

Many of the inverses above are multi-valued. The selection of which value to return, the principal value, so as to make the inverses into functions, is done in the conventional way. E.g., $\sqrt{x} \in [0, \infty[$ when $x \in [0, \infty[$.

Part 2 defines the mathematical *arc* function, which is used in this part to define the mathematical complex *signum* function.

4.1.4 Exceptional values

ISO/IEC 10967 uses the following five exceptional values:

underflow: the result has an absolute value that is smaller than the smallest positive normalised value, and the result may be inexact. This notification need not be given if the result is exact. In particular, if the result is zero, or exact and IEC 60559 is conformed to and trapping is not enabled.

overflow: the result, after rounding, has a magnitude larger than the maximum value representable in the result datatype.

infinitary: the operation result is (exactly) infinite (in its real or its imaginary part), while all the arguments (their real and imaginary parts) are finite. This is to say that the corresponding mathematical function has a pole at the finite argument point.

invalid: the operation is undefined, while not infinitary, for the given arguments.

absolute_precision_underflow: indicate that the argument is such that the density of representable argument values is too small in the neighbourhood of the given argument value for a numeric result to be considered appropriate to return. Used for operations that approximate trigonometric functions (part 2 and part 3), and hyperbolic and exponentiation functions (part 3).

The exceptional value **inexact** is not specified in ISO/IEC 10967, but IEC 60559 conforming implementations will provide it. It should then be used also for operations approximating transcendental functions, when the returned result may be approximate. This part of ISO/IEC 10967 does not specify when it is appropriate to return this exceptional value, but does specify an appropriate continuation value (see Definition 4.2.4). Thus, v is specified by ISO/IEC 10967 when v or **inexact**(v) should be returned by implementations that are based on IEC 60559, and ISO/IEC 10967 specifies **underflow**(v) when **underflow**(v) or **inexact**(**underflow**(v)) should be returned by implementations that are based on IEC 60559. Ideally, **underflow** should imply **inexact**.

For the exceptional values, a continuation value may be given in this part in parenthesis after the exceptional value.

4.1.5 Datatypes and special values

The datatype **Boolean** consists of the two values **true** and **false**.

NOTE 1 – Mathematical relations are true or false (or undefined, if an operand is undefined), which are not values. In contrast, **true** and **false** are values in **Boolean**.

Square brackets are used to write finite sequences of values. [] is the sequence containing no values. $[s]$, is the sequence of one value, s. $[s_1, s_2]$, is the sequence of two values, s_1 and then s_2, etc. The colon operator is used to prepend a value to a sequence: $x : [x_1, ..., x_n] = [x, x_1, ..., x_n]$. $[S]$, where S is a set, denotes the set of finite sequences, where each value in a sequence is in S.

NOTE 2 – It is always clear from context, in the text of this part, if $[X]$ is a sequence of one element, or the set of sequences with values from X. It is also clear from context if $[x_1, x_2]$ is a sequence of two values or an interval.

Integer datatypes and floating point datatypes are defined in part 1. Let I be the non-special value set for an integer datatype conforming to part 1. Let F be the non-special value set for a floating point datatype conforming to part 1 and part 2. The following symbols used in this part are defined in part 1 or part 2:

Exceptional values:
: **underflow**, **overflow**, **infinitary**, **invalid**, and **absolute_precision_underflow**.

Integer helper function:
: $result_I$.

Integer operations:
: eq_I, neq_I, lss_I, leq_I, gtr_I, geq_I, neg_I, add_I, sub_I, mul_I, abs_I, max_I, min_I, max_seq_I, min_seq_I, $divides_I$, $quot_I$, mod_I, $ratio_I$, $residue_I$, $group_I$, and pad_I.

Integer conversion operation:
: $convert_{I \to I'}$.

Floating point parameters:
: r_F, p_F, $emin_F$, $emax_F$, $denorm_F$, and iec_559_F.

Derived floating point constants:
: $fmax_F$, $fmin_F$, $fminN_F$, $fminD_F$, and $epsilon_F$.

Floating point value sets related to F:
: F^*, F_D, F_N, G_F, $F^{2\cdot\pi}$, and F^u (for a given u).

Floating point helper functions:
: up_F, $down_F$, $nearest_F$, $result_F$, $result^*_F$, no_result_F, and $no\_result2_F$.

Floating point operations from part 1:
: eq_F, neq_F, lss_F, leq_F, gtr_F, geq_F, neg_F, add_F, sub_F, mul_F, div_F, abs_F, max_F, $mmax_F$, min_F, $mmin_F$, max_seq_F, $mmax\_seq_F$, min_seq_F, $mmin\_seq_F$, $floor_F$, $ceiling_F$, and $rounding_F$.

Floating point operations from part 2:
: $sqrt_F$, $hypot_F$, $power_{F,I}$, exp_F, $power_F$, ln_F, $logbase_F$,
rad_F, sin_F, cos_F, tan_F, cot_F, sec_F, csc_F,
$arcsin_F$, $arccos_F$, $arctan_F$, $arccot_F$, $arcsec_F$, $arccsc_F$, arc_F, $arcu_F$,
$sinu_F$, $cosu_F$, $sinh_F$, $cosh_F$, $tanh_F$, $coth_F$, $sech_F$, $csch_F$,
$arcsinh_F$, $arccosh_F$, $arctanh_F$, $arccoth_F$, $arcsech_F$, and $arccsch_F$.

Angular parameters and maximum error parameters from part 2:
: $big\_angle\_r_F$, $big\_angle\_u_F$, $max\_error\_tan_F$, and $max\_error\_tanu_F$.

Floating point conversion operations:
: $convert_{F \to F'}$, $convert_{F \to D}$, and $convert_{D \to F}$.

Approximation helper functions from part 2:
: exp^*_F, $power^*_F$, ln^*_F, $logbase^*_F$,
sin^*_F, cos^*_F, tan^*_F, cot^*_F, sec^*_F, csc^*_F,
$arcsin^*_F$, $arccos^*_F$, $arctan^*_F$, $arccot^*_F$, $arcsec^*_F$, $arccsc^*_F$,
$sinh^*_F$, $cosh^*_F$, $tanh^*_F$, $coth^*_F$, $sech^*_F$, $csch^*_F$,
$arcsinh^*_F$, $arccosh^*_F$, $arctanh^*_F$, $arccoth^*_F$, $arcsech^*_F$, and $arccsch^*_F$.

Floating point datatypes that conform to part 1 shall, for use with this part, like for part 2, have a value for the parameter p_F such that $p_F \geqslant 2 \cdot \max\{1, \log_{r_F}(2 \cdot \pi)\}$, and have a value for the parameter $emin_F$ such that $emin_F \leqslant -p_F - 1$.

NOTES

3 This implies that $fminN_F < 0.5 \cdot epsilon_F / r_F$ in this part, rather than just $fminN_F \leqslant epsilon_F$.

4 These extra requirements, which do not limit the use of any existing floating point datatype, are made so that angles in radians are not too degenerate within the first two cycles, plus and minus, when represented in F.

5 F should also be such that $p_F \geqslant 2 + \log_{r_F}(1000)$, to allow for a not too coarse angle resolution anywhere in the interval $[-big\_angle\_r_F, big\_angle\_r_F]$ with the default value for $big\_angle\_r_F$. See Clause 5.3.9 of part 2.

The following symbols represent special values defined in IEC 60559 and are used in this part:

−0, **+∞**, **−∞**, **qNaN**, and **sNaN**.

These floating point values are not part of the set F, but if iec_559_F has the value **true**, these values are included in the floating point datatype corresponding to F.

NOTE 6 – This part uses the above five special values for compatibility with IEC 60559. In particular, the symbol **−0** (in bold) is not the application of (mathematical) unary − to the value 0, and is a value logically distinct from 0.

The specifications cover the results to be returned by an operation if given one or more of the IEC 60559 special values **−0**, **+∞**, **−∞**, or NaNs as input values. These specifications apply only to systems which provide and support these special values. If an implementation is not capable of representing a **−0** result or continuation value, 0 shall be used as the actual result or continuation value. If an implementation is not capable of representing a prescribed result or continuation value of the IEC 60559 special values **+∞**, **−∞**, or **qNaN**, the actual result or continuation value is binding or implementation defined.

If and only if an implementation is not capable of representing **−0**:

a) a 0 as the imaginary part of a complex argument (in c(F), see Clause 4.1.6) shall be interpreted as if it was **−0** if and only if the real part of that complex argument is greater than or equal to zero, and

b) a 0 as the real part of a complex argument (in c(F), see Clause 4.1.6) shall be interpreted as if it was **−0** if and only if the imaginary part of the complex argument is less than zero.

NOTES

7 Reinterpreting 0 as **−0** as required above is needed to follow the sign rules for inverse trigonometric and inverse hyperbolic operations, as well as the exact relations between trigonometric and hyperbolic operations also for argument parts (real and imaginary) that have a zero as value.

8 The rule above is sometimes referred to as *continuous when approaching an axis in a counterclockwise path*. This fits both with Common Lisp and C requirements when zeroes don't have a distinguishable sign.

9 For consistency, this rule also has implications for the operations that implicitly or explicitly extract an implicit real or implicit imaginary part (see for example the specifications for the $re_{\mathrm{i}(F)}$ and im_F operations in Clause 5.2.5).

4.1.6 Complex value constructors and complex datatype constructors

Let X be a set containing values in $\mathcal{R}$, and possibly also containing special values (such as IEC 60559 special values).

i(X) is a subset of values in an imaginary datatype, constructed from the datatype corresponding to X. $\hat{\mathbf{\imath}}\boldsymbol{\cdot}$ is a prefix constructor that takes one parameter.

$$\mathrm{i}(X) = \{\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y \mid y \in X\}$$

c(X) is a subset of values in a complex datatype, constructed from the datatype corresponding to X. $\mathbf{+\hat{\imath}\cdot}$ is an infix constructor that takes two parameters.

$$\mathrm{c}(X) = \{x \mathbin{\mathbf{+\hat{\imath}\cdot}} y \mid x, y \in X\}$$

NOTES

1 While $\mathbf{\hat{\imath}\cdot}$ and $\mathbf{+\hat{\imath}\cdot}$ (note that they are written in bold) have an appearance of being the imaginary unit together with the plus and times operators, that is not the case. For instance, $\mathbf{\hat{\imath}\cdot}\,2$ is an element of $\mathrm{i}(X)$ (if $2 \in X$), but not of $\mathcal{G}$ or $\mathcal{C}$. $\tilde{\imath} \cdot 2$, on the other hand, is an expression that denotes an element of $\mathcal{G}$ (and $\mathcal{C}$), but neither of $\mathrm{i}(X)$ nor $\mathrm{c}(X)$. Further, e.g., $4 + \tilde{\imath} \cdot 0 = 4$, but $4 \mathbin{\mathbf{+\hat{\imath}\cdot}} 0 \neq 4$.

2 A constructor that takes one argument is a one-tuple tag. A constructor that takes two arguments is a two-tuple (pair) tag. The arguments are part of the resulting value.

3 The tuple tags need not be explicitly represented in implementations. But if represented, there should be different tags for different argument types (which is not needed in this text).

Some of the helper function signatures use $\mathcal{C}_F$, where

$$\mathcal{C}_F = \{x + \tilde{\imath} \cdot y \mid x, y \in F\}$$

where $F \subset \mathcal{R}$.

4.2 Definitions of terms

For the purposes of this document, the following terms and definitions apply.

4.2.1
accuracy
The closeness between the true mathematical result and a computed result.

4.2.2
arithmetic datatype
A datatype whose non-special values are members of $\mathcal{Z}$, $\mathrm{i}(\mathcal{Z})$, $\mathrm{c}(\mathcal{Z})$, $\mathcal{R}$, $\mathrm{i}(\mathcal{R})$, or $\mathrm{c}(\mathcal{R})$.

NOTE 1 – $\mathrm{i}(\mathcal{Z})$ corresponds to imaginary integer values in $\mathcal{G}$. $\mathrm{c}(\mathcal{Z})$ corresponds to complex integer values in $\mathcal{G}$. $\mathrm{i}(\mathcal{R})$ corresponds to imaginary values in $\mathcal{C}$. $\mathrm{c}(\mathcal{R})$ corresponds to complex values in $\mathcal{C}$.

4.2.3
binding
Documentation that specifies which syntax (most often, operations or functions) of a programming language, or programming library, that is used for an operation specified in this part.

4.2.4
continuation value
A computational value used as the result of an arithmetic operation when an exception occurs. Continuation values are intended to be used in subsequent arithmetic processing. A continuation value can be a (in the datatype representable) value in $\mathcal{Z}$, $\mathrm{i}(\mathcal{Z})$, $\mathrm{c}(\mathcal{Z})$, $\mathcal{R}$, $\mathrm{i}(\mathcal{R})$, $\mathrm{c}(\mathcal{R})$, or a value where one or both parts of the value is an IEC 60559 special value (e.g. $\mathbf{-0}$, $\mathbf{\hat{\imath}\cdot}\,(\mathbf{-\infty})$), or $\mathbf{+\infty} \mathbin{\mathbf{+\hat{\imath}\cdot}} \mathbf{qNaN}$). (Contrast with *exceptional value*. See Clause 6.1.2 of part 1.)

4.2.5
denormalisation loss
A larger than normal rounding error caused by the fact that subnormal values have less than full precision. (See Clause 5.2.5 of part 1 for a full definition.)

4.2.6
error
⟨in computed value⟩ The difference between a computed value and the mathematically correct value. Used in phrases like "rounding error" or "error bound".

4.2.7
error
⟨computation gone awry⟩ A synonym for *exception* in phrases like "error message" or "error output". Error and exception are not synonyms in any other contexts.

4.2.8
exception
The inability of an operation to return a suitable finite numeric result from finite arguments. This might arise because no such finite result exists mathematically, or because the mathematical result cannot be represented with sufficient accuracy.

4.2.9
exceptional value
A non-numeric value produced by an arithmetic operation to indicate the occurrence of an exception. Exceptional values are not used in subsequent arithmetic processing. (See Clause 5 of part 1.)

NOTES

2 Exceptional values are used as part of the defining formalism only. With respect to this part, they do not represent values of any of the datatypes described. There is no requirement that they be represented or stored in the computing system.

3 Exceptional values are not to be confused with the NaNs and infinities defined in IEC 60559. Contrast this definition with that of *continuation value* above.

4.2.10
helper function
A function used solely to aid in the expression of a requirement. Helper functions are not visible to the programmer, and are not required to be part of an implementation.

4.2.11
implementation (of this part)
The total arithmetic environment presented to a programmer, including hardware, language processors, exception handling facilities, subroutine libraries, other software, and all pertinent documentation.

4.2.12
literal
A syntactic entity denoting a constant value without having proper sub-entities that are expressions.

4.2.13
monotonic approximation
An approximation helper function $h : ... \times S \times ... \to \mathcal{R}$, where the other arguments are kept constant, and where $S \subseteq \mathcal{R}$, is a monotonic approximation of a predetermined mathematical function $f : \mathcal{R} \to \mathcal{R}$ if, for every $a \in S$ and $b \in S$, where $a < b$,

a) f is monotonic non-decreasing on $[a, b]$ implies $h(..., a, ...) \leqslant h(..., b, ...)$,

b) f is monotonic non-increasing on $[a, b]$ implies $h(..., a, ...) \geqslant h(..., b, ...)$.

4.2.14
monotonic non-decreasing
A function $f : \mathcal{R} \to \mathcal{R}$ is monotonic non-decreasing on a real interval $[a, b]$ if for every x and y such that $a \leqslant x \leqslant y \leqslant b$, $f(x)$ and $f(y)$ are well-defined and $f(x) \leqslant f(y)$.

4.2.15
monotonic non-increasing
A function $f : \mathcal{R} \to \mathcal{R}$ is monotonic non-increasing on a real interval $[a, b]$ if for every x and y such that $a \leqslant x \leqslant y \leqslant b$, $f(x)$ and $f(y)$ are well-defined and $f(x) \geqslant f(y)$.

4.2.16
normalised
The non-zero values of a floating point type F that provide the full precision allowed by that type. (See F_N in Clause 5.2 of part 1 for a full definition.)

4.2.17
notification
The process by which a program (or that program's end user) is informed that an arithmetic exception has occurred. For example, dividing 2 by 0 results in a notification. (See Clause 6 of part 1

4.2.18
numeral
A numeric literal. It may denote a value in $\mathcal{Z}$, i($\mathcal{Z}$), c($\mathcal{Z}$), $\mathcal{R}$, i($\mathcal{R}$), or c($\mathcal{R}$), a value which is $\mathbf{-0}$, an infinity, or a NaN.

4.2.19
numerical function
A computer routine or other mechanism for the approximate evaluation of a mathematical function.

4.2.20
operation
A function directly available to the programmer, as opposed to helper functions or theoretical mathematical functions.

4.2.21
pole
A mathematical function f has a pole at x_0 if x_0 is finite, f is defined, finite, monotone, and continuous in at least part of the neighbourhood of x_0, and the imaginary or real part of $\lim_{x \to x_0} f(x)$ is infinite via at least one path to x_0.

4.2.22
precision
The number of digits in the fraction of a floating point number. (See Clause 5.2 of part 1.)

4.2.23
rounding
The act of computing a representable final result for an operation that is close to the exact (but unrepresentable in the result datatype) result for that operation. Note that a suitable representable result may not exist (see Clause 5.2.6 of part 1). (See also Annex A.5.2.6 of part 1 for some examples.)

4.2.24
rounding function
Any function $rnd : \mathcal{R} \to X$ (where X is a given discrete and unlimited subset of $\mathcal{R}$) that maps each element of X to itself, and is monotonic non-decreasing. Formally, if x and y are in $\mathcal{R}$,

$$x \in X \Rightarrow rnd(x) = x$$
$$x < y \Rightarrow rnd(x) \leqslant rnd(y)$$

Note that if $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects one of those adjacent values.

4.2.25
round to nearest
The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects the one nearest u. If the adjacent values are equidistant from u, either may be chosen deterministically.

4.2.26
round toward minus infinity
The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects the one less than u.

4.2.27
round toward plus infinity
The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects the one greater than u.

4.2.28
shall
A verbal form used to indicate requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted. (Quoted from the directives [1].)

4.2.29
should
A verbal form used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that (in the negative form) a certain possibility is deprecated but not prohibited. (Quoted from the directives [1].)

4.2.30
signature (of a function or operation)
A summary of information about an operation or function. A signature includes the function or operation name; a subset of allowed argument values to the function or operation; and a superset of results from the function or operation (including exceptional values if any), if the argument is in the subset of argument values given in the signature.

The signature

$$add_I : I \times I \to I \cup \{\mathbf{overflow}\}$$

states that the operation named add_I shall accept any pair of I values as input, and (when given such input) shall return either a single I value as its output or the exceptional value **overflow**.

A signature for an operation or function does not forbid the operation from accepting a wider range of arguments, nor does it guarantee that every value in the result range will actually be returned for some input. An operation given an argument outside the stipulated argument domain may produce a result outside the stipulated result range.

4.2.31
subnormal
denormal (obsolete)
The values of a floating point datatype F, as well as $\mathbf{-0}$, whose absolute values are strictly less than the smallest positive normal value in F.

NOTE 4 – Compare the definition of F_D in Clause 5.2 of part 1. In the first edition (1994) of part 1 and in IEC 60559:1989 this concept, then excepting the zero values, was called denormal.

4.2.32
ulp
The value of one "unit in the last place" of a floating point number. This value depends on the exponent, the radix, and the precision used in representing the number. Thus, the ulp of a

normalised value x (in F), with exponent t, precision p, and radix r, is r^{t-p}, and the ulp of a subnormal value is $fminD_F$. (See Clause 5.2 of part 1.)

5 Specifications for imaginary and complex datatypes and operations

This clause specifies imaginary and complex integer datatypes, imaginary and complex floating point datatypes and a number of helper functions and operations for imaginary and complex integer as well as imaginary and complex floating point datatypes.

Each operation is given a signature and is further specified by a number of cases. These cases may refer to other operations (specified in this document, in part 1, or in part 2), to mathematical functions (textbook elementary functions, or functions defined in ISO/IEC 10967), and to helper functions (specified in this document, in part 1, or in part 2). They also use special abstract values (**−∞**, **+∞**, **−0**, **qNaN**, **sNaN**). For each datatype, two of these abstract values, **qNaN** and **sNaN**, may represent several actual values each. Finally, the specifications may refer to exceptional values.

The signatures in the specifications in this clause specify only all non-special values as input values, and indicate as output values a superset of all non-special, special, and exceptional values that may result from these (non-special) input values. Exceptional and special values that can never result from non-special input values are not included in the signatures given. Also, signatures that, for example, include IEC 60559 special values as arguments are not given in the specifications below. This does not exclude such signatures from being valid for these operations.

NOTE – For instance, the realpart operation on complex floating point is given with the following signature:

$$re_{c(F)} : c(F) \to F$$

But the following signature is also valid, and takes some special values into account

$$re_{c(F)} : c(F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\}) \to F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\}$$

The following signature is also valid

$$re_{c(F)} : c(F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}, \mathbf{qNaN}, \mathbf{sNaN}\}) \to F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}, \mathbf{qNaN}, \mathbf{invalid}\}$$

5.1 Imaginary and complex integer datatypes and operations

Clause 5.1 of part 1 and clause 5.1 of part 2 specify integer datatypes and a number of operations on values of an integer datatype. In this clause imaginary and complex integer datatypes and operations on values of an imaginary or complex integer datatype are specified.

A complex integer datatype is constructed from an integer datatype. There should be at least one imaginary integer datatype and at least one complex integer datatype for each provided integer datatype.

I is the set of non-special values, $I \subseteq \mathcal{Z}$, for an integer datatype conforming to part 1. Integer datatypes conforming to part 1 often do not contain any NaN or infinity values, even though they may do so. Therefore this clause has no specifications for such values as arguments or results.

i(I) (see Clause 4.1.6) is the set of non-special values in an imaginary integer datatype, constructed from the integer datatype corresponding to non-special value set I.

$\mathrm{c}(I)$ (see Clause 4.1.6) is the set of non-special values in a complex integer datatype, constructed from the integer datatype corresponding to the non-special value set I.

NOTE – The operations that return zero for certain cases, according to the specifications below, may in a subset of those cases return negative zero instead, if negative zero can be represented. Compare the specifications for corresponding complex floating point operations in Clause 5.2.

5.1.1 The complex integer *result* helper function

The $result_{\mathrm{c}(I)}$ helper function:

$$result_{\mathrm{c}(I)} : \mathcal{G} \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$

$$result_{\mathrm{c}(I)}(z) \quad = result_I(\mathfrak{Re}(z)) \mathbin{\hat{+}} \hat{\imath} \cdot result_I(\mathfrak{Im}(z))$$

NOTE – If one or both of the $result_I$ (defined in part 2) function applications on the right side returns **overflow**, then the $result_{\mathrm{c}(I)}$ application returns **overflow**. The continuation values used when **overflow** occurs are to be specified by the binding or implementation. The same holds also for other exceptional values and also for the specifications below that do not use $result_{\mathrm{c}(I)}$ but specify the result parts directly.

5.1.2 Imaginary and complex integer operations

5.1.2.1 Complex integer comparisons

$$eq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$$

$$eq_{\mathrm{i}(I)}(\hat{\imath} \cdot y, \hat{\imath} \cdot w) \quad = eq_I(y, w)$$

$$eq_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$$

$$eq_{I,\mathrm{i}(I)}(x, \hat{\imath} \cdot w) \quad = eq_{\mathrm{c}(I)}(x \mathbin{\hat{+}} \hat{\imath} \cdot 0, 0 \mathbin{\hat{+}} \hat{\imath} \cdot w)$$

$$eq_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \rightarrow \mathbf{Boolean}$$

$$eq_{\mathrm{i}(I),I}(\hat{\imath} \cdot y, z) \quad = eq_{\mathrm{c}(I)}(0 \mathbin{\hat{+}} \hat{\imath} \cdot y, z \mathbin{\hat{+}} \hat{\imath} \cdot 0)$$

$$eq_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$$

$$eq_{I,\mathrm{c}(I)}(x, z \mathbin{\hat{+}} \hat{\imath} \cdot w)$$
$$= eq_{\mathrm{c}(I)}(x \mathbin{\hat{+}} \hat{\imath} \cdot 0, z \mathbin{\hat{+}} \hat{\imath} \cdot w)$$

$$eq_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \rightarrow \mathbf{Boolean}$$

$$eq_{\mathrm{c}(I),I}(x \mathbin{\hat{+}} \hat{\imath} \cdot y, z)$$
$$= eq_{\mathrm{c}(I)}(x \mathbin{\hat{+}} \hat{\imath} \cdot y, z \mathbin{\hat{+}} \hat{\imath} \cdot 0)$$

$$eq_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$$

$$eq_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath} \cdot y, z \mathbin{\hat{+}} \hat{\imath} \cdot w)$$
$$= eq_{\mathrm{c}(I)}(0 \mathbin{\hat{+}} \hat{\imath} \cdot y, z \mathbin{\hat{+}} \hat{\imath} \cdot w)$$

$$eq_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$$

$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$\quad = eq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, 0 \mathbin{+\hat{\imath}\cdot} w)$

$eq_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$

$eq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$= \mathbf{true}$	if $eq_I(x,z) = \mathbf{true}$ and $eq_I(y,w) = \mathbf{true}$
$= \mathbf{false}$	if $eq_I(x,z) = \mathbf{false}$ and $eq_I(y,w) = \mathbf{true}$
$= \mathbf{false}$	if $eq_I(x,z) = \mathbf{true}$ and $eq_I(y,w) = \mathbf{false}$
$= \mathbf{false}$	if $eq_I(x,z) = \mathbf{false}$ and $eq_I(y,w) = \mathbf{false}$

$neq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$neq_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = neq_I(y,w)$

$neq_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$neq_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot w) \; = neq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} 0, 0 \mathbin{+\hat{\imath}\cdot} w)$

$neq_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \rightarrow \mathbf{Boolean}$

$neq_{\mathrm{i}(I),I}(\hat{\imath}\cdot y, z) \; = neq_{\mathrm{c}(I)}(0 \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} 0)$

$neq_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$

$neq_{I,\mathrm{c}(I)}(x, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = neq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} 0, z \mathbin{+\hat{\imath}\cdot} w)$

$neq_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \rightarrow \mathbf{Boolean}$

$neq_{\mathrm{c}(I),I}(x \mathbin{+\hat{\imath}\cdot} y, z)$
$\quad = neq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} 0)$

$neq_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$

$neq_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath}\cdot y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = neq_{\mathrm{c}(I)}(0 \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$neq_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$\quad = neq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, 0 \mathbin{+\hat{\imath}\cdot} w)$

$neq_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$

$neq_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$= \mathbf{true}$	if $neq_I(x,z) = \mathbf{true}$ and $neq_I(y,w) = \mathbf{true}$
$= \mathbf{true}$	if $neq_I(x,z) = \mathbf{false}$ and $neq_I(y,w) = \mathbf{true}$
$= \mathbf{true}$	if $neq_I(x,z) = \mathbf{true}$ and $neq_I(y,w) = \mathbf{false}$
$= \mathbf{false}$	if $neq_I(x,z) = \mathbf{false}$ and $neq_I(y,w) = \mathbf{false}$

$lss_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$lss_{\mathrm{i}(I)}(\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y, \hat{\mathbf{\imath}}\boldsymbol{\cdot}\, w) \quad = lss_I(y, w)$

$leq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$leq_{\mathrm{i}(I)}(\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y, \hat{\mathbf{\imath}}\boldsymbol{\cdot}\, w) \quad = leq_I(y, w)$

$gtr_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$gtr_{\mathrm{i}(I)}(\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y, \hat{\mathbf{\imath}}\boldsymbol{\cdot}\, w) \quad = gtr_I(y, w)$

$geq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$geq_{\mathrm{i}(I)}(\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y, \hat{\mathbf{\imath}}\boldsymbol{\cdot}\, w) = geq_I(y, w)$

5.1.2.2 Multiplication by the imaginary unit

$itimes_{I \rightarrow \mathrm{i}(I)} : I \rightarrow \mathrm{i}(I)$

$itimes_{I \rightarrow \mathrm{i}(I)}(x) \quad = \hat{\mathbf{\imath}}\boldsymbol{\cdot}\, x$

$itimes_{\mathrm{i}(I) \rightarrow I} : \mathrm{i}(I) \rightarrow I \cup \{\mathbf{overflow}\}$

$itimes_{\mathrm{i}(I) \rightarrow I}(\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y)$

$\qquad\qquad = neg_I(y)$

$itimes_{\mathrm{c}(I)} : \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}\}$

$itimes_{\mathrm{c}(I)}(x \,\boldsymbol{+}\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y)$

$\qquad\qquad = neg_I(y) \,\boldsymbol{+}\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, x$

5.1.2.3 The real and imaginary parts of a complex value

$re_I : I \rightarrow I$

$re_I(x) \quad = x \qquad\qquad$ if $x \in I$

$re_{\mathrm{i}(I)} : \mathrm{i}(I) \rightarrow \{0\}$

$re_{\mathrm{i}(I)}(\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y) \quad = 0 \qquad\qquad$ if $y \in I$

$re_{\mathrm{c}(I)} : \mathrm{c}(I) \rightarrow I$

$re_{\mathrm{c}(I)}(x \,\boldsymbol{+}\hat{\mathbf{\imath}}\boldsymbol{\cdot}\, y) \quad = x \qquad\qquad$ if $x \in I$

$im_I : I \rightarrow \{0\}$

$im_I(x) \quad = 0 \qquad\qquad$ if $x \in I$

$$im_{\mathrm{i}(I)} : \mathrm{i}(I) \to I$$
$$im_{\mathrm{i}(I)}(\hat{\imath}\cdot y) = y \qquad \text{if } y \in I$$

$$im_{\mathrm{c}(I)} : \mathrm{c}(I) \to I$$
$$im_{\mathrm{c}(I)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y) = y \qquad \text{if } y \in I$$

5.1.2.4 Formation of a complex integer from two real valued integers

$$plusitimes_I : I \times I \to \mathrm{c}(I)$$
$$plusitimes_I(x, z) = x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot z$$

5.1.2.5 Basic complex integer arithmetic

$$neg_{\mathrm{i}(I)} : \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\mathbf{overflow}\}$$
$$neg_{\mathrm{i}(I)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot neg_I(y)$$

$$neg_{\mathrm{c}(I)} : \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$neg_{\mathrm{c}(I)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y) = neg_I(x) \mathbin{\boldsymbol{+}} \hat{\imath}\cdot neg_I(y)$$

$$conj_I : I \to I$$
$$conj_I(x) = x$$

$$conj_{\mathrm{i}(I)} : \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\mathbf{overflow}\}$$
$$conj_{\mathrm{i}(I)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot neg_I(y)$$

$$conj_{\mathrm{c}(I)} : \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$conj_{\mathrm{c}(I)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y) = x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot neg_I(y)$$

$$add_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\mathbf{overflow}\}$$
$$add_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = \hat{\imath}\cdot add_I(y, w)$$

$$add_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \to \mathrm{c}(I)$$
$$add_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot w) = x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot w$$

$$add_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \to \mathrm{c}(I)$$
$$add_{\mathrm{i}(I),I}(\hat{\imath}\cdot y, z) = z \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y$$

$$add_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$add_{I,\mathrm{c}(I)}(x, z \,\hat{+}\mathbf{\hat{\imath}}\cdot\, w) = add_I(x, z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, w$$

$$add_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$add_{\mathrm{c}(I),I}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y, z) = add_I(x, z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y$$

$$add_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$add_{\mathrm{i}(I),\mathrm{c}(I)}(\mathbf{\hat{\imath}}\cdot\, y, z \,\hat{+}\mathbf{\hat{\imath}}\cdot\, w) = z \,\hat{+}\mathbf{\hat{\imath}}\cdot\, add_I(y, w)$$

$$add_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$add_{\mathrm{c}(I),\mathrm{i}(I)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y, \mathbf{\hat{\imath}}\cdot\, w) = x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, add_I(y, w)$$

$$add_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$add_{\mathrm{c}(I)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y, z \,\hat{+}\mathbf{\hat{\imath}}\cdot\, w) = add_I(x, z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, add_I(y, w)$$

$$sub_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{\mathrm{i}(I)}(\mathbf{\hat{\imath}}\cdot\, y, \mathbf{\hat{\imath}}\cdot\, w) = \mathbf{\hat{\imath}}\cdot\, sub_I(y, w)$$

$$sub_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{I,\mathrm{i}(I)}(x, \mathbf{\hat{\imath}}\cdot\, w) = x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, neg_I(w)$$

$$sub_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{\mathrm{i}(I),I}(\mathbf{\hat{\imath}}\cdot\, y, z) = neg_I(z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y$$

$$sub_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{I,\mathrm{c}(I)}(x, z \,\hat{+}\mathbf{\hat{\imath}}\cdot\, w) = sub_I(x, z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, neg_I(w)$$

$$sub_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{\mathrm{c}(I),I}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y, z) = sub_I(x, z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y$$

$$sub_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{\mathrm{i}(I),\mathrm{c}(I)}(\mathbf{\hat{\imath}}\cdot\, y, z \,\hat{+}\mathbf{\hat{\imath}}\cdot\, w) = neg_I(z) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, sub_I(y, w)$$

$$sub_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} y, \hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, w)$$
$$= x \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} sub_I(y, w)$$

$$sub_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$sub_{\mathrm{c}(I)}(x \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} y, z \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} w)$$
$$= sub_I(x, z) \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} sub_I(y, w)$$

$$mul_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to I \cup \{\mathbf{overflow}\}$$
$$mul_{\mathrm{i}(I)}(\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y, \hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, w)$$
$$= result_I(-(y \cdot w)) \qquad \text{if } y, w \in I$$

$$mul_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{I,\mathrm{i}(I)}(x, \hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, w) = \hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, mul_I(x, w)$$

$$mul_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \to \mathrm{i}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{\mathrm{i}(I),I}(\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y, z) = \hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, mul_I(y, z)$$

$$mul_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{I,\mathrm{c}(I)}(x, z \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} w)$$
$$= mul_I(x, z) \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} mul_I(x, w)$$

$$mul_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{\mathrm{c}(I),I}(x \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} y, z)$$
$$= mul_I(x, z) \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} mul_I(y, z)$$

$$mul_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y, z \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} w)$$
$$= result_I(-(y \cdot w)) \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} mul_I(y, z)$$
$$\text{if } y, z, w \in I$$

$$mul_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} y, \hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, w)$$
$$= result_I(-(y \cdot w)) \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} mul_I(x, w)$$
$$\text{if } x, y, w \in I$$

$$mul_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$$
$$mul_{\mathrm{c}(I)}(x \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} y, z \mathbin{\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}} w)$$
$$= result_{\mathrm{c}(I)}((x + (\tilde{\imath} \cdot y)) \cdot (z + (\tilde{\imath} \cdot w)))$$
$$\text{if } x, y, z, w \in I$$

5.1.2.6 Absolute value and signum of integers and imaginary integers

NOTE – abs_I is specified in part 1.

$abs_{\mathrm{i}(I)} : \mathrm{i}(I) \rightarrow I \cup \{\mathbf{overflow}\}$

$abs_{\mathrm{i}(I)}(\hat{\imath}\cdot y) \quad = abs_I(y)$

$signum_I : I \rightarrow \{-1, 1\}$

$$\begin{aligned} signum_I(x) &= 1 && \text{if } (x \in I \text{ and } x \geqslant 0)\\ &= -1 && \text{if } (x \in I \text{ and } x < 0)\end{aligned}$$

$signum_{\mathrm{i}(I)} : \mathrm{i}(I) \rightarrow \{\hat{\imath}\cdot(-1), \hat{\imath}\cdot 1\}$

$signum_{\mathrm{i}(I)}(\hat{\imath}\cdot y) \quad = \hat{\imath}\cdot signum_I(y)$

5.1.2.7 Divisibility interrogation

$divides_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$divides_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$\quad = divides_I(y, w)$

$divides_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$divides_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot w)$
$\quad = divides_I(x, w)$

$divides_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \rightarrow \mathbf{Boolean}$

$divides_{\mathrm{i}(I),I}(\hat{\imath}\cdot y, z)$
$\quad = divides_I(y, z)$

$divides_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$

$divides_{I,\mathrm{c}(I)}(x, z + \hat{\imath}\cdot w)$
$\quad = divides_{\mathrm{c}(I)}(x + \hat{\imath}\cdot 0, z + \hat{\imath}\cdot w)$

$divides_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \rightarrow \mathbf{Boolean}$

$divides_{\mathrm{c}(I),I}(x + \hat{\imath}\cdot y, z)$
$\quad = divides_{\mathrm{c}(I)}(x + \hat{\imath}\cdot y, z + \hat{\imath}\cdot 0)$

$divides_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \rightarrow \mathbf{Boolean}$

$divides_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath}\cdot y, z + \hat{\imath}\cdot w)$
$\quad = divides_{\mathrm{c}(I)}(0 + \hat{\imath}\cdot y, z + \hat{\imath}\cdot w)$

$divides_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \rightarrow \mathbf{Boolean}$

$divides_{c(I),i(I)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$\quad = divides_{c(I)}(x \mathbin{+\hat{\imath}\cdot} y, 0 \mathbin{+\hat{\imath}\cdot} w)$

$divides_{c(I)} : c(I) \times c(I) \to \mathbf{Boolean}$

$divides_{c(I)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = \mathbf{true}$ if $x, y, z, w \in I$ and $(x + (\tilde{\imath} \cdot y)) \mid (z + (\tilde{\imath} \cdot w))$
$\quad = \mathbf{false}$ if $x, y, z, w \in I$ and not $(x + (\tilde{\imath} \cdot y)) \mid (z + (\tilde{\imath} \cdot w))$

5.1.2.8 Integer division and remainder extended to imaginary and complex integers

For these operations, I shall be signed.

NOTE – Even though the integer division operations in principle could be included also for unsigned integer datatypes, that would result in operations that would overflow (mathematically have negative subresults) for so many argument values, that the inclusion of those operations would be pointless. The same goes for the remainder operations.

$quot_{i(I)} : i(I) \times i(I) \to I \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{i(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$\quad = quot_I(y, w)$

$quot_{I,i(I)} : I \times i(I) \to i(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{I,i(I)}(x, \hat{\imath}\cdot w)$
$\quad = \hat{\imath}\cdot minint_I$ if $x = minint_I$ and $w = -1$
$\quad = \hat{\imath}\cdot neg_I(group_I(x, w))$ otherwise

$quot_{i(I),I} : i(I) \times I \to i(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{i(I),I}(\hat{\imath}\cdot y, z) = \hat{\imath}\cdot quot_I(y, z)$

$quot_{I,c(I)} : I \times c(I) \to c(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{I,c(I)}(x, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = quot_{c(I)}(x \mathbin{+\hat{\imath}\cdot} 0, z \mathbin{+\hat{\imath}\cdot} w)$

$quot_{c(I),I} : c(I) \times I \to c(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{c(I),I}(x \mathbin{+\hat{\imath}\cdot} y, z)$
$\quad = quot_I(x, z) \mathbin{+\hat{\imath}\cdot} quot_I(y, z)$

$quot_{i(I),c(I)} : i(I) \times c(I) \to c(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{i(I),c(I)}(\hat{\imath}\cdot y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = quot_{c(I)}(0 \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$quot_{c(I),i(I)} : c(I) \times i(I) \to c(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{c(I),i(I)}(x \mathbin{+\!\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$= neg_I(y) \mathbin{+\!\hat{\imath}\cdot} minint_I$ if $x = minint_I$ and $w = -1$
$= quot_I(y, w) \mathbin{+\!\hat{\imath}\cdot} neg_I(group_I(x, w))$
otherwise

$quot_{c(I)} : c(I) \times c(I) \to c(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$quot_{c(I)}(x \mathbin{+\!\hat{\imath}\cdot} y, z \mathbin{+\!\hat{\imath}\cdot} w)$
$= result_{c(I)}(\lfloor (x + (\tilde{\imath} \cdot y))/(z + (\tilde{\imath} \cdot w)) \rfloor)$
if $x, y, z, w \in I$ and $z + (\tilde{\imath} \cdot w) \neq 0$
$= quot_I(x, 0) \mathbin{+\!\hat{\imath}\cdot} quot_I(y, 0)$
otherwise

$mod_{i(I)} : i(I) \times i(I) \to i(I) \cup \{\mathbf{invalid}\}$

$mod_{i(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$= \hat{\imath}\cdot mod_I(y, w)$

$mod_{I,i(I)} : I \times i(I) \to I \cup \{\mathbf{invalid}\}$

$mod_{I,i(I)}(x, \hat{\imath}\cdot w)$
$= neg_I(pad_I(x, w))$

$mod_{i(I),I} : i(I) \times I \to i(I) \cup \{\mathbf{invalid}\}$

$mod_{i(I),I}(\hat{\imath}\cdot y, z) = \hat{\imath}\cdot mod_I(y, z)$

$mod_{I,c(I)} : I \times c(I) \to c(I) \cup \{\mathbf{invalid}\}$

$mod_{I,c(I)}(x, z \mathbin{+\!\hat{\imath}\cdot} w)$
$= mod_{c(I)}(x \mathbin{+\!\hat{\imath}\cdot} 0, z \mathbin{+\!\hat{\imath}\cdot} w)$

$mod_{c(I),I} : c(I) \times I \to c(I) \cup \{\mathbf{invalid}\}$

$mod_{c(I),I}(x \mathbin{+\!\hat{\imath}\cdot} y, z)$
$= mod_I(x, z) \mathbin{+\!\hat{\imath}\cdot} mod_I(y, z)$

$mod_{i(I),c(I)} : i(I) \times c(I) \to c(I) \cup \{\mathbf{invalid}\}$

$mod_{i(I),c(I)}(\hat{\imath}\cdot y, z \mathbin{+\!\hat{\imath}\cdot} w)$
$= mod_{c(I)}(0 \mathbin{+\!\hat{\imath}\cdot} y, z \mathbin{+\!\hat{\imath}\cdot} w)$

$mod_{c(I),i(I)} : c(I) \times i(I) \to c(I) \cup \{\mathbf{invalid}\}$

$mod_{c(I),i(I)}(x \mathbin{+\!\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$= neg_I(pad_I(x, w)) \mathbin{+\!\hat{\imath}\cdot} mod_I(y, w)$

$mod_{c(I)} : c(I) \times c(I) \to c(I) \cup \{\mathbf{invalid}\}$

$$
\begin{aligned}
&mod_{\mathrm{c}(I)}(x \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, y, z \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, w)\\
&\quad = result_{\mathrm{c}(I)}((x + (\tilde{\imath} \cdot y)) - (\lfloor (x + (\tilde{\imath} \cdot y))/(z + (\tilde{\imath} \cdot w))\rfloor \cdot (z + (\tilde{\imath} \cdot w))))\\
&\qquad\qquad \text{if } x, y, z, w \in I \text{ and } z + (\tilde{\imath} \cdot w) \neq 0\\
&\quad = \mathbf{invalid}(\mathbf{qNaN} \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, \mathbf{qNaN})\\
&\qquad\qquad \text{otherwise}
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to I \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{\mathrm{i}(I)}(\hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w)\\
&\quad = ratio_I(y, w)
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot\, w)\\
&\quad = \hat{\imath}\cdot\, minint_I \qquad \text{if } x = minint_I \text{ and } w = -1\\
&\quad = \hat{\imath}\cdot\, neg_I(ratio_I(x, w)) \qquad \text{otherwise}
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \to \mathrm{i}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{\mathrm{i}(I),I}(\hat{\imath}\cdot\, y, z) = \hat{\imath}\cdot\, ratio_I(y, z)
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{I,\mathrm{c}(I)}(x, z \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, w)\\
&\quad = ratio_{\mathrm{c}(I)}(x \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, 0, z \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, w)
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{\mathrm{c}(I),I}(x \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, y, z)\\
&\quad = ratio_I(x, z) \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, ratio_I(y, z)
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath}\cdot\, y, z \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, w)\\
&\quad = ratio_{\mathrm{c}(I)}(0 \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, y, z \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, w)
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{\mathrm{c}(I),\mathrm{i}(I)}(x \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w)\\
&\quad = neg_I(y) \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, minint_I \qquad \text{if } x = minint_I \text{ and } w = -1\\
&\quad = ratio_I(y, w) \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, neg_I(ratio_I(x, w))\\
&\qquad\qquad \text{otherwise}
\end{aligned}
$$

$$
\begin{aligned}
&ratio_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}\\
&ratio_{\mathrm{c}(I)}(x \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, y, z \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, w)\\
&\quad = result_{\mathrm{c}(I)}(\mathrm{round}((x + (\tilde{\imath} \cdot y))/(z + (\tilde{\imath} \cdot w))))\\
&\qquad\qquad \text{if } x, y, z, w \in I \text{ and } z + (\tilde{\imath} \cdot w) \neq 0\\
&\quad = ratio_I(x, 0) \,\hat{\textbf{+}}\, \hat{\imath}\cdot\, ratio_I(y, 0)\\
&\qquad\qquad \text{otherwise}
\end{aligned}
$$

$residue_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathrm{i}(I) \cup \{\mathbf{invalid}\}$

$residue_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$\quad = \hat{\imath}\cdot residue_I(y, w)$

$residue_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \rightarrow I \cup \{\mathbf{invalid}\}$

$residue_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot w)$
$\quad = residue_I(x, w)$

$residue_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \rightarrow \mathrm{i}(I) \cup \{\mathbf{invalid}\}$

$residue_{\mathrm{i}(I),I}(\hat{\imath}\cdot y, z)$
$\quad = \hat{\imath}\cdot residue_I(y, z)$

$residue_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{invalid}\}$

$residue_{I,\mathrm{c}(I)}(x, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = residue_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} 0, z \mathbin{+\hat{\imath}\cdot} w)$

$residue_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \rightarrow \mathrm{c}(I) \cup \{\mathbf{invalid}\}$

$residue_{\mathrm{c}(I),I}(x \mathbin{+\hat{\imath}\cdot} y, z)$
$\quad = residue_I(x, z) \mathbin{+\hat{\imath}\cdot} residue_I(y, z)$

$residue_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{invalid}\}$

$residue_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath}\cdot y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = residue_{\mathrm{c}(I)}(0 \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$residue_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{invalid}\}$

$residue_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$\quad = residue_I(x, w) \mathbin{+\hat{\imath}\cdot} residue_I(y, w)$

$residue_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{invalid}\}$

$residue_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = result_{\mathrm{c}(I)}((x + (\tilde{\imath} \cdot y)) - (\mathrm{round}((x + (\tilde{\imath} \cdot y))/(z + (\tilde{\imath} \cdot w))) \cdot (z + (\tilde{\imath} \cdot w))))$
$\qquad$ if $x, y, z, w \in I$ and $z + (\tilde{\imath} \cdot w) \neq 0$
$\quad = \mathbf{invalid}(\mathbf{qNaN} \mathbin{+\hat{\imath}\cdot} \mathbf{qNaN})$
$\qquad$ otherwise

$group_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow I \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$group_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$\quad = group_I(y, w)$

$group_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \rightarrow \mathrm{i}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$$
\begin{aligned}
&group_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot w)\\
&\qquad = \hat{\imath}\cdot minint_I && \text{if } x = minint_I \text{ and } w = -1\\
&\qquad = \hat{\imath}\cdot neg_I(quot_I(x, w)) && \text{otherwise}
\end{aligned}
$$

$$group_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \rightarrow \mathrm{i}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$
\begin{aligned}
&group_{\mathrm{i}(I),I}(\hat{\imath}\cdot y, z)\\
&\qquad = \hat{\imath}\cdot group_I(y, z)
\end{aligned}
$$

$$group_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$
\begin{aligned}
&group_{I,\mathrm{c}(I)}(x, z +\hat{\imath}\cdot w)\\
&\qquad = group_{\mathrm{c}(I)}(x +\hat{\imath}\cdot 0, z +\hat{\imath}\cdot w)
\end{aligned}
$$

$$group_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$
\begin{aligned}
&group_{\mathrm{c}(I),I}(x +\hat{\imath}\cdot y, z)\\
&\qquad = group_I(x, z) +\hat{\imath}\cdot group_I(y, z)
\end{aligned}
$$

$$group_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$
\begin{aligned}
&group_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath}\cdot y, z +\hat{\imath}\cdot w)\\
&\qquad = group_{\mathrm{c}(I)}(0 +\hat{\imath}\cdot y, z +\hat{\imath}\cdot w)
\end{aligned}
$$

$$group_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$
\begin{aligned}
&group_{\mathrm{c}(I),\mathrm{i}(I)}(x +\hat{\imath}\cdot y, \hat{\imath}\cdot w)\\
&\qquad = neg_I(y) +\hat{\imath}\cdot minint_I && \text{if } x = minint_I \text{ and } w = -1\\
&\qquad = group_I(y, w) +\hat{\imath}\cdot neg_I(quot_I(x, w))\\
& && \text{otherwise}
\end{aligned}
$$

$$group_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \rightarrow \mathrm{c}(I) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$
\begin{aligned}
&group_{\mathrm{c}(I)}(x +\hat{\imath}\cdot y, z +\hat{\imath}\cdot w)\\
&\qquad = result_{\mathrm{c}(I)}(\lceil (x + (\tilde{\imath}\cdot y))/(z + (\tilde{\imath}\cdot w)) \rceil)\\
& && \text{if } x, y, z, w \in I \text{ and } z + (\tilde{\imath}\cdot w) \neq 0\\
&\qquad = group_I(x, 0) +\hat{\imath}\cdot group_I(y, 0)\\
& && \text{otherwise}
\end{aligned}
$$

$$pad_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \rightarrow \mathrm{i}(I) \cup \{\mathbf{invalid}\}$$

$$pad_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = \hat{\imath}\cdot pad_I(y, w)$$

$$pad_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \rightarrow I \cup \{\mathbf{invalid}\}$$

$$pad_{I,\mathrm{i}(I)}(x, \hat{\imath}\cdot w) \;= neg_I(mod_I(x, w))$$

$$pad_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \rightarrow \mathrm{i}(I) \cup \{\mathbf{invalid}\}$$

$$pad_{\mathrm{i}(I),I}(\hat{\imath}\cdot y, z) \;= \hat{\imath}\cdot pad_I(y, z)$$

$pad_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\textbf{invalid}\}$

$pad_{I,\mathrm{c}(I)}(x, z \mathbin{\hat{+}\mathbf{\hat{\imath}}\cdot} w)$
$= pad_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} 0, z \mathbin{+\hat{\imath}\cdot} w)$

$pad_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \mathrm{c}(I) \cup \{\textbf{invalid}\}$

$pad_{\mathrm{c}(I),I}(x \mathbin{+\hat{\imath}\cdot} y, z)$
$= pad_I(x, z) \mathbin{+\hat{\imath}\cdot} pad_I(y, z)$

$pad_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\textbf{invalid}\}$

$pad_{\mathrm{i}(I),\mathrm{c}(I)}(\hat{\imath}\cdot y, z \mathbin{+\hat{\imath}\cdot} w)$
$= pad_{\mathrm{c}(I)}(0 \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$pad_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\textbf{invalid}\}$

$pad_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$= neg_I(mod_I(x, w)) \mathbin{+\hat{\imath}\cdot} pad_I(y, w)$

$pad_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\textbf{invalid}\}$

$pad_{\mathrm{c}(I)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$
$= result_{\mathrm{c}(I)}((\lceil (x + (\tilde{\imath} \cdot y))/(z + (\tilde{\imath} \cdot w)) \rfloor \cdot (z + (\tilde{\imath} \cdot w))) - (x + (\tilde{\imath} \cdot y)))$
if $x, y, z, w \in I$ and $z + (\tilde{\imath} \cdot w) \neq 0$
$= \textbf{invalid}(\textbf{qNaN} \mathbin{+\hat{\imath}\cdot} \textbf{qNaN})$
otherwise

5.1.2.9 Maximum and minimum

$max_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \mathrm{i}(I)$

$max_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$= \hat{\imath}\cdot max_I(y, w)$

$min_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \mathrm{i}(I)$

$min_{\mathrm{i}(I)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$= \hat{\imath}\cdot min_I(y, w)$

$max\_seq_{\mathrm{i}(I)} : [\mathrm{i}(I)] \to \mathrm{i}(I) \cup \{\textbf{infinitary}\}$

$max\_seq_{\mathrm{i}(I)}([\hat{\imath}\cdot y_1, ..., \hat{\imath}\cdot y_n])$
$= \hat{\imath}\cdot max\_seq_I([y_1, ..., y_n])$

$min\_seq_{\mathrm{i}(I)} : [\mathrm{i}(I)] \to \mathrm{i}(I) \cup \{\textbf{infinitary}\}$

$min\_seq_{\mathrm{i}(I)}([\hat{\imath}\cdot y_1, ..., \hat{\imath}\cdot y_n])$
$= \hat{\imath}\cdot min\_seq_I([y_1, ..., y_n])$

5.2 Imaginary and complex floating point datatypes and operations

Clause 5.2 of part 1 and Clause 5.2 of part 2 of ISO/IEC 10967 specify floating point datatypes and a number of operations on values of a floating point datatype. In this clause imaginary and complex floating point datatypes and operations on values of an imaginary or complex floating point datatype are specified.

NOTE – Further operations on values of an imaginary or complex floating point datatype, for elementary complex floating point numerical functions, are specified in Clause 5.3.

An imaginary or complex floating point datatype is constructed from a floating point datatype. There should be at least one imaginary floating point datatype and at least one complex floating point datatype for each provided floating point datatype.

F is the non-special value set, $F \subset \mathcal{R}$, for a floating point datatype conforming to part 1 of ISO/IEC 10967. Floating point datatypes conforming to part 1 often do contain $\mathbf{-0}$, infinity, and NaN values. Therefore, in this clause there are specifications for such values as arguments.

$\mathrm{i}(F)$ (see Clause 4.1.6) is the set of non-special values in an imaginary floating point datatype, constructed from the floating point datatype corresponding to the non-special value set F.

$\mathrm{c}(F)$ (see Clause 4.1.6) is the set of non-special values in a complex floating point datatype, constructed from the floating point datatype corresponding to the non-special value set F.

5.2.1 Maximum error requirements

Some of the operations are exact, such as taking the imaginary part of a complex value. Some operations are approximate, with maximum error requirements implied by their exact relationships with operations defined in part 1, part 2, or this part of this International Standard. Some other approximate operations have new error parameters associated with them as detailed in the specifications.

The approximation helper functions for the individual operations in these subclauses have maximum error parameters that describe the maximum *relative* error, in ulps, of the helper function composed with $nearest_F$, for non-subnormal results. The maximum error parameters also describe the maximum *absolute* error, in ulps, for $-fminN_F$, $fminN_F$, subnormal results and underflow continuation values if $denorm_F = \mathbf{true}$. All maximum error parameters shall have a value that is $\geqslant 0.5$. For the maximum value of the maximum error parameters, see the specification of each of the maximum error parameters. See also Annex A, on partial conformity. The relevant maximum error parameters shall be made available to programs.

When the maximum error for an approximation helper function $h_{\mathrm{c}(F)}$, approximating f, is $max\_error\_op_{\mathrm{c}(F)}$, then for all arguments $z, ... \in \mathcal{C}_F \times ...$ the following equations shall hold:

$$|\mathfrak{Re}(f(z,...)) - nearest_F(\mathfrak{Re}(h_{\mathrm{c}(F)}(z,...)))| \leqslant max\_error\_op_{\mathrm{c}(F)} \cdot r_F^{e_F(\mathfrak{Re}(f(z,...)))-p_F}$$
$$|\mathfrak{Im}(f(z,...)) - nearest_F(\mathfrak{Im}(h_{\mathrm{c}(F)}(z,...)))| \leqslant max\_error\_op_{\mathrm{c}(F)} \cdot r_F^{e_F(\mathfrak{Im}(f(z,...)))-p_F}$$

Some operations have a **Boolean** parameter $box\_error\_mode\_op_{\mathrm{c}(F)}$. If such a parameter is present for an operation and that parameter has the value **true**, then the sign requirements need not be fulfilled (see below) and the maximum error requirements are modified to the following:

$$|\mathfrak{Re}(f(z,...)) - nearest_F(\mathfrak{Re}(h_{c(F)}(z,...)))| \leqslant max\_error\_op_{c(F)} \cdot r_F^{e_F(|f(z,...)|)-p_F}$$
$$|\mathfrak{Im}(f(z,...)) - nearest_F(\mathfrak{Im}(h_{c(F)}(z,...)))| \leqslant max\_error\_op_{c(F)} \cdot r_F^{e_F(|f(z,...)|)-p_F}$$

The default value for a $box\_error\_mode\_op_{c(F)}$ parameter should be **false**.

NOTES

1 Partially conforming implementations may have greater values for maximum error parameters than stipulated below. See Annex A.

2 Multiplication and division of complex values have the box error mode parameter. See 5.2.6.

3 The relative error requirement results in a 'rectangular' error bound, which can only span over a zero (in either dimension) when the returned result is very close to 0 (but see 'Sign requirements' below).

4 The box error requirement results in a 'square-formed' error bound, which for cancellation cases can span over zero in either or both dimensions. Relative error requirements in each axis can be fulfilled also for multiplication and division, but that is often considered too inefficient, and an implementation that may suffer from cancellation is often used instead.

5.2.2 Sign requirements

The following sign requirements shall hold:

a) The real or imaginary part of the result of an approximation helper function shall be zero exactly at the points where the real or imaginary part (respectively) of the approximated mathematical function is exactly zero.

b) At a point where an approximation helper function result part is not zero, the result part shall have the same sign as the corresponding result part of the approximated mathematical function at that point.

However, the following exceptions are made:

a) For the trigonometric helper functions, these zero and sign requirements are imposed only for arguments, $x + (\tilde{\imath} \cdot y)$, such that $|x| \leqslant big\_angle\_r_F$ (see Clause 5.3.2; $big\_angle\_r_F$ is specified in part 2).

b) If there is a $box\_error\_mode\_op_{c(F)}$ parameter for an operation and that parameter has the value **true**, then the sign requirements are not imposed for that operation.

NOTES

1 For the operations, the continuation value after an **underflow** may be zero (including negative zero) as given by $result_F^*$ (see part 2), even though the approximation helper function is not zero at that point. Such zero results are required to be accompanied by an **underflow** notification. When appropriate, zero may also be returned for IEC 60559 infinities arguments. See the individual specifications.

2 Multiplication and division of complex values have modified sign requirements. See above and 5.2.6.

5.2.3 Monotonicity requirements

For this document, each approximation helper function shall be a monotonic approximation, for each real and imaginary argument part individually, and the real and imaginary result parts individually, to the mathematical function it is approximating.

For the trigonometric approximation helper functions, the monotonic approximation requirement is imposed only for arguments, $x + (\tilde{\imath} \cdot y)$, such that $|x| \leqslant big\_angle\_r_F$ (see Clause 5.3.2). Similarly, for the exponentiation approximation helper functions, the monotonic approximation requirement is imposed only for arguments, $x + (\tilde{\imath} \cdot y)$, such that $|y| \leqslant big\_angle\_r_F$ (see Clause 5.3.1).

NOTES

1 As in part 2, the monotonicity requirement applies to each real dimension individually. For the complex operations, it thus applies to each real and imaginary part of the argument(s) individually, and individually to the real and imaginary parts of the result.

2 As in part 2, the monotonicity requirement applies individually to each monotonic interval of the approximated mathematical function.

3 There is an exact *relationship* between the trigonometric and hyperbolic *operations* specified in this document, and thus there are no complex hyperbolic approximation helper functions used in this document.

5.2.4 The complex floating point *result* helper functions

$$result^*_{c(F)} : \mathcal{C} \times (\mathcal{R} \rightarrow F^*) \rightarrow c(F) \cup \{\textbf{underflow}, \textbf{overflow}\}$$

$$result^*_{c(F)}(z, rnd) = result^*_F(\mathfrak{Re}(z), rnd) \,\hat{+}\hat{\imath}\cdot\, result^*_F(\mathfrak{Im}(z), rnd)$$

NOTES

1 **overflow** and **underflow** can both occur for a single application of a complex operation. Likewise, e.g., one part may overflow, while the other produced a value in F. This is not exposed accurately by the mathematical framework used in this document. However, handling that accurately, would add complexity to the framework with little gain.

2 $result^*_F$ is defined in part 2.

The $no\_result_{c(F)}$, $no\_result_{i(F)}$, $no\_result_{F\rightarrow c(F)}$, $no\_result_{i(F)\rightarrow c(F)}$, and $no\_result2_{c(F)}$ helper functions are defined as follows:

$$no\_result_{c(F)} : c(F) \rightarrow \{\textbf{invalid}\}$$

$$\begin{aligned} no\_result_{c(F)}(x \,\hat{+}\hat{\imath}\cdot\, y) &= \textbf{invalid}(\textbf{qNaN} \,\hat{+}\hat{\imath}\cdot\, \textbf{qNaN}) && \text{if } x, y \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \\ &= \textbf{qNaN} \,\hat{+}\hat{\imath}\cdot\, \textbf{qNaN} && \text{if at least one of } x \text{ and } y \text{ is a quiet NaN and neither is a signalling NaN} \\ &= \textbf{invalid}(\textbf{qNaN} \,\hat{+}\hat{\imath}\cdot\, \textbf{qNaN}) && \text{if } x \text{ is a signalling NaN or } y \text{ is a signalling NaN} \end{aligned}$$

$$no\_result_{i(F)} : i(F) \rightarrow \{\textbf{invalid}\}$$

$$
\begin{aligned}
&\mathit{no\_result}_{\mathrm{i}(F)}(\hat{\textbf{\i}}\cdot y)\\
&\qquad = \mathbf{invalid}(\hat{\textbf{\i}}\cdot \mathbf{qNaN}) && \text{if } y \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\}\\
&\qquad = \hat{\textbf{\i}}\cdot \mathbf{qNaN} && \text{if } y \text{ is a quiet NaN}\\
&\qquad = \mathbf{invalid}(\hat{\textbf{\i}}\cdot \mathbf{qNaN}) && \text{if } y \text{ is a signalling NaN}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{no\_result}_{F\to\mathrm{c}(F)} : F \to \{\mathbf{invalid}\}\\
&\mathit{no\_result}_{F\to\mathrm{c}(F)}(x)\\
&\qquad = \mathit{no\_result}_{\mathrm{c}(F)}(x \mathbin{+\hat{\textbf{\i}}\cdot} \mathit{im}_F(x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{no\_result}_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \{\mathbf{invalid}\}\\
&\mathit{no\_result}_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\textbf{\i}}\cdot y)\\
&\qquad = \mathit{no\_result}_{\mathrm{c}(F)}(\mathit{re}_{\mathrm{i}(F)}(y) \mathbin{+\hat{\textbf{\i}}\cdot} y)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{no\_result2}_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \{\mathbf{invalid}\}\\
&\mathit{no\_result2}_{\mathrm{c}(F)}(x \mathbin{+\hat{\textbf{\i}}\cdot} y, z \mathbin{+\hat{\textbf{\i}}\cdot} w)\\
&\qquad = \mathbf{invalid}(\mathbf{qNaN} \mathbin{+\hat{\textbf{\i}}\cdot} \mathbf{qNaN})\\
&\qquad && \text{if } x, y, z, w \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\}\\
&\qquad = \mathbf{qNaN} \mathbin{+\hat{\textbf{\i}}\cdot} \mathbf{qNaN} && \text{if at least one of } x, y, z, \text{ and } w \text{ is a quiet NaN}\\
&\qquad && \quad \text{and neither is a signalling NaN}\\
&\qquad = \mathbf{invalid}(\mathbf{qNaN} \mathbin{+\hat{\textbf{\i}}\cdot} \mathbf{qNaN})\\
&\qquad && \text{if at least one of } x, y, z, \text{ and } w \text{ is a signalling NaN}
\end{aligned}
$$

These helper functions are used to specify both NaN argument handling and to handle non-NaN-argument cases where $\mathbf{invalid}(\mathbf{qNaN} \mathbin{+\hat{\textbf{\i}}\cdot} \mathbf{qNaN})$ or $\mathbf{invalid}(\hat{\textbf{\i}}\cdot \mathbf{qNaN})$ is the appropriate result.

NOTES

3 The handling of other special values, if available, is left unspecified by this document. Other special values may be exact representations for π or e. At a lower level, such special values may be represented by IEC 60559 signalling NaNs, but which aren't regarded as NaNs at the LIA level.

4 $\mathit{re}_{\mathrm{i}(F)}$ and im_F are defined below.

5.2.5 Basic arithmetic for complex floating point

5.2.5.1 Complex floating point comparisons

$$
\begin{aligned}
&\mathit{eq}_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}\\
&\mathit{eq}_{\mathrm{i}(F)}(\hat{\textbf{\i}}\cdot y, \hat{\textbf{\i}}\cdot w) \quad = \mathit{eq}_F(y, w)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{eq}_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \to \mathbf{Boolean}\\
&\mathit{eq}_{F,\mathrm{i}(F)}(x, \hat{\textbf{\i}}\cdot w) \quad = \mathit{eq}_{\mathrm{c}(F)}(x \mathbin{+\hat{\textbf{\i}}\cdot} 0, 0 \mathbin{+\hat{\textbf{\i}}\cdot} w)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{eq}_{\mathrm{i}(F),F} : \mathrm{i}(F) \times F \to \mathbf{Boolean}\\
&\mathit{eq}_{\mathrm{i}(F),F}(\hat{\textbf{\i}}\cdot y, z) \quad = \mathit{eq}_{\mathrm{c}(F)}(0 \mathbin{+\hat{\textbf{\i}}\cdot} y, z \mathbin{+\hat{\textbf{\i}}\cdot} 0)
\end{aligned}
$$

$$
\mathit{eq}_{F,\mathrm{c}(F)} : F \times \mathrm{c}(F) \to \mathbf{Boolean}
$$

$eq_{F,\mathrm{c}(F)}(x, z \,\hat{+}\,\hat{\imath}\cdot w)$
$\qquad = eq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot 0, z \,\hat{+}\,\hat{\imath}\cdot w)$

$eq_{\mathrm{c}(F),F} : \mathrm{c}(F) \times F \to \mathbf{Boolean}$
$eq_{\mathrm{c}(F),F}(x \,\hat{+}\,\hat{\imath}\cdot y, z)$
$\qquad = eq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot y, z \,\hat{+}\,\hat{\imath}\cdot 0)$

$eq_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \to \mathbf{Boolean}$
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot y, z \,\hat{+}\,\hat{\imath}\cdot w)$
$\qquad = eq_{\mathrm{c}(F)}(0 \,\hat{+}\,\hat{\imath}\cdot y, z \,\hat{+}\,\hat{\imath}\cdot w)$

$eq_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x \,\hat{+}\,\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$\qquad = eq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot y, 0 \,\hat{+}\,\hat{\imath}\cdot w)$

$eq_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathbf{Boolean}$
$eq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot y, z \,\hat{+}\,\hat{\imath}\cdot w)$

$= \mathbf{true}$	if $eq_F(x, z) = \mathbf{true}$ and $eq_F(y, w) = \mathbf{true}$
$= \mathbf{false}$	if $eq_F(x, z) = \mathbf{false}$ and $eq_F(y, w) = \mathbf{true}$
$= \mathbf{false}$	if $eq_F(x, z) = \mathbf{true}$ and $eq_F(y, w) = \mathbf{false}$
$= \mathbf{false}$	if $eq_F(x, z) = \mathbf{false}$ and $eq_F(y, w) = \mathbf{false}$
$= \mathbf{invalid}(\mathbf{false})$	otherwise

$neq_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$
$neq_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = neq_F(y, w)$

$neq_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \to \mathbf{Boolean}$
$neq_{F,\mathrm{i}(F)}(x, \hat{\imath}\cdot w) = neq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot 0, 0 \,\hat{+}\,\hat{\imath}\cdot w)$

$neq_{\mathrm{i}(F),F} : \mathrm{i}(F) \times F \to \mathbf{Boolean}$
$neq_{\mathrm{i}(F),F}(\hat{\imath}\cdot y, z) = neq_{\mathrm{c}(F)}(0 \,\hat{+}\,\hat{\imath}\cdot y, z \,\hat{+}\,\hat{\imath}\cdot 0)$

$neq_{F,\mathrm{c}(F)} : F \times \mathrm{c}(F) \to \mathbf{Boolean}$
$neq_{F,\mathrm{c}(F)}(x, z \,\hat{+}\,\hat{\imath}\cdot w)$
$\qquad = neq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot 0, z \,\hat{+}\,\hat{\imath}\cdot w)$

$neq_{\mathrm{c}(F),F} : \mathrm{c}(F) \times F \to \mathbf{Boolean}$
$neq_{\mathrm{c}(F),F}(x \,\hat{+}\,\hat{\imath}\cdot y, z)$
$\qquad = neq_{\mathrm{c}(F)}(x \,\hat{+}\,\hat{\imath}\cdot y, z \,\hat{+}\,\hat{\imath}\cdot 0)$

$neq_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \to \mathbf{Boolean}$

$$neq_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot y, z \mathbin{\hat{+}\!\hat{\imath}\cdot} w) = neq_{\mathrm{c}(F)}(0 \mathbin{\hat{+}\!\hat{\imath}\cdot} y, z \mathbin{\hat{+}\!\hat{\imath}\cdot} w)$$

$$neq_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$$
$$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{\hat{+}\!\hat{\imath}\cdot} y, \hat{\imath}\cdot w) = neq_{\mathrm{c}(F)}(x \mathbin{\hat{+}\!\hat{\imath}\cdot} y, 0 \mathbin{\hat{+}\!\hat{\imath}\cdot} w)$$

$$neq_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathbf{Boolean}$$
$$\begin{aligned} neq_{\mathrm{c}(F)}(x \mathbin{\hat{+}\!\hat{\imath}\cdot} y, z \mathbin{\hat{+}\!\hat{\imath}\cdot} w) &= \mathbf{true} && \text{if } neq_F(x,z) = \mathbf{true} \text{ and } neq_F(y,w) = \mathbf{true} \\ &= \mathbf{true} && \text{if } neq_F(x,z) = \mathbf{false} \text{ and } neq_F(y,w) = \mathbf{true} \\ &= \mathbf{true} && \text{if } neq_F(x,z) = \mathbf{true} \text{ and } neq_F(y,w) = \mathbf{false} \\ &= \mathbf{false} && \text{if } neq_F(x,z) = \mathbf{false} \text{ and } neq_F(y,w) = \mathbf{false} \\ &= \mathbf{invalid}(\mathbf{true}) && \text{otherwise} \end{aligned}$$

$$lss_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$$
$$lss_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = lss_F(y,w)$$

$$leq_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$$
$$leq_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = leq_F(y,w)$$

$$gtr_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$$
$$gtr_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = gtr_F(y,w)$$

$$geq_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathbf{Boolean}$$
$$geq_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = geq_F(y,w)$$

5.2.5.2 Multiplication by the imaginary unit

$$itimes_{F\to\mathrm{i}(F)} : F \to \mathrm{i}(F)$$
$$itimes_{F\to\mathrm{i}(F)}(x) = \hat{\imath}\cdot x$$

$$itimes_{\mathrm{i}(F)\to F} : \mathrm{i}(F) \to F \cup \{\mathbf{-0}\}$$
$$itimes_{\mathrm{i}(F)\to F}(\hat{\imath}\cdot y) = neg_F(y)$$

$$itimes_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\})$$
$$itimes_{\mathrm{c}(F)}(x \mathbin{\hat{+}\!\hat{\imath}\cdot} y) = neg_F(y) \mathbin{\hat{+}\!\hat{\imath}\cdot} x$$

5.2.5.3 The real and imaginary parts of a complex value

$re_F : F \to F$

$re_F(x)$	$= x$	if $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
	$= no\_result_F(x)$	otherwise

$re_{\mathrm{i}(F)} : \mathrm{i}(F) \to \{\mathbf{-0}, 0\}$

$re_{\mathrm{i}(F)}(\hat{\imath}\cdot y)$	$= 0$	if $(y \in F$ and $y \geqslant 0)$ or $y = +\infty$
	$= \mathbf{-0}$	if $(y \in F$ and $y < 0)$ or $y \in \{-\infty, \mathbf{-0}\}$
	$= no\_result_F(y)$	otherwise

$re_{\mathrm{c}(F)} : \mathrm{c}(F) \to F$

$re_{\mathrm{c}(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot y)$	$= x$	if $x \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
	$= no\_result_F(x)$	otherwise

$im_F : F \to \{\mathbf{-0}, 0\}$

$im_F(x)$	$= \mathbf{-0}$	if $(x \in F$ and $x \geqslant 0)$ or $x = +\infty$
	$= 0$	if $(x \in F$ and $x < 0)$ or $x \in \{-\infty, \mathbf{-0}\}$
	$= no\_result_F(x)$	otherwise

$im_{\mathrm{i}(F)} : \mathrm{i}(F) \to F$

$im_{\mathrm{i}(F)}(\hat{\imath}\cdot y)$	$= y$	if $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
	$= no\_result_F(y)$	otherwise

$im_{\mathrm{c}(F)} : \mathrm{c}(F) \to F$

$im_{\mathrm{c}(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot y)$	$= y$	if $y \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
	$= no\_result_F(y)$	otherwise

5.2.5.4 Formation of a complex floating point from two floating point values

$plusitimes_F : F \times F \to \mathrm{c}(F)$

$plusitimes_F(x, z)$
$\quad = x \mathbin{\hat{+}} \hat{\imath}\cdot z$

5.2.5.5 Fundamental complex floating point arithmetic

NOTE 1 – The addition and subtraction operations never underflow for an IEC 60559 implementation.

$neg_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F \cup \{\mathbf{-0}\})$

$neg_{\mathrm{i}(F)}(\hat{\imath}\cdot y) \quad = \hat{\imath}\cdot neg_F(y)$

$$neg_{c(F)} : c(F) \to c(F \cup \{\mathbf{-0}\})$$
$$neg_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y) = neg_F(x) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} neg_F(y)$$

$$conj_F : F \to F$$
$$conj_F(x) = x$$

$$conj_{i(F)} : i(F) \to i(F \cup \{\mathbf{-0}\})$$
$$conj_{i(F)}(\hat{\imath}\boldsymbol{\cdot}\, y) = \hat{\imath}\boldsymbol{\cdot}\, neg_F(y)$$

$$conj_{c(F)} : c(F) \to c(F \cup \{\mathbf{-0}\})$$
$$conj_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y) = x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} neg_F(y)$$

$$add_{i(F)} : i(F) \times i(F) \to i(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$add_{i(F)}(\hat{\imath}\boldsymbol{\cdot}\, y, \hat{\imath}\boldsymbol{\cdot}\, w) = \hat{\imath}\boldsymbol{\cdot}\, add_F(y, w)$$

$$add_{F,i(F)} : F \times i(F) \to c(F)$$
$$add_{F,i(F)}(x, \hat{\imath}\boldsymbol{\cdot}\, w) = x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} w$$

$$add_{i(F),F} : i(F) \times F \to c(F)$$
$$add_{i(F),F}(\hat{\imath}\boldsymbol{\cdot}\, y, z) = z \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y$$

$$add_{F,c(F)} : F \times c(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$add_{F,c(F)}(x, z \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} w) = add_F(x, z) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} w$$

$$add_{c(F),F} : c(F) \times F \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$add_{c(F),F}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y, z) = add_F(x, z) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y$$

$$add_{i(F),c(F)} : i(F) \times c(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$add_{i(F),c(F)}(\hat{\imath}\boldsymbol{\cdot}\, y, z \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} w) = z \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} add_F(y, w)$$

$$add_{c(F),i(F)} : c(F) \times i(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$add_{c(F),i(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y, \hat{\imath}\boldsymbol{\cdot}\, w) = x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} add_F(y, w)$$

$$add_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$add_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w) = add_F(x, z) \mathbin{+\hat{\imath}\cdot} add_F(y, w)$$

$$sub_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$sub_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w) = \hat{\imath}\cdot\, sub_F(y, w)$$

$$sub_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\})$$
$$sub_{F,\mathrm{i}(F)}(x, \hat{\imath}\cdot\, w) = x \mathbin{+\hat{\imath}\cdot} neg_F(w)$$

$$sub_{\mathrm{i}(F),F} : \mathrm{i}(F) \times F \to \mathrm{c}(F \cup \{\mathbf{-0}\})$$
$$sub_{\mathrm{i}(F),F}(\hat{\imath}\cdot\, y, z) = neg_F(z) \mathbin{+\hat{\imath}\cdot} y$$

$$sub_{F,\mathrm{c}(F)} : F \times \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$sub_{F,\mathrm{c}(F)}(x, z \mathbin{+\hat{\imath}\cdot} w) = sub_F(x, z) \mathbin{+\hat{\imath}\cdot} neg_F(w)$$

$$sub_{\mathrm{c}(F),F} : \mathrm{c}(F) \times F \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$sub_{\mathrm{c}(F),F}(x \mathbin{+\hat{\imath}\cdot} y, z) = sub_F(x, z) \mathbin{+\hat{\imath}\cdot} y$$

$$sub_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$sub_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot\, y, z \mathbin{+\hat{\imath}\cdot} w) = neg_F(z) \mathbin{+\hat{\imath}\cdot} sub_F(y, w)$$

$$sub_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$sub_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot\, w) = x \mathbin{+\hat{\imath}\cdot} sub_F(y, w)$$

$$sub_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$sub_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w) = sub_F(x, z) \mathbin{+\hat{\imath}\cdot} sub_F(y, w)$$

$$mul_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{overflow}\}$$
$$mul_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w) = neg_F(mul_F(y, w))$$

$$mul_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \to \mathrm{i}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$
$$mul_{F,\mathrm{i}(F)}(x, \hat{\imath}\cdot\, w) = \hat{\imath}\cdot\, mul_F(x, w)$$

$mul_{\mathrm{i}(F),F} : \mathrm{i}(F) \times F \rightarrow \mathrm{i}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$mul_{\mathrm{i}(F),F}(\hat{\imath}\cdot\, y, z) = \hat{\imath}\cdot\, mul_F(y, z)$

$mul_{F,\mathrm{c}(F)} : F \times \mathrm{c}(F) \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$mul_{F,\mathrm{c}(F)}(x, z \mathbin{+\hat{\imath}\cdot}\, w)$
$= mul_F(x, z) \mathbin{+\hat{\imath}\cdot}\, mul_F(x, w)$

$mul_{\mathrm{c}(F),F} : \mathrm{c}(F) \times F \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$mul_{\mathrm{c}(F),F}(x \mathbin{+\hat{\imath}\cdot}\, y, z)$
$= mul_F(x, z) \mathbin{+\hat{\imath}\cdot}\, mul_F(y, z)$

$mul_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$mul_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot\, y, z \mathbin{+\hat{\imath}\cdot}\, w)$
$= neg_F(mul_F(y, w)) \mathbin{+\hat{\imath}\cdot}\, mul_F(y, z)$

$mul_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$mul_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{+\hat{\imath}\cdot}\, y, \hat{\imath}\cdot\, w)$
$= neg_F(mul_F(y, w)) \mathbin{+\hat{\imath}\cdot}\, mul_F(x, w)$

NOTE 2 – $mul_{\mathrm{c}(F)}$ is specified in clause 5.2.6.

$div_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \rightarrow F \cup \{\mathbf{-0}, \mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w) = div_F(y, w)$

$div_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \rightarrow \mathrm{i}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{F,\mathrm{i}(F)}(x, \hat{\imath}\cdot\, w) = \hat{\imath}\cdot\, neg_F(div_F(x, w))$

$div_{\mathrm{i}(F),F} : \mathrm{i}(F) \times F \rightarrow \mathrm{i}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{\mathrm{i}(F),F}(\hat{\imath}\cdot\, y, z) = \hat{\imath}\cdot\, div_F(y, z)$

$div_{F,\mathrm{c}(F)} : F \times \mathrm{c}(F) \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{F,\mathrm{c}(F)}(x, z \mathbin{+\hat{\imath}\cdot}\, w)$
$= div_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot}\, im_F(x), z \mathbin{+\hat{\imath}\cdot}\, w)$

$div_{\mathrm{c}(F),F} : \mathrm{c}(F) \times F \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{\mathrm{c}(F),F}(x \mathbin{+\hat{\imath}\cdot}\, y, z)$
$= div_F(x, z) \mathbin{+\hat{\imath}\cdot}\, div_F(y, z)$

$div_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \rightarrow \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot\, y, z \mathbin{+\hat{\imath}\cdot}\, w)$
$= div_{\mathrm{c}(F)}(re_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) \mathbin{+\hat{\imath}\cdot}\, y, z \mathbin{+\hat{\imath}\cdot}\, w)$

$div_{c(F),i(F)} : c(F) \times i(F) \rightarrow c(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$div_{c(F),i(F)}(x \mathbin{\hat{+}\mathbf{\hat{\imath}}\cdot} y, \mathbf{\hat{\imath}}\cdot w)$
$\quad = div_F(y, w) \mathbin{\mathbf{+\hat{\imath}\cdot}} neg_F(div_F(x, w))$

NOTE 3 – $div_{c(F)}$ is specified in clause 5.2.6.

5.2.5.6 Absolute value, phase and signum of complex floating point values

NOTES

1 This clause also includes $signum_F$.

2 abs_F is specified in part 1.

$abs_{i(F)} : i(F) \rightarrow F$

$abs_{i(F)}(\mathbf{\hat{\imath}\cdot}\, y) \quad = abs_F(y)$

$abs_{c(F)} : c(F) \rightarrow F \cup \{\mathbf{underflow}, \mathbf{overflow}\}$

$abs_{c(F)}(x \mathbin{\mathbf{+\hat{\imath}\cdot}} y) \quad = hypot_F(x, y)$

$phase_F : F \rightarrow F$

$phase_F(x) \quad = arc_F(x, im_F(x))$

NOTE 3 – The arc_F (and similarly $arcu_F$) operation as specified in clause 5.3.8.15 of the *first edition* (issued in 2001) of part 2 has a minor flaw; it is not properly limited. See annex F for a corrected version. Implementations are recommended to follow this improvement, if possible, for arc_F, $arcu_F$, and all operations that reference those specifications.

$phase_{i(F)} : i(F) \rightarrow F$

$phase_{i(F)}(\mathbf{\hat{\imath}\cdot}\, y) \quad = arc_F(re_{i(F)}(\mathbf{\hat{\imath}\cdot}\, y), y)$

$phase_{c(F)} : c(F) \rightarrow F \cup \{\mathbf{underflow}\}$

$phase_{c(F)}(x \mathbin{\mathbf{+\hat{\imath}\cdot}} y)$
$\quad = arc_F(x, y)$

$phaseu_F : F \times F \rightarrow F \cup \{\mathbf{invalid}\}$

$phaseu_F(u, x) \quad = arcu_F(u, x, im_F(x))$

$phaseu_{i(F)} : F \times i(F) \rightarrow F \cup \{\mathbf{invalid}\}$

$phaseu_{i(F)}(u, \mathbf{\hat{\imath}\cdot}\, y)$
$\quad = arcu_F(u, re_{i(F)}(\mathbf{\hat{\imath}\cdot}\, y), y)$

$phaseu_{c(F)} : F \times c(F) \rightarrow F \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$phaseu_{c(F)}(u, x \mathbin{\mathbf{+\hat{\imath}\cdot}} y)$
$\quad = arcu_F(u, x, y)$

$signum_F : F \to \{-1, 1\}$

$$
\begin{array}{lll}
signum_F(x) & = 1 & \text{if } (x \in F \text{ and } x \geqslant 0) \text{ or } x = \mathbf{+\infty} \\
 & = -1 & \text{if } (x \in F \text{ and } x < 0) \text{ or } x \in \{\mathbf{-\infty}, \mathbf{-0}\} \\
 & = no\_result_F(x) & \text{otherwise}
\end{array}
$$

$signum_{\mathrm{i}(F)} : \mathrm{i}(F) \to \{\hat{\imath}\cdot\, (-1), \hat{\imath}\cdot\, 1\}$

$signum_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, signum_F(y)$

NOTE 4 – $signum_{\mathrm{c}(F)}$ is specified in clause 5.2.6.

5.2.5.7 Floor, round, and ceiling

$floor_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F)$

$floor_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, floor_F(y)$

$floor_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F)$

$floor_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot}\, y)$
$= floor_F(x) \mathbin{+\hat{\imath}\cdot}\, floor_F(y)$

$rounding_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F)$

$rounding_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y)$
$= \hat{\imath}\cdot\, rounding_F(y)$

$rounding_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F)$

$rounding_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot}\, y)$
$= rounding_F(x) \mathbin{+\hat{\imath}\cdot}\, rounding_F(y)$

$ceiling_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F)$

$ceiling_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, ceiling_F(y)$

$ceiling_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F)$

$ceiling_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot}\, y)$
$= ceiling_F(x) \mathbin{+\hat{\imath}\cdot}\, ceiling_F(y)$

5.2.5.8 Maximum and minimum

$max_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathrm{i}(F)$

$max_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w)$
$= \hat{\imath}\cdot\, max_F(y, w)$

$mmax_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathrm{i}(F)$

$mmax_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y, \hat{\imath}\cdot\, w)$
$= \hat{\imath}\cdot\, mmax_F(y, w)$

$min_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \rightarrow \mathrm{i}(F)$

$min_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$= \hat{\imath}\cdot min_F(y, w)$

$mmin_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \rightarrow \mathrm{i}(F)$

$mmin_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w)$
$= \hat{\imath}\cdot mmin_F(y, w)$

$max\_seq_{\mathrm{i}(F)} : [\mathrm{i}(F)] \rightarrow \mathrm{i}(F) \cup \{\mathbf{infinitary}\}$

$max\_seq_{\mathrm{i}(F)}([\hat{\imath}\cdot y_1, ..., \hat{\imath}\cdot y_n])$
$= \hat{\imath}\cdot max\_seq_F([y_1, ..., y_n])$

$mmax\_seq_{\mathrm{i}(F)} : [\mathrm{i}(F)] \rightarrow \mathrm{i}(F) \cup \{\mathbf{infinitary}\}$

$mmax\_seq_{\mathrm{i}(F)}([\hat{\imath}\cdot y_1, ..., \hat{\imath}\cdot y_n])$
$= \hat{\imath}\cdot mmax\_seq_F([y_1, ..., y_n])$

$min\_seq_{\mathrm{i}(F)} : [\mathrm{i}(F)] \rightarrow \mathrm{i}(F) \cup \{\mathbf{infinitary}\}$

$min\_seq_{\mathrm{i}(F)}([\hat{\imath}\cdot y_1, ..., \hat{\imath}\cdot y_n])$
$= \hat{\imath}\cdot min\_seq_F([y_1, ..., y_n])$

$mmin\_seq_{\mathrm{i}(F)} : [\mathrm{i}(F)] \rightarrow \mathrm{i}(F) \cup \{\mathbf{infinitary}\}$

$mmin\_seq_{\mathrm{i}(F)}([\hat{\imath}\cdot y_1, ..., \hat{\imath}\cdot y_n])$
$= \hat{\imath}\cdot mmin\_seq_F([y_1, ..., y_n])$

5.2.6 Complex sign, multiplication, and division

This clause gives specifications for some operations that have been deferred above, due to the possibility of lesser accuracy.

There shall be a box error mode parameter each for the multiplication and division operations on a complex datatype:

$box\_error\_mode\_mul_{\mathrm{c}(F)} \in \mathbf{Boolean}$
$box\_error\_mode\_div_{\mathrm{c}(F)} \in \mathbf{Boolean}$

There shall be a maximum error parameter each for multiplication and division on a complex datatype:

$max\_error\_mul_{\mathrm{c}(F)} \in F$
$max\_error\_div_{\mathrm{c}(F)} \in F$

The $max\_error\_mul_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 5$. The $max\_error\_div_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 13$.

For use in the specification below, define the mathematical complex sign function

$signum : \mathcal{C} \rightarrow \mathcal{C}$

The *signum* function is defined by

$signum(z) = \cos(arc(\mathfrak{Re}(z), \mathfrak{Im}(z))) + (\tilde{\imath} \cdot \sin(arc(\mathfrak{Re}(z), \mathfrak{Im}(z))))$

NOTES

1 The *arc* function is defined in part 2, Clause 5.3.7.

2 Note that $z = |z| \cdot signum(z)$ if $z \in \mathcal{C}$, and $signum(x + (\tilde{\imath} \cdot y)) = e^{\tilde{\imath} \cdot arc(x,y)}$ if $x, y \in \mathcal{R}$.

5.2.6.1 Complex signum

The $signum^*_{\mathrm{c}(F)}$ approximation helper function:

$signum^*_{\mathrm{c}(F)} : \mathcal{C}_F \to \mathcal{C}$

$signum^*_{\mathrm{c}(F)}(z)$ returns a close approximation to $signum(z)$ in $\mathcal{C}$, with maximum error $max\_error\_tan_F$ (interpreted as an error parameter for a complex floating point operation).

Further requirements on the $signum^*_{\mathrm{c}(F)}$ approximation helper function are:

$signum^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(signum^*_{\mathrm{c}(F)}(z))$ if $z \in \mathcal{C}_F$

$signum^*_{\mathrm{c}(F)}(-z) = -signum^*_{\mathrm{c}(F)}(z)$ if $z \in \mathcal{C}_F$

$\mathfrak{Re}(signum^*_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot y))) = \mathfrak{Im}(signum^*_{\mathrm{c}(F)}(y + (\tilde{\imath} \cdot x)))$ if $x, y \in F$

$signum^*_{\mathrm{c}(F)}(x) = 1$ if $x \in F$ and $x > 0$

$signum^*_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot x)) = (1/\sqrt{2}) + (\tilde{\imath} \cdot (1/\sqrt{2}))$ if $x \in F$ and $x > 0$

The $signum_{\mathrm{c}(F)}$ operation:

$signum_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}\}$

$$\begin{aligned}
signum_{\mathrm{c}(F)}(x \,\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y) &= result^*_{\mathrm{c}(F)}(signum^*_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) && \text{if } x, y \in F \text{ and } x + (\tilde{\imath} \cdot y) \neq 0 \\
&= conj_{\mathrm{c}(F)}(signum_{\mathrm{c}(F)}(conj_{\mathrm{c}(F)}(x \,\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y))) && \text{if } y \in \{\mathbf{-\infty}, \mathbf{-0}\} \\
&= neg_{\mathrm{c}(F)}(signum_{\mathrm{c}(F)}(neg_{\mathrm{c}(F)}(x \,\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y))) && \text{if } x \in \{\mathbf{-\infty}, \mathbf{-0}\} \text{ and } y \notin \{\mathbf{-\infty}, \mathbf{-0}\} \\
&= sin_F(arc_F(y, x)) \,\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, sin_F(arc_F(x, y)) && \text{otherwise}
\end{aligned}$$

NOTE – $signum_{\mathrm{c}(F)}(x \,\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, y) \approx cos_F(arc_F(x, y)) \,\boldsymbol{+}\hat{\boldsymbol{\imath}}\boldsymbol{\cdot}\, sin_F(arc_F(x, y))$. The specification is more complicated, in order to allow higher accuracy, including the sign of zero result parts.

5.2.6.2 Complex multiplication

The $mul^*_{\mathrm{c}(F)}$ approximation helper function:

$mul^*_{\mathrm{c}(F)} : \mathcal{C}_F \times \mathcal{C}_F \to \mathcal{C}$

$mul^*_{\mathrm{c}(F)}(x, z)$ returns a close approximation to $x \cdot z$ in $\mathcal{C}$ with maximum error $max\_error\_mul_{\mathrm{c}(F)}$, interpreted according to the value of $box\_error\_mode\_mul_{\mathrm{c}(F)}$.

Further requirements on the $mul^*_{\mathrm{c}(F)}$ approximation helper function are:

$$mul^*_{\mathrm{c}(F)}(\mathrm{conj}(z), \mathrm{conj}(z')) = \mathrm{conj}(mul^*_{\mathrm{c}(F)}(z, z')) \quad \text{if } z, z' \in \mathcal{C}_F$$
$$mul^*_{\mathrm{c}(F)}(-z, z') = -mul^*_{\mathrm{c}(F)}(z, z') \quad \text{if } z, z' \in \mathcal{C}_F$$
$$mul^*_{\mathrm{c}(F)}(z, z') = mul^*_{\mathrm{c}(F)}(z', z) \quad \text{if } z, z' \in \mathcal{C}_F$$

The $mul_{\mathrm{c}(F)}$ operation:

$$mul_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$

$$\begin{aligned} mul_{\mathrm{c}(F)}(x \,\hat{+}\!\hat{\imath}\cdot\, y, z \,\hat{+}\!\hat{\imath}\cdot\, w) \\ &= result^*_{\mathrm{c}(F)}(mul^*_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot y), z + (\tilde{\imath} \cdot w)), nearest_F) \\ &\qquad \text{if } x \,\hat{+}\!\hat{\imath}\cdot\, y, z \,\hat{+}\!\hat{\imath}\cdot\, w \in \mathrm{c}(F) \text{ and} \\ &\qquad\quad x \neq 0 \text{ and } y \neq 0 \text{ and } z \neq 0 \text{ and } w \neq 0 \\ &= sub_F(mul_F(x, z), mul_F(y, w)) \,\hat{+}\!\hat{\imath}\cdot\, add_F(mul_F(y, z), mul_F(x, w)) \\ &\qquad \text{otherwise} \end{aligned}$$

NOTE – NaN (and **invalid**) is not avoided for the result in the "otherwise" case here. Note in particular cases like $mul_{\mathrm{c}(F)}(2 \,\hat{+}\!\hat{\imath}\cdot\, (\mathbf{-0}), 3 \,\hat{+}\!\hat{\imath}\cdot\, (+\infty))$ which is **invalid** with a continuation value of $\mathbf{qNaN} \,\hat{+}\!\hat{\imath}\cdot\, (+\infty)$. However, $mul_{F,\mathrm{c}(F)}(2, 3 \,\hat{+}\!\hat{\imath}\cdot\, (+\infty))$ returns $6 \,\hat{+}\!\hat{\imath}\cdot\, (+\infty)$, due to the strong implicit zero (see B.5.2.4).

5.2.6.3 Complex division

The $div^*_{\mathrm{c}(F)}$ approximation helper function:

$$div^*_{\mathrm{c}(F)} : \mathcal{C}_F \times \mathcal{C}_F \to \mathcal{C}$$

$div^*_{\mathrm{c}(F)}(x, z)$ returns a close approximation to x/z in $\mathcal{C}$ with maximum error $max\_error\_div_{\mathrm{c}(F)}$, interpreted according to the value of $box\_error\_mode\_div_{\mathrm{c}(F)}$.

Further requirements on the $div^*_{\mathrm{c}(F)}$ approximation helper function are:

$$div^*_{\mathrm{c}(F)}(\mathrm{conj}(z), \mathrm{conj}(z')) = \mathrm{conj}(div^*_{\mathrm{c}(F)}(z, z')) \quad \text{if } z, z' \in \mathcal{C}_F \text{ and } z' \neq 0$$
$$div^*_{\mathrm{c}(F)}(-z, z') = -div^*_{\mathrm{c}(F)}(z, z') \quad \text{if } z, z' \in \mathcal{C}_F \text{ and } z' \neq 0$$
$$div^*_{\mathrm{c}(F)}(z, -z') = -div^*_{\mathrm{c}(F)}(z, z') \quad \text{if } z, z' \in \mathcal{C}_F \text{ and } z' \neq 0$$

The $div_{\mathrm{c}(F)}$ operation:

$$div_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$

$$\begin{aligned} div_{\mathrm{c}(F)}(x \,\hat{+}\!\hat{\imath}\cdot\, y, z \,\hat{+}\!\hat{\imath}\cdot\, w) \\ &= result^*_{\mathrm{c}(F)}(div^*_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot y), z + (\tilde{\imath} \cdot w)), nearest_F) \\ &\qquad \text{if } x \,\hat{+}\!\hat{\imath}\cdot\, y, z \,\hat{+}\!\hat{\imath}\cdot\, w \in \mathrm{c}(F) \text{ and} \\ &\qquad\quad x \neq 0 \text{ and } y \neq 0 \text{ and } z \neq 0 \text{ and } w \neq 0 \\ &= div_{\mathrm{c}(F),F}(add_F(mul_F(x, z), mul_F(y, w)) \,\hat{+}\!\hat{\imath}\cdot\, sub_F(mul_F(y, z), mul_F(x, w)), \\ &\qquad add_F(mul_F(z, z), mul_F(w, w))) \\ &\qquad \text{otherwise} \end{aligned}$$

5.2.7 Operations for conversion from polar to Cartesian

The $polar^*_{c(F)}$ approximation helper function:

$$polar^*_{c(F)} : F \times F^{2\cdot\pi} \to \mathcal{C}$$

$polar^*_{c(F)}(x, z)$ returns a close approximation to $x \cdot e^{\tilde{\imath}\cdot z}$ in $\mathcal{C}$ with maximum error $max\_error\_tan_F$ (interpreted as an error parameter for a complex floating point operation).

Further requirements on the $polar^*_{c(F)}$ approximation helper function are:

$polar^*_{c(F)}(-x, z) = -polar^*_{c(F)}(x, z)$ if $x, z \in F$ and $x \geqslant 0$
$polar^*_{c(F)}(x, -z) = \mathrm{conj}(polar^*_{c(F)}(x, z))$ if $x, z \in F$

$polar^*_{c(F)}(x, n \cdot 2 \cdot \pi) = x$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$
$\mathfrak{Im}(polar^*_{c(F)}(x, n \cdot 2 \cdot \pi + \pi/6)) = x/2$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$
$\mathfrak{Re}(polar^*_{c(F)}(x, n \cdot 2 \cdot \pi + \pi/3)) = x/2$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$
$polar^*_{c(F)}(x, n \cdot 2 \cdot \pi + \pi/2) = \tilde{\imath} \cdot x$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$
$\mathfrak{Re}(polar^*_{c(F)}(x, n \cdot 2 \cdot \pi + 2 \cdot \pi/3)) = -x/2$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$
$\mathfrak{Im}(polar^*_{c(F)}(x, n \cdot 2 \cdot \pi + 5 \cdot \pi/6)) = x/2$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$
$polar^*_{c(F)}(x, n \cdot 2 \cdot \pi + \pi) = -x$ if $x \in F$ and $n \in \mathcal{Z}$ and $|n \cdot 2 \cdot \pi| \leqslant big\_angle\_r_F$

The $polar_{c(F)}$ operation:

$$polar_{c(F)} : F \times F \to c(F) \cup \{\textbf{underflow}, \textbf{absolute\_precision\_underflow}\}$$

$polar_{c(F)}(x, z) = result^*_{c(F)}(polar^*_{c(F)}(x, z), nearest_F)$
if $x, z \in F$ and $|z| \leqslant big\_angle\_r_F$ and $x \neq 0$ and $z \neq 0$
$= mul_F(x, cos_F(z)) \,\hat{+}\hat{\imath}\cdot\, mul_F(x, sin_F(z))$
otherwise

The $polaru^*_{c(F)}$ approximation helper function:

$$polaru^*_{c(F)} : (u : F) \times F \times F^u \to \mathcal{C}$$

$polaru^*_{c(F)}(u, x, z)$ returns a close approximation to $x \cdot e^{\tilde{\imath}\cdot 2\cdot\pi\cdot z/u}$ in $\mathcal{C}$ with maximum error $max\_error\_tanu_F(u)$ (interpreted as an error parameter for a complex floating point operation).

Further requirements on the $polaru^*_{c(F)}$ approximation helper function are:

$polaru^*_{c(F)}(u, -x, z) = -polaru^*_{c(F)}(u, x, z)$ if $u, x \in F$ and $z \in F^u$ and $u \neq 0$ and $x \geqslant 0$
$polaru^*_{c(F)}(u, x, -z) = \mathrm{conj}(polaru^*_{c(F)}(u, x, z))$ if $u, x \in F$ and $z \in F^u$ and $u \neq 0$
$polaru^*_{c(F)}(-u, x, z) = polaru^*_{c(F)}(u, x, -z)$ if $u, x \in F$ and $z \in F^u$ and $u \neq 0$

$polaru^*_{c(F)}(u, x, (n \cdot u) + z) = polaru^*_{c(F)}(u, x, z)$ if $u, x \in F$ and $z, (n \cdot u) + z \in F^u$ and $u \neq 0$ and $n \in \mathcal{Z}$

$polaru^*_{c(F)}(u, x, 0) = x$ if $u, x \in F$ and $u \neq 0$
$\mathfrak{Im}(polaru^*_{c(F)}(u, x, u/12)) = x/2$ if $u, x \in F$ and $u \neq 0$
$\mathfrak{Re}(polaru^*_{c(F)}(u, x, u/6)) = x/2$ if $u, x \in F$ and $u \neq 0$
$polaru^*_{c(F)}(u, x, u/4) = \tilde{\imath} \cdot x$ if $u, x \in F$ and $u \neq 0$
$\mathfrak{Re}(polaru^*_{c(F)}(u, x, u/3)) = -x/2$ if $u, x \in F$ and $u \neq 0$

$\mathfrak{Im}(polaru^*_{\mathrm{c}(F)}(u, x, 5 \cdot u/12)) = x/2$ if $u, x \in F$ and $u \neq 0$
$polaru^*_{\mathrm{c}(F)}(u, x, n \cdot u/2) = -x$ if $u, x \in F$ and $u \neq 0$

The $polaru_{\mathrm{c}(F)}$ operation:

$polaru_{\mathrm{c}(F)} : F \times F \times F \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{invalid}\}$

$polaru_{\mathrm{c}(F)}(u, x, z)$
$= result^*_{\mathrm{c}(F)}(polaru^*_{\mathrm{c}(F)}(u, x, z), nearest_F)$
if $u \in G_F$ and $x, z \in F$ and $|z/u| \leqslant big\_angle\_u_F$ and $x \neq 0$ and $z \neq 0$
$= mul_F(x, cosu_F(u, z)) \,\mathbf{+\hat{\imath}\cdot}\, mul_F(x, sinu_F(u, z))$
otherwise

NOTE – G_F is defined in part 2.

5.3 Elementary transcendental imaginary and complex floating point operations

Clause 5.3 of part 2 specifies a number of transcendental floating point operations. In this clause a number of transcendental imaginary and complex floating point operations are specified.

Several of the specifications below include relationships to specifications for operations specified in part 2. The requirements implied by these relationships and the requirements from part 2 shall hold even if none or only some of the mentioned operations are included in an associated library (which may be the same library as for the complex operations) for real-valued operations or even if there is no associated library for real-valued operations.

5.3.1 Operations for exponentiations and logarithms

There shall be two maximum error parameters for complex exponentiations and logarithms.

$max\_error\_exp_{\mathrm{c}(F)} \in F$
$max\_error\_power_{\mathrm{c}(F)} \in F$

The $max\_error\_exp_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 7$. The $max\_error\_power_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 15$.

5.3.1.1 Exponentiation of imaginary base to integer power

The $poweri_{\mathrm{i}(F),I}$ operation:

$poweri_{\mathrm{i}(F),I} : \mathrm{i}(F) \times I \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}\}$

$poweri_{\mathrm{i}(F),I}(\hat{\imath}\cdot\, x, y)$
$= re_{\mathrm{i}(F)}(\hat{\imath}\cdot\, x) \,\mathbf{+\hat{\imath}\cdot}\, power_{F,I}(x, y)$
if $y \in I$ and $4|(y + 3)$
$= neg_F(power_{F,I}(abs_F(x), y)) \,\mathbf{+\hat{\imath}\cdot}\, 0$
if $y \in I$ and $4|(y + 2)$
$= neg_{\mathrm{c}(F)}(re_{\mathrm{i}(F)}(\hat{\imath}\cdot\, x) \,\mathbf{+\hat{\imath}\cdot}\, power_{F,I}(x, y))$
if $y \in I$ and $4|(y + 1)$
$= power_{F,I}(abs_F(x), y) \,\mathbf{+\hat{\imath}\cdot}\, (\mathbf{-0})$
if $y \in I$ and $4|y$

5.3.1.2 Natural exponentiation

The $exp_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$$exp_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\}$$

$$\begin{aligned} exp_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y) & \\ &= cos_F(y) + \hat{\imath}\cdot sin_F(y) \end{aligned}$$

NOTE 1 – Some programming languages have the operation `cis`. `cis`(x) is $exp_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot x)$.

The $exp^*_{\mathrm{c}(F)}$ approximation helper function:

$$exp^*_{\mathrm{c}(F)} : \mathcal{C}_F \to \mathcal{C}$$

$exp^*_{\mathrm{c}(F)}(z)$ returns a close approximation to e^z in $\mathcal{C}$ with maximum error $max\_error\_exp_{\mathrm{c}(F)}$.

A further requirement on the $exp^*_{\mathrm{c}(F)}$ approximation helper function is:

$$exp^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(exp^*_{\mathrm{c}(F)}(z)) \qquad \text{if } z \in \mathcal{C}_F$$

The relationships to the cos^*_F, sin^*_F, and exp^*_F approximation helper functions for the cos_F, sin_F, and exp_F operations in an associated library for real-valued operations shall be:

$$\begin{aligned} exp^*_{\mathrm{c}(F)}(\tilde{\imath}\cdot y) &= cos^*_F(y) + (\tilde{\imath}\cdot sin^*_F(y)) && \text{if } y \in F \\ exp^*_{\mathrm{c}(F)}(x) &= exp^*_F(x) && \text{if } x \in F \end{aligned}$$

The $exp_{\mathrm{c}(F)}$ operation:

$$exp_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$

$$\begin{aligned} exp_{\mathrm{c}(F)}(x + \hat{\imath}\cdot y) &= result^*_{\mathrm{c}(F)}(exp^*_{\mathrm{c}(F)}(x + (\tilde{\imath}\cdot y)), nearest_F) \\ & \qquad \text{if } x, y \in F \text{ and } |y| \leqslant big\_angle\_r_F \\ &= exp_{\mathrm{c}(F)}(0 + \hat{\imath}\cdot y) \qquad \text{if } x = \mathbf{-0} \\ &= conj_{\mathrm{c}(F)}(exp_{\mathrm{c}(F)}(x + \hat{\imath}\cdot 0)) \\ & \qquad \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\ &= mul_F(0, cos_F(y)) + \hat{\imath}\cdot mul_F(0, sin_F(y)) \\ & \qquad \text{if } x = \mathbf{-\infty} \text{ and } y \in F \text{ and } |y| \leqslant big\_angle\_r_F \\ &= mul_F(\mathbf{+\infty}, cos_F(y)) + \hat{\imath}\cdot mul_F(\mathbf{+\infty}, sin_F(y)) \\ & \qquad \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } |y| \leqslant big\_angle\_r_F \text{ and} \\ & \qquad\quad y \neq 0 \\ &= (\mathbf{+\infty}) + \hat{\imath}\cdot 0 \qquad \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y = 0 \\ &= radh_{\mathrm{c}(F)}(x + \hat{\imath}\cdot y) \qquad \text{otherwise} \end{aligned}$$

NOTES

2 $radh_{\mathrm{c}(F)}$ is specified in Clause 5.3.3.1.

3 **invalid** is avoided here for the cases $exp_{\mathrm{c}(F)}((\mathbf{+\infty}) + \hat{\imath}\cdot 0)$ and $exp_{\mathrm{c}(F)}((\mathbf{+\infty}) + \hat{\imath}\cdot (\mathbf{-0}))$.

5.3.1.3 Complex exponentiation of argument base

The $power^*_{F\to\mathrm{c}(F)}$ helper function:

$$power^*_{F\to\mathrm{c}(F)} : F \times F \to \mathcal{C}$$

$power^*_{F\to\mathrm{c}(F)}(x, z)$ returns a close approximation to x^z in $\mathcal{C}$ with maximum error $max\_error\_power_{\mathrm{c}(F)}$.

NOTE – $x^z = (-1\cdot(-x))^z = (-1)^z \cdot (-x)^z = e^{\tilde{\imath}\cdot\pi\cdot z}\cdot(-x)^z$ if $x < 0$

Further requirements on the $power^*_{F\to\mathrm{c}(F)}$ are:

$\mathfrak{Re}(power^*_{F\to c(F)}(x,z)) = \mathfrak{Im}(power^*_{F\to c(F)}(x,z))$
  if $x, z \in F$ and $x < 0$ and $z - 0.25 \in \mathcal{Z}$
$\mathfrak{Re}(power^*_{F\to c(F)}(x,z)) = -\mathfrak{Im}(power^*_{F\to c(F)}(x,z))$
  if $x, z \in F$ and $x < 0$ and $z - 0.75 \in \mathcal{Z}$

The relationships to the $power^*_F$, $sinu^*_F$, and $cosu^*_F$ helper functions for the $power_F$, $sinu_F$, and $cosu_F$ operations in an associated library for real-valued operations shall be:

$power^*_{F\to c(F)}(x,z) = power^*_F(x,z)$ if $x, z \in F$ and $x \geqslant 0$

$power^*_{F\to c(F)}(x,z) = power^*_F(-x,z)$ if $x, z \in F$ and $x < 0$ and $z/2 \in \mathcal{Z}$
$power^*_{F\to c(F)}(x,z) = \tilde{\imath}\cdot power^*_F(-x,z)$ if $x, z \in F$ and $x < 0$ and $(z - 0.5)/2 \in \mathcal{Z}$
$power^*_{F\to c(F)}(x,z) = -power^*_F(-x,z)$ if $x, z \in F$ and $x < 0$ and $(z - 1)/2 \in \mathcal{Z}$
$power^*_{F\to c(F)}(x,z) = -\tilde{\imath}\cdot power^*_F(-x,z)$ if $x, z \in F$ and $x < 0$ and $(z - 1.5)/2 \in \mathcal{Z}$

$power^*_{F\to c(F)}(-1,z) = cosu^*_F(2,z) + (\tilde{\imath}\cdot sinu^*_F(2,z))$

The $power_{F\to c(F)}$ operation:

$power_{F\to c(F)} : F \times F \to c(F) \cup$
  $\{\textbf{underflow}, \textbf{overflow}, \textbf{absolute\_precision\_underflow}, \textbf{infinitary}, \textbf{invalid}\}$

$power_{F\to c(F)}(x,z)$
$= result^*_{c(F)}(power^*_{F\to c(F)}(x,z), nearest_F)$
  if $x, z \in F$ and ($x \geqslant 0$ or
  ($x < 0$ and $|z/2| \leqslant big\_angle\_u_F$))
$= power_F(x,z) \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; 0$ if $x \in F$ and $x \geqslant 0$ and $z \in \{-\infty, \mathbf{-0}, +\infty\}$
$= power_F(x,z) \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; 0$ if $x = +\infty$ and $z \in F \cup \{-\infty, \mathbf{-0}, +\infty\}$
$= mul_F(power_F(-x,z), cosu_F(2,z)) \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; sinu_F(2,z)$
  if $x \in F$ and $x < 0$ and $z \in \{-\infty, \mathbf{-0}, +\infty\}$
$= mul_F(power_F(neg_F(x),z), cosu_F(2,z)) \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; sinu_F(2,z)$
  if $x \in \{-\infty, \mathbf{-0}\}$ and $z \in \{-\infty, \mathbf{-0}, +\infty\}$
$= mul_{F,c(F)}(power_F(neg_F(x),z), cosu_F(2,z) \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; sinu_F(2,z))$
  if $x \in \{-\infty, \mathbf{-0}\}$ and $z \in F$ and $|z/2| \leqslant big\_angle\_u_F$
$= \textbf{absolute\_precision\_underflow}(\textbf{qNaN} \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; \textbf{qNaN})$
  if $x \in F$ and $x < 0$ and $z \in F$ and $|z/2| > big\_angle\_u_F$
$= \textbf{absolute\_precision\_underflow}(\textbf{qNaN} \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; \textbf{qNaN})$
  if $x \in \{-\infty, \mathbf{-0}\}$ and $z \in F$ and $|z/2| > big\_angle\_u_F$
$= no\_result2_{c(F)}(x,z)$ otherwise

5.3.1.4 Complex square root

The $sqrt_{F\to c(F)}$ operation:

$sqrt_{F\to c(F)} : F \to c(F)$

$sqrt_{F\to c(F)}(x)$ $= 0 \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; sqrt_F(neg_F(x))$ if ($x \in F$ and $x < 0$) or $x \in \{-\infty, \mathbf{-0}\}$
$= sqrt_F(x) \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; (\mathbf{-0})$ if ($x \in F$ and $x \geqslant 0$) or $x = +\infty$
$= no\_result_{c(F)}(x \;\hat{+}\boldsymbol{\hat{\imath}}\cdot\; im_F(x))$
  otherwise

The $sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F)$$

$$\begin{aligned} &sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y) \\ &\qquad = sqrt_{\mathrm{c}(F)}(re_{\mathrm{i}(F)}(\hat{\imath}\cdot y) \mathbin{+\hat{\imath}\cdot} y) \end{aligned}$$

The $sqrt^*_{\mathrm{c}(F)}$ approximation helper function:

$$sqrt^*_{\mathrm{c}(F)} : \mathcal{C}_F \to \mathcal{C}$$

$sqrt^*_{\mathrm{c}(F)}(z)$ returns a close approximation to $\sqrt{z}$ in $\mathcal{C}$ with maximum error $max\_error\_exp_{\mathrm{c}(F)}$.

Further requirements on the $sqrt^*_{\mathrm{c}(F)}$ approximation helper function are:

$$\begin{array}{ll} sqrt^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(sqrt^*_{\mathrm{c}(F)}(z)) & \text{if } z \in \mathcal{C}_F \\ sqrt^*_{\mathrm{c}(F)}(x) = \sqrt{x} & \text{if } x \in F \text{ and } x \geqslant 0 \\ sqrt^*_{\mathrm{c}(F)}(x) = \tilde{\imath}\cdot\sqrt{-x} & \text{if } x \in F \text{ and } x < 0 \\ \mathfrak{Re}(sqrt^*_{\mathrm{c}(F)}(\tilde{\imath}\cdot y)) = \mathfrak{Im}(sqrt^*_{\mathrm{c}(F)}(\tilde{\imath}\cdot y)) & \text{if } y \in F \text{ and } y \geqslant 0 \end{array}$$

The $sqrt_{\mathrm{c}(F)}$ operation:

$$sqrt_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F)$$

$$\begin{array}{lll} sqrt_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) & & \\ & = result^*_{\mathrm{c}(F)}(sqrt^*_{\mathrm{c}(F)}(x + (\tilde{\imath}\cdot y)), nearest_F) & \\ & & \text{if } x, y \in F \\ & = sqrt_{\mathrm{c}(F)}(0 \mathbin{+\hat{\imath}\cdot} y) & \text{if } x = \mathbf{-0} \\ & = conj_{\mathrm{c}(F)}(sqrt_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0)) & \\ & & \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y = \mathbf{-0} \\ & = (\mathbf{+\infty}) \mathbin{+\hat{\imath}\cdot} (\mathbf{+\infty}) & \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y = \mathbf{+\infty} \\ & = (\mathbf{+\infty}) \mathbin{+\hat{\imath}\cdot} 0 & \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y \geqslant 0 \\ & = (\mathbf{+\infty}) \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}) & \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y < 0 \\ & = (\mathbf{+\infty}) \mathbin{+\hat{\imath}\cdot} (\mathbf{-\infty}) & \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y = \mathbf{-\infty} \\ & = 0 \mathbin{+\hat{\imath}\cdot} (\mathbf{+\infty}) & \text{if } x = \mathbf{-\infty} \text{ and } y \in F \text{ and } y \geqslant 0 \\ & = 0 \mathbin{+\hat{\imath}\cdot} (\mathbf{-\infty}) & \text{if } x = \mathbf{-\infty} \text{ and } y \in F \text{ and } y < 0 \\ & = no\_result_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) & \text{otherwise} \end{array}$$

NOTE – The inverse of complex square is multi-valued. The principal result is given by $\sqrt{b} = e^{0.5\cdot\ln(b)}$. The $\sqrt{\ }$ function branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } x < 0\}$. Thus $sqrt_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0) \neq sqrt_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}))$ when $x < 0$.

5.3.1.5 Natural logarithm

The $ln_{F\to\mathrm{c}(F)}$ operation:

$$ln_{F\to\mathrm{c}(F)} : F \to \mathrm{c}(F) \cup \{\mathbf{infinitary}\}$$

$$ln_{F\to\mathrm{c}(F)}(x) \quad = ln_F(abs_F(x)) \mathbin{+\hat{\imath}\cdot} arc_F(x, im_F(x))$$

NOTE 1 – The arc_F (and similarly $arcu_F$) operation as specified in Clause 5.3.8.15 of the *first edition* (issued in 2001) of part 2 has a minor flaw as noted in Annex F of this document.

The $ln_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$$ln_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{infinitary}\}$$

$$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y) = ln_F(abs_F(y)) \mathbin{+\hat{\imath}\cdot} arc_F(re_{\mathrm{i}(F)}(\hat{\imath}\cdot y), y)$$

The $ln^*_{\mathrm{c}(F)}$ approximation helper function:

$ln^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$

$ln^*_{c(F)}(z)$ returns a close approximation to $\ln(z)$ in $\mathcal{C}$ with maximum error $max\_error\_exp_{c(F)}$ in the real part.

NOTE 2 – Since the imaginary part of the result of $ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot y)$ is $arc_F(x, y)$, the maximum error in the imaginary part is $max\_error\_tan_F$. Thus $max\_error\_exp_{c(F)}$ reflects the maximum error in the real part of the result.

A further requirement on the $ln^*_{c(F)}$ approximation helper function is:

$ln^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(ln^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$

The relationships to the arc^*_F and ln^*_F approximation helper functions for the arc_F and ln_F operations in an associated library for real-valued operations shall be:

$\mathfrak{Im}(ln^*_{c(F)}(x + (\tilde{\imath} \cdot y))) = arc^*_F(x, y)$ if $x, y \in F$ and $x + (\tilde{\imath} \cdot y) \neq 0$
$\mathfrak{Re}(ln^*_{c(F)}(x)) = ln^*_F(|x|)$ if $x \in F$ and $x \neq 0$
$\mathfrak{Re}(ln^*_{c(F)}(\tilde{\imath} \cdot y)) = ln^*_F(|y|)$ if $y \in F$ and $y \neq 0$

The $ln^{\#}_{c(F)}$ range limitation helper function (for $z \in \mathcal{C}_F$):

$ln^{\#}_{c(F)}(z) = \mathfrak{Re}(ln^*_{c(F)}(z)) + (\tilde{\imath} \cdot \max\{up_F(-\pi), \min\{\mathfrak{Im}(ln^*_{c(F)}(z)), down_F(-\pi/2)\}\})$
if $\mathfrak{Re}(z) < 0$ and $\mathfrak{Im}(z) < 0$
$= \mathfrak{Re}(ln^*_{c(F)}(z)) + (\tilde{\imath} \cdot \max\{up_F(-\pi/2), \min\{\mathfrak{Im}(ln^*_{c(F)}(z)), down_F(\pi/2)\}\})$
if $\mathfrak{Re}(z) \geqslant 0$
$= \mathfrak{Re}(ln^*_{c(F)}(z)) + (\tilde{\imath} \cdot \max\{up_F(\pi/2), \min\{\mathfrak{Im}(ln^*_{c(F)}(z)), down_F(\pi)\}\})$
if $\mathfrak{Re}(z) < 0$ and $\mathfrak{Im}(z) \geqslant 0$

The $ln_{c(F)}$ operation:

$ln_{c(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\textbf{infinitary}\}$

$ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot y) = result^*_{c(F)}(ln^{\#}_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F)$
if $x, y \in F$ and $(x \neq 0$ or $y \neq 0)$
$= \textbf{infinitary}((-\infty) \,\hat{+}\,\hat{\imath}\cdot arc_F(x, y))$
if $x \in \{-\mathbf{0}, 0\}$ and $y = 0$
$= conj_{c(F)}(ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot 0))$
if $y = -\mathbf{0}$
$= ln_F(y) \,\hat{+}\,\hat{\imath}\cdot up_F(\pi/2)$ if $x = -\mathbf{0}$ and $((y \in F$ and $y > 0)$ or $y = +\infty)$
$= ln_F(neg_F(y)) \,\hat{+}\,\hat{\imath}\cdot down_F(-\pi/2)$
if $x = -\mathbf{0}$ and $((y \in F$ and $y < 0)$ or $y = -\infty)$
$= (+\infty) \,\hat{+}\,\hat{\imath}\cdot arc_F(x, y)$ if $x \in \{-\infty, +\infty\}$ and $y \in F \cup \{-\infty, +\infty\}$
$= (+\infty) \,\hat{+}\,\hat{\imath}\cdot arc_F(x, y)$ if $x \in F$ and $y \in \{-\infty, +\infty\}$
$= no\_result_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot y)$ otherwise

NOTES

3 The inverse of natural exponentiation is multi-valued: the imaginary part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The ln function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R}$ and $x < 0\}$, is continuous on the rest of $\mathcal{C}$, and $\ln(z) \in \mathcal{R}$ if $x \in \mathcal{R}$ and $x > 0$. Thus $ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot 0) \neq ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot (-\mathbf{0}))$ when $x < 0$.

4 $re_{c(F)}(ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot y)) \approx ln_F(hypot_F(x, y))$ and $im_{c(F)}(ln_{c(F)}(x \,\hat{+}\,\hat{\imath}\cdot y)) = arc_F(x, y)$ when there is no notification (if the specification of arc_F is corrected as noted in Clause 5.2.5).

5.3.2 Operations for radian trigonometric elementary functions

There shall be two maximum error parameters for complex trigonometric operations.

$max\_error\_sin_{\mathrm{c}(F)} \in F$
$max\_error\_tan_{\mathrm{c}(F)} \in F$

The $max\_error\_sin_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 11$. The $max\_error\_tan_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 14$.

5.3.2.1 Radian angle normalisation

$$rad_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F)$$
$$rad_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot y$$

$$rad_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$rad_{\mathrm{c}(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot y) = rad_F(x) \mathbin{\hat{+}} \hat{\imath}\cdot y$$

5.3.2.2 Radian sine

The $sin_{\mathrm{i}(F)}$ operation:

$$sin_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{overflow}\}$$
$$sin_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot sinh_F(y)$$

The $sin^*_{\mathrm{c}(F)}$ approximation helper function:

$$sin^*_{\mathrm{c}(F)} : \mathcal{C}_F \to \mathcal{C}$$

$sin^*_{\mathrm{c}(F)}(z)$ returns a close approximation to $\sin(z)$ in $\mathcal{C}$ with maximum error $max\_error\_sin_{\mathrm{c}(F)}$.

Further requirements on the $sin^*_{\mathrm{c}(F)}$ approximation helper function are:

$$sin^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(sin^*_{\mathrm{c}(F)}(z)) \qquad \text{if } z \in \mathcal{C}_F$$
$$sin^*_{\mathrm{c}(F)}(-z) = -sin^*_{\mathrm{c}(F)}(z) \qquad \text{if } z \in \mathcal{C}_F$$

The relationships to the sin^*_F and $sinh^*_F$ approximation helper functions for sin_F and $sinh_F$ operations in an associated library for real-valued operations shall be:

$$sin^*_{\mathrm{c}(F)}(x) = sin^*_F(x) \qquad \text{if } x \in F$$
$$sin^*_{\mathrm{c}(F)}(\tilde{\imath}\cdot y) = \tilde{\imath}\cdot sinh^*_F(y) \qquad \text{if } y \in F$$

The $sin_{\mathrm{c}(F)}$ operation:

$$sin_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$

$$\begin{aligned}
sin_{\mathrm{c}(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot y) &= result^*_{\mathrm{c}(F)}(sin^*_{\mathrm{c}(F)}(x + (\tilde{\imath}\cdot y)), nearest_F) \\
&\qquad \text{if } x, y \in F \text{ and } |x| \leqslant big\_angle\_r_F \\
&= conj_{\mathrm{c}(F)}(sin_{\mathrm{c}(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot 0)) \\
&\qquad \text{if } y = \mathbf{-0} \\
&= neg_{\mathrm{c}(F)}(sin_{\mathrm{c}(F)}(0 \mathbin{\hat{+}} \hat{\imath}\cdot neg_F(y))) \\
&\qquad \text{if } x = \mathbf{-0} \text{ and } y \neq \mathbf{-0} \\
&= mul_F(sin_F(x), \mathbf{+\infty}) \mathbin{\hat{+}} \hat{\imath}\cdot mul_F(cos_F(x), y) \\
&\qquad \text{if } x \notin \{\mathbf{-0}, 0\} \text{ and } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \\
&= 0 \mathbin{\hat{+}} \hat{\imath}\cdot y \qquad \text{if } x = 0 \text{ and } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\}
\end{aligned}$$

$= rad_{c(F)}(x + \hat{\imath}\cdot y)$ otherwise

5.3.2.3 Radian cosine

The $cos_{i(F)}$ operation:

$cos_{i(F)} : i(F) \to F \cup \{\textbf{overflow}\}$

$cos_{i(F)}(\hat{\imath}\cdot y) = cosh_F(y)$

The $cos^*_{c(F)}$ approximation helper function:

$cos^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$

$cos^*_{c(F)}(z)$ returns a close approximation to $\cos(z)$ in $\mathcal{C}$ with maximum error $max\_error\_sin_{c(F)}$.

Further requirements on the $cos^*_{c(F)}$ approximation helper function are:

$cos^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(cos^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$
$cos^*_{c(F)}(-z) = cos^*_{c(F)}(z)$ if $z \in \mathcal{C}_F$

The relationships to the cos^*_F and $cosh^*_F$ approximation helper functions for cos_F and $cosh_F$ operations in an associated library for real-valued operations shall be:

$cos^*_{c(F)}(x) = cos^*_F(x)$ if $x \in F$
$cos^*_{c(F)}(\tilde{\imath}\cdot y) = cosh^*_F(y)$ if $y \in F$

The $cos_{c(F)}$ operation:

$cos_{c(F)} : c(F) \to c(F) \cup \{\textbf{underflow}, \textbf{overflow}, \textbf{absolute\_precision\_underflow}\}$

$cos_{c(F)}(x + \hat{\imath}\cdot y) = result^*_{c(F)}(cos^*_{c(F)}(x + (\tilde{\imath}\cdot y)), nearest_F)$
if $x, y \in F$ and $|x| \leqslant big\_angle\_r_F$
$= conj_{c(F)}(cos_{c(F)}(x + \hat{\imath}\cdot 0))$
if $y = \mathbf{-0}$
$= cos_{c(F)}(0 + \hat{\imath}\cdot neg_F(y))$ if $x = \mathbf{-0}$ and $y \neq \mathbf{-0}$
$= mul_F(cos_F(x), +\infty) + \hat{\imath}\cdot mul_F(sin_F(x), neg_F(y))$
if $x \notin \{\mathbf{-0}, 0\}$ and $y \in \{-\infty, +\infty\}$
$= (+\infty) + \hat{\imath}\cdot div_F(-1, y)$ if $x = 0$ and $y \in \{-\infty, +\infty\}$
$= rad_{c(F)}(x + \hat{\imath}\cdot y)$ otherwise

5.3.2.4 Radian tangent

The $tan_{i(F)}$ operation:

$tan_{i(F)} : i(F) \to i(F)$

$tan_{i(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot tanh_F(y)$

The $tan^*_{c(F)}$ approximation helper function:

$tan^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$

$tan^*_{c(F)}(z)$ returns a close approximation to $\tan(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $tan^*_{c(F)}$ approximation helper function are:

$tan^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(tan^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$
$tan^*_{c(F)}(-z) = -tan^*_{c(F)}(z)$ if $z \in \mathcal{C}_F$

The relationships to the tan^*_F and $tanh^*_F$ approximation helper functions for tan_F and $tanh_F$ operations in an associated library for real-valued operations shall be:

$$\begin{array}{ll} tan^*_{c(F)}(x) = tan^*_F(x) & \text{if } x \in F \\ tan^*_{c(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot tanh^*_F(y) & \text{if } y \in F \end{array}$$

The $tan_{c(F)}$ operation:

$$tan_{c(F)} : c(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$

$$\begin{array}{lll} tan_{c(F)}(x \,\hat{+}\hat{\imath}\cdot\, y) & = result^*_{c(F)}(tan^*_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) & \\ & & \text{if } x, y \in F \text{ and } |x| \leqslant big\_angle\_r_F \\ & = conj_{c(F)}(tan_{c(F)}(x \,\hat{+}\hat{\imath}\cdot\, 0)) & \\ & & \text{if } y = \mathbf{-0} \\ & = neg_F(tan_{c(F)}(0 \,\hat{+}\hat{\imath}\cdot\, neg_F(y))) & \\ & & \text{if } x = \mathbf{-0} \text{ and } y \neq \mathbf{-0} \\ & = mul_F(tan_F(x), 0) \,\hat{+}\hat{\imath}\cdot\, signum_F(y) & \\ & & \text{if } x \neq \mathbf{-0} \text{ and } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \\ & = rad_{c(F)}(x \,\hat{+}\hat{\imath}\cdot\, y) & \text{otherwise} \end{array}$$

5.3.2.5 Radian cotangent

The $cot_{i(F)}$ operation:

$$cot_{i(F)} : i(F) \to i(F) \cup \{\mathbf{overflow}, \mathbf{infinitary}\}$$

$$cot_{i(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, neg_F(coth_F(y))$$

The $cot^*_{c(F)}$ approximation helper function:

$$cot^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$$

$cot^*_{c(F)}(z)$ returns a close approximation to $\cot(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $cot^*_{c(F)}$ approximation helper function are:

$$\begin{array}{ll} cot^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(cot^*_{c(F)}(z)) & \text{if } z \in \mathcal{C}_F \\ cot^*_{c(F)}(-z) = -cot^*_{c(F)}(z) & \text{if } z \in \mathcal{C}_F \end{array}$$

The relationships to the cot^*_F and $coth^*_F$ approximation helper functions for cot_F and $coth_F$ operations in an associated library for real-valued operations shall be:

$$\begin{array}{ll} cot^*_{c(F)}(x) = cot^*_F(x) & \text{if } x \in F \\ cot^*_{c(F)}(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot coth^*_F(y) & \text{if } y \in F \end{array}$$

The $cot_{c(F)}$ operation:

$$cot_{c(F)} : c(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$$

$$\begin{array}{lll} cot_{c(F)}(x \,\hat{+}\hat{\imath}\cdot\, y) & = result^*_{c(F)}(cot^*_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) & \\ & & \text{if } x, y \in F \text{ and } |x| \leqslant big\_angle\_r_F \text{ and} \\ & & \quad (x \neq 0 \text{ or } y \neq 0) \\ & = \mathbf{infinitary}((\mathbf{+\infty}) \,\hat{+}\hat{\imath}\cdot\, (\mathbf{-\infty})) & \\ & & \text{if } x = 0 \text{ and } y = 0 \\ & = conj_{c(F)}(cot_{c(F)}(x \,\hat{+}\hat{\imath}\cdot\, 0)) & \\ & & \text{if } y = \mathbf{-0} \\ & = neg_F(cot_{c(F)}(0 \,\hat{+}\hat{\imath}\cdot\, neg_F(y))) & \\ & & \text{if } x = \mathbf{-0} \text{ and } y \neq \mathbf{-0} \end{array}$$

$= mul_F(tan_F(x), 0) \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, neg_F(signum_F(y))$
  if $x \neq \mathbf{-0}$ and $y \in \{\mathbf{-\infty}, \mathbf{+\infty}\}$
$= rad_{c(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, y)$ otherwise

NOTE – The reference to tan_F (instead of cot_F) for the case when x is different from $\mathbf{-0}$ and y is an infinity, is in order to avoid **invalid** for $cot_{c(F)}$.

5.3.2.6 Radian secant

The $sec_{i(F)}$ operation:

$sec_{i(F)} : i(F) \rightarrow F \cup \{\textbf{underflow}\}$

$sec_{i(F)}(\hat{\imath}\cdot\, y) = sech_F(y)$

The $sec^*_{c(F)}$ approximation helper function:

$sec^*_{c(F)} : \mathcal{C}_F \rightarrow \mathcal{C}$

$sec^*_{c(F)}(z)$ returns a close approximation to $\sec(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $sec^*_{c(F)}$ approximation helper function are:

$sec^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(sec^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$
$sec^*_{c(F)}(-z) = sec^*_{c(F)}(z)$ if $z \in \mathcal{C}_F$

The relationships to the sec^*_F and $sech^*_F$ approximation helper functions for sec_F and $sech_F$ operations in an associated library for real-valued operations shall be:

$sec^*_{c(F)}(x) = sec^*_F(x)$ if $x \in F$
$sec^*_{c(F)}(\tilde{\imath} \cdot y) = sech^*_F(y)$ if $y \in F$

The $sec_{c(F)}$ operation:

$sec_{c(F)} : c(F) \rightarrow c(F) \cup \{\textbf{underflow}, \textbf{overflow}, \textbf{absolute\_precision\_underflow}\}$

$sec_{c(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, y) = result^*_{c(F)}(sec^*_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F)$
  if $x, y \in F$ and $|x| \leqslant big\_angle\_r_F$
$= conj_{c(F)}(sec_{c(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, 0))$
  if $y = \mathbf{-0}$
$= sec_{c(F)}(0 \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, neg_F(y))$ if $x = \mathbf{-0}$ and $y \neq \mathbf{-0}$
$= mul_F(cos_F(x), 0) \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, div_F(sin_F(x), y)$
  if $x \neq \mathbf{-0}$ and $y \in \{\mathbf{-\infty}, \mathbf{+\infty}\}$
$= rad_{c(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot\, y)$ otherwise

5.3.2.7 Radian cosecant

The $csc_{i(F)}$ operation:

$csc_{i(F)} : i(F) \rightarrow i(F) \cup \{\textbf{underflow}, \textbf{overflow}, \textbf{infinitary}\}$

$csc_{i(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, neg_F(csch_F(y))$

The $csc^*_{c(F)}$ approximation helper function:

$csc^*_{c(F)} : \mathcal{C}_F \rightarrow \mathcal{C}$

$csc^*_{c(F)}(z)$ returns a close approximation to $\csc(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $csc^*_{c(F)}$ approximation helper function are:

$csc^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(csc^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$
$csc^*_{c(F)}(-z) = -csc^*_{c(F)}(z)$ if $z \in \mathcal{C}_F$

The relationships to the csc^*_F and $csch^*_F$ approximation helper functions for csc_F and $csch_F$ operations in an associated library for real-valued operations shall be:

$csc^*_{c(F)}(x) = csc^*_F(x)$ if $x \in F$
$csc^*_{c(F)}(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot csch^*_F(y)$ if $y \in F$

The $csc_{c(F)}$ operation:

$$csc_{c(F)} : c(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$$

$$\begin{aligned}
csc_{c(F)}(x \mathbin{+\hat{\imath}\cdot} y) &= result^*_{c(F)}(csc^*_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) \\
&\qquad \text{if } x, y \in F \text{ and } |x| \leqslant big\_angle\_r_F \text{ and } (x \neq 0 \text{ or } y \neq 0) \\
&= \mathbf{infinitary}((+\infty) \mathbin{+\hat{\imath}\cdot} (-\infty)) \\
&\qquad \text{if } x = 0 \text{ and } y = 0 \\
&= conj_{c(F)}(csc_{c(F)}(x \mathbin{+\hat{\imath}\cdot} 0)) \\
&\qquad \text{if } y = \mathbf{-0} \\
&= neg_F(csc_{c(F)}(0 \mathbin{+\hat{\imath}\cdot} neg_F(y))) \\
&\qquad \text{if } x = \mathbf{-0} \text{ and } y \neq \mathbf{-0} \\
&= mul_F(sin_F(x), 0) \mathbin{+\hat{\imath}\cdot} div_F(cos_F(x), neg_F(y)) \\
&\qquad \text{if } x \neq \mathbf{-0} \text{ and } y \in \{-\infty, +\infty\} \\
&= rad_{c(F)}(x \mathbin{+\hat{\imath}\cdot} y) \qquad \text{otherwise}
\end{aligned}$$

5.3.2.8 Radian arc sine

The $arcsin_{F \to c(F)}$ operation:

$$arcsin_{F \to c(F)} : F \to c(F \cup \{\mathbf{-0}\})$$

$$\begin{aligned}
arcsin_{F \to c(F)}(x) &= up_F(-\pi/2) \mathbin{+\hat{\imath}\cdot} arccosh_F(neg_F(x)) \\
&\qquad \text{if } (x \in F \text{ and } x < -1) \text{ or } x = -\infty \\
&= arcsin_F(x) \mathbin{+\hat{\imath}\cdot} mul_F(\mathbf{-0}, x) \\
&\qquad \text{if } (x \in F \text{ and } |x| \leqslant 1) \text{ or } x = \mathbf{-0} \\
&= down_F(\pi/2) \mathbin{+\hat{\imath}\cdot} neg_F(arccosh_F(x)) \\
&\qquad \text{if } (x \in F \text{ and } x > 1) \text{ or } x = +\infty \\
&= no\_result_{F \to c(F)}(x) \qquad \text{otherwise}
\end{aligned}$$

The $arcsin_{i(F)}$ operation:

$$arcsin_{i(F)} : i(F) \to i(F)$$

$$arcsin_{i(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot arcsinh_F(y)$$

The $arcsin^*_{c(F)}$ approximation helper function:

$$arcsin^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$$

$arcsin^*_{c(F)}(z)$ returns a close approximation to $\arcsin(z)$ in $\mathcal{C}$ with maximum error $max\_error\_sin_{c(F)}$.

Further requirements on the $arcsin^*_{c(F)}$ approximation helper function are:

$arcsin^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arcsin^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$ and $(|\mathfrak{Re}(z)| \leqslant 1$ or $\mathfrak{Im}(z) \neq 0)$
$arcsin^*_{c(F)}(-z) = -arcsin^*_{c(F)}(z)$ if $z \in \mathcal{C}_F$ and $(|\mathfrak{Re}(z)| \leqslant 1$ or $\mathfrak{Im}(z) \neq 0)$

The relationships to the $arcsin^*_F$, $arcsinh^*_F$, and $arccosh^*_F$ approximation helper functions for $arcsin_F$, $arcsinh_F$, and $arccosh_F$ operations in an associated library for real-valued operations shall be:

$$\begin{aligned}
&arcsin^*_{\mathrm{c}(F)}(x) = (-\pi/2) + (\tilde{\imath} \cdot arccosh^*_F(-x)) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{if } x \in F \text{ and } x < -1 \\
&arcsin^*_{\mathrm{c}(F)}(x) = arcsin^*_F(x) \qquad\qquad\qquad \text{if } x \in F \text{ and } |x| \leqslant 1 \\
&arcsin^*_{\mathrm{c}(F)}(x) = (\pi/2) + (\tilde{\imath} \cdot arccosh^*_F(x)) \quad \text{if } x \in F \text{ and } x > 1 \\
&arcsin^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot arcsinh^*_F(y) \qquad\quad \text{if } y \in F
\end{aligned}$$

The $arcsin^{\#}_{\mathrm{c}(F)}$ range limitation helper function:

$$arcsin^{\#}_{\mathrm{c}(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arcsin^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\}\} + (\tilde{\imath} \cdot \mathfrak{Im}(arcsin^*_{\mathrm{c}(F)}(z)))$$

The $arcsin_{\mathrm{c}(F)}$ operation:

$$arcsin_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}\}$$

$$\begin{aligned}
&arcsin_{\mathrm{c}(F)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y) \\
&\qquad = result^*_{\mathrm{c}(F)}(arcsin^{\#}_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } x, y \in F \\
&\qquad = neg_{\mathrm{c}(F)}(arcsin_{\mathrm{c}(F)}(0 \,\hat{+}\mathbf{\hat{\imath}}\cdot\, neg_F(y))) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } x = \mathbf{-0} \\
&\qquad = conj_{\mathrm{c}(F)}(arcsin_{\mathrm{c}(F)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, 0)) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\
&\qquad = arc_F(abs_F(y), x) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, mul_F(signum_F(y), \mathbf{+\infty}) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } (x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\}) \text{ or} \\
&\qquad\qquad\qquad\qquad\qquad\ (y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } x \in F) \\
&\qquad = no\_result_{\mathrm{c}(F)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, y) \quad \text{otherwise}
\end{aligned}$$

NOTE – The inverse of sin is multi-valued, the real part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The arcsin function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } |x| > 1\}$. Thus $arcsin_{\mathrm{c}(F)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, 0) \neq arcsin_{\mathrm{c}(F)}(x \,\hat{+}\mathbf{\hat{\imath}}\cdot\, (\mathbf{-0}))$ when $|x| > 1$.

5.3.2.9 Radian arc cosine

The $arccos_{F\to\mathrm{c}(F)}$ operation:

$$arccos_{F\to\mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{\mathbf{-0}\})$$

$$\begin{aligned}
&arccos_{F\to\mathrm{c}(F)}(x) = down_F(\pi) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, neg_F(arccosh_F(neg_F(x))) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } (x \in F \text{ and } x < -1) \text{ or } x = \mathbf{-\infty} \\
&\qquad = arccos_F(x) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, (\mathbf{-0}) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } (x \in F \text{ and } |x| \leqslant 1) \text{ or } x = \mathbf{-0} \\
&\qquad = 0 \,\hat{+}\mathbf{\hat{\imath}}\cdot\, arccosh_F(x) \qquad \text{if } (x \in F \text{ and } x > 1) \text{ or } x = \mathbf{+\infty} \\
&\qquad = no\_result_{F\to\mathrm{c}(F)}(x) \qquad \text{otherwise}
\end{aligned}$$

The $arccos_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$$arccos_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F)$$

$$\begin{aligned}
&arccos_{\mathrm{i}(F)\to\mathrm{c}(F)}(\mathbf{\hat{\imath}}\cdot\, y) \\
&\qquad = up_F(\pi/2) \,\hat{+}\mathbf{\hat{\imath}}\cdot\, neg_F(arcsinh_F(y)) \\
&\qquad\qquad\qquad\qquad\qquad \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{\mathbf{-\infty}, \mathbf{-0}\}
\end{aligned}$$

$$\begin{aligned}
&= down_F(\pi/2) \mathbin{+\hat{\imath}\cdot} neg_F(arcsinh_F(y)) \\
&\qquad\qquad \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = +\infty \\
&= no\_result_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot\, y) \quad \text{otherwise}
\end{aligned}$$

The $arccos^*_{\mathrm{c}(F)}$ approximation helper function:

$$arccos^*_{\mathrm{c}(F)} : \mathcal{C}_F \to \mathcal{C}$$

$arccos^*_{\mathrm{c}(F)}(z)$ returns a close approximation to $\arccos(z)$ in $\mathcal{C}$ with maximum error $max\_error\_sin_{\mathrm{c}(F)}$.

Further requirements on the $arccos^*_{\mathrm{c}(F)}$ approximation helper function are:

$$\begin{aligned}
&arccos^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arccos^*_{\mathrm{c}(F)}(z)) && \text{if } z \in \mathcal{C}_F \text{ and } (|\mathfrak{Re}(z)| \leqslant 1 \text{ or } \mathfrak{Im}(z) \neq 0) \\
&\mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(-z)) = -\mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z)) && \text{if } z \in \mathcal{C}_F \text{ and } (|\mathfrak{Re}(z)| \leqslant 1 \text{ or } \mathfrak{Im}(z) \neq 0)
\end{aligned}$$

The relationships to the $arccos^*_F$, $arccosh^*_F$, and $arcsinh^*_F$ approximation helper functions for $arccos_F$, $arccosh_F$, and $arcsinh_F$ operations in an associated library for real-valued operations shall be:

$$\begin{aligned}
&arccos^*_{\mathrm{c}(F)}(x) = \pi - (\tilde{\imath} \cdot arccosh^*_F(-x)) && \text{if } x \in F \text{ and } x \leqslant -1 \\
&arccos^*_{\mathrm{c}(F)}(x) = arccos^*_F(x) && \text{if } x \in F \text{ and } |x| < 1 \\
&arccos^*_{\mathrm{c}(F)}(x) = -(\tilde{\imath} \cdot arccosh^*_F(x)) && \text{if } x \in F \text{ and } x \geqslant 1 \\
&arccos^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y) = (\pi/2) - (\tilde{\imath} \cdot arcsinh^*_F(y)) && \text{if } y \in F
\end{aligned}$$

The $arccos^{\#}_{\mathrm{c}(F)}$ range limitation helper function:

$$\begin{aligned}
arccos^{\#}_{\mathrm{c}(F)}(z) &= \max\{up_F(\pi/2), \min\{\mathfrak{Re}(arccos^*_{\mathrm{c}(F)}(z)), down_F(\pi)\}\} + (\tilde{\imath} \cdot \mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z))) \\
&\qquad\qquad \text{if } \mathfrak{Re}(z) < 0 \\
&= \min\{\mathfrak{Re}(arccos^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\} + (\tilde{\imath} \cdot \mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z))) \\
&\qquad\qquad \text{if } \mathfrak{Re}(z) \geqslant 0
\end{aligned}$$

The $arccos_{\mathrm{c}(F)}$ operation:

$$arccos_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\})$$

$$\begin{aligned}
&arccos_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \\
&\quad = result^*_{\mathrm{c}(F)}(arccos^{\#}_{\mathrm{c}(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) \\
&\qquad\qquad \text{if } x, y \in F \text{ and } (y \neq 0 \text{ or } |x| > 1) \\
&\quad = arccos_F(x) \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}) \quad \text{if } x \in F \text{ and } y = 0 \text{ and } |x| \leqslant 1 \\
&\quad = up_F(\pi/2) \mathbin{+\hat{\imath}\cdot} neg_F(arcsinh_F(y)) \\
&\qquad\qquad \text{if } x = \mathbf{-0} \\
&\quad = conj_{\mathrm{c}(F)}(arccos_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0)) \\
&\qquad\qquad \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\
&\quad = arc_F(x, y) \mathbin{+\hat{\imath}\cdot} (-\infty) \quad \text{if } x \in \{-\infty, +\infty\} \text{ and} \\
&\qquad\qquad ((y \in F \text{ and } y \geqslant 0) \text{ or } y = +\infty) \\
&\quad = arc_F(x, neg_F(y)) \mathbin{+\hat{\imath}\cdot} (+\infty) \\
&\qquad\qquad \text{if } x \in \{-\infty, +\infty\} \text{ and} \\
&\qquad\qquad ((y \in F \text{ and } y < 0) \text{ or } y = -\infty) \\
&\quad = no\_result_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \quad \text{otherwise}
\end{aligned}$$

NOTE – The inverse of cos is multi-valued, the real part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The arccos function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } |x| > 1\}$. Thus $arccos_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0) \neq arccos_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}))$ when $|x| > 1$.

5.3.2.10 Radian arc tangent

The $arctan_{\mathrm{i}(F)}$ operation:

$$arctan_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{infinitary}, \mathbf{invalid}\}$$

$$arctan_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) \quad = \hat{\imath}\cdot\, arctanh_F(y)$$

The $arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$$arctan_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{infinitary}\}$$

$$\begin{aligned} arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}&(\hat{\imath}\cdot\, y) \\ &= up_F(-\pi/2) \mathbin{+\hat{\imath}\cdot}\, arccoth_F(y) && \\ & && \text{if } (y \in F \text{ and } y < -1) \text{ or } y = \mathbf{-\infty} \\ &= mul_F(0, y) \mathbin{+\hat{\imath}\cdot}\, arctanh_F(y) && \\ & && \text{if } (y \in F \text{ and } |y| \leqslant 1) \text{ or } y = \mathbf{-0} \\ &= down_F(\pi/2) \mathbin{+\hat{\imath}\cdot}\, arccoth_F(y) && \\ & && \text{if } (y \in F \text{ and } y > 1) \text{ or } y = \mathbf{+\infty} \\ &= no\_result_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot\, y) && \text{otherwise} \end{aligned}$$

The $arctan^*_{\mathrm{c}(F)}$ approximation helper function:

$$arctan^*_{\mathrm{c}(F)} : \mathcal{C}_F \to \mathcal{C}$$

$arctan^*_{\mathrm{c}(F)}(z)$ returns a close approximation to $\arctan(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{\mathrm{c}(F)}$.

Further requirements on the $arctan^*_{\mathrm{c}(F)}$ approximation helper function are:

$$\begin{aligned} & arctan^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arctan^*_{\mathrm{c}(F)}(z)) && \text{if } z \in \mathcal{C}_F \\ & arctan^*_{\mathrm{c}(F)}(-z) = -arctan^*_{\mathrm{c}(F)}(z) && \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Re}(z) < 1 \text{ or } \mathfrak{Im}(z) \neq 0) \end{aligned}$$

The relationships to the $arctan^*_F$, $arctanh^*_F$, and $arccoth^*_F$ approximation helper functions for $arctan_F$, $arctanh_F$, and $arccoth_F$ operations in an associated library for real-valued operations shall be:

$$\begin{aligned} & arctan^*_{\mathrm{c}(F)}(x) = arctan^*_F(x) && \text{if } x \in F \\ & arctan^*_{\mathrm{c}(F)}(\tilde{\imath}\cdot y) = \tilde{\imath}\cdot arctanh^*_F(y) && \text{if } y \in F \text{ and } |y| < 1 \\ & arctan^*_{\mathrm{c}(F)}(\tilde{\imath}\cdot y)) = (\pi/2) + (\tilde{\imath}\cdot arccoth^*_F(y)) && \\ & && \text{if } y \in F \text{ and } |y| > 1 \end{aligned}$$

The $arctan^\#_{\mathrm{c}(F)}$ range limitation helper function:

$$arctan^\#_{\mathrm{c}(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arctan^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\}\} + (\tilde{\imath}\cdot\mathfrak{Im}(arctan^*_{\mathrm{c}(F)}(z)))$$

The $arctan_{\mathrm{c}(F)}$ operation:

$$arctan_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$$

$$\begin{aligned} arctan_{\mathrm{c}(F)}&(x \mathbin{+\hat{\imath}\cdot}\, y) \\ &= result^*_{\mathrm{c}(F)}(arctan^\#_{\mathrm{c}(F)}(x + (\tilde{\imath}\cdot y)), nearest_F) && \\ & && \text{if } x, y \in F \text{ and } (x \neq 0 \text{ or } |y| \neq 1) \\ &= \mathbf{infinitary}(0 \mathbin{+\hat{\imath}\cdot}\, mul_F(y, \mathbf{+\infty})) && \\ & && \text{if } x = 0 \text{ and } y \in \{-1, 1\} \\ &= neg_{\mathrm{c}(F)}(arctan_{\mathrm{c}(F)}(0 \mathbin{+\hat{\imath}\cdot}\, neg_F(y))) && \\ & && \text{if } x = \mathbf{-0} \\ &= conj_{\mathrm{c}(F)}(arctan_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot}\, 0)) && \\ & && \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \end{aligned}$$

$= mul_F(signum_F(x), down_F(\pi/2)) \mathbin{+\hat{\imath}\cdot} mul_F(signum_F(y), 0)$
if $(x \in \{-\infty, +\infty\}$ and $y \in F \cup \{-\infty, +\infty\})$ or $(x \in F$ and $y \in \{-\infty, +\infty\})$

$= no\_result_{c(F)}(x \mathbin{+\hat{\imath}\cdot} y)$ otherwise

NOTE – The inverse of tan is multi-valued, the real part may have any integer multiple of $2 \cdot \pi$ (even any integer multiple of π) added to it, and the result is also in the solution set. The arctan function (returning the principal value for the inverse) branch cuts at $\{\tilde{\imath} \cdot y \mid y \in F \text{ and } |y| > 1\}$. Thus $arctan_{c(F)}(0 \mathbin{+\hat{\imath}\cdot} y) \neq arctan_{c(F)}((\mathbf{-0}) \mathbin{+\hat{\imath}\cdot} y)$ when $|y| > 1$.

5.3.2.11 Radian arc cotangent

The $arccot_{i(F)}$ operation:

$arccot_{i(F)} : i(F) \to i(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$arccot_{i(F)}(\hat{\imath}\cdot\, y) \quad = \hat{\imath}\cdot\, arccoth_F(neg_F(y))$

The $arccot_{i(F)\to c(F)}$ operation:

$arccot_{i(F)\to c(F)} : i(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccot_{i(F)\to c(F)}(\hat{\imath}\cdot\, y)$

$= mul_F(signum_F(y), 0) \mathbin{+\hat{\imath}\cdot} arccoth_F(neg_F(y))$
if $y \in F$ and $|y| \geqslant 1$ or $y \in \{-\infty, +\infty\}$

$= up_F(-\pi/2) \mathbin{+\hat{\imath}\cdot} arctanh_F(y)$
if $(y \in F$ and $-1 < y$ and $y < 0)$ or $y = \mathbf{-0}$

$= down_F(\pi/2) \mathbin{+\hat{\imath}\cdot} arctanh_F(y)$
if $y \in F$ and $0 \leqslant y$ and $y < 1$

$= no\_result_{i(F)\to c(F)}(\hat{\imath}\cdot\, y)$ otherwise

The $arccot^*_{c(F)}$ approximation helper function:

$arccot^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$

$arccot^*_{c(F)}(z)$ returns a close approximation to $\mathrm{arccot}(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $arccot^*_{c(F)}$ approximation helper function are:

$arccot^*_{c(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arccot^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$ and $(\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| > 1)$
$arccot^*_{c(F)}(-z) = -arccot^*_{c(F)}(z)$ if $z \in \mathcal{C}_F$ and $(\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| > 1)$
$\mathfrak{Re}(arccot^*_{c(F)}(\tilde{\imath} \cdot y)) = \pi/2$ if $y \in F$ and $|y| < 1$

The relationships to the $arccot^*_F$, $arccoth^*_F$, and $arctanh^*_F$ approximation helper functions for $arccot_F$, $arccoth_F$, and $arctanh_F$ operations in an associated library for real-valued operations shall be:

$arccot^*_{c(F)}(x) = arccot^*_F(x)$ if $x \in F$
$arccot^*_{c(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot arccoth^*_F(-y)$ if $y \in F$ and $|y| > 1$
$arccot^*_{c(F)}(\tilde{\imath} \cdot y) = (-\pi/2) + (\tilde{\imath} \cdot arctanh^*_F(y))$
if $y \in F$ and $|y| < 1$

The $arccot^{\#}_{c(F)}$ range limitation helper function:

$arccot^{\#}_{c(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arccot^*_{c(F)}(z)), down_F(\pi/2)\}\} + (\tilde{\imath} \cdot \mathfrak{Im}(arccot^*_{c(F)}(z)))$

The $arccot_{c(F)}$ operation:

$arccot_{c(F)} : c(F) \to c(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccot_{c(F)}(x \mathbin{\hat{\imath}\cdot} y)$

$= result^*_{c(F)}(arccot^\#_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F)$ if $x, y \in F$ and $(|y| \neq 1$ or $x \neq 0)$ and $y \neq 0$

$= arccot_F(x) \mathbin{\hat{\imath}\cdot} (\mathbf{-0})$ if $x \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\}$ and $y = 0$

$= neg_{c(F)}(arccot_{c(F)}(0 \mathbin{\hat{\imath}\cdot} neg_F(y)))$ if $x = \mathbf{-0}$

$= conj_{c(F)}(arccot_{c(F)}(x \mathbin{\hat{\imath}\cdot} 0))$ if $y = \mathbf{-0}$ and $x \neq \mathbf{-0}$

$= mul_F(signum_F(x), 0) \mathbin{\hat{\imath}\cdot} mul_F(signum_F(y), 0)$ if $(x \in \{\mathbf{-\infty}, \mathbf{+\infty}\}$ and $y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\})$ or $(x \in F$ and $y \in \{\mathbf{-\infty}, \mathbf{+\infty}\})$

$= \mathbf{infinitary}(0 \mathbin{\hat{\imath}\cdot} mul_F(y, \mathbf{-\infty}))$ if $x = 0$ and $y \in \{-1, 1\}$

$= no\_result_{c(F)}(x \mathbin{\hat{\imath}\cdot} y)$ otherwise

NOTE – The inverse of cot is multi-valued, the real part may have any integer multiple of $2 \cdot \pi$ (even any integer multiple of π) added to it, and the result is also in the solution set. The arccot function (returning the principal value for the inverse) branch cuts at $\{\tilde{\imath} \cdot y \mid y \in \mathcal{R}$ and $|y| < 1\}$. Thus $arccot_{c(F)}(0 \mathbin{\hat{\imath}\cdot} y) \neq arccot_{c(F)}((\mathbf{-0}) \mathbin{\hat{\imath}\cdot} y)$ when $|y| < 1$ or $y = \mathbf{-0}$.

5.3.2.12 Radian arc secant

The $arcsec_{F \to c(F)}$ operation:

$arcsec_{F \to c(F)} : F \to c(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arcsec_{F \to c(F)}(x) = arcsec_F(x) \mathbin{\hat{\imath}\cdot} mul_F(signum_F(x), \mathbf{-0})$ if $(x \in F$ and $|x| \geqslant 1)$ or $x \in \{\mathbf{-\infty}, \mathbf{+\infty}\}$

$= 0 \mathbin{\hat{\imath}\cdot} neg_F(arcsech_F(x))$ if $(x \in F$ and $-1 < x < 0)$ or $x = \mathbf{-0}$

$= down_F(\pi) \mathbin{\hat{\imath}\cdot} arcsech_F(neg_F(x))$ if $x \in F$ and $0 \leqslant x < 1$

$= no\_result_{F \to c(F)}(x)$ otherwise

The $arcsec_{i(F) \to c(F)}$ operation:

$arcsec_{i(F) \to c(F)} : i(F) \to c(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arcsec_{i(F) \to c(F)}(\hat{\imath}\cdot y)$

$= down_F(\pi/2) \mathbin{\hat{\imath}\cdot} arccsch_F(y)$ if $(y \in F$ and $y < 0)$ or $y \in \{\mathbf{-\infty}, \mathbf{-0}\}$

$= up_F(\pi/2) \mathbin{\hat{\imath}\cdot} arccsch_F(y)$ if $(y \in F$ and $y \geqslant 0)$ or $y = \mathbf{+\infty}$

$= no\_result_{i(F) \to c(F)}(\hat{\imath}\cdot y)$ otherwise

The $arcsec^*_{c(F)}$ approximation helper function:

$arcsec^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$

$arcsec^*_{c(F)}(z)$ returns a close approximation to $\text{arcsec}(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $arcsec^*_{c(F)}$ approximation helper function are:

$arcsec^*_{c(F)}(\text{conj}(z)) = \text{conj}(arcsec^*_{c(F)}(z))$ if $z \in \mathcal{C}_F$ and $(\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$

The relationships to the $arcsec^*_F$, $arcsech^*_F$, and $arccsch^*_F$ approximation helper functions for $arcsec_F$, $arcsech_F$, and $arccsch_F$ operations in an associated library for real-valued operations shall be:

$$\begin{array}{ll}
arcsec^*_{c(F)}(x) = arcsec^*_F(x) & \text{if } x \in F \text{ and } |x| \geqslant 1 \\
arcsec^*_{c(F)}(x) = -\tilde{\imath} \cdot arcsech^*_F(x) & \text{if } x \in F) \text{ and } |x| < 1 \text{ and } x \neq 0 \\
arcsec^*_{c(F)}(\tilde{\imath} \cdot y) = (\pi/2) + (\tilde{\imath} \cdot arccsch^*_F(y)) & \text{if } y \in F
\end{array}$$

The $arcsec^\#_{c(F)}$ range limitation helper function:

$$\begin{array}{lll}
arcsec^\#_{c(F)}(z) & = \min\{\mathfrak{Re}(arcsec^*_{c(F)}(z)), down_F(\pi/2)\} + (\tilde{\imath} \cdot \mathfrak{Im}(arcsec^*_{c(F)}(z))) & \\
& & \text{if } \mathfrak{Re}(z) \geqslant 0 \\
& = \max\{up_F(\pi/2), \min\{\mathfrak{Re}(arcsec^*_{c(F)}(z)), down_F(\pi)\}\} + (\tilde{\imath} \cdot \mathfrak{Im}(arcsec^*_{c(F)}(z))) & \\
& & \text{if } \mathfrak{Re}(z) < 0
\end{array}$$

The $arcsec_{c(F)}$ operation:

$$arcsec_{c(F)} : c(F) \rightarrow c(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$$

$$\begin{array}{ll}
arcsec_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y) & \\
\quad = result^*_{c(F)}(arcsec^\#_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) & \\
 & \text{if } x, y \in F \text{ and } (x \neq 0 \text{ or } y \neq 0) \\
\quad = \mathbf{infinitary}(down(\pi/2) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} (\boldsymbol{+\infty})) & \\
 & \text{if } y = 0 \text{ and } x = 0 \\
\quad = arcsec_{c(F)}(0 \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} neg_F(y)) & \\
 & \text{if } x = \mathbf{-0} \\
\quad = conj_{c(F)}(arcsec_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} 0)) & \\
 & \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\
\quad = mul_F(signum_F(x), down_F(\pi/2)) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} mul_F(signum_F(y), 0) & \\
 & \text{if } (x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\}) \text{ or} \\
 & \quad (y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } x \in F) \\
\quad = no\_result_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} y) & \text{otherwise}
\end{array}$$

NOTE – The inverse of sec is multi-valued, the real part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The arcsec function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } |x| < 1\}$. Thus $arcsec_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} 0) \neq arcsec_{c(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} (\mathbf{-0}))$ when $|x| < 1$ or $x = \mathbf{-0}$.

5.3.2.13 Radian arc cosecant

The $arccsc_{F \rightarrow c(F)}$ operation:

$$arccsc_{F \rightarrow c(F)} : F \rightarrow c(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$$

$$\begin{array}{ll}
arccsc_{F \rightarrow c(F)}(x) = arccsc_F(x) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} mul_F(signum_F(x), 0) & \\
 & \text{if } (x \in F \text{ and } |x| \geqslant 1) \text{ or } x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \\
\quad = up_F(-\pi/2) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} neg_F(arcsech_F(neg_F(x))) & \\
 & \text{if } (x \in F \text{ and } -1 < x < 0) \text{ or } x = \mathbf{-0} \\
\quad = down_F(\pi/2) \mathbin{\boldsymbol{+}\hat{\imath}\boldsymbol{\cdot}} arcsech_F(x) & \\
 & \text{if } x \in F \text{ and } 0 \leqslant x < 1 \\
\quad = no\_result_{F \rightarrow c(F)}(x) & \text{otherwise}
\end{array}$$

The $arccsc_{i(F)}$ operation:

$$arccsc_{i(F)} : i(F) \to i(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$$

$$arccsc_{i(F)}(\hat{\imath}\cdot y) \quad = \hat{\imath}\cdot arccsch_F(neg_F(y))$$

The $arccsc^*_{c(F)}$ approximation helper function:

$$arccsc^*_{c(F)} : \mathcal{C}_F \to \mathcal{C}$$

$arccsc^*_{c(F)}(z)$ returns a close approximation to $\operatorname{arccsc}(z)$ in $\mathcal{C}$ with maximum error $max\_error\_tan_{c(F)}$.

Further requirements on the $arccsc^*_{c(F)}$ approximation helper function are:

$$\begin{array}{ll} arccsc^*_{c(F)}(\operatorname{conj}(z)) = \operatorname{conj}(arccsc^*_{c(F)}(z)) & \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Im}(z) \neq 0 \text{ or } |\mathfrak{Re}(z)| \geqslant 0) \\ arccsc^*_{c(F)}(-z) = -arccsc^*_{c(F)}(z) & \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Im}(z) \neq 0 \text{ or } |\mathfrak{Re}(z)| \geqslant 0) \end{array}$$

The relationships to the $arccsc^*_F$, $arccsch^*_F$, and $arcsec^*_F$ approximation helper functions for $arccsc_F$, $arccsch_F$, and $arcsec_F$ operations in an associated library for real-valued operations shall be:

$$\begin{array}{ll} arccsc^*_{c(F)}(x) = arccsc^*_F(x) & \text{if } x \in F \text{ and } |x| \geqslant 1 \\ arccsc^*_{c(F)}(x) = (\pi/2) + (\tilde{\imath} \cdot arcsec^*_F(x)) & \text{if } x \in F \text{ and } |x| < 1 \\ arccsc^*_{c(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot arccsch^*_F(-y) & \text{if } y \in F \end{array}$$

The $arccsc^{\#}_{c(F)}$ range limitation helper function:

$$arccsc^{\#}_{c(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arccsc^*_{c(F)}(z)), down_F(\pi/2)\}\} + (\tilde{\imath} \cdot \mathfrak{Im}(arccsc^*_{c(F)}(z)))$$

The $arccsc_{c(F)}$ operation:

$$arccsc_{c(F)} : c(F) \to c(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$$

$$\begin{array}{ll} arccsc_{c(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot y) & \\ \quad = result^*_{c(F)}(arccsc^{\#}_{c(F)}(x + (\tilde{\imath} \cdot y)), nearest_F) & \\ & \text{if } x, y \in F \text{ and } (y \neq 0 \text{ or } 0 < |x| < 1) \\ \quad = arccsc_F(x) \mathbin{\hat{+}} \hat{\imath}\cdot (\mathbf{-0}) & \text{if } x \in F \text{ and } y = 0 \text{ and } |x| \geqslant 1 \\ \quad = \mathbf{infinitary}(0 \mathbin{\hat{+}} \hat{\imath}\cdot (\mathbf{-\infty})) & \\ & \text{if } x = 0 \text{ and } y = 0 \\ \quad = neg_{c(F)}(arccsc_{c(F)}(0 \mathbin{\hat{+}} \hat{\imath}\cdot neg_F(y))) & \\ & \text{if } x = \mathbf{-0} \\ \quad = conj_{c(F)}(arccsc_{c(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot 0)) & \\ & \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\ \quad = mul_F(signum_F(x), 0) \mathbin{\hat{+}} \hat{\imath}\cdot mul_F(signum_F(y), \mathbf{-0}) & \\ & \text{if } (x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\}) \text{ or} \\ & \quad y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } x \in F \\ \quad = no\_result_{c(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot y) & \text{otherwise} \end{array}$$

NOTE – The inverse of csc is multi-valued, the real part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The arccsc function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } |x| < 1\}$. Thus $arccsc_{c(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot 0) \neq arccsc_{c(F)}(x \mathbin{\hat{+}} \hat{\imath}\cdot (\mathbf{-0}))$ when $|x| < 1$ or $x = \mathbf{-0}$.

5.3.3 Operations for hyperbolic elementary functions

NOTE – The *correspondences* specified below to other ISO/IEC 10967 operations are exact, not approximate.

5.3.3.1 Hyperbolic normalisation

$$radh_F : F \to F$$
$$radh_F(x) = x$$

$$radh_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$radh_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, rad_F(y)$$

$$radh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$radh_{\mathrm{c}(F)}(x + \hat{\imath}\cdot\, y) = x + \hat{\imath}\cdot\, rad_F(y)$$

5.3.3.2 Hyperbolic sine

The $sinh_{\mathrm{i}(F)}$ operation:

$$sinh_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$sinh_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, sin_F(y)$$

The $sinh_{\mathrm{c}(F)}$ operation:

$$sinh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$sinh_{\mathrm{c}(F)}(x + \hat{\imath}\cdot\, y) = itimes_{\mathrm{c}(F)}(sin_{\mathrm{c}(F)}(y + \hat{\imath}\cdot\, neg_F(x)))$$

5.3.3.3 Hyperbolic cosine

The $cosh_{\mathrm{i}(F)}$ operation:

$$cosh_{\mathrm{i}(F)} : \mathrm{i}(F) \to F \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$cosh_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = cos_F(y)$$

The $cosh_{\mathrm{c}(F)}$ operation:

$$cosh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$cosh_{\mathrm{c}(F)}(x + \hat{\imath}\cdot\, y) = cos_{\mathrm{c}(F)}(y + \hat{\imath}\cdot\, neg_F(x))$$

5.3.3.4 Hyperbolic tangent

The $tanh_{\mathrm{i}(F)}$ operation:

$$tanh_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$tanh_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, tan_F(y)$$

The $tanh_{\mathrm{c}(F)}$ operation:

$$tanh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$
$$tanh_{\mathrm{c}(F)}(x + \hat{\imath}\cdot\, y) = itimes_{\mathrm{c}(F)}(tan_{\mathrm{c}(F)}(y + \hat{\imath}\cdot\, neg_F(x)))$$

5.3.3.5 Hyperbolic cotangent

The $coth_{\mathrm{i}(F)}$ operation:

$$coth_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$$

$$coth_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot cot_F(neg_F(y))$$

The $coth_{\mathrm{c}(F)}$ operation:

$$coth_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$$

$$coth_{\mathrm{c}(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y) = itimes_{\mathrm{c}(F)}(cot_{\mathrm{c}(F)}(neg_F(y) \mathbin{\boldsymbol{+}} \hat{\imath}\cdot x))$$

5.3.3.6 Hyperbolic secant

The $sech_{\mathrm{i}(F)}$ operation:

$$sech_{\mathrm{i}(F)} : \mathrm{i}(F) \to F \cup \{\mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$

$$sech_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = sec_F(neg_F(y))$$

The $sech_{\mathrm{c}(F)}$ operation:

$$sech_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$$

$$sech_{\mathrm{c}(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y) = sec_{\mathrm{c}(F)}(neg_F(y) \mathbin{\boldsymbol{+}} \hat{\imath}\cdot x)$$

5.3.3.7 Hyperbolic cosecant

The $csch_{\mathrm{i}(F)}$ operation:

$$csch_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$$

$$csch_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot csc_F(neg_F(y))$$

The $csch_{\mathrm{c}(F)}$ operation:

$$csch_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$$

$$csch_{\mathrm{c}(F)}(x \mathbin{\boldsymbol{+}} \hat{\imath}\cdot y) = itimes_{\mathrm{c}(F)}(csc_{\mathrm{c}(F)}(neg_F(y) \mathbin{\boldsymbol{+}} \hat{\imath}\cdot x))$$

5.3.3.8 Inverse hyperbolic sine

The $arcsinh_{\mathrm{i}(F)}$ operation:

$$arcsinh_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{invalid}\}$$

$$arcsinh_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot arcsin_F(y)$$

The $arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F)$$

$$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y) = itimes_{\mathrm{i}(F)}(arcsin_{F\to\mathrm{c}(F)}(y))$$

The $arcsinh_{\mathrm{c}(F)}$ operation:

$arcsinh_{c(F)} : c(F) \to c(F) \cup \{\mathbf{underflow}\}$

$arcsinh_{c(F)}(x \,\hat{+}\, \hat{\imath} \cdot y)$
$\quad = itimes_{c(F)}(arcsin_{c(F)}(y \,\hat{+}\, \hat{\imath} \cdot neg_F(x)))$

NOTE – The inverse of sinh is multi-valued, the imaginary part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The arcsinh function (returning the principal value for the inverse) branch cuts at $\{\tilde{\imath} \cdot y \mid y \in \mathcal{R} \text{ and } |y| > 1\}$. Thus $arcsinh_{c(F)}(0 \,\hat{+}\, \hat{\imath} \cdot y) \neq arcsinh_{c(F)}(\mathbf{-0} \,\hat{+}\, \hat{\imath} \cdot y)$ when $|y| > 1$.

5.3.3.9 Inverse hyperbolic cosine

The $arccosh_{F \to c(F)}$ operation:

$arccosh_{F \to c(F)} : F \to c(F \cup \{\mathbf{-0}\})$

$arccosh_{F \to c(F)}(x)$
$\quad = itimes_{c(F)}(arccos_{F \to c(F)}(x))$ if $(x \in F$ and $x < 0)$ or $x \in \{\mathbf{-\infty}, \mathbf{-0}\}$
$\quad = neg_{c(F)}(itimes_{c(F)}(arccos_{F \to c(F)}(x)))$ otherwise

The $arccosh_{i(F) \to c(F)}$ operation:

$arccosh_{i(F) \to c(F)} : i(F) \to c(F)$

$arccosh_{i(F) \to c(F)}(\hat{\imath} \cdot y)$
$\quad = itimes_{c(F)}(arccos_{i(F) \to c(F)}(\hat{\imath} \cdot y))$ if $(y \in F$ and $y \geqslant 0)$ or $y = \mathbf{+\infty}$
$\quad = neg_{c(F)}(itimes_{c(F)}(arccos_{i(F) \to c(F)}(\hat{\imath} \cdot y)))$ otherwise

The $arccosh_{c(F)}$ operation:

$arccosh_{c(F)} : c(F) \to c(F)$

$arccosh_{c(F)}(x \,\hat{+}\, \hat{\imath} \cdot y)$
$\quad = itimes_{c(F)}(arccos_{c(F)}(x \,\hat{+}\, \hat{\imath} \cdot y))$ if $(y \in F$ and $y \geqslant 0)$ or $y = \mathbf{+\infty}$
$\quad = neg_{c(F)}(itimes_{c(F)}(arccos_{c(F)}(x \,\hat{+}\, \hat{\imath} \cdot y)))$ otherwise

NOTE – The inverse of cosh is multi-valued, the imaginary part may have any integer multiple of $2 \cdot \pi$ added to it, and the result is also in the solution set. The arccosh function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } x < 1\}$. Thus $arccosh_{c(F)}(x \,\hat{+}\, \hat{\imath} \cdot 0) \neq arccosh_{c(F)}(x \,\hat{+}\, \hat{\imath} \cdot (\mathbf{-0}))$ when $x < 1$ or $x = \mathbf{-0}$.

5.3.3.10 Inverse hyperbolic tangent

The $arctanh_{F \to c(F)}$ operation:

$arctanh_{F \to c(F)} : F \to c(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arctanh_{F \to c(F)}(x)$
$\quad = \hat{\imath} \cdot arctan_{i(F) \to c(F)}(\hat{\imath} \cdot neg_F(x))$

The $arctanh_{i(F)}$ operation:

$arctanh_{i(F)} : i(F) \to i(F)$

$arctanh_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, arctan_F(y)$

The $arctanh_{\mathrm{c}(F)}$ operation:

$arctanh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned} &arctanh_{\mathrm{c}(F)}(x + \hat{\imath}\cdot\, y) \\ &\qquad = itimes_{\mathrm{c}(F)}(arctan_{\mathrm{c}(F)}(y + \hat{\imath}\cdot\, neg_F(x))) \end{aligned}$$

NOTE – The inverse of tanh is multi-valued, the imaginary part may have any integer multiple of $2 \cdot \pi$ (even any integer multiple of π) added to it, and the result is also in the solution set. The arctanh function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } |x| > 1\}$. Thus $arctanh_{\mathrm{c}(F)}(x+\hat{\imath}\cdot 0) \neq arctanh_{\mathrm{c}(F)}(x+\hat{\imath}\cdot(-\mathbf{0}))$ when $|x| > 1$.

5.3.3.11 Inverse hyperbolic cotangent

The $arccoth_{F\to\mathrm{c}(F)}$ operation:

$arccoth_{F\to\mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{-\mathbf{0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned} &arccoth_{F\to\mathrm{c}(F)}(x) \\ &\qquad = \hat{\imath}\cdot\, arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot\, x) \end{aligned}$$

The $arccoth_{\mathrm{i}(F)}$ operation:

$arccoth_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}\}$

$arccoth_{\mathrm{i}(F)}(\hat{\imath}\cdot\, y) = \hat{\imath}\cdot\, arccot_F(neg_F(y))$

The $arccoth_{\mathrm{c}(F)}$ operation:

$arccoth_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned} &arccoth_{\mathrm{c}(F)}(x + \hat{\imath}\cdot\, y) \\ &\qquad = itimes_{\mathrm{c}(F)}(arccot_{\mathrm{c}(F)}(neg_F(y) + \hat{\imath}\cdot\, x)) \end{aligned}$$

NOTE – The inverse of coth is multi-valued, the imaginary part may have any integer multiple of $2 \cdot \pi$ (even any integer multiple of π) added to it, and the result is also in the solution set. The arccoth function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } |x| < 1\}$. Thus $arccoth_{\mathrm{c}(F)}(x + \hat{\imath}\cdot 0) \neq arccoth_{\mathrm{c}(F)}(x + \hat{\imath}\cdot(-\mathbf{0}))$ when $|x| < 1$ or $x = -\mathbf{0}$.

5.3.3.12 Inverse hyperbolic secant

The $arcsech_{F\to\mathrm{c}(F)}$ operation:

$arcsech_{F\to\mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{-\mathbf{0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned} &arcsech_{F\to\mathrm{c}(F)}(x) \\ &\qquad = itimes_{\mathrm{c}(F)}(arcsec_{F\to\mathrm{c}(F)}(x)) \\ &\qquad\qquad\qquad \text{if } (x \in F \text{ and } x < 0) \text{ or } x \in \{-\infty, -\mathbf{0}\} \\ &\qquad = neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arcsec_{F\to\mathrm{c}(F)}(x))) \\ &\qquad\qquad\qquad \text{otherwise} \end{aligned}$$

The $arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned}
&arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y)\\
&\quad = itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y)) && \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = +\infty\\
&\quad = neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y))) && \text{otherwise}
\end{aligned}$$

The $arcsech_{\mathrm{c}(F)}$ operation:

$arcsech_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned}
&arcsech_{\mathrm{c}(F)}(x + \hat{\imath}\cdot y)\\
&\quad = itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{c}(F)}(x + \hat{\imath}\cdot y)) && \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = +\infty\\
&\quad = neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{c}(F)}(x + \hat{\imath}\cdot y))) && \text{otherwise}
\end{aligned}$$

NOTE – The inverse of sech is multi-valued, the imaginary part may have any integer multiple of $2\cdot\pi$ added to it, and the result is also in the solution set. The arcsech function (returning the principal value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } x \leqslant 0 \text{ or } x > 1\}$. Thus $arcsech_{\mathrm{c}(F)}(x + \hat{\imath}\cdot 0) \neq arcsech_{\mathrm{c}(F)}(x + \hat{\imath}\cdot(-\mathbf{0}))$ when $x \leqslant 0$ or $x = -\mathbf{0}$ or $x > 1$.

5.3.3.13 Inverse hyperbolic cosecant

The $arccsch_{\mathrm{i}(F)}$ operation:

$arccsch_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$arccsch_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot arccsc_F(neg_F(y))$

The $arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}$ operation:

$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned}
&arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot y)\\
&\quad = itimes_{\mathrm{c}(F)}(arccsc_{F\to\mathrm{c}(F)}(y))
\end{aligned}$$

The $arccsch_{\mathrm{c}(F)}$ operation:

$arccsch_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned}
&arccsch_{\mathrm{c}(F)}(x + \hat{\imath}\cdot y)\\
&\quad = itimes_{\mathrm{c}(F)}(arccsc_{\mathrm{c}(F)}(neg_F(y) + \hat{\imath}\cdot x))
\end{aligned}$$

NOTE – The inverse of csch is multi-valued, the imaginary part may have any integer multiple of $2\cdot\pi$ added to it, and the result is also in the solution set. The arccsch function (returning the principal value for the inverse) branch cuts at $\{\tilde{\imath}\cdot y \mid y \in \mathcal{R} \text{ and } |y| < 1\}$. Thus $arccsch_{\mathrm{c}(F)}(0 + \hat{\imath}\cdot y) \neq arccsch_{\mathrm{c}(F)}(-\mathbf{0} + \hat{\imath}\cdot y)$ when $|y| < 1$ or $y = -\mathbf{0}$.

5.4 Operations for conversion between imaginary and complex numeric datatypes

5.4.1 Integer to complex integer conversions

Let I and I' be the non-special value sets for two integer datatypes, at least one of which of those occurring in each operation's signature expression conforms to part 1 of ISO/IEC 10967.

$convert_{I\to\mathrm{c}(I)} : I \to \mathrm{c}(I)$

$$convert_{I\to c(I)}(x) = x \,\hat{\imath}\!\cdot\! 0$$

$$convert_{i(I)\to c(I)} : i(I) \to c(I)$$

$$convert_{i(I)\to c(I)}(\hat{\imath}\cdot y) = 0 + \hat{\imath}\cdot y$$

$$convert_{i(I)\to i(I')} : i(I) \to i(I') \cup \{\mathbf{overflow}\}$$

$$convert_{i(I)\to i(I')}(\hat{\imath}\cdot y) = \hat{\imath}\cdot convert_{I\to I'}(y)$$

$$convert_{c(I)\to c(I')} : c(I) \to c(I') \cup \{\mathbf{overflow}\}$$

$$convert_{c(I)\to c(I')}(x + \hat{\imath}\cdot y) = convert_{I\to I'}(x) + \hat{\imath}\cdot convert_{I\to I'}(y)$$

5.4.2 Floating point to complex floating point conversions

Let F and F' be the non-special value sets for two floating point datatypes, at least one of which of those occurring in each operation's signature expression conforms to part 1. Let D be the non-special value set for a fixed point datatype (see Clause 5.4.5 in part 2).

The $convert_{F\to c(F)}$ operation:

$$convert_{F\to c(F)} : F \to c(F \cup \{\mathbf{-0}\})$$

$$convert_{F\to c(F)}(x) = \begin{cases} x + \hat{\imath}\cdot im_F(x) & \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \\ no\_result_{F\to c(F)}(x) & \text{otherwise} \end{cases}$$

The $convert_{i(F)\to c(F)}$ operation:

$$convert_{i(F)\to c(F)} : i(F) \to c(F \cup \{\mathbf{-0}\})$$

$$convert_{i(F)\to c(F)}(\hat{\imath}\cdot y) = \begin{cases} re_{i(F)}(\hat{\imath}\cdot y) + \hat{\imath}\cdot y & \text{if } y \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \\ no\_result_{i(F)\to c(F)}(\hat{\imath}\cdot y) & \text{otherwise} \end{cases}$$

The $convert_{i(F)\to i(F')}$ operation:

$$convert_{i(F)\to i(F')} : i(F) \to i(F') \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$

$$convert_{i(F)\to i(F')}(\hat{\imath}\cdot y) = \hat{\imath}\cdot convert_{F\to F'}(y)$$

The $convert_{c(F)\to c(F')}$ operation:

$$convert_{c(F)\to c(F')} : c(F) \to c(F') \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$

$$convert_{c(F)\to c(F')}(x + \hat{\imath}\cdot y) = convert_{F\to F'}(x) + \hat{\imath}\cdot convert_{F\to F'}(y)$$

The $convert_{\mathrm{i}(F)\to\mathrm{i}(D)}$ operation:

$$convert_{\mathrm{i}(F)\to\mathrm{i}(D)} : \mathrm{i}(F) \to \mathrm{i}(D) \cup \{\mathbf{overflow}\}$$

$$convert_{\mathrm{i}(F)\to\mathrm{i}(D)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot convert_{F\to D}(y)$$

The $convert_{\mathrm{c}(F)\to\mathrm{c}(D)}$ operation:

$$convert_{\mathrm{c}(F)\to\mathrm{c}(D)} : \mathrm{c}(F) \to \mathrm{c}(D) \cup \{\mathbf{overflow}\}$$

$$convert_{\mathrm{c}(F)\to\mathrm{c}(D)}(x \mathbin{\boldsymbol{+}\hat{\imath}\cdot} y) = convert_{F\to D}(x) \mathbin{\boldsymbol{+}\hat{\imath}\cdot} convert_{F\to D}(y)$$

The $convert_{\mathrm{i}(D)\to\mathrm{i}(F)}$ operation:

$$convert_{\mathrm{i}(D)\to\mathrm{i}(F)} : \mathrm{i}(D) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$

$$convert_{\mathrm{i}(D)\to\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot convert_{D\to F}(y)$$

The $convert_{\mathrm{c}(D)\to\mathrm{c}(F)}$ operation:

$$convert_{\mathrm{c}(D)\to\mathrm{c}(F)} : \mathrm{c}(D) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$$

$$convert_{\mathrm{c}(D)\to\mathrm{c}(F)}(x \mathbin{\boldsymbol{+}\hat{\imath}\cdot} y) = convert_{D\to F}(x) \mathbin{\boldsymbol{+}\hat{\imath}\cdot} convert_{D\to F}(y)$$

5.5 Support for imaginary and complex numerals

Rather than specifying special numerals for imaginary and complex values, imaginary and complex units are specified in this document. These imaginary and complex units can be used as a basis for expressions that transform ordinary numerals to imaginary or complex numerals. Many programming languages also specify special numeral forms for imaginary or complex numerals, either for the programming language syntax or for input/output formats. These can be expressed using the units specified here.

The imaginary and complex unit operations are as follows:

$$imaginary\_unit_{\mathrm{i}(I)} = \hat{\imath}\cdot 1$$

$$imaginary\_unit_{\mathrm{c}(I)} = 0 \mathbin{\boldsymbol{+}\hat{\imath}\cdot} 1$$

$$imaginary\_unit_{\mathrm{i}(F)} = \hat{\imath}\cdot 1$$

$$imaginary\_unit_{\mathrm{c}(F)} = 0 \mathbin{\boldsymbol{+}\hat{\imath}\cdot} 1$$

6 Notification

Notification is the process by which a user or program is informed that an arithmetic operation cannot return a suitable numeric result. Specifically, a notification shall occur when any arithmetic operation returns an exceptional value. Notification shall be performed according to the requirements of Clause 6 of part 1 of ISO/IEC 10967.

An implementation shall not give notifications for operations conforming to this document, unless the specification requires that an exceptional value results for the given arguments.

The default method of notification should be recording of indicators.

6.1 Continuation values

If notifications are handled by a recording of indicators, in the event of notification the implementation shall provide a *continuation value* to be used in subsequent arithmetic operations. Continuation values may be in I, $\mathrm{i}(I)$, $\mathrm{c}(I)$, F, $\mathrm{i}(F)$ or $\mathrm{c}(F)$ (as appropriate), or be special values (where the real or imaginary component is $\mathbf{-0}$, $\mathbf{-\infty}$, $\mathbf{+\infty}$, or a **qNaN**).

Floating point datatypes that satisfy the requirements of IEC 60559 have special values in addition to the values in F. These are: $\mathbf{-0}$, $\mathbf{+\infty}$, $\mathbf{-\infty}$, *signaling* **NaN**s (**sNaN**), and *quiet* **NaN**s (**qNaN**). Such values may be components of complex floating point datatypes, and may be included in values passed as arguments to operations, and used in results or continuation values. Floating point types that do not fully conform to IEC 60559 can also have values corresponding to $\mathbf{-0}$, $\mathbf{+\infty}$, $\mathbf{-\infty}$, or **NaN**.

7 Relationship with language standards

A computing system often provides some of the operations specified in this document within the context of a programming language. The requirements of this document shall be in addition to those imposed by the relevant programming language standards.

This document does not define the syntax of arithmetic expressions. However, programmers need to know how to reliably access the operations specified in this document.

NOTE 1 – Providing the information required in this clause is properly the responsibility of programming language standards (or a binding standard between this document and a programming language). An individual implementation would only need to provide details if it could not cite an appropriate clause of the language or binding standard.

An implementation shall document the notation that should be used to invoke an operation or to access a parameter specified in this document and made available. An implementation should document the notation that should be used to invoke an operation or to access a parameter specified in this document and that could be made available.

NOTES

2 For example, the complex radian arc sine operation for an argument x ($arcsin_{c(F)}(x)$) might be invoked as

`arcsin(`x`)`	in Ada [2]
`casin(`x`)`	in C [4]
`asin(`x`)`	in Fortran [6] and C++ [5]
`(asin `x`)`	in Common Lisp [10]

with a suitable expression of the argument (x).

3 The requirement that the notation that should be used for accessing operations or parameters that are not made available should also be documented is a recommendation to reserve that notation for future use (or present use by some implementations). Thus, doing so would be a way of saying that that notation should not be used for something else, nor, e.g., for a less accurate but otherwise similar operation.

An implementation shall document the semantics of arithmetic expressions in terms of compositions of the operations specified in Clause 5 of this document and in Clause 5 of part 1 and Clause 5 of part 2 of ISO/IEC 10967.

Compilers often "optimize" code as part of compilation. Thus, an arithmetic expression might not be executed as written. An implementation shall document the possible transformations of arithmetic expressions (or groups of expressions) that it permits. Typical transformations include:

a) Insertion of operations, such as datatype conversions or changes in precision.

b) Replacing operations (or entire subexpressions) with others, such as "`cos(-x)`" → "`cos(x)`" (exactly the same result) or "`pi - arccos(x)`" → "`arccos(-x)`" (more accurate result).

c) Evaluating constant subexpressions.

d) Eliminating unneeded subexpressions.

Only transformations which alter the semantics of an expression (the values produced, and the notifications generated) need be documented. Only the kinds of permitted transformations need be documented. It is not necessary to describe the specific choice of transformations that will be applied to a particular expression.

The textual scope of such transformations shall be documented, and any mechanisms that provide programmer control over this process should be documented as well.

NOTE 4 – It is highly desirable that programming languages intended for numerical use provide means for limiting the transformations applied to particular arithmetic expressions. Control over changes of precision is particularly useful.

8 Documentation requirements

In order to conform to this part, an implementation shall include documentation providing the following information to programmers.

NOTE 1 – Much of the documentation required in this clause is properly the responsibility of programming language or binding standards. An individual implementation would only need to provide details if it could not cite an appropriate clause of the language or binding standard.

a) A list of the provided operations that conform to this document.

b) For each box error mode parameter, the value of that parameter. Only box error mode parameters that are relevant to the provided operations need be given.

c) For each maximum error parameter, the value of that parameter or definition of that parameter function. Only maximum error parameters that are relevant to the provided operations need be given.

d) The value of the parameters $big\_angle\_r_F$ and $big\_angle\_u_F$. Only big angle parameters that are relevant to the provided operations need be given.

e) For the $nearest_F$ function, the rule used for rounding halfway cases, unless iec_559_F is fixed to **true**.

f) For each conforming operation, the continuation value provided for each notification condition. Specific continuation values that are required by this document need not be documented. If the notification mechanism does not make use of continuation values (see Clause 6), continuation values need not be documented.

NOTE 2 – Implementations that do not provide infinities or NaNs will have to document any continuation values used in place of such values.

g) For each conforming operation, how the results depend on the rounding mode, if rounding modes are provided. Operations may be insensitive to the rounding mode, or sensitive to it, but even then need not heed the rounding mode.

h) For each conforming operation, the notation to be used for invoking that operation.

i) For each box error mode parameter, the notation to be used to access that parameter.

j) For each maximum error parameter, the notation to be used to access that parameter.

k) The notation to be used to access the parameters $big\_angle\_r_F$ and $big\_angle\_u_F$.

l) For each of the provided operations where this document specifies a relation to another operation specified in this International Standard, the binding for that other operation.

m) For numerals conforming to this International Standard, which available string conversion operations, including reading from input, give exactly the same conversion results, even if the string syntaxes for 'internal' and 'external' numerals are different.

Since the integer and floating point datatypes used in conforming operations shall satisfy the requirements of part 1 of ISO/IEC 10967, the following information shall also be provided by any conforming implementation.

n) The means for selecting the modes of operation that ensure conformity.

o) The translation of arithmetic expressions into combinations of the operations provided by any part of ISO/IEC 10967, including any use made of higher precision. (See Clause 7 of part 1.)

p) The methods used for notification, and the information made available about the notification. (See Clause 6 of part 1.)

q) The means for selecting among the notification methods, and the notification method used in the absence of a user selection. (See Clause 6.3 of part 1.)

r) When "recording of indicators" is the method of notification, the datatype used to represent Ind (see Clause 6.1.2 of part 1), the method for denoting the values of Ind, and the notation for invoking each of the "indicator" operations. E is the set of notification indicators. The association of values in Ind with subsets of E must be clear. In interpreting Clause 6.1.2 of part 1, the set of indicators E shall be interpreted as including all exceptional values listed in the signatures of conforming operations. In particular, E may need to contain **infinitary** and **absolute_precision_underflow**.

Annex A
(normative)

Partial conformity

If an implementation of an operation fulfills all relevant requirements according to the main normative text in this part, except the ones relaxed in this annex, the implementation of that operation is said to *partially conform* to this part.

Conformity to this part shall not be claimed for operations that only fulfill partial conformity.

Partial conformity shall not be claimed for operations that relax other requirements than those relaxed in this annex.

A.1 Maximum error relaxation

This part has the following maximum error requirements for conformity.

$$max\_error\_mul_{c(F)} \in [0.5, 5]$$
$$max\_error\_div_{c(F)} \in [0.5, 13]$$

$$max\_error\_exp_{c(F)} \in [0.5, 7]$$
$$max\_error\_power_{c(F)} \in [0.5, 15]$$

$$max\_error\_sin_{c(F)} \in [0.5, 11]$$
$$max\_error\_tan_{c(F)} \in [0.5, 14]$$

In a partially conforming implementation the maximum error parameters may be greater than what is specified by this part. The maximum error parameter values given by an implementation shall still adequately reflect the accuracy of the relevant operations, if a claim of partial conformity is made.

A partially conforming implementation shall document which maximum error parameters have greater values than specified by this part, and their values.

A.2 Extra accuracy requirements relaxation

This part has a number of extra accuracy requirements. These are detailed in the paragraphs beginning "Further requirements on the op^*_F approximation helper function are:".

In a partially conforming implementation these further requirements need not be fulfilled. The values returned must still be within the maximum error bounds that are given by the maximum error parameters, if a claim of partial conformity is made.

A partially conforming implementation shall document which extra accuracy requirements are not fulfilled by the implementation.

A.3 Relationships to other operations relaxation

This part has a number of requirements giving relations to other operations. These are detailed in the paragraphs beginning with wordings like "Relationship to the op^{*}_{F} approximation helper function for operations in an associated library shall be:".

In a partially conforming implementation these relationships need not be fulfilled. The values returned must still be within the maximum error bounds that are given by the values provided for the maximum error parameters, if a claim of partial conformity is made.

A partially conforming implementation shall document which operation relationships are not fulfilled by the implementation.

A.4 Part 1 and part 2 requirements relaxation

Part 3 depends on the datatypes and operations specified in part 1, as well as the operations (and approximation helper functions) specified in part 2. Part 1 and part 2 allow for partial conformity. Part 3 operations may thus be only partially conforming if a relevant datatype or operation is only partially conforming to part 1 or part 2.

Annex B
(informative)

Rationale

In the informative annexes of this document, LIA-1 refers to part 1 of ISO/IEC 10967, LIA-2 refers to part 2, and LIA-3 refers to this document.

This annex explains and clarifies some of the ideas behind *Information technology – Language independent arithmetic – Part 3: Complex floating point arithmetic and complex elementary numerical functions*. The clause numbering matches that of the standard.

B.1 Scope

B.1.1 Inclusions

Integer complex datatypes are included in Common Lisp [10]. Integer complex numbers are sometimes referred to as Gaussian numbers.

Imaginary datatypes are included informatively in the current version of the C programming language standard (annex G of [4]) and normatively in the current version of the Ada language standard (annex G of [2]).

B.1.2 Exclusions

LIA-3 only covers Cartesian complex datatypes. Polar complex datatypes are not included since no widely known programming language provides them. However, operations for conversion from polar representation to Cartesian representation are included both in some programming languages and in LIA-3.

Neither rational nor fixed point complex datatypes are covered by LIA-3 since (as yet) no part of LIA covers rational or fixed point datatypes. (Note that Common Lisp includes complex datatypes that have rational datatype values in the parts.)

B.2 Conformity

Conformity to this standard is dependent on the existence of language binding standards. Each programming language committee (or other organization responsible for a programming language or other specification to which LIA-1, LIA-2, and LIA-3 may apply) is encouraged to produce a binding covering at least those operations already required by the programming language (or similar) and also specified in LIA-3.

The term “programming language” is here used in a generalised sense to include other computing entities such as calculators, spread sheets, page description languages, web-script languages, and database query languages to the extent that they provide the operations covered by LIA-3.

Suggestions for bindings are provided in Annex C. Annex C has partial binding examples for a number of existing programming languages and LIA-3. In addition to the bindings for the operations in LIA-3, it is also necessary to provide bindings for the maximum error parameters specified by LIA-3. Annex C contains suggestions for these bindings. To conform to this standard, in the absence of a binding standard, an implementation should create a binding, following the suggestions in annex C.

B.3 Normative references

The referenced IEC 60559 standard is identical to the IEEE 754 standard and the former IEC 559 standard.

The origin of IEC 60559:1989, IEEE 754:1984, was undergoing revision during the development of this part.

See Annex E of ISO/IEC 10967-2:2001 regarding the first edition (published in 1994) of ISO/IEC 10967-1.

See Annex F of this part regarding the first edition (published in 2001) of ISO/IEC 10967-2.

B.4 Symbols and definitions

B.4.1 Symbols

B.4.1.1 Sets and intervals

The interval notation is in common use. It has been chosen over the other commonly used interval notation because the chosen notation has no risk of confusion with the pair notation.

B.4.1.2 Operators and relations

Note that all operators, relations, and other mathematical notation used in LIA-3 are used in their conventional exact mathematical sense. They are not used to stand for operations specified by IEC 60559, LIA-1, LIA-2, LIA-3, or, with the exception of program excerpts which are clearly marked, any programming language. E.g. x/u stands for the mathematically exact result of dividing x by u, whether that value is representable in any floating point datatype or not, and $x/u \neq div_F(x, u)$ is often the case. Likewise, $=$ is the mathematical equality, not the eq_F operation: $0 \neq \mathbf{-0}$, while $eq_F(0, \mathbf{-0}) = \mathbf{true}$.

For mathematical complex values, conventional notation with $+$ and $\cdot$ is used. For the imaginary unit, the symbol $\tilde{\imath}$ is used.

It is important to distinguish this mathematical notation for values in $\mathcal{C}$ and the notation used to express values in $\mathrm{c}(X)$ or $\mathrm{i}(X)$. For $\mathrm{c}(X)$ the binary operator $\mathbf{+\hat{\imath}\cdot}$ is used. Its only function is to create a pair of values corresponding to a value in $\mathcal{C}$. Similarly, for $\mathrm{i}(X)$, the unary operator $\mathbf{\hat{\imath}\cdot}$ is used. It creates a 1-tuple corresponding to a value in $\mathcal{C}$ where the real part is 0.

B.4.1.3 Mathematical functions

The elementary functions named sin, cos, etc., used in LIA-3 are the exact mathematical functions, not any approximation. The approximations to these mathematical functions are introduced in clauses 5.3 and 5.4 and are written in a way clearly distinct from the exact mathematical functions. E.g., $sin^*_{\mathrm{c}(F)}$, $cos^*_{\mathrm{c}(F)}$, etc., which are unspecified mathematical functions approximating the targeted exact mathematical functions to a specified degree; $sin_{\mathrm{c}(F)}$, $cos_{\mathrm{c}(F)}$, etc., which are the operations specified by LIA-3 based on the respective approximating function; `sin`, `cos`, etc., which are programming language names bound to LIA-3 operations. `sin` is thus very different from sin.

B.4.1.4 Exceptional values

LIA-3 uses the same set of exceptional values as LIA-2.

B.4.1.5 Datatypes and special values

LIA allows for three methods for handing notifications: recording of indicators, change of control flow (returnable or not), and termination of program. The preferred method is recording of indicators. This allows the computation to continue using the continuation values. For **underflow** and **infinitary** notifications this course of action is strongly preferred, provided that a suitable continuation value can be represented in the result datatype.

Not all occurrences of the same exceptional value need be handled the same. There may be explicit mode changes in how notifications are handled, and there may be implicit changes. E.g., **invalid** without a specified (by LIA-3 or binding) continuation value to cause change of control flow (like an Ada [2] exception), while **invalid** with a specified continuation value to use recording of indicators. This should be specified by bindings or by implementations.

The operations may return any of the exceptional values **overflow**, **underflow**, **invalid**, **infinitary**, or **absolute_precision_underflow**. This does *not* imply that the implemented operations are to actually *return* any of these values. When these values are returned according to the LIA specification, that means that the implementation is to perform a notification handling for that exceptional value. If the notification handling is by recording of indicators, then what is actually returned by the implemented operation is the continuation value.

B.4.1.6 Complex value constructors and complex datatype constructors

For integer and floating point datatypes, the set of non-special values are subsets of $\mathcal{Z}$ and $\mathcal{R}$ respectively (LIA-1). However, if we do the parallel here, defining the set of non-special values as subsets of $\mathcal{G}$ and $\mathcal{C}$, the handling of special values becomes awkward. The latter would need a pair representation for the full complex case. Instead, we use a pair representation for all complex values. In addition, we then also get an easy notational difference between imaginary and full complex datatypes. Unfortunately, we also get a distinction between this pair formation, and the arithmetic expression resulting in a complex value in $\mathcal{G}$ or $\mathcal{C}$. This distinction has to be carefully maintained in the specifications in the rest of the document.

B.4.2 Definitions of terms

Note the LIA distinction between exceptional values, exceptions, and exception handling (handling of notification by non-returnable change of control flow; as in e.g. Ada). LIA exceptional values are not the same as Ada exceptions, nor are they the same as IEC 60559 special values.

B.5 Specifications for the imaginary and complex datatypes and operations

B.5.1 Imaginary and complex integer datatypes and operations

Of the programming languages covered in annex C, only Common Lisp has support for complex integers, but not imaginary integers as separate from complex integers.

Since LIA-3 distinguishes between complex and imaginary floating point, that same distinction is also made for the integer case.

“Real” integers and imaginary integers can be compared for order, while complex integers only can be compared for equality or inequality.

The re_I operation is specified for completeness. It is the identity function on integers, except possibly for NaN values (which are rare in integer datatypes) Likewise is the im_I operation specified for completeness. It always returns zero. Note, however, that the imaginary part of a “real” floating point value is not the same for all arguments, even though it is always a zero. Similarly, im_I could return signed zero as im_F does, if the integer datatype supports signed zero. This is, however, rare, so is not specified explicitly by LIA-3. Also the $conj_I$ and $conj_{\mathrm{i}(I)}$ operations are specified for completeness.

The binary operations, except comparisons for order, allow for combining “real” integer, imaginary integer and complex integer of the same base integer datatype. Bindings may generalise this to also allow different base integer datatypes, converting the smaller ones to the largest one occurring as arguments to the operation.

$signum_I$ is specified in this part of LIA, since the $sign_I$ operation specified in the first edition of LIA-1 is not consistent with the $signum_F$ operation, also specified in this part. The $sign_I$ and $sign_F$ of the first edition of LIA-1 return zero for a zero argument. This is not consistent with the branch cuts specified for elementary operations in this part. For $signum_I$ and $signum_F$ the branch cut view is used, and they only return −1 or 1, never zero (of any sign).

There is no $signum_{\mathrm{c}(I)}$ operation. The mathematical function would normalise the value to the unit circle. But since only four values in $\mathrm{c}(I)$ is on, or even close, to the unit circle, the approximation for returning a $\mathrm{c}(I)$ value is too great. If one wants to compute an approximation to the signum of a $\mathrm{c}(I)$ value, first convert the value to $\mathrm{c}(F)$, and then compute $signum_{\mathrm{c}(F)}$ of that value.

$divides_{\mathrm{c}(I)}$ is the operation version of the divides relation specified in clause 4.1.

Max and min operations are defined only for imaginary integer values, since complex integers are not ordered in mathematics. A lexicographic order is easy to define, but that is another kind of ordering that is out of scope for LIA.

B.5.2 Imaginary and complex floating point datatypes and operations

B.5.2.1 Maximum error requirements

B.5.2.2 Sign requirements

B.5.2.3 Maximum error requirements

The values for the error parameters given later in the normative part of this document are largely taken from the Ada standard [2].

B.5.2.4 Basic arithmetic for complex floating point

F must be a subset of $\mathcal{R}$. Floating point datatypes can have infinity values as well as NaN values, and also may have a $\mathbf{-0}$. These values are not in F. The special values are, however, commonly available in floating point datatypes today, thanks to the wide adoption of IEC 60599.

Note that for some operations the exceptional value **invalid** is produced only for argument values involving $\mathbf{-0}$, $\mathbf{+\infty}$, $\mathbf{-\infty}$, or **sNaN**. For these operations the signature given in LIA-3 does not contain **invalid**.

The report *Floating-Point C Extensions* [16], issued by the ANSI X3J11 committee, discusses possible ways of exploiting the IEC 60559 special values. The report identifies some of its suggestions as controversial and cites *Branch Cuts for Complex Elementary Functions, or Much Ado about Nothing's Sign Bit* [14] as justification. The report *Complex Floating-Point C Extensions* [17], also issued by the ANSI X3J11 committee, extends these recommendations to imaginary and complex datatypes. These reports have strongly influenced the C99 [4] programming language.

The implicit zero imaginary part of a value in F and the implicit zero real part of a value in $\mathrm{i}(F)$ is a very strong zero, here written **00**. Note that **00** is *not* represented in the datatype corresponding to F, nor in $\mathrm{i}(F)$ or $\mathrm{c}(F)$, not even as a component. **00** is strong in the sense that **00** overtrumps NaN and infinity values, which neither 0 nor $\mathbf{-0}$ does. I.e. the implicit zeroes (**00**) work *as if* the rules of the following table applies (plus commutativity):

$mul_F(0, \mathbf{+\infty}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{-0}, \mathbf{+\infty}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{00}, \mathbf{+\infty}) = \mathbf{00}$
$mul_F(0, \mathbf{-\infty}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{-0}, \mathbf{-\infty}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{00}, \mathbf{-\infty}) = \mathbf{00}$
$mul_F(0, \mathbf{qNaN}) = \mathbf{qNaN}$	$mul_F(\mathbf{-0}, \mathbf{qNaN}) = \mathbf{qNaN}$	$mul_F(\mathbf{00}, \mathbf{qNaN}) = \mathbf{00}$
$mul_F(0, \mathbf{sNaN}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{-0}, \mathbf{sNaN}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{00}, \mathbf{sNaN}) = \mathbf{00}$

and further as if $add_F(\mathbf{00}, x) = x$, even for signalling NaNs. The basic idea here is that **00** is not only mostly treated as exact even though it *may* be the result of an approximation (as other values are handled), but that **00** really is exact, never the result of an approximation. Instead of representing **00** in any floating point datatype, which would necessitate new floating point datatypes, several programming languages have chosen to use triplets of datatypes (corresponding to F, $\mathrm{i}(F)$ and $\mathrm{c}(F)$, but adding special values). LIA-3 adheres to this convention.

When the implicit **00** is made explicit, one has to choose between 0 and $\mathbf{-0}$ (if the latter is representable). In doing so, sign symmetry is observed by the specifications in LIA-3. This is done in such a way that it is coherent with the sign symmetry of some trigonometric (and hyperbolic) operations that accept values in F (or $\mathrm{i}(F)$) but return a value in $\mathrm{c}(F)$. This is in order to maintain also the sign symmetry, even when $\mathbf{-0}$ is not representable. This also adheres to the sign symmetry conventions of Common Lisp and C99. Further, when negative zero ($\mathbf{-0}$) is not representable, a 0 as the real part or as the imaginary part of the argument is to be considered to be $\mathbf{-0}$ consistently with the above.

Another approach would be not to regard implicit zeroes as **00**, but as ordinary zero. This renders the $\mathrm{i}(F)$ datatypes superfluous, and they are usually omitted in this approach. This approach is both less efficient and looses information needlessly. In this approach one also uses just one complex NaN, only one (unsigned) complex infinity, and only one (unsigned) complex 0. Again this looses information needlessly, in particular if the infinities or zeroes are the result of approximations. Again that speaks against this approach. In this approach one sometimes also do not distinguish the signs of zero, forcing programmers to write expressions like `max(posvalue, FMIN)` and `min(negvalue, -FMIN)`, to avoid zero but (with error prone programming) nearly mimic the signed zeroes. In this approach one does also not always follow conjugate symmetry and sign symmetry rules when part of the argument is zero, which for this approach leads to surprising changes of results when doing algebraic manipulations based on those rules. For the reasons given here, this approach is not taken for LIA-3.

B.5.3 Elementary transcendental imaginary and complex floating point operations

B.5.3.1 Operations for exponentiations and logarithms

The LIA-3 exponentiation and inverse exponentiation operations that take imaginary or complex arguments interpret the imaginary part (of the argument or result respectively) as in radians. Operations that take an extra floating point argument, giving the non-radian cyclic unit, can be made, paralleling such operations (for trigonometric operations) in LIA-2. Such operations are not included in LIA-3, because no programming language requires them for complex arguments.

However, the complex result power operation uses $|z/2| \leqslant big\_angle\_u_F$, as the "angle" cutoff, since that is more logical in connection with the power operation, as well as being easier to implement (no multiplication by π).

B.5.3.2 Operations for radian trigonometric elementary functions

The LIA-3 trigonometric and inverse trigonometric operations that take imaginary or complex arguments interpret the real part (of the argument or result respectively) as in radians. Operations that take an extra floating point argument, giving the non-radian cyclic unit, can be made, paralleling such operations in LIA-2. Such operations are not included in LIA-3, because no programming language require them (with the exception of *phaseu* and *polaru* which are required by Ada).

The notation here is taken from *Handbook of mathematical functions with graphs, formulas, and mathematical tables*. The equalities that involve an uppercased elementary function name refer the multi-valued inverses. The equalities then say the values on the left side (seen as a set) is equal to the values on the right side (seen as a set). This could be expressed more conventionally, using set notation, but this notation is borrowed since it is convenient. The lowercase names for inverses are the corresponding functions, denoting the principal value for the multi-valued inverse.

B.5.3.2.1 Radian angle normalisation

The radian angle normalisation operations normalise the angle part (the real part) of the argument to be within $[down_F(-\pi), up_F(\pi)]$ (or, if the correction given in annex F is used: $[up_F(-\pi), down_F(\pi)]$). While useful for programs where one wishes to keep angle information at a high accuracy, it is not, and cannot be, sufficiently accurate for implementing the trigonometric operations specified in LIA-3. A normalisation function that is almost sufficient for that is specified in LIA-2: $axis\_rad_F$. Some extra accuracy bits for the normalisation are still needed for implementing the trigonometric operations with respect to sign and accuracy requirements. $axis\_rad_F$ is not generalised to the complex case, since there is little or no convenience to be gained compared to using $axis\_rad_F$ directly on the relevant part of a complex value.

B.5.3.2.2 Radian sine

The following equalities have been used to derive the LIA-3 requirements:

$\sin(-z) = -\sin(z)$
$\sin(\mathrm{conj}(z)) = \mathrm{conj}(\sin(z))$
$\sin(z + k \cdot 2 \cdot \pi) = \sin(z)$ if $k \in \mathcal{Z}$
$\sin(z) = \cos(\pi/2 - z)$

$\sin(x) = -\tilde{\imath} \cdot \sinh(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sinh(-\tilde{\imath} \cdot x)$
$\sin(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sinh(-y) = \tilde{\imath} \cdot \sinh(y)$
$\sin(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sinh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sinh(y - \tilde{\imath} \cdot x)$

$\sin(x + \tilde{\imath} \cdot y) = \sin(x) \cdot \cosh(y) + \tilde{\imath} \cdot \cos(x) \cdot \sinh(y)$

B.5.3.2.3 Radian cosine

The following equalities have been used to derive the LIA-3 requirements:

$\cos(-z) = \cos(z)$
$\cos(\mathrm{conj}(z)) = \mathrm{conj}(\cos(z))$
$\cos(z + k \cdot 2 \cdot \pi) = \cos(z)$ if $k \in \mathcal{Z}$
$\cos(z) = \sin(\pi/2 - z)$

$\cos(x) = \cosh(\tilde{\imath} \cdot x) = \cosh(-\tilde{\imath} \cdot x)$
$\cos(\tilde{\imath} \cdot y) = \cosh(y)$
$\cos(x + \tilde{\imath} \cdot y) = \cosh(-y + \tilde{\imath} \cdot x) = \cosh(y - \tilde{\imath} \cdot x)$

$\cos(x + \tilde{\imath} \cdot y) = \cos(x) \cdot \cosh(y) - \tilde{\imath} \cdot \sin(x) \cdot \sinh(y)$

B.5.3.2.4 Radian tangent

The following equalities have been used to derive the LIA-3 requirements:

$\tan(-z) = -\tan(z)$
$\tan(\mathrm{conj}(z)) = \mathrm{conj}(\tan(z))$
$\tan(z + k \cdot 2 \cdot \pi) = \tan(z)$ if $k \in \mathcal{Z}$
$\tan(z) = \cot(\pi/2 - z)$

$\tan(x) = -\tilde{\imath} \cdot \tanh(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \tanh(-\tilde{\imath} \cdot x)$
$\tan(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \tanh(-y) = \tilde{\imath} \cdot \tanh(y)$
$\tan(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \tanh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \tanh(y - \tilde{\imath} \cdot x)$

Note that every non-zero integer multiple of π is a cycle for tangent. Since $2 \cdot \pi$ is the smallest positive cycle that is common to all of the trigonometric functions, that is the cycle concentrated on for all of them, even though some have π as cycle too.

B.5.3.2.5 Radian cotangent

The following equalities have been used to derive the LIA-3 requirements:

$\cot(-z) = -\cot(z)$
$\cot(\mathrm{conj}(z)) = \mathrm{conj}(\cot(z))$
$\cot(z + k \cdot 2 \cdot \pi) = \cot(z)$ if $k \in \mathcal{Z}$
$\cot(z) = \tan(\pi/2 - z)$
$\cot(z) = 1/\tan(z)$

$\cot(x) = \tilde{\imath} \cdot \coth(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \coth(-\tilde{\imath} \cdot x)$
$\cot(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \coth(-y) = -\tilde{\imath} \cdot \coth(y)$
$\cot(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \coth(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \coth(y - \tilde{\imath} \cdot x)$

B.5.3.2.6 Radian secant

The following equalities have been used to derive the LIA-3 requirements:

$\sec(-z) = \sec(z)$
$\sec(\mathrm{conj}(z)) = \mathrm{conj}(\sec(z))$
$\sec(z + k \cdot 2 \cdot \pi) = \sec(z)$ if $k \in \mathcal{Z}$
$\sec(z) = \csc(\pi/2 - z)$
$\sec(z) = 1/\cos(z)$

$\sec(x) = \mathrm{sech}(\tilde{\imath} \cdot x) = \mathrm{sech}(-\tilde{\imath} \cdot x)$
$\sec(\tilde{\imath} \cdot y) = \mathrm{sech}(-y) = \mathrm{sech}(y)$
$\sec(x + \tilde{\imath} \cdot y) = \mathrm{sech}(-y + \tilde{\imath} \cdot x) = \mathrm{sech}(y - \tilde{\imath} \cdot x)$

B.5.3.2.7 Radian cosecant

The following equalities have been used to derive the LIA-3 requirements:

$\csc(-z) = -\csc(z)$
$\csc(\mathrm{conj}(z)) = \mathrm{conj}(\csc(z))$
$\csc(z + k \cdot 2 \cdot \pi) = \csc(z)$ if $k \in \mathcal{Z}$
$\csc(z) = \sec(\pi/2 - z)$
$\csc(z) = 1/\sin(z)$

$\csc(x) = \tilde{\imath} \cdot \mathrm{csch}(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{csch}(-\tilde{\imath} \cdot x)$
$\csc(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{csch}(-y) = -\tilde{\imath} \cdot \mathrm{csch}(y)$
$\csc(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{csch}(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{csch}(y - \tilde{\imath} \cdot x)$

B.5.3.2.8 Radian arc sine

The following equalities have been used to derive the LIA-3 requirements:

$\arcsin(-z) = -\arcsin(z)$
$\arcsin(z) = \pi/2 - \arccos(z)$
$\arcsin(\mathrm{conj}(z)) = \mathrm{conj}(\arcsin(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \leqslant 1$
$Arcsin(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arcsinh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arcsinh(y - \tilde{\imath} \cdot x)$

B.5.3.2.9 Radian arc cosine

The following equalities have been used to derive the LIA-3 requirements:

$\arccos(-z) = \tilde{\imath} \cdot \pi - \arccos(z)$
$\arccos(\mathrm{conj}(z)) = \mathrm{conj}(\arccos(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \leqslant 1$
$\arccos(z) = \pi/2 - \arcsin(z)$
$Arccos(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arccosh(x + \tilde{\imath} \cdot y)$

B.5.3.2.10 Radian arc tangent

The following equalities have been used to derive the LIA-3 requirements:

$\arctan(-z) = -\arctan(z)$
$\arctan(\mathrm{conj}(z)) = \mathrm{conj}(\arctan(z))$ if $\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| \leqslant 1$
$\arctan(z) = \pm\pi/2 - \mathrm{arccot}(z)$
$Arctan(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arctanh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arctanh(y - \tilde{\imath} \cdot x)$

B.5.3.2.11 Radian arc cotangent

The following equalities have been used to derive the LIA-3 requirements:

$\mathrm{arccot}(-z) = -\mathrm{arccot}(z)$
$\mathrm{arccot}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arccot}(z))$ if $\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| \geqslant 1$
$\mathrm{arccot}(z) = \pm\pi/2 - \arctan(z)$
$Arccot(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arccoth(-y + \tilde{\imath} \cdot x)$
$\mathrm{arccot}(z) = \arctan(1/z)$

B.5.3.2.12 Radian arc secant

The following equalities have been used to derive the LIA-3 requirements:

$\mathrm{arcsec}(-z) = \pi - \mathrm{arcsec}(z)$
$\mathrm{arcsec}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arcsec}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$
$\mathrm{arcsec}(z) = \pi/2 - \mathrm{arccsc}(z)$
$Arcsec(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arcsech(x + \tilde{\imath} \cdot y)$
$\mathrm{arcsec}(z) = \arccos(1/z)$

B.5.3.2.13 Radian arc cosecant

The following equalities have been used to derive the LIA-3 requirements:

$\mathrm{arccsc}(-z) = -\mathrm{arccsc}(z)$
$\mathrm{arccsc}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arccsc}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$
$\mathrm{arccsc}(z) = \pi/2 - \mathrm{arcsec}(z)$
$Arccsc(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arccsch(-y + \tilde{\imath} \cdot x)$
$\mathrm{arccsc}(z) = \arcsin(1/z)$

B.5.3.3 Operations for hyperbolic elementary functions

The operations for hyperbolic elementary functions are all defined directly in terms of "turned" imaginary and complex trigonometric operations of a "turned" argument. Their specifications are therefore rather short. It's done this way for several reasons:

a) The hyperbolic operations are less commonly used than the trigonometric ones.

b) The connections with the corresponding trigonometric operations are exact rather than approximate.

c) The hyperbolic functions have some 'irregularities' (expressed as conditionals in the specifications for the inverse hyperbolic operations for arccosh and arcsech) that the trigonometric operations don't have. It is therefore slightly easier to specify the hyperbolic operations in terms of the trigonometric ones rather than the other way around.

d) C also specifies exact correspondences between the complex hyperbolic operations and the complex trigonometric operations (though the other way around, and skipping the "irregular" operations, compared to the specification in LIA-3).

The LIA-3 hyperbolic and inverse hyperbolic operations that take imaginary or complex arguments interpret the imaginary part (of the argument or result respectively) as in radians. Operations that take an extra floating point argument, giving the non-radian cyclic unit, can be made, paralleling such operations (for trigonometric operations) in LIA-2. Such operations are not included in LIA-3, because no programming language require them.

B.5.3.3.1 Hyperbolic normalisation

B.5.3.3.2 Hyperbolic sine

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$$\sinh(x) = -\tilde{\imath} \cdot \sin(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sin(-\tilde{\imath} \cdot x)$$
$$\sinh(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sin(-y) = \tilde{\imath} \cdot \sin(y)$$
$$\sinh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sin(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sin(y - \tilde{\imath} \cdot x)$$

B.5.3.3.3 Hyperbolic cosine

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$$\cosh(x) = \cos(\tilde{\imath} \cdot x) = \cos(-\tilde{\imath} \cdot x)$$
$$\cosh(\tilde{\imath} \cdot y) = \cos(y) = \cos(-y)$$
$$\cosh(x + \tilde{\imath} \cdot y) = \cos(-y + \tilde{\imath} \cdot x) = \cos(y - \tilde{\imath} \cdot x)$$

B.5.3.3.4 Hyperbolic tangent

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$\mathrm{tanh}(x) = -\tilde{\imath} \cdot \mathrm{tan}(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \mathrm{tan}(-\tilde{\imath} \cdot x)$
$\mathrm{tanh}(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \mathrm{tan}(-y) = \tilde{\imath} \cdot \mathrm{tan}(y)$
$\mathrm{tanh}(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \mathrm{tan}(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \mathrm{tan}(y - \tilde{\imath} \cdot x)$

B.5.3.3.5 Hyperbolic cotangent

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$\mathrm{coth}(z) = \mathrm{tanh}(\tilde{\imath} \cdot \pi/2 - z)$
$\mathrm{coth}(x) = \tilde{\imath} \cdot \mathrm{cot}(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{cot}(-\tilde{\imath} \cdot x)$
$\mathrm{coth}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{cot}(-y) = -\tilde{\imath} \cdot \mathrm{cot}(y)$
$\mathrm{coth}(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{cot}(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{cot}(y - \tilde{\imath} \cdot x)$

B.5.3.3.6 Hyperbolic secant

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$\mathrm{sech}(x) = \mathrm{sec}(\tilde{\imath} \cdot x) = \mathrm{sec}(-\tilde{\imath} \cdot x)$
$\mathrm{sech}(\tilde{\imath} \cdot y) = \mathrm{sec}(-y) = \mathrm{sec}(y)$
$\mathrm{sech}(x + \tilde{\imath} \cdot y) = \mathrm{sec}(-y + \tilde{\imath} \cdot x) = \mathrm{sec}(y - \tilde{\imath} \cdot x)$

B.5.3.3.7 Hyperbolic cosecant

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$\mathrm{csch}(z) = \mathrm{sech}(\tilde{\imath} \cdot \pi/2 - z)$
$\mathrm{csch}(x) = \tilde{\imath} \cdot \mathrm{csc}(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{csc}(-\tilde{\imath} \cdot x)$
$\mathrm{csch}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{csc}(-y) = -\tilde{\imath} \cdot \mathrm{csc}(y)$
$\mathrm{csch}(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{csc}(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{csc}(y - \tilde{\imath} \cdot x)$

B.5.3.3.8 Inverse hyperbolic sine

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$Arcsinh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arcsin(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arcsin(y - \tilde{\imath} \cdot x)$

B.5.3.3.9 Inverse hyperbolic cosine

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$Arccosh(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arccos(x + \tilde{\imath} \cdot y)$

B.5.3.3.10 Inverse hyperbolic tangent

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$$Arctanh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arctan(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arctan(y - \tilde{\imath} \cdot x)$$

B.5.3.3.11 Inverse hyperbolic cotangent

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$$Arccoth(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arccot(-y + \tilde{\imath} \cdot x)$$
$$\mathrm{arccoth}(z) = \mathrm{arctanh}(1/z)$$

B.5.3.3.12 Inverse hyperbolic secant

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$$\mathrm{arcsech}(z) = \tilde{\imath} \cdot \pi/2 - \mathrm{arccsch}(z) \text{ if } \mathfrak{Re}(z) > 0$$
$$Arcsech(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arcsec(x + \tilde{\imath} \cdot y)$$
$$\mathrm{arcsech}(z) = \mathrm{arccosh}(1/z)$$

B.5.3.3.13 Inverse hyperbolic cosecant

Only the relationship to the corresponding trigonometric function is used in deriving the LIA-3 specification.

$$\mathrm{arccsch}(z) = \tilde{\imath} \cdot \pi/2 - \mathrm{arcsech}(z) \text{ if } \mathfrak{Re}(z) > 0$$
$$Arccsch(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arccsc(-y + \tilde{\imath} \cdot x)$$
$$\mathrm{arccsch}(z) = \mathrm{arcsinh}(1/z)$$

B.5.4 Operations for conversion between imaginary and complex numeric datatypes

B.5.5 Support for imaginary and complex numerals

B.6 Notification

B.6.1 Continuation values

Some operations are specified applications of one or more other operations. In these cases the notifications of the "suboperations" are recorded, if recording of indicators are used, and a result (continuation value) is to be computed from the continuation values of the "suboperations".

B.7 Relationship with language standards

B.8 Documentation requirements

Annex C
(informative)

Example bindings for specific languages

This annex describes how a computing system can simultaneously conform to a language standard (or publicly available specification) and to LIA-3. It contains suggestions for binding the "abstract" operations specified in LIA-3 to concrete language syntax. The format used for these example bindings in this annex is a short form version, suitable for the purposes of this annex. An actual binding is under no obligation to follow this format. An actual binding should, however, as in the bindings examples, give the LIA-3 operation name, or parameter name, bound to an identifier (or expression) by the binding.

Portability of programs can be improved if two conforming LIA-3 systems using the same programming language agree in the manner with which they adhere to LIA-3. For instance, LIA-3 requires that the parameter $big\_angle\_r_F$ be provided (if any conforming complex radian trigonometric operations are provided), but if one system provides it by means of the identifier `BigAngle` and another by the identifier `MaxAngle` for the same programming language, portability is impaired. Clearly, it would be best if such names were defined in the relevant language standards or binding standards, but in the meantime, suggestions are given here to aid portability. Name consistency cannot, however, be fully maintained between different programming languages, due to already existing differences in naming conventions, and LIA does *not* require wholesale naming changes, nor expression syntax changes.

The following clauses are suggestions rather than requirements because the areas covered are the responsibility of the various programming language standards committees. Until binding standards are in place, implementors can promote "de facto" portability by following these suggestions on their own.

The languages covered in this annex are:

Ada
C
C++
Fortran
Common Lisp

This list is not exhaustive. Other languages and other computing devices (like 'scientific' calculators, 'web script' languages, and database 'query languages') may be suitable for conformity to LIA-3.

In this annex, the parameters, operations, and exception behaviour of each language are examined to see how closely they fit the requirements of LIA-3.

a) For parameters and operations provided (semanticswise, or with a very close semantics) via the specification of a programming language, their syntax is marked with a ⋆ below, occasionally noting minor deviations (which bindings are allowed to have).

b) Where parameters or operations are not provided by the language, suggested names and syntax for them are given but marked with a †.

c) Some provided operations are closely related to LIA-3 operations, but not specified by this version of LIA-3; the syntax for such operations are in addition to the † noted as "Not LIA-3".

This annex describes only the language-level support for LIA-3. An implementation that wishes to conform must ensure that the underlying hardware and software is also configured to conform to LIA-3 requirements.

A complete binding for LIA-3 will include, or refer to, a binding for LIA-2 and LIA-1. In turn, a complete binding for LIA-2/LIA-1 may include, or refer to, a binding for IEC 60559. Such a joint LIA-3/LIA-2/LIA-1/IEC 60559 binding should be developed as a single binding standard. To avoid conflict with ongoing development, only the LIA-3 specific portions of such a binding are exemplified in this annex.

Most language standards permit an implementation to provide, by some means, the parameters and operations required by LIA-3 that are not already part of the language. The method for accessing these additional parameters and operations depends on the implementation and language, and is not specified in LIA-3 nor exemplified in this annex. It could include external subroutine libraries; new intrinsic functions supported by the compiler; constants and functions provided as global "macros"; and so on. The actual method of access through libraries, macros, etc. should of course be given in a real binding.

Most language standards do not constrain the accuracy of elementary numerical functions, or specify the subsequent behaviour after an arithmetic notification occurs.

In the event that there is a conflict between the requirements of the language standard and the requirements of LIA-3, the language binding standard should clearly identify the conflict and state its resolution of the conflict.

C.1 Ada

The programming language Ada is defined by ISO/IEC 8652:1995, *Information Technology – Programming Languages – Ada* [2].

An implementation should follow all the requirements of LIA-3 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to LIA-3 for that operation or parameter. For each of the marked items a suggested identifier is provided.

The Ada datatype `Boolean` corresponds to the LIA datatype **Boolean**.

Every implementation of Ada has at least one integer datatype, and at least one floating point datatype. The notations *INT* and *FLT* are used to stand for the names of one of these datatypes (respectively) in what follows.

Ada has imaginary and complex floating point datatypes. Standard Ada does not have any imaginary and complex integer datatypes, but such can be added as libraries (including infix syntax for certain operators). The notations *IINT* and *IFLT* are used to stand for the names of one of the imaginary datatypes (for integers and floating point respectively), and the notations *CINT* and *CFLT* are used to stand for the names of one of the complex datatypes (for integers and floating point respectively).

Ada has an overloading system, so that the same name can be used for different types of arguments to the operations. However, the constants `i` and `j` are defined in a generic package, but do not have any arguments whose types can determine which package instance is used (unless there is only one instance). The constants `i` and `j` may therefore need to be qualified with which package instance in used, even if that package is 'open'.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them:

$eq_{\mathrm{i}(I)}(x, y)$	x `=` y	†
$eq_{I,\mathrm{i}(I)}(x, y)$	x `=` y	†
$eq_{\mathrm{i}(I),I}(x, y)$	x `=` y	†
$eq_{I,\mathrm{c}(I)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(I),I}(x, y)$	x `=` y	†
$eq_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(I)}(x, y)$	x `=` y	†
$neq_{\mathrm{i}(I)}(x, y)$	x `/=` y	†
$neq_{I,\mathrm{i}(I)}(x, y)$	x `/=` y	†
$neq_{\mathrm{i}(I),I}(x, y)$	x `/=` y	†
$neq_{I,\mathrm{c}(I)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(I),I}(x, y)$	x `/=` y	†
$neq_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(I)}(x, y)$	x `/=` y	†
$lss_{\mathrm{i}(I)}(x, y)$	x `<` y	†
$leq_{\mathrm{i}(I)}(x, y)$	x `<=` y	†
$gtr_{\mathrm{i}(I)}(x, y)$	x `>` y	†
$geq_{\mathrm{i}(I)}(x, y)$	x `>=` y	†
$itimes_{I\to\mathrm{i}(I)}(x)$	`ii *` x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`ii *` x	†
$itimes_{\mathrm{c}(I)}(x)$	`ii *` x	†
$re_I(x)$	`Re(`x`)`	†
$re_{\mathrm{i}(I)}(x)$	`Re(`x`)`	†
$re_{\mathrm{c}(I)}(x)$	`Re(`x`)`	†
$im_I(x)$	`Im(`x`)`	†
$im_{\mathrm{i}(I)}(x)$	`Im(`x`)`	†
$im_{\mathrm{c}(I)}(x)$	`Im(`x`)`	†
$plusitimes_I(x, y)$	`Compose_From_Cartesian(`x`,`y`)` or	†
	x `+ ii *` y	†
$neg_{\mathrm{i}(I)}(x)$	`-`x	†
$neg_{\mathrm{c}(I)}(x)$	`-`x	†
$conj_I(x)$	`Conjugate(`x`)`	†
$conj_{\mathrm{i}(I)}(x)$	`Conjugate(`x`)`	†
$conj_{\mathrm{c}(I)}(x)$	`Conjugate(`x`)`	†

$add_{\mathrm{i}(I)}(x, y)$	`x + y`	†
$add_{I,\mathrm{i}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{i}(I),I}(x, y)$	`x + y`	†
$add_{I,\mathrm{c}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{c}(I),I}(x, y)$	`x + y`	†
$add_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{c}(I)}(x, y)$	`x + y`	†
$sub_{\mathrm{i}(I)}(x, y)$	`x - y`	†
$sub_{I,\mathrm{i}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{i}(I),I}(x, y)$	`x - y`	†
$sub_{I,\mathrm{c}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I),I}(x, y)$	`x - y`	†
$sub_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I)}(x, y)$	`x - y`	†
$mul_{\mathrm{i}(I)}(x, y)$	`x * y`	†
$mul_{I,\mathrm{i}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{i}(I),I}(x, y)$	`x * y`	†
$mul_{I,\mathrm{c}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{c}(I),I}(x, y)$	`x * y`	†
$mul_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{c}(I)}(x, y)$	`x * y`	†
$abs_{\mathrm{i}(I)}(x)$	`abs x`	†
$signum_I(x)$	`Signum(x)`	†
$signum_{\mathrm{i}(I)}(x)$	`Signum(x)`	†
$divides_{\mathrm{i}(I)}(x, y)$	`Divides(x, y)`	†
$divides_{I,\mathrm{i}(I)}(x, y)$	`Divides(x, y)`	†
$divides_{\mathrm{i}(I),I}(x, y)$	`Divides(x, y)`	†
$divides_{I,\mathrm{c}(I)}(x, y)$	`Divides(x, y)`	†
$divides_{\mathrm{c}(I),I}(x, y)$	`Divides(x, y)`	†
$divides_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`Divides(x, y)`	†
$divides_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`Divides(x, y)`	†
$divides_{\mathrm{c}(I)}(x, y)$	`Divides(x, y)`	†
$quot_{\mathrm{i}(I)}(x, y)$	`Quotient(x, y)`	†
$quot_{I,\mathrm{i}(I)}(x, y)$	`Quotient(x, y)`	†
$quot_{\mathrm{i}(I),I}(x, y)$	`Quotient(x, y)`	†
$quot_{I,\mathrm{c}(I)}(x, y)$	`Quotient(x, y)`	†
$quot_{\mathrm{c}(I),I}(x, y)$	`Quotient(x, y)`	†
$quot_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`Quotient(x, y)`	†
$quot_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`Quotient(x, y)`	†

$quot_{c(I)}(x, y)$	`Quotient(`x`, `y`)`	†
$mod_{i(I)}(x, y)$	x `mod` y	†
$mod_{I,i(I)}(x, y)$	x `mod` y	†
$mod_{i(I),I}(x, y)$	x `mod` y	†
$mod_{I,c(I)}(x, y)$	x `mod` y	†
$mod_{c(I),I}(x, y)$	x `mod` y	†
$mod_{i(I),c(I)}(x, y)$	x `mod` y	†
$mod_{c(I),i(I)}(x, y)$	x `mod` y	†
$mod_{c(I)}(x, y)$	x `mod` y	†
$ratio_{i(I)}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{I,i(I)}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{i(I),I}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{I,c(I)}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{c(I),I}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{i(I),c(I)}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{c(I),i(I)}(x, y)$	`Ratio(`x`, `y`)`	†
$ratio_{c(I)}(x, y)$	`Ratio(`x`, `y`)`	†
$residue_{i(I)}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{I,i(I)}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{i(I),I}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{I,c(I)}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{c(I),I}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{i(I),c(I)}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{c(I),i(I)}(x, y)$	`Residue(`x`, `y`)`	†
$residue_{c(I)}(x, y)$	`Residue(`x`, `y`)`	†
$group_{i(I)}(x, y)$	`Group(`x`, `y`)`	†
$group_{I,i(I)}(x, y)$	`Group(`x`, `y`)`	†
$group_{i(I),I}(x, y)$	`Group(`x`, `y`)`	†
$group_{I,c(I)}(x, y)$	`Group(`x`, `y`)`	†
$group_{c(I),I}(x, y)$	`Group(`x`, `y`)`	†
$group_{i(I),c(I)}(x, y)$	`Group(`x`, `y`)`	†
$group_{c(I),i(I)}(x, y)$	`Group(`x`, `y`)`	†
$group_{c(I)}(x, y)$	`Group(`x`, `y`)`	†
$pad_{i(I)}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{I,i(I)}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{i(I),I}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{I,c(I)}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{c(I),I}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{i(I),c(I)}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{c(I),i(I)}(x, y)$	`Pad(`x`, `y`)`	†
$pad_{c(I)}(x, y)$	`Pad(`x`, `y`)`	†

$max_{\mathrm{i}(I)}(x, y)$	*IINT*`'Max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x, y)$	*IINT*`'Min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`Max(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`Min(`xs`)`	†

where x and y are expressions of type *INT*, *IINT*, or *CINT* as appropriate, and xs is an expression of type `array (Integer range <>) of` *IINT*.

The parameters for LIA-3 operations approximating real complex valued functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{\mathrm{c}(F)}$	`Box_Err_Mul(`x`)`	†
$max\_err\_mul_{\mathrm{c}(F)}$	`Err_Mul(`x`)`	†
$box\_error\_mode\_div_{\mathrm{c}(F)}$	`Box_Err_Div(`x`)`	†
$max\_err\_div_{\mathrm{c}(F)}$	`Err_Div(`x`)`	†
$max\_err\_exp_{\mathrm{c}(F)}$	`Err_Exp(`x`)`	†
$max\_err\_power_{\mathrm{c}(F)}$	`Err_Power(`x`)`	†
$max\_err\_sin_{\mathrm{c}(F)}$	`Err_Sin(`x`)` or `Err_Sinh(`x`)`	†
$max\_err\_tan_{\mathrm{c}(F)}$	`Err_Tan(`x`)` or `Err_Tanh(`x`)`	†

where x is an expression of type *CFLT*. The parameter functions are constant for each type (and library); the argument is used only to differentiate among the floating point types for the overloading resolution.

The LIA-3 basic complex floating point operations are listed below, along with the syntax used to invoke them:

$eq_{\mathrm{i}(F)}(x, y)$	x `=` y	⋆
$eq_{F,\mathrm{i}(F)}(x, y)$	x `=` y	†
$eq_{\mathrm{i}(F),F}(x, y)$	x `=` y	†
$eq_{F,\mathrm{c}(F)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(F),F}(x, y)$	x `=` y	†
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(F)}(x, y)$	x `=` y	⋆
$neq_{\mathrm{i}(F)}(x, y)$	x `/=` y	⋆
$neq_{F,\mathrm{i}(F)}(x, y)$	x `/=` y	†
$neq_{\mathrm{i}(F),F}(x, y)$	x `/=` y	†
$neq_{F,\mathrm{c}(F)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(F),F}(x, y)$	x `/=` y	†
$neq_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(F)}(x, y)$	x `/=` y	⋆
$lss_{\mathrm{i}(F)}(x, y)$	x `<` y	⋆
$leq_{\mathrm{i}(F)}(x, y)$	x `<=` y	⋆
$gtr_{\mathrm{i}(F)}(x, y)$	x `>` y	⋆
$geq_{\mathrm{i}(F)}(x, y)$	x `>=` y	⋆

$itimes_{F\to i(F)}(x)$	`i * `x or `j * `x	⋆
$itimes_{i(F)\to F}(x)$	`i * `x or `j * `x	⋆
$itimes_{c(F)}(x)$	`i * `x or `j * `x	⋆
$re_F(x)$	`Re(`x`)`	†
$re_{i(F)}(x)$	`Re(`x`)`	†
$re_{c(F)}(x)$	`Re(`x`)`	⋆
$im_F(x)$	`Im(`x`)`	†
$im_{i(F)}(x)$	`Im(`x`)`	⋆
$im_{c(F)}(x)$	`Im(`x`)`	⋆
$plusitimes_F(x,y)$	`Compose_From_Cartesian(`x`,`y`)` or	⋆
	x` + i * `y or x` + j * `y	⋆
$neg_{i(F)}(x)$	`-`x	⋆
$neg_{c(F)}(x)$	`-`x	⋆
$conj_F(x)$	`Conjugate(`x`)`	†
$conj_{i(F)}(x)$	`Conjugate(`x`)`	⋆
$conj_{c(F)}(x)$	`Conjugate(`x`)`	⋆
$add_{i(F)}(x,y)$	x` + `y	⋆
$add_{F,i(F)}(x,y)$	x` + `y	⋆
$add_{i(F),F}(x,y)$	x` + `y	⋆
$add_{F,c(F)}(x,y)$	x` + `y	⋆
$add_{c(F),F}(x,y)$	x` + `y	⋆
$add_{i(F),c(F)}(x,y)$	x` + `y	⋆
$add_{c(F),i(F)}(x,y)$	x` + `y	⋆
$add_{c(F)}(x,y)$	x` + `y	⋆
$sub_{i(F)}(x,y)$	x` - `y	⋆
$sub_{F,i(F)}(x,y)$	x` - `y	⋆
$sub_{i(F),F}(x,y)$	x` - `y	⋆
$sub_{F,c(F)}(x,y)$	x` - `y	⋆
$sub_{c(F),F}(x,y)$	x` - `y	⋆
$sub_{i(F),c(F)}(x,y)$	x` - `y	⋆
$sub_{c(F),i(F)}(x,y)$	x` - `y	⋆
$sub_{c(F)}(x,y)$	x` - `y	⋆
$mul_{i(F)}(x,y)$	x` * `y	⋆
$mul_{F,i(F)}(x,y)$	x` * `y	⋆
$mul_{i(F),F}(x,y)$	x` * `y	⋆
$mul_{F,c(F)}(x,y)$	x` * `y	⋆
$mul_{c(F),F}(x,y)$	x` * `y	⋆
$mul_{i(F),c(F)}(x,y)$	x` * `y	⋆
$mul_{c(F),i(F)}(x,y)$	x` * `y	⋆
$mul_{c(F)}(x,y)$	x` * `y	⋆
$div_{i(F)}(x,y)$	x` / `y	⋆

$div_{F,\mathrm{i}(F)}(x, y)$	`x / y`	⋆
$div_{\mathrm{i}(F),F}(x, y)$	`x / y`	⋆
$div_{F,\mathrm{c}(F)}(x, y)$	`x / y`	⋆
$div_{\mathrm{c}(F),F}(x, y)$	`x / y`	⋆
$div_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	`x / y`	⋆
$div_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	`x / y`	⋆
$div_{\mathrm{c}(F)}(x, y)$	`x / y`	⋆
$abs_{\mathrm{i}(F)}(x)$	`abs x`	⋆
$abs_{\mathrm{c}(F)}(x)$	`abs x` or `Modulus(x)`	⋆
$phase_F(x)$	`Argument(x)`	†
$phase_{\mathrm{i}(F)}(x)$	`Argument(x)`	†
$phase_{\mathrm{c}(F)}(x)$	`Argument(x)`	⋆
$phaseu_F(u, x)$	`Argument(x, u)`	†
$phaseu_{\mathrm{i}(F)}(u, x)$	`Argument(x, u)`	†
$phaseu_{\mathrm{c}(F)}(u, x)$	`Argument(x, u)`	⋆
$signum_F(x)$	`Signum(x)`	†
$signum_{\mathrm{i}(F)}(x)$	`Signum(x)`	†
$signum_{\mathrm{c}(F)}(x)$	`Signum(x)`	†
$floor_{\mathrm{i}(F)}(x)$	*IFLT*`'Floor(x)`	†
$floor_{\mathrm{c}(F)}(x)$	*CFLT*`'Floor(x)`	†
$rounding_{\mathrm{i}(F)}(x)$	*IFLT*`'Unbiased_Rounding(x)`	†
$rounding_{\mathrm{c}(F)}(x)$	*CFLT*`'Unbiased_Rounding(x)`	†
$ceiling_{\mathrm{i}(F)}(x)$	*IFLT*`'Ceiling(x)`	†
$ceiling_{\mathrm{c}(F)}(x)$	*CFLT*`'Ceiling(x)`	†
$max_{\mathrm{i}(F)}(x, y)$	*IFLT*`'Max(x, y)`	†
$mmax_{\mathrm{i}(F)}(x, y)$	`Mmax(x, y)`	†
$min_{\mathrm{i}(F)}(x, y)$	*IFLT*`'Min(x, y)`	†
$mmin_{\mathrm{i}(F)}(x, y)$	`Mmin(x, y)`	†
$max\_seq_{\mathrm{i}(F)}(xs)$	`Max(xs)`	†
$mmax\_seq_{\mathrm{i}(F)}(xs)$	`Mmax(xs)`	†
$min\_seq_{\mathrm{i}(F)}(xs)$	`Min(xs)`	†
$mmin\_seq_{\mathrm{i}(F)}(xs)$	`Mmin(xs)`	†
$polar_F(x, y)$	`Compose_From_Polar(x, y)`	⋆
$polaru_F(u, x, y)$	`Compose_From_Polar(x, y, u)`	⋆

where x and y are expressions of type *FLT*, *IFLT*, or *CFLT* as appropriate, and u is an expression of type *FLT*.

The LIA-3 elementary floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(x, y)$	`x ** y`	⋆
$power_{\mathrm{c}(F),I}(x, y)$	`x ** y`	⋆ Not LIA-3

$exp_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Exp(`x`)`	⋆
$exp_{\mathrm{c}(F)}(x)$	`Exp(`x`)`	⋆
$power_{F\to\mathrm{c}(F)}(x, y)$	`Compose_From_Cartesian(`x`) **` y	⋆
$power_{\mathrm{c}(F)}(x, y)$	x `**` y	⋆ Not LIA-3
$sqrt_{F\to\mathrm{c}(F)}(x)$	`Sqrtc(`x`)`	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Sqrt(`x`)`	†
$sqrt_{\mathrm{c}(F)}(x)$	`Sqrt(`x`)`	⋆
$ln_{F\to\mathrm{c}(F)}(x)$	`Logc(`x`)`, or	†
	`Log(abs` x`) + i*Arctan(Im(`x`),` x`)`	⋆
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Log(`x`)`	†
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Log(abs` x`) + i*Arctan(Im(`x`), Re(`x`))`	⋆
$ln_{\mathrm{c}(F)}(x)$	`Log(`x`)`	⋆
$rad_{\mathrm{i}(F)}(x)$	`Rad(`x`)`	†
$rad_{\mathrm{c}(F)}(x)$	`Rad(`x`)`	†
$sin_{\mathrm{i}(F)}(x)$	`Sin(`x`)` or	†
	`i * Sinh(Im(`x`))`	⋆
$sin_{\mathrm{c}(F)}(x)$	`Sin(`x`)`	⋆
$cos_{\mathrm{i}(F)}(x)$	`Cos(`x`)` or	†
	`Cosh(Im(`x`))`	⋆
$cos_{\mathrm{c}(F)}(x)$	`Cos(`x`)`	⋆
$tan_{\mathrm{i}(F)}(x)$	`Tan(`x`)` or	†
	`i * Tanh(Im(`x`))`	⋆
$tan_{\mathrm{c}(F)}(x)$	`Tan(`x`)`	⋆
$cot_{\mathrm{i}(F)}(x)$	`Cot(`x`)` or	†
	`-i * Coth(Im(`x`))`	⋆
$cot_{\mathrm{c}(F)}(x)$	`Cot(`x`)`	⋆
$sec_{\mathrm{i}(F)}(x)$	`Sec(`x`)`	†
$sec_{\mathrm{c}(F)}(x)$	`Sec(`x`)`	†
$csc_{\mathrm{i}(F)}(x)$	`Csc(`x`)`	†
$csc_{\mathrm{c}(F)}(x)$	`Csc(`x`)`	†
$arcsin_{F\to\mathrm{c}(F)}(x)$	`Arcsinc(`x`)`	†
$arcsin_{\mathrm{i}(F)}(x)$	`Arcsin(`x`)`	†
$arcsin_{\mathrm{c}(F)}(x)$	`Arcsin(`x`)`	⋆
$arccos_{F\to\mathrm{c}(F)}(x)$	`Arccosc(`x`)`	†
$arccos_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arccos(`x`)`	†
$arccos_{\mathrm{c}(F)}(x)$	`Arccos(`x`)`	⋆
$arctan_{\mathrm{i}(F)}(x)$	`Arctan(`x`)`	†
$arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arctanc(`x`)`	†
$arctan_{\mathrm{c}(F)}(x)$	`Arctan(`x`)`	⋆
$arccot_{\mathrm{i}(F)}(x)$	`Arccot(`x`)`	†

$arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arccotc(`x`)`	†
$arccot_{\mathrm{c}(F)}(x)$	`Arccot(`x`)`	⋆
$arcsec_{F\to\mathrm{c}(F)}(x)$	`Arcsecc(`x`)`	†
$arcsec_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arcsecc(`x`)`	†
$arcsec_{\mathrm{c}(F)}(x)$	`Arcsec(`x`)`	†
$arccsc_{F\to\mathrm{c}(F)}(x)$	`Arccscc(`x`)`	†
$arccsc_{\mathrm{i}(F)}(x)$	`Arccsc(`x`)`	†
$arccsc_{\mathrm{c}(F)}(x)$	`Arccsc(`x`)`	†
$radh_F(x)$	`Radh(`x`)`	†
$radh_{\mathrm{i}(F)}(x)$	`Radh(`x`)`	†
$radh_{\mathrm{c}(F)}(x)$	`Radh(`x`)`	†
$sinh_{\mathrm{i}(F)}(x)$	`Sinh(`x`)`	†
$sinh_{\mathrm{c}(F)}(x)$	`Sinh(`x`)`	⋆
$cosh_{\mathrm{i}(F)}(x)$	`Cosh(`x`)`	†
$cosh_{\mathrm{c}(F)}(x)$	`Cosh(`x`)`	⋆
$tanh_{\mathrm{i}(F)}(x)$	`Tanh(`x`)`	†
$tanh_{\mathrm{c}(F)}(x)$	`Tanh(`x`)`	⋆
$coth_{\mathrm{i}(F)}(x)$	`Coth(`x`)`	†
$coth_{\mathrm{c}(F)}(x)$	`Coth(`x`)`	⋆
$sech_{\mathrm{i}(F)}(x)$	`Sech(`x`)`	†
$sech_{\mathrm{c}(F)}(x)$	`Sech(`x`)`	†
$csch_{\mathrm{i}(F)}(x)$	`Csch(`x`)`	†
$csch_{\mathrm{c}(F)}(x)$	`Csch(`x`)`	†
$arcsinh_{\mathrm{i}(F)}(x)$	`Arcsinh(`x`)`	†
$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arcsinhc(`x`)`	†
$arcsinh_{\mathrm{c}(F)}(x)$	`ArcSinh(`x`)`	⋆
$arccosh_{F\to\mathrm{c}(F)}(x)$	`Arccoshc(`x`)`	†
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arccoshc(`x`)`	†
$arccosh_{\mathrm{c}(F)}(x)$	`Arccosh(`x`)`	⋆
$arctanh_{F\to\mathrm{c}(F)}(x)$	`Arctanhc(`x`)`	†
$arctanh_{\mathrm{i}(F)}(x)$	`Arctanh(`x`)`	†
$arctanh_{\mathrm{c}(F)}(x)$	`Arctanh(`x`)`	⋆
$arccoth_{F\to\mathrm{c}(F)}(x)$	`Arccothc(`x`)`	†
$arccoth_{\mathrm{i}(F)}(x)$	`Arccoth(`x`)`	†
$arccoth_{\mathrm{c}(F)}(x)$	`Arccoth(`x`)`	⋆
$arcsech_{F\to\mathrm{c}(F)}(x)$	`Arcsechc(`x`)`	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arcsechc(`x`)`	†
$arcsech_{\mathrm{c}(F)}(x)$	`Arcsech(`x`)`	†
$arccsch_{\mathrm{i}(F)}(x)$	`Arccsch(`x`)`	†
$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Arccschc(`x`)`	†
$arccsch_{\mathrm{c}(F)}(x)$	`Arccsch(`x`)`	†

where x and y are expressions of type *FLT*, *IFLT*, or *CFLT* as appropriate.

Arithmetic value conversions in Ada are always explicit.

$convert_{I \to c(I)}(x)$	`Compose_From_Cartesian(x)`	†
$convert_{i(I) \to c(I)}(x)$	`Compose_From_Cartesian(x)`	†
$convert_{i(I) \to i(I')}(x)$	`ii * INT2(Im(x))`	†
$convert_{c(I) \to c(I')}(x)$	`Compose_From_Cartesian(INT2(Re(x)), INT2(Im(x)))`	†
$convert_{F \to c(F)}(x)$	`Compose_From_Cartesian(x)`	⋆
$convert_{F \to c(F)}(x)$	`x + i * Im(x)` or `x + j * Im(x)`	†
$convert_{i(F) \to c(F)}(x)$	`Compose_From_Cartesian(x)`	⋆
$convert_{i(F) \to c(F)}(x)$	`Re(x) + x`	†
$convert_{i(F) \to i(F')}(x)$	`i * FLT2(Im(x))`	⋆
$convert_{c(F) \to c(F')}(x)$	`Compose_From_Cartesian(FLT2(Re(x)), FLT2(Im(x)))`	⋆
$convert_{c(F) \to c(F')}(x)$	`FLT2(Re(x)) + i * FLT2(Im(x)))`	⋆

where x is of type *IFLT* or *CFLT* as appropriate.

Ada has standard support for input of complex numerals and output of complex values, though not for imaginary values, both to/from strings as well as to/from files, both in "narrow" characters (Latin-1) and "wide" characters (UCS-2/UTF-16):

$convert_{c(F'') \to c(F)}(s)$	`Get(s, g, w?);`	⋆
$convert_{c(F'') \to c(F)}(f)$	`Get(f?, g, w?);`	⋆
$convert_{c(F) \to c(F'')}(y)$	`Put(s, y, Aft=>a?, Exp=>e?);`	⋆
$convert_{c(F) \to c(F'')}(y)$	`Put(h?, y, Fore=>i?, Aft=>a?, Exp=>e?);`	⋆
$convert_{c(D') \to c(F)}(s)$	`Get(s, g, w?);`	⋆
$convert_{c(D') \to c(F)}(f)$	`Get(f?, g, w?);`	⋆
$convert_{c(F) \to c(D')}(y)$	`Put(s, y, Aft=>a?, Exp=>0);`	⋆
$convert_{c(F) \to c(D')}(y)$	`Put(h?, y, Fore=>i?, Aft=>a?, Exp=>0);`	⋆

where y is an expression of type *CFLT*. *base*, w, i, a, and e are expressions for non-negative integers. e is greater than 0. *base* is greater than 1. g is a variable of type *CFLT*. A ? above indicates that the parameter is optional. f is an opened input file (default is the default input file). h is an opened output file (default is the default output file). s is of type `String` or `Wide_String`. For `Get` of a floating point or fixed point numeral, the base is indicated in the numeral (default 10). For `Put` of a floating point or fixed point numeral, only base 10 is required to be supported. For details on `Get` and `Put`, see clause G.1.3 Complex Input-Output, and G.1.4, of ISO/IEC 8652:1995. The Ada standard is not explicit about the use of UCS-2/UTF-16 for input/output of "wide" characters. However, there is an implicit assumption of no data loss for data output and later input again. Though there are several standard external encodings that satisfy that (e.g. UTF-8), one may also assume that there is no encoding change during IO. Hence, big-endian UCS-2/UTF-16 is the only reasonable option (otherwise explicit mention in the Ada standard would have been necessary).

Ada does not have any special programming language syntax (as opposed to I/O syntax) for complex, or imaginary, numerals in general. Instead imaginary and complex numerals are expressed as (compile-time evaluatable) expressions involving a notation for the imaginary unit.

$imaginary\_unit_{i(I)}$	`ii`	†
$imaginary\_unit_{c(I)}$	`0+ii`	†
$imaginary\_unit_{i(F)}$	`i` or `j`	⋆
$imaginary\_unit_{c(F)}$	`0+i` or `0+j`	⋆

C.2 C

The programming language C is defined by ISO/IEC 9899:1999, *Information technology – Programming languages – C* [4].

An implementation should follow all the requirements of LIA-3 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-3 for that operation. For each of the marked items a suggested identifier is provided.

The LIA datatype **Boolean** is implemented by the C datatype `int` (1 = **true** and 0 = **false**), or the new `_Bool` datatype (usually referred to via the macro `bool`).

C defines numerous integer datatypes. They may be aliases of each other in an implementation defined way. The description here is not complete. See the C standard. Some of the integer datatypes have a predetermined bit width, and the signed ones use 2's complement for representation of negative values: `int`n`_t` and `uint`n`_t`, where n is the bit width expressed as a decimal numeral. Some bit widths are required. There are also minimum width, fastest minimum width, and special purpose integer datatypes (like `size_t`). Also provided are the more well-known integer datatypes `char`, `short int`, `int`, `long int`, `long long int`, `unsigned char`, `unsigned short int`, `unsigned int`, `unsigned long int`, and `unsigned long long int`. Finally there are the integer datatypes `intmax_t` and `uintmax_t` that are the largest provided signed and unsigned integer datatypes. `intmax_t` and `uintmax_t` may even be unbounded with a negative integer infinity for the signed variety as `INTMAX_MIN` and a positive integer infinity as `INTMAX_MAX` and `UINTMAX_MAX`. *INT* is used below to designate one of the integer datatypes.

C has no standard imaginary integer or complex integer datatypes. However, in the suggested binding below, *IINT* and *CINT* are used to designate such potential datatypes.

C names three floating point datatypes: `float`, `double`, and `long double`. *FLT* is used below to designate one of the floating point datatypes.

C has three complex floating point datatypes: `float _Complex`, `double _Complex`, and `long double _Complex`, and *may* have three imaginary floating point datatypes: `float _Imaginary`, `double _Imaginary`, and `long double _Imaginary`. `_Complex` and `_Imaginary` are usually written `complex` and `imaginary` respectively. *CFLT* and *IFLT* are used below to designate one of the complex or imaginary (respectively) floating point datatypes.

For some of the operations below, the C standard defines 'type generic macros'. These are fixed by the C standard, and new ones cannot be defined in program libraries.

The operations marked "(⋆)" are available only when the implementation provides imaginary floating point datatypes.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them (the binding for `II` is given towards the end of this binding example):

$eq_{\mathrm{i}(I)}(x, y)$	x `==` y	†
$eq_{I,\mathrm{i}(I)}(x, y)$	x `==` y	†
$eq_{\mathrm{i}(I),I}(x, y)$	x `==` y	†
$eq_{I,\mathrm{c}(I)}(x, y)$	x `==` y	†
$eq_{\mathrm{c}(I),I}(x, y)$	x `==` y	†
$eq_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `==` y	†
$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `==` y	†

$eq_{\mathrm{c}(I)}(x, y)$	`x == y`	†
$neq_{\mathrm{i}(I)}(x, y)$	`x != y`	†
$neq_{I,\mathrm{i}(I)}(x, y)$	`x != y`	†
$neq_{\mathrm{i}(I),I}(x, y)$	`x != y`	†
$neq_{I,\mathrm{c}(I)}(x, y)$	`x != y`	†
$neq_{\mathrm{c}(I),I}(x, y)$	`x != y`	†
$neq_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x != y`	†
$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x != y`	†
$neq_{\mathrm{c}(I)}(x, y)$	`x != y`	†
$lss_{\mathrm{i}(I)}(x, y)$	`x < y`	†
$leq_{\mathrm{i}(I)}(x, y)$	`x <= y`	†
$gtr_{\mathrm{i}(I)}(x, y)$	`x > y`	†
$geq_{\mathrm{i}(I)}(x, y)$	`x >= y`	†
$itimes_{I\rightarrow\mathrm{i}(I)}(x)$	`II * x`	†
$itimes_{\mathrm{i}(I)\rightarrow I}(x)$	`II * x`	†
$itimes_{\mathrm{c}(I)}(x)$	`II * x`	†
$re_I(x)$	`crealit(x)`	†
$re_{\mathrm{i}(I)}(x)$	`crealit(x)`	†
$re_{\mathrm{c}(I)}(x)$	`crealit(x)`	†
$im_I(x)$	`cimagit(x)`	†
$im_{\mathrm{i}(I)}(x)$	`cimagit(x)`	†
$im_{\mathrm{c}(I)}(x)$	`cimagit(x)`	†
$plusitimes_I(x, y)$	`x + II * y`	†
$neg_{\mathrm{i}(I)}(x)$	`-x`	†
$neg_{\mathrm{c}(I)}(x)$	`-x`	†
$conj_I(x)$	`conjit(x)`	†
$conj_{\mathrm{i}(I)}(x)$	`conjit(x)`	†
$conj_{\mathrm{c}(I)}(x)$	`conjit(x)`	†
$add_{\mathrm{i}(I)}(x, y)$	`x + y`	†
$add_{I,\mathrm{i}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{i}(I),I}(x, y)$	`x + y`	†
$add_{I,\mathrm{c}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{c}(I),I}(x, y)$	`x + y`	†
$add_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x + y`	†
$add_{\mathrm{c}(I)}(x, y)$	`x + y`	†
$sub_{\mathrm{i}(I)}(x, y)$	`x - y`	†
$sub_{I,\mathrm{i}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{i}(I),I}(x, y)$	`x - y`	†
$sub_{I,\mathrm{c}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I),I}(x, y)$	`x - y`	†

$sub_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x - y	†
$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x - y	†
$sub_{\mathrm{c}(I)}(x, y)$	x - y	†
$mul_{\mathrm{i}(I)}(x, y)$	x * y	†
$mul_{I,\mathrm{i}(I)}(x, y)$	x * y	†
$mul_{\mathrm{i}(I),I}(x, y)$	x * y	†
$mul_{I,\mathrm{c}(I)}(x, y)$	x * y	†
$mul_{\mathrm{c}(I),I}(x, y)$	x * y	†
$mul_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x * y	†
$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x * y	†
$mul_{\mathrm{c}(I)}(x, y)$	x * y	†
$abs_{\mathrm{i}(I)}(x)$	iabs$t$(x)	†
$signum_{I}(x)$	signum$t$(x)	†
$signum_{\mathrm{i}(I)}(x)$	signum$t$(x)	†
$divides_{\mathrm{i}(I)}(x, y)$	divides$t$(x, y)	†
$divides_{I,\mathrm{i}(I)}(x, y)$	divides$t$(x, y)	†
$divides_{\mathrm{i}(I),I}(x, y)$	divides$t$(x, y)	†
$divides_{I,\mathrm{c}(I)}(x, y)$	divides$t$(x, y)	†
$divides_{\mathrm{c}(I),I}(x, y)$	divides$t$(x, y)	†
$divides_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	divides$t$(x, y)	†
$divides_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	divides$t$(x, y)	†
$divides_{\mathrm{c}(I)}(x, y)$	divides$t$(x, y)	†
$quot_{\mathrm{i}(I)}(x, y)$	quot$t$(x, y)	†
$quot_{I,\mathrm{i}(I)}(x, y)$	quot$t$(x, y)	†
$quot_{\mathrm{i}(I),I}(x, y)$	quot$t$(x, y)	†
$quot_{I,\mathrm{c}(I)}(x, y)$	quot$t$(x, y)	†
$quot_{\mathrm{c}(I),I}(x, y)$	quot$t$(x, y)	†
$quot_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	quot$t$(x, y)	†
$quot_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	quot$t$(x, y)	†
$quot_{\mathrm{c}(I)}(x, y)$	quot$t$(x, y)	†
$mod_{\mathrm{i}(I)}(x, y)$	mod$t$(x, y)	†
$mod_{I,\mathrm{i}(I)}(x, y)$	mod$t$(x, y)	†
$mod_{\mathrm{i}(I),I}(x, y)$	mod$t$(x, y)	†
$mod_{I,\mathrm{c}(I)}(x, y)$	mod$t$(x, y)	†
$mod_{\mathrm{c}(I),I}(x, y)$	mod$t$(x, y)	†
$mod_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	mod$t$(x, y)	†
$mod_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	mod$t$(x, y)	†
$mod_{\mathrm{c}(I)}(x, y)$	mod$t$(x, y)	†
$ratio_{\mathrm{i}(I)}(x, y)$	ratio$t$(x, y)	†
$ratio_{I,\mathrm{i}(I)}(x, y)$	ratio$t$(x, y)	†
$ratio_{\mathrm{i}(I),I}(x, y)$	ratio$t$(x, y)	†

$ratio_{I,\mathrm{c}(I)}(x,y)$	`ratio`t`(`x`, `y`)`	†
$ratio_{\mathrm{c}(I),I}(x,y)$	`ratio`t`(`x`, `y`)`	†
$ratio_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`ratio`t`(`x`, `y`)`	†
$ratio_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`ratio`t`(`x`, `y`)`	†
$ratio_{\mathrm{c}(I)}(x,y)$	`ratio`t`(`x`, `y`)`	†
$residue_{\mathrm{i}(I)}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{I,\mathrm{i}(I)}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{\mathrm{i}(I),I}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{I,\mathrm{c}(I)}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{\mathrm{c}(I),I}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$residue_{\mathrm{c}(I)}(x,y)$	`iremainder`t`(`x`, `y`)`	†
$group_{\mathrm{i}(I)}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{I,\mathrm{i}(I)}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{\mathrm{i}(I),I}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{I,\mathrm{c}(I)}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{\mathrm{c}(I),I}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`group`t`(`x`, `y`)`	†
$group_{\mathrm{c}(I)}(x,y)$	`group`t`(`x`, `y`)`	†
$pad_{\mathrm{i}(I)}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{I,\mathrm{i}(I)}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{\mathrm{i}(I),I}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{I,\mathrm{c}(I)}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{\mathrm{c}(I),I}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`pad`t`(`x`, `y`)`	†
$pad_{\mathrm{c}(I)}(x,y)$	`pad`t`(`x`, `y`)`	†
$max_{\mathrm{i}(I)}(x,y)$	`imax`t`(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x,y)$	`imin`t`(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`imax_arr`t`(`xs`, `nr_of_items`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`imax_arr`t`(`xs`, `nr_of_items`)`	†

where x and y are expressions of type *INT*, *IINT*, or *CINT* as appropriate, and xs is an expression of type array of *IINT*, and t is a string (part of the operation name in C), that is the empty string for `int`, is `l` for `long int`, is `u` for `unsigned int`, and is `ul` for `unsigned long int`.

The parameters for LIA-3 operations approximating complex real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{\mathrm{c}(F)}$	`box_err_cmul`t	†
$max\_err\_mul_{\mathrm{c}(F)}$	`err_cmul`t	†
$box\_error\_mode\_div_{\mathrm{c}(F)}$	`box_err_cdiv`t	†
$max\_err\_div_{\mathrm{c}(F)}$	`err_cdiv`t	†

$max\_err\_exp_{c(F)}$	`err_cexp`t	†
$max\_err\_power_{c(F)}$	`err_cpower`t	†
$max\_err\_sin_{c(F)}$	`err_csin`t	†
$max\_err\_tan_{c(F)}$	`err_ctan`t	†

where t is a string (part of the parameter name in C), that is the empty string for `int`, is `l` for `long int`, is `u` for `unsigned int`, and is `ul` for `unsigned long int`.

The LIA-3 non-transcendental complex floating point operations are listed below, along with the syntax used to invoke them (the binding for `I` is given towards the end of this binding example):

$eq_{i(F)}(x,y)$	x `==` y	(⋆)
$eq_{F,i(F)}(x,y)$	x `==` y	(⋆)
$eq_{i(F),F}(x,y)$	x `==` y	(⋆)
$eq_{F,c(F)}(x,y)$	x `==` y	⋆
$eq_{c(F),F}(x,y)$	x `==` y	⋆
$eq_{i(F),c(F)}(x,y)$	x `==` y	(⋆)
$eq_{c(F),i(F)}(x,y)$	x `==` y	(⋆)
$eq_{c(F)}(x,y)$	x `==` y	⋆
$neq_{i(F)}(x,y)$	x `!=` y	(⋆)
$neq_{F,i(F)}(x,y)$	x `!=` y	(⋆)
$neq_{i(F),F}(x,y)$	x `!=` y	(⋆)
$neq_{F,c(F)}(x,y)$	x `!=` y	⋆
$neq_{c(F),F}(x,y)$	x `!=` y	⋆
$neq_{i(F),c(F)}(x,y)$	x `!=` y	(⋆)
$neq_{c(F),i(F)}(x,y)$	x `!=` y	(⋆)
$neq_{c(F)}(x,y)$	x `!=` y	⋆
$lss_{i(F)}(x,y)$	x `<` y	†
$leq_{i(F)}(x,y)$	x `<=` y	†
$gtr_{i(F)}(x,y)$	x `>` y	†
$geq_{i(F)}(x,y)$	x `>=` y	†
$itimes_{F\to i(F)}(x)$	`I *` x	(⋆)
$itimes_{i(F)\to F}(x)$	`I *` x	(⋆)
$itimes_{c(F)}(x)$	`I *` x	⋆
$re_F(x)$	`creal`t`(`x`)`	†
$re_{i(F)}(x)$	`creal`t`(`x`)`	†
$re_{c(F)}(x)$	`creal`t`(`x`)` or `creal(`x`)`	⋆
$im_F(x)$	`cimag`t`(`x`)`	†
$im_{i(F)}(x)$	`cimag`t`(`x`)`	†
$im_{c(F)}(x)$	`cimag`t`(`x`)` or `cimag(`x`)`	⋆
$plusitimes_F(x,y)$	x `+ I *` y	⋆
$neg_{i(F)}(x)$	`-` x	(⋆)
$neg_{c(F)}(x)$	`-` x	⋆
$conj_F(x)$	`conj`t`(`x`)` or `conj(`x`)`	†

$conj_{\mathrm{i}(F)}(x)$	`conjt(x)` or `conj(x)`	$(\star)$
$conj_{\mathrm{c}(F)}(x)$	`conjt(x)` or `conj(x)`	$\star$
$add_{\mathrm{i}(F)}(x,y)$	`x + y`	$(\star)$
$add_{F,\mathrm{i}(F)}(x,y)$	`x + y`	$(\star)$
$add_{\mathrm{i}(F),F}(x,y)$	`x + y`	$(\star)$
$add_{F,\mathrm{c}(F)}(x,y)$	`x + y`	$\star$
$add_{\mathrm{c}(F),F}(x,y)$	`x + y`	$\star$
$add_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	`x + y`	$(\star)$
$add_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	`x + y`	$(\star)$
$add_{\mathrm{c}(F)}(x,y)$	`x + y`	$\star$
$sub_{\mathrm{i}(F)}(x,y)$	`x - y`	$(\star)$
$sub_{F,\mathrm{i}(F)}(x,y)$	`x - y`	$(\star)$
$sub_{\mathrm{i}(F),F}(x,y)$	`x - y`	$(\star)$
$sub_{F,\mathrm{c}(F)}(x,y)$	`x - y`	$\star$
$sub_{\mathrm{c}(F),F}(x,y)$	`x - y`	$\star$
$sub_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	`x - y`	$(\star)$
$sub_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	`x - y`	$(\star)$
$sub_{\mathrm{c}(F)}(x,y)$	`x - y`	$\star$
$mul_{\mathrm{i}(F)}(x,y)$	`x * y`	$(\star)$
$mul_{F,\mathrm{i}(F)}(x,y)$	`x * y`	$(\star)$
$mul_{\mathrm{i}(F),F}(x,y)$	`x * y`	$(\star)$
$mul_{F,\mathrm{c}(F)}(x,y)$	`x * y`	$\star$
$mul_{\mathrm{c}(F),F}(x,y)$	`x * y`	$\star$
$mul_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	`x * y`	$(\star)$
$mul_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	`x * y`	$(\star)$
$mul_{\mathrm{c}(F)}(x,y)$	`x * y`	$\star$
$div_{\mathrm{i}(F)}(x,y)$	`x / y`	$(\star)$
$div_{F,\mathrm{i}(F)}(x,y)$	`x / y`	$(\star)$
$div_{\mathrm{i}(F),F}(x,y)$	`x / y`	$(\star)$
$div_{F,\mathrm{c}(F)}(x,y)$	`x / y`	$\star$
$div_{\mathrm{c}(F),F}(x,y)$	`x / y`	$\star$
$div_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	`x / y`	$(\star)$
$div_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	`x / y`	$(\star)$
$div_{\mathrm{c}(F)}(x,y)$	`x / y`	$\star$
$abs_{\mathrm{i}(F)}(x)$	`fabs(x)`	$(\star)$
$abs_{\mathrm{c}(F)}(x)$	`cabst(x)` or `fabs(x)`	$\star$
$phase_F(x)$	`cargt(x)` or `carg(x)`	†
$phase_{\mathrm{i}(F)}(x)$	`carg(x)`	$(\star)$
$phase_{\mathrm{c}(F)}(x)$	`cargt(x)` or `carg(x)`	$\star$
$phaseu_F(u,x)$	`cargut(x, u)`	†
$phaseu_{\mathrm{i}(F)}(u,x)$	`cargut(x, u)`	†

$phaseu_{\mathrm{c}(F)}(u,x)$	`cargut(x, u)`	†
$signum_F(x)$	`signbit(x)`	⋆
$signum_{\mathrm{i}(F)}(x)$	`I*signbit(cimag(x))`	⋆
$signum_{\mathrm{c}(F)}(x)$	`csignt(x)`	†
$floor_{\mathrm{i}(F)}(x)$	`floort(x)`	†
$floor_{\mathrm{c}(F)}(x)$	`floort(x)`	†
$rounding_{\mathrm{i}(F)}(x)$	`nearbyintt(x)`	†
$rounding_{\mathrm{c}(F)}(x)$	`nearbyintt(x)`	†
$ceiling_{\mathrm{i}(F)}(x)$	`ceilingt(x)`	†
$ceiling_{\mathrm{c}(F)}(x)$	`ceilingt(x)`	†
$max_{\mathrm{i}(F)}(x,y)$	`nmaxt(x, y)`	†
$mmax_{\mathrm{i}(F)}(x,y)$	`fmaxt(x, y)`	†
$min_{\mathrm{i}(F)}(x,y)$	`nmint(x, y)`	†
$mmin_{\mathrm{i}(F)}(x,y)$	`fmint(x, y)`	†
$max\_seq_{\mathrm{i}(F)}(xs)$	`nmax_arrt(xs, nr_of_items)`	†
$mmax\_seq_{\mathrm{i}(F)}(xs)$	`fmax_arrt(xs, nr_of_items)`	†
$min\_seq_{\mathrm{i}(F)}(xs)$	`nmin_arrt(xs, nr_of_items)`	†
$mmin\_seq_{\mathrm{i}(F)}(xs)$	`fmin_arrt(xs, nr_of_items)`	†
$polar_F(x,y)$	`polart(x, y)`	†
$polaru_F(u,x,y)$	`polarut(x, y, u)`	†
$proj_F(x)$	`projt(x)` or `cproj(x)`	† Not LIA-3
$proj_{\mathrm{i}(F)}(x)$	`cproj(x)`	† Not LIA-3
$proj_{\mathrm{c}(F)}(x)$	`cprojt(x)` or `cproj(x)`	⋆ Not LIA-3

where x and y are expressions of type *FLT*, *IFLT*, or *CFLT* as appropriate, u is an expression of type *FLT*, and t is a string (part of the operation name in C), that is the empty string for `double`, is "`l`" for `long double`, and is "`f`" for `float`.

The LIA-3 elementary complex floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b,i)$	`cpowit(b, i)`	†
$exp_{\mathrm{i}(F)}(x)$	`exp(x)`	(⋆)
$exp_{\mathrm{c}(F)}(x)$	`cexpt(x)` or `exp(x)`	⋆
$power_{F\to\mathrm{c}(F)}(x,y)$	`powc(x, y)`	†
$power_{\mathrm{c}(F)}(b,y)$	`cpowt(x, y)` or `pow(x, y)`	⋆ Not LIA-3
$sqrt_{F\to\mathrm{c}(F)}(x)$	`sqrtc(x)`	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`sqrt(x)`	(⋆)
$sqrt_{\mathrm{c}(F)}(x)$	`csqrtt(x)` or `sqrt(x)`	⋆
$ln_{F\to\mathrm{c}(F)}(x)$	`logct(x)`	†

$ln_{F\to c(F)}(x)$	`log(abs(`x`))+I*atan2(cimag(`x`),`x`)`	⋆
$ln_{i(F)\to c(F)}(x)$	`log(`x`)`	(⋆)
$ln_{i(F)\to c(F)}(x)$	`log(abs(`x`))+I*atan2(cimag(`x`),creal(`x`))`	(⋆)
$ln_{c(F)}(x)$	`clog`t`(`x`)` or `log(`x`)`	⋆
$rad_{i(F)}(x)$	`radian`t`(`x`)`	†
$rad_{c(F)}(x)$	`radian`t`(`x`)`	†
$sin_{i(F)}(x)$	`sin(`x`)`	(⋆)
$sin_{c(F)}(x)$	`csin`t`(`x`)` or `sin(`x`)`	⋆
$cos_{i(F)}(x)$	`cos(`x`)`	(⋆)
$cos_{c(F)}(x)$	`ccos`t`(`x`)` or `cos(`x`)`	⋆
$tan_{i(F)}(x)$	`tan(`x`)`	(⋆)
$tan_{c(F)}(x)$	`ctan`t`(`x`)` or `tan(`x`)`	⋆
$cot_{i(F)}(x)$	`cot(`x`)`	†
$cot_{c(F)}(x)$	`ccot`t`(`x`)` or `cot(`x`)`	†
$sec_{i(F)}(x)$	`sec(`x`)`	†
$sec_{c(F)}(x)$	`csec`t`(`x`)` or `sec(`x`)`	†
$csc_{i(F)}(x)$	`csc(`x`)`	†
$csc_{c(F)}(x)$	`ccsc`t`(`x`)` or `csc(`x`)`	†
$arcsin_{F\to c(F)}(x)$	`casinc`t`(`x`)`	†
$arcsin_{i(F)}(x)$	`asin(`x`)`	(⋆)
$arcsin_{c(F)}(x)$	`casin`t`(`x`)` or `asin(`x`)`	⋆
$arccos_{F\to c(F)}(x)$	`acosc`t`(`x`)`	†
$arccos_{i(F)\to c(F)}(x)$	`acos(`x`)`	(⋆)
$arccos_{c(F)}(x)$	`cacos`t`(`x`)` or `acos(`x`)`	⋆
$arctan_{i(F)}(x)$	`atan(`x`)`	(⋆)
$arctan_{i(F)\to c(F)}(x)$	`atanc`t`(`x`)`	†
$arctan_{c(F)}(x)$	`catan`t`(`x`)` or `atan(`x`)`	⋆
$arccot_{i(F)}(x)$	`acot(`x`)`	†
$arccot_{i(F)\to c(F)}(x)$	`acot(`x`)`	†
$arccot_{c(F)}(x)$	`cacot`t`(`x`)` or `acot(`x`)`	†
$arcsec_{F\to c(F)}(x)$	`asecc`t`(`x`)`	†
$arcsec_{i(F)\to c(F)}(x)$	`asecc`t`(`x`)`	†
$arcsec_{c(F)}(x)$	`casec`t`(`x`)` or `asec(`x`)`	†
$arccsc_{F\to c(F)}(x)$	`acscc`t`(`x`)`	†
$arccsc_{i(F)}(x)$	`acsc(`x`)`	†
$arccsc_{c(F)}(x)$	`cacsc`t`(`x`)` or `acsc(`x`)`	†
$radh_{F}(x)$	`radianh`t`(`x`)`	†
$radh_{i(F)}(x)$	`radianh`t`(`x`)`	†
$radh_{c(F)}(x)$	`radianh`t`(`x`)`	†
$sinh_{i(F)}(x)$	`sinh(`x`)`	(⋆)
$sinh_{c(F)}(x)$	`csinh`t`(`x`)` or `sinh(`x`)`	⋆
$cosh_{i(F)}(x)$	`cosh(`x`)`	(⋆)

$cosh_{c(F)}(x)$	ccosh$t$(x)	or	cosh(x)	⋆
$tanh_{i(F)}(x)$	tanh(x)			(⋆)
$tanh_{c(F)}(x)$	ctanh$t$(x)	or	tanh(x)	⋆
$coth_{i(F)}(x)$	coth(x)			†
$coth_{c(F)}(x)$	ccoth$t$(x)	or	coth(x)	†
$sech_{i(F)}(x)$	sech(x)			†
$sech_{c(F)}(x)$	csech$t$(x)	or	sech(x)	†
$csch_{i(F)}(x)$	csch(x)			†
$csch_{c(F)}(x)$	ccsch$t$(x)	or	csch(x)	†

$arcsinh_{i(F)}(x)$	asinh(x)			(⋆)
$arcsinh_{i(F)\to c(F)}(x)$	asinhc(x)			†
$arcsinh_{c(F)}(x)$	casinh$t$(x)	or	asinh(x)	⋆
$arccosh_{F\to c(F)}(x)$	acoshc$t$(x)			†
$arccosh_{i(F)\to c(F)}(x)$	acosh(x)			(⋆)
$arccosh_{c(F)}(x)$	cacosh$t$(x)	or	acosh(x)	⋆
$arctanh_{F\to c(F)}(x)$	atanhc$t$(x)			†
$arctanh_{i(F)}(x)$	atanh(x)			(⋆)
$arctanh_{c(F)}(x)$	catanh$t$(x)	or	atanh(x)	⋆
$arccoth_{F\to c(F)}(x)$	acothc$t$(x)			†
$arccoth_{i(F)}(x)$	acoth(x)			†
$arccoth_{c(F)}(x)$	cacoth$t$(x)	or	acoth(x)	†
$arcsech_{F\to c(F)}(x)$	asechc$t$(x)			†
$arcsech_{i(F)\to c(F)}(x)$	asech(x)			†
$arcsech_{c(F)}(x)$	casech$t$(x)	or	asech(x)	†
$arccsch_{i(F)}(x)$	acsch(x)			†
$arccsch_{i(F)\to c(F)}(x)$	acschc$t$(x)			†
$arccsch_{c(F)}(x)$	cacsch$t$(x)	or	acsch(x)	†

where x and y are expressions of type *FLT*, *IFLT*, or *CFLT* as appropriate. t is a string, part of the operation name, and is "`f`" for `float`, the empty string for `double`, and "`l`" for `long double`.

Arithmetic value conversions in C can be explicit or implicit. The explicit arithmetic value conversions are usually expressed as 'casts', except when converting to/from strings. The rules for when implicit conversions are applied is not repeated here, but work as if a cast had been applied.

$convert_{I\to c(I)}(x)$	(INT `_Complex`)x	†
$convert_{i(I)\to c(I)}(x)$	(INT `_Complex`)x	†
$convert_{i(I)\to i(I')}(x)$	($INT2$ `_Imaginary`)x	†
$convert_{c(I)\to c(I')}(x)$	($INT2$ `_Complex`)x	†

$convert_{F\to c(F)}(y)$	(FLT `_Complex`)x	⋆
$convert_{i(F)\to c(F)}(y)$	(FLT `_Complex`)x	(⋆)
$convert_{i(F)\to i(F')}(y)$	($FLT2$ `_Imaginary`)x	(⋆)
$convert_{c(F)\to c(F')}(y)$	($FLT2$ `_Complex`)x	⋆

where x is an expression of type *INT*, y is an expression of type *FLT*. *INT2* is the integer datatype that corresponds to I'. *FLT2* is the floating point datatype that corresponds to F'.

When converting to/from string formats, format strings are used. The format string is used as a pattern for the string format generated or parsed. Please see the C standard for a full description. There are no special format indicators for imaginary or complex values. Instead, the real part and the imaginary part are formatted separately, and any syntax before/between/after those numerals must be specified in the format string.

C does not have any special syntax for complex, or imaginary, numerals in general. Instead imaginary and complex numerals are expressed as (compile-time evaluable) expressions involving a notation for the imaginary unit (of types `const int _Imaginary` and `const float _Imaginary` or `const int _Complex` and `const float _Complex`).

$imaginary\_unit_{\mathrm{i}(I)}$	`II` or `_Imaginary_II`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`II` or `_Complex_II`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`I` or `_Imaginary_I`	(⋆)
$imaginary\_unit_{\mathrm{c}(F)}$	`I` or `_Complex_I`	⋆

The `I` macro is defined as `_Imaginary_I` if imaginary floating point datatypes are provided by the implementations, but defined as `_Complex_I` if imaginary floating point datatypes are not provided. Similarly can be done for the `II` macro.

C.3 C++

The programming language C++ is defined by ISO/IEC 14882:2003, *Programming languages – C++* [5].

An implementation should follow all the requirements of LIA-3 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-3 for that operation. For each of the marked items a suggested identifier is provided.

This example binding recommends that all identifiers suggested here be defined in the namespace `std::math`.

The LIA datatype **Boolean** is implemented by the C++ datatype `bool`.

Every implementation of C++ has integral datatypes `int`, `long int`, `unsigned int`, and `unsigned long int`. *INT* is used below to designate one of the integer datatypes.

C++ has no standard imaginary integer or complex integer datatypes. However, in the suggested binding below, *IINT* and *CINT* are used to designate such potential datatypes.

C++ names two floating point datatypes: `float` and `double`. *FLT* is used below to designate one of the floating point datatypes.

C++ has three complex floating point datatypes: `complex<float>`, `complex<double>`, and `complex<long double>`. *CFLT* is used below to designate one of the complex floating point datatypes.

C++ does not have imaginary floating point, nor any imaginary or complex integer datatype.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them:

$eq_{\mathrm{i}(I)}(x, y)$	x `==` y	†
$eq_{I,\mathrm{i}(I)}(x, y)$	x `==` y	†
$eq_{\mathrm{i}(I),I}(x, y)$	x `==` y	†
$eq_{I,\mathrm{c}(I)}(x, y)$	x `==` y	†
$eq_{\mathrm{c}(I),I}(x, y)$	x `==` y	†

$eq_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x == y	†
$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x == y	†
$eq_{\mathrm{c}(I)}(x,y)$	x == y	†
$neq_{\mathrm{i}(I)}(x,y)$	x != y	†
$neq_{I,\mathrm{i}(I)}(x,y)$	x != y	†
$neq_{\mathrm{i}(I),I}(x,y)$	x != y	†
$neq_{I,\mathrm{c}(I)}(x,y)$	x != y	†
$neq_{\mathrm{c}(I),I}(x,y)$	x != y	†
$neq_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x != y	†
$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x != y	†
$neq_{\mathrm{c}(I)}(x,y)$	x != y	†
$lss_{\mathrm{i}(I)}(x,y)$	x < y	†
$leq_{\mathrm{i}(I)}(x,y)$	x <= y	†
$gtr_{\mathrm{i}(I)}(x,y)$	x > y	†
$geq_{\mathrm{i}(I)}(x,y)$	x >= y	†
$itimes_{I\to\mathrm{i}(I)}(x)$	II * x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	II * x	†
$itimes_{\mathrm{c}(I)}(x)$	II * x	†
$re_{I}(x)$	x.real() or real(x)	†
$re_{\mathrm{i}(I)}(x)$	x.real() or real(x)	†
$re_{\mathrm{c}(I)}(x)$	x.real() or real(x)	†
$im_{I}(x)$	x.imag() or imag(x)	†
$im_{\mathrm{i}(I)}(x)$	x.imag() or imag(x)	†
$im_{\mathrm{c}(I)}(x)$	x.imag() or imag(x)	†
$plusitimes_{I}(x,y)$	complex(x, y)	†
$neg_{\mathrm{i}(I)}(x)$	-x	†
$neg_{\mathrm{c}(I)}(x)$	-x	†
$conj_{I}(x)$	conj(x)	†
$conj_{\mathrm{i}(I)}(x)$	conj(x)	†
$conj_{\mathrm{c}(I)}(x)$	conj(x)	†
$add_{\mathrm{i}(I)}(x,y)$	x + y	†
$add_{I,\mathrm{i}(I)}(x,y)$	x + y	†
$add_{\mathrm{i}(I),I}(x,y)$	x + y	†
$add_{I,\mathrm{c}(I)}(x,y)$	x + y	†
$add_{\mathrm{c}(I),I}(x,y)$	x + y	†
$add_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x + y	†
$add_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x + y	†
$add_{\mathrm{c}(I)}(x,y)$	x + y	†
$sub_{\mathrm{i}(I)}(x,y)$	x - y	†
$sub_{I,\mathrm{i}(I)}(x,y)$	x - y	†
$sub_{\mathrm{i}(I),I}(x,y)$	x - y	†

$sub_{I,\mathrm{c}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I),I}(x, y)$	`x - y`	†
$sub_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x - y`	†
$sub_{\mathrm{c}(I)}(x, y)$	`x - y`	†
$mul_{\mathrm{i}(I)}(x, y)$	`x * y`	†
$mul_{I,\mathrm{i}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{i}(I),I}(x, y)$	`x * y`	†
$mul_{I,\mathrm{c}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{c}(I),I}(x, y)$	`x * y`	†
$mul_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`x * y`	†
$mul_{\mathrm{c}(I)}(x, y)$	`x * y`	†
$abs_{\mathrm{i}(I)}(x)$	`abs(x)`	†
$signum_{I}(x)$	`signum(x)`	†
$signum_{\mathrm{i}(I)}(x)$	`signum(x)`	†
$divides_{\mathrm{i}(I)}(x, y)$	`divides(x, y)`	†
$divides_{I,\mathrm{i}(I)}(x, y)$	`divides(x, y)`	†
$divides_{\mathrm{i}(I),I}(x, y)$	`divides(x, y)`	†
$divides_{I,\mathrm{c}(I)}(x, y)$	`divides(x, y)`	†
$divides_{\mathrm{c}(I),I}(x, y)$	`divides(x, y)`	†
$divides_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`divides(x, y)`	†
$divides_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`divides(x, y)`	†
$divides_{\mathrm{c}(I)}(x, y)$	`divides(x, y)`	†
$quot_{\mathrm{i}(I)}(x, y)$	`quot(x, y)`	†
$quot_{I,\mathrm{i}(I)}(x, y)$	`quot(x, y)`	†
$quot_{\mathrm{i}(I),I}(x, y)$	`quot(x, y)`	†
$quot_{I,\mathrm{c}(I)}(x, y)$	`quot(x, y)`	†
$quot_{\mathrm{c}(I),I}(x, y)$	`quot(x, y)`	†
$quot_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`quot(x, y)`	†
$quot_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`quot(x, y)`	†
$quot_{\mathrm{c}(I)}(x, y)$	`quot(x, y)`	†
$mod_{\mathrm{i}(I)}(x, y)$	`mod(x, y)`	†
$mod_{I,\mathrm{i}(I)}(x, y)$	`mod(x, y)`	†
$mod_{\mathrm{i}(I),I}(x, y)$	`mod(x, y)`	†
$mod_{I,\mathrm{c}(I)}(x, y)$	`mod(x, y)`	†
$mod_{\mathrm{c}(I),I}(x, y)$	`mod(x, y)`	†
$mod_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`mod(x, y)`	†
$mod_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	`mod(x, y)`	†
$mod_{\mathrm{c}(I)}(x, y)$	`mod(x, y)`	†
$ratio_{\mathrm{i}(I)}(x, y)$	`ratio(x, y)`	†

$ratio_{I,i(I)}(x,y)$	`ratio(`x`, `y`)`	†
$ratio_{i(I),I}(x,y)$	`ratio(`x`, `y`)`	†
$ratio_{I,c(I)}(x,y)$	`ratio(`x`, `y`)`	†
$ratio_{c(I),I}(x,y)$	`ratio(`x`, `y`)`	†
$ratio_{i(I),c(I)}(x,y)$	`ratio(`x`, `y`)`	†
$ratio_{c(I),i(I)}(x,y)$	`ratio(`x`, `y`)`	†
$ratio_{c(I)}(x,y)$	`ratio(`x`, `y`)`	†
$residue_{i(I)}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{I,i(I)}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{i(I),I}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{I,c(I)}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{c(I),I}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{i(I),c(I)}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{c(I),i(I)}(x,y)$	`iremainder(`x`, `y`)`	†
$residue_{c(I)}(x,y)$	`iremainder(`x`, `y`)`	†
$group_{i(I)}(x,y)$	`group(`x`, `y`)`	†
$group_{I,i(I)}(x,y)$	`group(`x`, `y`)`	†
$group_{i(I),I}(x,y)$	`group(`x`, `y`)`	†
$group_{I,c(I)}(x,y)$	`group(`x`, `y`)`	†
$group_{c(I),I}(x,y)$	`group(`x`, `y`)`	†
$group_{i(I),c(I)}(x,y)$	`group(`x`, `y`)`	†
$group_{c(I),i(I)}(x,y)$	`group(`x`, `y`)`	†
$group_{c(I)}(x,y)$	`group(`x`, `y`)`	†
$pad_{i(I)}(x,y)$	`pad(`x`, `y`)`	†
$pad_{I,i(I)}(x,y)$	`pad(`x`, `y`)`	†
$pad_{i(I),I}(x,y)$	`pad(`x`, `y`)`	†
$pad_{I,c(I)}(x,y)$	`pad(`x`, `y`)`	†
$pad_{c(I),I}(x,y)$	`pad(`x`, `y`)`	†
$pad_{i(I),c(I)}(x,y)$	`pad(`x`, `y`)`	†
$pad_{c(I),i(I)}(x,y)$	`pad(`x`, `y`)`	†
$pad_{c(I)}(x,y)$	`pad(`x`, `y`)`	†
$max_{i(I)}(x,y)$	`max(`x`, `y`)`	†
$min_{i(I)}(x,y)$	`min(`x`, `y`)`	†
$max\_seq_{i(I)}(xs)$	xs`.max()`	†
$min\_seq_{i(I)}(xs)$	xs`.min()`	†

where x and y are expressions of imaginary or complex integer type, or an integer datatype as appropriate. xs is an expression having the type of an array of an imaginary integer datatype.

The parameters for operations approximating complex valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{c(F)}$	`numeric_limits<`$CFLT$`>::box_err_cmul()`	†
$max\_err\_mul_{c(F)}$	`numeric_limits<`$CFLT$`>::err_cmul()`	†
$box\_error\_mode\_div_{c(F)}$	`numeric_limits<`$CFLT$`>::box_err_cdiv()`	†

$max\_err\_div_{c(F)}$	`numeric_limits<`$CFLT$`>::err_cdiv()`	†
$max\_err\_exp_{c(F)}$	`numeric_limits<`$CFLT$`>::err_exp()`	†
$max\_err\_power_{c(F)}$	`numeric_limits<`$CFLT$`>::err_power()`	†
$max\_err\_sin_{c(F)}$	`numeric_limits<`$CFLT$`>::err_sin()`	†
$max\_err\_tan_{c(F)}$	`numeric_limits<`$CFLT$`>::err_tan()`	†

The LIA-3 non-transcendental complex floating point operations are listed below, along with the syntax used to invoke them:

$eq_{i(F)}(x,y)$	x `==` y	†
$eq_{F,i(F)}(x,y)$	x `==` y	†
$eq_{i(F),F}(x,y)$	x `==` y	†
$eq_{F,c(F)}(x,y)$	x `==` y	⋆
$eq_{c(F),F}(x,y)$	x `==` y	⋆
$eq_{i(F),c(F)}(x,y)$	x `==` y	†
$eq_{c(F),i(F)}(x,y)$	x `==` y	†
$eq_{c(F)}(x,y)$	x `==` y	⋆
$neq_{i(F)}(x,y)$	x `!=` y	†
$neq_{F,i(F)}(x,y)$	x `!=` y	†
$neq_{i(F),F}(x,y)$	x `!=` y	†
$neq_{F,c(F)}(x,y)$	x `!=` y	⋆
$neq_{c(F),F}(x,y)$	x `!=` y	⋆
$neq_{i(F),c(F)}(x,y)$	x `!=` y	†
$neq_{c(F),i(F)}(x,y)$	x `!=` y	†
$neq_{c(F)}(x,y)$	x `!=` y	⋆
$lss_{i(F)}(x,y)$	x `<` y	†
$leq_{i(F)}(x,y)$	x `<=` y	†
$gtr_{i(F)}(x,y)$	x `>` y	†
$geq_{i(F)}(x,y)$	x `>=` y	†
$itimes_{F\to i(F)}(x)$	`I *` x	†
$itimes_{i(F)\to F}(x)$	`I *` x	†
$itimes_{c(F)}(x)$	`I *` x	†
$re_F(x)$	x`.real()` or `real(`x`)`	†
$re_{i(F)}(x)$	x`.real()` or `real(`x`)`	†
$re_{c(F)}(x)$	x`.real()` or `real(`x`)`	⋆
$im_F(x)$	x`.imag()` or `imag(`x`)`	†
$im_{i(F)}(x)$	x`.imag()` or `imag(`x`)`	†
$im_{c(F)}(x)$	x`.imag()` or `imag(`x`)`	⋆
$plusitimes_F(x,y)$	`complex(`x`, `y`)`	⋆
$plusitimes_F(x,y)$	x `+ I *` y	†
$neg_{i(F)}(x)$	`-` x	†
$neg_{c(F)}(x)$	`-` x	⋆
$conj_F(x)$	`conj(`x`)`	†

$conj_{\mathrm{i}(F)}(x)$	`conj(`x`)`	†
$conj_{\mathrm{c}(F)}(x)$	`conj(`x`)`	⋆
$add_{\mathrm{i}(F)}(x, y)$	x `+` y	†
$add_{F,\mathrm{i}(F)}(x, y)$	x `+` y	†
$add_{\mathrm{i}(F),F}(x, y)$	x `+` y	†
$add_{F,\mathrm{c}(F)}(x, y)$	x `+` y	⋆
$add_{\mathrm{c}(F),F}(x, y)$	x `+` y	⋆
$add_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `+` y	†
$add_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `+` y	†
$add_{\mathrm{c}(F)}(x, y)$	x `+` y	⋆
$sub_{\mathrm{i}(F)}(x, y)$	x `-` y	†
$sub_{F,\mathrm{i}(F)}(x, y)$	x `-` y	†
$sub_{\mathrm{i}(F),F}(x, y)$	x `-` y	†
$sub_{F,\mathrm{c}(F)}(x, y)$	x `-` y	⋆
$sub_{\mathrm{c}(F),F}(x, y)$	x `-` y	⋆
$sub_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `-` y	†
$sub_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `-` y	†
$sub_{\mathrm{c}(F)}(x, y)$	x `-` y	⋆
$mul_{\mathrm{i}(F)}(x, y)$	x `*` y	†
$mul_{F,\mathrm{i}(F)}(x, y)$	x `*` y	†
$mul_{\mathrm{i}(F),F}(x, y)$	x `*` y	†
$mul_{F,\mathrm{c}(F)}(x, y)$	x `*` y	⋆
$mul_{\mathrm{c}(F),F}(x, y)$	x `*` y	⋆
$mul_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `*` y	†
$mul_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `*` y	†
$mul_{\mathrm{c}(F)}(x, y)$	x `*` y	⋆
$div_{\mathrm{i}(F)}(x, y)$	x `/` y	†
$div_{F,\mathrm{i}(F)}(x, y)$	x `/` y	†
$div_{\mathrm{i}(F),F}(x, y)$	x `/` y	†
$div_{F,\mathrm{c}(F)}(x, y)$	x `/` y	⋆
$div_{\mathrm{c}(F),F}(x, y)$	x `/` y	⋆
$div_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `/` y	†
$div_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `/` y	†
$div_{\mathrm{c}(F)}(x, y)$	x `/` y	⋆
$abs_{\mathrm{i}(F)}(x)$	`abs(`x`)`	⋆
$abs_{\mathrm{c}(F)}(x)$	`abs(`x`)`	⋆
$sqr\_abs_{\mathrm{c}(F)}(x)$	`norm(`x`)`	⋆ Not LIA-3
$phase_F(x)$	`arg(`x`)`	†
$phase_{\mathrm{i}(F)}(x)$	`arg(`x`)`	†
$phase_{\mathrm{c}(F)}(x)$	`arg(`x`)`	⋆
$phaseu_F(u, x)$	`argu(`x`, `u`)`	†

$phaseu_{\mathrm{i}(F)}(u, x)$	`argu(x, u)`	†
$phaseu_{\mathrm{c}(F)}(u, x)$	`argu(x, u)`	†
$signum_{F}(x)$	`sign(x)`	†
$signum_{\mathrm{i}(F)}(x)$	`sign(x)`	†
$signum_{\mathrm{c}(F)}(x)$	`sign(x)`	†
$floor_{\mathrm{i}(F)}(x)$	`floor(x)`	†
$floor_{\mathrm{c}(F)}(x)$	`floor(x)`	†
$rounding_{\mathrm{i}(F)}(x)$	`nearbyint(x)`	†
$rounding_{\mathrm{c}(F)}(x)$	`nearbyint(x)`	†
$ceiling_{\mathrm{i}(F)}(x)$	`ceiling(x)`	†
$ceiling_{\mathrm{c}(F)}(x)$	`ceiling(x)`	†
$max_{\mathrm{i}(F)}(x, y)$	`nmax(x, y)`	†
$mmax_{\mathrm{i}(F)}(x, y)$	`fmax(x, y)`	†
$min_{\mathrm{i}(F)}(x, y)$	`nmin(x, y)`	†
$mmin_{\mathrm{i}(F)}(x, y)$	`fmin(x, y)`	†
$max\_seq_{\mathrm{i}(F)}(xs)$	`nmax(xs, nr_of_items)`	†
$mmax\_seq_{\mathrm{i}(F)}(xs)$	`fmax(xs, nr_of_items)`	†
$min\_seq_{\mathrm{i}(F)}(xs)$	`nmin(xs, nr_of_items)`	†
$mmin\_seq_{\mathrm{i}(F)}(xs)$	`fmin(xs, nr_of_items)`	†
$polar_{F}(x, y)$	`polar(x, y)`	⋆
$polaru_{F}(u, x, y)$	`polaru(x, y, u)`	†

where x and y are expressions of an imaginary or complex floating point type, and u is of the corresponding real floating point type.

The LIA-3 elementary complex floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(x, i)$	`pow(x, i)`	†
$power_{\mathrm{c}(F),I}(x, i)$	`pow(x, i)`	⋆ (for `int`) Not LIA-3
$exp_{\mathrm{i}(F)}(x)$	`exp(x)`	†
$exp_{\mathrm{c}(F)}(x)$	`exp(x)`	⋆
$power_{F \to \mathrm{c}(F)}(x, y)$	`powc(x, y)`	†
$power_{\mathrm{c}(F),F}(x, y)$	`pow(x, y)`	⋆ Not LIA-3
$power_{F,\mathrm{c}(F)}(x, y)$	`pow(x, y)`	⋆ Not LIA-3
$power_{\mathrm{c}(F)}(x, y)$	`pow(x, y)`	⋆ Not LIA-3
$sqrt_{F \to \mathrm{c}(F)}(x)$	`sqrtc(x)`	†
$sqrt_{\mathrm{i}(F) \to \mathrm{c}(F)}(x)$	`sqrt(x)`	†
$sqrt_{\mathrm{c}(F)}(x)$	`sqrt(x)`	⋆
$ln_{F \to \mathrm{c}(F)}(x)$	`logc(x)`	†
$ln_{\mathrm{i}(F) \to \mathrm{c}(F)}(x)$	`complex(log(abs(x)), atan2(imag(x),real(x)))`	⋆

$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`log(`x`)`	†
$ln_{\mathrm{c}(F)}(x)$	`log(`x`)`	⋆
$rad_{\mathrm{i}(F)}(x)$	`radian(`x`)`	†
$rad_{\mathrm{c}(F)}(x)$	`radian(`x`)`	†
$sin_{\mathrm{i}(F)}(x)$	`sin(`x`)`	†
$sin_{\mathrm{c}(F)}(x)$	`sin(`x`)`	⋆
$cos_{\mathrm{i}(F)}(x)$	`cos(`x`)`	†
$cos_{\mathrm{c}(F)}(x)$	`cos(`x`)`	⋆
$tan_{\mathrm{i}(F)}(x)$	`tan(`x`)`	†
$tan_{\mathrm{c}(F)}(x)$	`tan(`x`)`	⋆
$cot_{\mathrm{i}(F)}(x)$	`cot(`x`)`	†
$cot_{\mathrm{c}(F)}(x)$	`cot(`x`)`	†
$sec_{\mathrm{i}(F)}(x)$	`sec(`x`)`	†
$sec_{\mathrm{c}(F)}(x)$	`sec(`x`)`	†
$csc_{\mathrm{i}(F)}(x)$	`csc(`x`)`	†
$csc_{\mathrm{c}(F)}(x)$	`csc(`x`)`	†
$arcsin_{F\to\mathrm{c}(F)}(x)$	`asinc(`x`)`	†
$arcsin_{\mathrm{i}(F)}(x)$	`asin(`x`)`	†
$arcsin_{\mathrm{c}(F)}(x)$	`asin(`x`)`	⋆
$arccos_{F\to\mathrm{c}(F)}(x)$	`acosc(`x`)`	†
$arccos_{\mathrm{c}(F)\to\mathrm{c}(F)}(x)$	`acos(`x`)`	†
$arccos_{\mathrm{c}(F)}(x)$	`acos(`x`)`	⋆
$arctan_{\mathrm{i}(F)}(x)$	`atan(`x`)`	†
$arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`atanc(`x`)`	†
$arctan_{\mathrm{c}(F)}(x)$	`atan(`x`)`	⋆
$arccot_{\mathrm{i}(F)}(x)$	`acot(`x`)`	†
$arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`acotc(`x`)`	†
$arccot_{\mathrm{c}(F)}(x)$	`acot(`x`)`	†
$arcsec_{F\to\mathrm{c}(F)}(x)$	`asecc(`x`)`	†
$arcsec_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`asec(`x`)`	†
$arcsec_{\mathrm{c}(F)}(x)$	`asec(`x`)`	†
$arccsc_{F\to\mathrm{c}(F)}(x)$	`acscc(`x`)`	†
$arccsc_{\mathrm{i}(F)}(x)$	`acsc(`x`)`	†
$arccsc_{\mathrm{c}(F)}(x)$	`acsc(`x`)`	†
$radh_F(x)$	`radianh(`x`)`	†
$radh_{\mathrm{i}(F)}(x)$	`radianh(`x`)`	†
$radh_{\mathrm{c}(F)}(x)$	`radianh(`x`)`	†
$sinh_{\mathrm{i}(F)}(x)$	`sinh(`x`)`	†
$sinh_{\mathrm{c}(F)}(x)$	`sinh(`x`)`	⋆
$cosh_{\mathrm{i}(F)}(x)$	`cosh(`x`)`	†
$cosh_{\mathrm{c}(F)}(x)$	`cosh(`x`)`	⋆
$tanh_{\mathrm{i}(F)}(x)$	`tanh(`x`)`	†

$tanh_{c(F)}(x)$	`tanh(`x`)`	⋆
$coth_{i(F)}(x)$	`coth(`x`)`	†
$coth_{c(F)}(x)$	`coth(`x`)`	†
$sech_{i(F)}(x)$	`sech(`x`)`	†
$sech_{c(F)}(x)$	`sech(`x`)`	†
$csch_{i(F)}(x)$	`csch(`x`)`	†
$csch_{c(F)}(x)$	`csch(`x`)`	†
$arcsinh_{i(F)}(x)$	`asinh(`x`)`	†
$arcsinh_{i(F)\to c(F)}(x)$	`asinhc(`x`)`	†
$arcsinh_{c(F)}(x)$	`asinh(`x`)`	⋆
$arccosh_{F\to c(F)}(x)$	`acoshc(`x`)`	†
$arccosh_{i(F)\to c(F)}(x)$	`acoshc(`x`)`	†
$arccosh_{c(F)}(x)$	`acosh(`x`)`	⋆
$arctanh_{F\to c(F)}(x)$	`atanhc(`x`)`	†
$arctanh_{i(F)}(x)$	`atanh(`x`)`	†
$arctanh_{c(F)}(x)$	`atanh(`x`)`	⋆
$arccoth_{F\to c(F)}(x)$	`acothc(`x`)`	†
$arccoth_{i(F)}(x)$	`acoth(`x`)`	†
$arccoth_{c(F)}(x)$	`acoth(`x`)`	†
$arcsech_{F\to c(F)}(x)$	`asechc(`x`)`	†
$arcsech_{i(F)\to c(F)}(x)$	`asech(`x`)`	†
$arcsech_{c(F)}(x)$	`asech(`x`)`	†
$arccsch_{i(F)}(x)$	`acsch(`x`)`	†
$arccsch_{i(F)\to c(F)}(x)$	`acschc(`x`)`	†
$arccsch_{c(F)}(x)$	`acsch(`x`)`	†

where x, and y are expressions of an imaginary, complex, or plain floating point type (as appropriate).

When converting to/from string formats when using the C printf/scanf family of routines, format strings are used. The format string is used as a pattern for the string format generated or parsed. Please see the C99 standard for a full description. There are no special format indicators for imaginary or complex values. Instead, the real part and the imaginary part are formatted separately, and any syntax before/between/after those numerals must be specified in the format string. In addition, C++ has a predefined format for complex (but not imaginary) values for I/O when using the stream formatting operators "<<" and ">>". For details, see clause 26.2.5 of the C++ standard [5].

$convert_{c(F)\to c(F')}(x)$	is `>>` y	⋆
$convert_{c(F')\to c(F)}(x)$	os `<<` x	⋆

C++ does not have any special syntax for complex numerals in general. Instead the `complex` function is used on floating point values (not necessarily syntactically numerals).

C.4 Fortran

The programming language Fortran is defined by ISO/IEC 1539-1:1997, *Information technology – Programming languages – Fortran – Part 1: Base language* [6].

An implementation should follow all the requirements of LIA-3 unless otherwise specified by this language binding.

The operations or parameters marked "†" are not part of the language and should be provided by an implementation that wishes to conform to the LIA-3 for that operation. For each of the marked items a suggested identifier is provided.

The Fortran datatype `LOGICAL` corresponds to the LIA datatype **Boolean**.

Every implementation of Fortran has at least one integer datatype, denoted as `INTEGER`, and at least two floating point datatypes denoted as `REAL` (single precision, also denoted `REAL(KIND(0.0))` and `DOUBLE PRECISION` (also denoted `REAL(KIND(0d0)`, and at least two complex floating point datatypes denoted as `COMPLEX` (also denoted `COMPLEX(KIND(0.0))`, and `COMPLEX(KIND(0d0))`. Standard Fortran does not have any imaginary datatypes, nor any complex integer datatypes.

An implementation is permitted to offer additional `INTEGER` types with a different range and additional `REAL` and `COMPLEX` types with different precision or range, parameterised with the `KIND` parameter.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them:

$eq_{\mathrm{i}(I)}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{I,\mathrm{i}(I)}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{i}(I),I}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{I,\mathrm{c}(I)}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{c}(I),I}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{c}(I)}(x, y)$	x `==` y or x `.EQ.` y	†
$neq_{\mathrm{i}(I)}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{I,\mathrm{i}(I)}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{i}(I),I}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{I,\mathrm{c}(I)}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{c}(I),I}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{c}(I)}(x, y)$	x `/=` y or x `.NE.` y	†
$lss_{\mathrm{i}(I)}(x, y)$	x `<` y or x `.LT.` y	†
$leq_{\mathrm{i}(I)}(x, y)$	x `<=` y or x `.LE.` y	†
$gtr_{\mathrm{i}(I)}(x, y)$	x `>` y or x `.GT.` y	†
$geq_{\mathrm{i}(I)}(x, y)$	x `>=` y or x `.GE.` y	†
$itimes_{I\to\mathrm{i}(I)}(x)$	`IMUL(`x`)`	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`IMUL(`x`)`	†
$itimes_{\mathrm{c}(I)}(x)$	`IMUL(`x`)`	†
$re_I(x)$	`IREAL(`x`)`	†
$re_{\mathrm{i}(I)}(x)$	`IREAL(`x`)`	†
$re_{\mathrm{c}(I)}(x)$	`IREAL(`x`)`	†
$im_I(x)$	`IMAG(`x`)`	†
$im_{\mathrm{i}(I)}(x)$	`IMAG(`x`)`	†

$im_{\mathrm{c}(I)}(x)$	`IMAG(`x`)`	†
$plusitimes_I(x, y)$	`CMPLX(`x`, `y`)`	†
$neg_{\mathrm{i}(I)}(x)$	`-`x	†
$neg_{\mathrm{c}(I)}(x)$	`-`x	†
$conj_I(x)$	`CONJG(`x`)`	†
$conj_{\mathrm{i}(I)}(x)$	`CONJG(`x`)`	†
$conj_{\mathrm{c}(I)}(x)$	`CONJG(`x`)`	†
$add_{\mathrm{i}(I)}(x, y)$	x `+` y	†
$add_{I,\mathrm{i}(I)}(x, y)$	x `+` y	†
$add_{\mathrm{i}(I),I}(x, y)$	x `+` y	†
$add_{I,\mathrm{c}(I)}(x, y)$	x `+` y	†
$add_{\mathrm{c}(I),I}(x, y)$	x `+` y	†
$add_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `+` y	†
$add_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `+` y	†
$add_{\mathrm{c}(I)}(x, y)$	x `+` y	†
$sub_{\mathrm{i}(I)}(x, y)$	x `-` y	†
$sub_{I,\mathrm{i}(I)}(x, y)$	x `-` y	†
$sub_{\mathrm{i}(I),I}(x, y)$	x `-` y	†
$sub_{I,\mathrm{c}(I)}(x, y)$	x `-` y	†
$sub_{\mathrm{c}(I),I}(x, y)$	x `-` y	†
$sub_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `-` y	†
$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `-` y	†
$sub_{\mathrm{c}(I)}(x, y)$	x `-` y	†
$mul_{\mathrm{i}(I)}(x, y)$	x `*` y	†
$mul_{I,\mathrm{i}(I)}(x, y)$	x `*` y	†
$mul_{\mathrm{i}(I),I}(x, y)$	x `*` y	†
$mul_{I,\mathrm{c}(I)}(x, y)$	x `*` y	†
$mul_{\mathrm{c}(I),I}(x, y)$	x `*` y	†
$mul_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	x `*` y	†
$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x, y)$	x `*` y	†
$mul_{\mathrm{c}(I)}(x, y)$	x `*` y	†
$abs_{\mathrm{i}(I)}(x)$	`ABS(`x`)`	†
$signum_I(x)$	`SIGN(1, `x`)`	⋆
$signum_{\mathrm{i}(I)}(x)$	`SIGN(1, `x`)`	†
$mul_I(abs_I(y), signum_I(x))$	`SIGN(`y`, `x`)`	⋆
	(`SIGN(`y`, `x`)` is the `copysign` operation of IEC 60559)	
$divides_{\mathrm{i}(I)}(x, y)$	`DIVIDES(`x`, `y`)`	†
$divides_{I,\mathrm{i}(I)}(x, y)$	`DIVIDES(`x`, `y`)`	†
$divides_{\mathrm{i}(I),I}(x, y)$	`DIVIDES(`x`, `y`)`	†
$divides_{I,\mathrm{c}(I)}(x, y)$	`DIVIDES(`x`, `y`)`	†
$divides_{\mathrm{c}(I),I}(x, y)$	`DIVIDES(`x`, `y`)`	†
$divides_{\mathrm{i}(I),\mathrm{c}(I)}(x, y)$	`DIVIDES(`x`, `y`)`	†

$divides_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`DIVIDES(`x`, `y`)`	†
$divides_{\mathrm{c}(I)}(x,y)$	`DIVIDES(`x`, `y`)`	†
$quot_{\mathrm{i}(I)}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{I,\mathrm{i}(I)}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{\mathrm{i}(I),I}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{I,\mathrm{c}(I)}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{\mathrm{c}(I),I}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$quot_{\mathrm{c}(I)}(x,y)$	`QUOTIENT(`x`, `y`)`	†
$mod_{\mathrm{i}(I)}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{I,\mathrm{i}(I)}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{\mathrm{i}(I),I}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{I,\mathrm{c}(I)}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{\mathrm{c}(I),I}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`MODULO(`x`, `y`)`	†
$mod_{\mathrm{c}(I)}(x,y)$	`MODULO(`x`, `y`)`	†
$ratio_{\mathrm{i}(I)}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{I,\mathrm{i}(I)}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{\mathrm{i}(I),I}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{I,\mathrm{c}(I)}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{\mathrm{c}(I),I}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`RATIO(`x`, `y`)`	†
$ratio_{\mathrm{c}(I)}(x,y)$	`RATIO(`x`, `y`)`	†
$residue_{\mathrm{i}(I)}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{I,\mathrm{i}(I)}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{\mathrm{i}(I),I}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{I,\mathrm{c}(I)}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{\mathrm{c}(I),I}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`RESIDUE(`x`, `y`)`	†
$residue_{\mathrm{c}(I)}(x,y)$	`RESIDUE(`x`, `y`)`	†
$group_{\mathrm{i}(I)}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{I,\mathrm{i}(I)}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{\mathrm{i}(I),I}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{I,\mathrm{c}(I)}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{\mathrm{c}(I),I}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`GROUP(`x`, `y`)`	†
$group_{\mathrm{c}(I)}(x,y)$	`GROUP(`x`, `y`)`	†

$pad_{\mathrm{i}(I)}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{I,\mathrm{i}(I)}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{\mathrm{i}(I),I}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{I,\mathrm{c}(I)}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{\mathrm{c}(I),I}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`PAD(`x`, `y`)`	†
$pad_{\mathrm{c}(I)}(x,y)$	`PAD(`x`, `y`)`	†
$max_{\mathrm{i}(I)}(x,y)$	`MAX(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x,y)$	`MIN(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}([x_1,...,x_n])$	`MAX(`$x_1,...,x_n$`)`	†
$min\_seq_{\mathrm{i}(I)}([x_1,...,x_n])$	`MIN(`$x_1,...,x_n$`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`MAXVAL(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`MINVAL(`xs`)`	†

where x and y are expressions of complex integer, imaginary integer, or plain integer type (as appropriate) and where xs is an expression of type array of imaginary integer.

The LIA-3 parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{\mathrm{c}(F)}$	`BOX_ERR_MUL(`x`)`	†
$max\_err\_mul_{\mathrm{c}(F)}$	`ERR_MUL(`x`)`	†
$box\_error\_mode\_div_{\mathrm{c}(F)}$	`BOX_ERR_DIV(`x`)`	†
$max\_err\_div_{\mathrm{c}(F)}$	`ERR_DIV(`x`)`	†
$max\_err\_exp_{\mathrm{c}(F)}$	`ERR_EXP(`x`)`	†
$max\_err\_power_{\mathrm{c}(F)}$	`ERR_POWER(`x`)`	†
$max\_err\_sin_{\mathrm{c}(F)}$	`ERR_SIN(`x`)`	†
$max\_err\_tan_{\mathrm{c}(F)}$	`ERR_TAN(`x`)`	†

where x is an expression of type `COMPLEX`. Several of the parameter functions are constant for each type (and library), the argument is then used only to differentiate among the floating point types.

The LIA-3 non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$eq_{\mathrm{i}(F)}(x,y)$	x `==` y or x `.EQ.` y	†
$eq_{F,\mathrm{i}(F)}(x,y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{i}(F),F}(x,y)$	x `==` y or x `.EQ.` y	†
$eq_{F,\mathrm{c}(F)}(x,y)$	x `==` y or x `.EQ.` y	⋆
$eq_{\mathrm{c}(F),F}(x,y)$	x `==` y or x `.EQ.` y	⋆
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `==` y or x `.EQ.` y	†
$eq_{\mathrm{c}(F)}(x,y)$	x `==` y or x `.EQ.` y	⋆
$neq_{\mathrm{i}(F)}(x,y)$	x `/=` y or x `.NE.` y	†
$neq_{F,\mathrm{i}(F)}(x,y)$	x `/=` y or x `.NE.` y	†

$neq_{\mathrm{i}(F),F}(x,y)$	x `/=` y or x `.NE.` y	†
$neq_{F,\mathrm{c}(F)}(x,y)$	x `/=` y or x `.NE.` y	⋆
$neq_{\mathrm{c}(F),F}(x,y)$	x `/=` y or x `.NE.` y	⋆
$neq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `/=` y or x `.NE.` y	†
$neq_{\mathrm{c}(F)}(x,y)$	x `/=` y or x `.NE.` y	⋆
$lss_{\mathrm{i}(F)}(x,y)$	x `<` y or x `.LT.` y	†
$leq_{\mathrm{i}(F)}(x,y)$	x `<=` y or x `.LE.` y	†
$gtr_{\mathrm{i}(F)}(x,y)$	x `>` y or x `.GT.` y	†
$geq_{\mathrm{i}(F)}(x,y)$	x `>=` y or x `.GE.` y	†
$itimes_{F\rightarrow\mathrm{i}(F)}(x)$	`IMUL(`x`)`	†
$itimes_{\mathrm{i}(F)\rightarrow F}(x)$	`IMUL(`x`)`	†
$itimes_{\mathrm{c}(F)}(x)$	`IMUL(`x`)`	†
$re_F(x)$	`REAL(`x`)`	⋆
$re_{\mathrm{i}(F)}(x)$	`REAL(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	`REAL(`x`)`	⋆
$im_F(x)$	`AIMAG(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	`AIMAG(`x`)`	⋆
$im_{\mathrm{c}(F)}(x)$	`AIMAG(`x`)`	⋆
$plusitimes_F(x,y)$	`CMPLX(`x`, `y`)`	⋆
$plusitimes_F(x,0)$	`CMPLX(`x`)`	⋆
$neg_{\mathrm{i}(F)}(x)$	`-`x	†
$neg_{\mathrm{c}(F)}(x)$	`-`x	⋆
$conj_F(x)$	`CONJG(`x`)`	†
$conj_{\mathrm{i}(F)}(x)$	`CONJG(`x`)`	†
$conj_{\mathrm{c}(F)}(x)$	`CONJG(`x`)`	⋆
$add_{\mathrm{i}(F)}(x,y)$	x `+` y	†
$add_{F,\mathrm{i}(F)}(x,y)$	x `+` y	†
$add_{\mathrm{i}(F),F}(x,y)$	x `+` y	†
$add_{F,\mathrm{c}(F)}(x,y)$	x `+` y	⋆
$add_{\mathrm{c}(F),F}(x,y)$	x `+` y	⋆
$add_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `+` y	†
$add_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `+` y	†
$add_{\mathrm{c}(F)}(x,y)$	x `+` y	⋆
$sub_{\mathrm{i}(F)}(x,y)$	x `-` y	†
$sub_{F,\mathrm{i}(F)}(x,y)$	x `-` y	†
$sub_{\mathrm{i}(F),F}(x,y)$	x `-` y	†
$sub_{F,\mathrm{c}(F)}(x,y)$	x `-` y	⋆
$sub_{\mathrm{c}(F),F}(x,y)$	x `-` y	⋆
$sub_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `-` y	†
$sub_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `-` y	†
$sub_{\mathrm{c}(F)}(x,y)$	x `-` y	⋆