SC22/WG11 N481β

DRAFT INTERNATIONAL **ISO/IEC**
STANDARD **CD 10967-3**

Working draft for the First edition
2002-07-10

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 languages —

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

EDITOR'S WORKING DRAFT
July 10, 2002 11:07

Editor:
Kent Karlsson
Department of Computing Sciences
Chalmers University of Technology
SE-412 96 Göteborg
SWEDEN
E-mail: kentk@cs.chalmers.se

Reference number
ISO/IEC CD 10967-3.1:2002(E)

Copyright notice

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialised system for world-wide 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 organisations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules in the ISO/IEC Directives, part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. 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 part of ISO/IEC 10967 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/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*

Additional parts will specify other arithmetic datatypes or arithmetic operations.

Annex A forms a normative part of this part of ISO/IEC 10967. Annexes B to E are for information only.

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 part is based on part 1 and part 2, and extends part 1's and part 2's specifications to specifications for 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 part 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 part 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.

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 defines the properties of numerical approximations for complex arithmetic operations and many of the complex elementary numerical functions available in standard libraries for 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 part. 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 part provides specifications for numerical functions for which operand or result values are of complex integer or complex floating point datatypes constructed from integer and floating point datatypes satisfying the requirements of part 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, is prescribed for each specified floating point operation.

This part provides specifications for:

a) Basic imaginary integer and complex integer operations.

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

c) Exponentiations, logarithms, and hyperbolics for imaginary floating point and Cartesian complex floating point.

d) Radian trigonometric operations for imaginary floating point and Cartesian complex floating point.

This part also provides specifications for:

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

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

1.2 Exclusions

This part 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 data. This part neither requires nor excludes such data or operations.

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

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

f) Matrix, statistical, or symbolic operations. This part neither requires nor excludes such data or operations.

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 part of ISO/IEC 10967 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 abstract datatypes, parameters, and operations specified in this part 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 part if and only if it conforms to the binding standard. In case of conflict between a binding standard and this part, the specifications of the binding standard takes precedence.

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

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

When no binding standard for a language and some datatypes or operations specified in this part exists, an implementation conforms to this part 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 part is always with respect to a specified set of datatypes and set of operations. Conformity to this part implies conformity to part 1 and part 2 for the integer and floating point datatypes and operations used.

An implementation is free to provide datatypes or operations that do not conform to this part, or that are beyond the scope of this part. The implementation shall not claim or imply conformity to this part with respect to such datatypes or operations.

An implementation is permitted to have modes of operation that do not conform to this part. 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 supply a binding if no binding standard exists. See annex C for suggested language bindings.

2 A complete binding for this part 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 part does not require a particular set of operations to be provided. It is not possible to conform to this part without specifying to which datatypes and set of operations (and modes of operation) conformity is claimed.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 10967. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 10967 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

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

ISO/IEC 10967-1:1994, *Information technology – Language independent arithmetic – Part 1: Integer and floating point arithmetic.*

NOTE 1 – See also Annex E of ISO/IEC 10967-2:2001.

ISO/IEC 10967-2:2001, *Information technology – Language independent arithmetic – Part 2: Elementary numerical functions.*

NOTE 2 – See also Annex E of this part.

4 Symbols and definitions

4.1 Symbols

4.1.1 Sets and intervals

In this part, $\mathcal{Z}$ 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.

In this part, 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 this part uses

$\Rightarrow$ and $\Leftrightarrow$ for logical implication and equivalence
$+$, $-$, $/$, $|x|$, and conj on complex values
$\cdot$ for multiplication on complex values
$<$, $\leqslant$, $\geqslant$, and $>$ between reals
$=$ and $\neq$ between complex as well as special values
$\cup$, $\cap$, $\times$, $\in$, $\notin$, $\subset$, $\subseteq$, $\nsubseteq$, $\neq$, and $=$ with sets
$\times$ for the Cartesian product of sets
$\rightarrow$ for a mapping between sets
$\tilde{\imath}$ as the imaginary unit
$\mathfrak{Re}$ to extract the real part of a complex value
$\mathfrak{Im}$ to extract the imaginary part of a complex value

4.1.3 Mathematical functions

This part 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* [43] (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 $x \in [0, \infty[$.

4.1.4 Datatypes and exceptional 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). 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. 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:

neg_I, add_I, sub_I, mul_I, eq_I, neq_I, lss_I, leq_I, gtr_I, and geq_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 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:

neg_F, add_F, sub_F, mul_F, div_F, eq_F, neq_F, lss_F, leq_F, gtr_F, and geq_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, $arcu_F$,
$sinh_F$, $cosh_F$, $tanh_F$, $coth_F$, $sech_F$, $csch_F$,
$arcsinh_F$, $arccosh_F$, $arctanh_F$, $arccoth_F$, $arcsech_F$, $arccsch_F$.

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,
$arcsin^*_F$, $arccos^*_F$, $arctan^*_F$, $arccot^*_F$, $arcsec^*_F$, $arccsc^*_F$,
$sinh^*_F$, $cosh^*_F$, $tanh^*_F$, $coth^*_F$, $sech^*_F$, $csch^*_F$,
$arcsinh^*_F$, $arccosh^*_F$, $arctanh^*_F$, $arccoth^*_F$, $arcsech^*_F$, $arccsch^*_F$.

Floating point datatypes that conform to part 1 shall, for use with this part, 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 $[-big\_angle\_r_F, big\_angle\_r_F]$. See clause 5.3.9 of part 2.

The following symbols represent special values defined in IEC 60559 and 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 $-\mathbf{0}$:

a) a 0 as the imaginary part of a complex argument (in c(F), see 4.1.5) shall be interpreted as if it was $-\mathbf{0}$ 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 4.1.5) shall be interpreted as if it was $-\mathbf{0}$ if and only if the imaginary part of the complex argument is less than zero.

NOTES

7 Reinterpreting 0 as $-\mathbf{0}$ 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 signed zeroes.

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 C99 requirements when zeroes don't have a distinguishable sign.

9 For consistency, this rule also has implications for the operations that implicitly or explicitly take out 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).

4.1.5 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 a imaginary datatype, constructed from the datatype corresponding to X. $\mathbf{\hat{\imath}\cdot}$ is a constructor that takes one parameter.

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

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.

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

NOTE – 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 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 i(X) nor c(X). Further, e.g., $4 + \tilde{\imath} \cdot 0 = 4$, but $4 \mathbf{+\hat{\imath}\cdot}\, 0 \neq 4$.

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 part, the following definitions apply:

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

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

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

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 $\mathcal{R}$ or an IEC 60559 special value. (Contrast with *exceptional value*. See 6.1.2 of part 1.)

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

error: (1) The difference between a computed value and the correct value. (Used in phrases like "rounding error" or "error bound".)

(2) A synonym for *exception* in phrases like "error message" or "error output". Error and exception are not synonyms in any other context.

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.

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.

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.

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.

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

monotonic approximation: An approximation helper function $h : ... \times S \times ... \rightarrow \mathcal{R}$, where the other arguments are kept constant, and where $S \subseteq \mathcal{R}$, is a monotonic approximation of a predetermined mathematical function $f : \mathcal{R} \rightarrow \mathcal{R}$ 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 $h(..., a, ...) \geqslant h(..., b, ...)$.

monotonic non-decreasing: A function $f : \mathcal{R} \rightarrow \mathcal{R}$ 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 $f(x) \leqslant f(y)$.

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 $f(x) \geqslant f(y)$.

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

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 for details.)

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.

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

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

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 one side of the neighbourhood of x_0, and $\lim\limits_{x \to x_0} f(x)$ is infinite.

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

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 5.2.6 of part 1). (See also A.5.2.6 of part 1 for some examples.)

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.

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.

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.

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.

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].)

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].)

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 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

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

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.

subnormal: The non-zero values of a floating point type F that provide less than the full precision allowed by that type. (See F_D in 5.2 of part 1 for a full definition.)

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 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 part, in part 1, or in part 2), to mathematical functions, and to helper functions (specified in this part, in part 1, or in part 2). They also use special abstract values (**−∞**, **+∞**, **−0**, **qNaN**, **sNaN**). For each datatype, two of these abstract values may represent several actual values each: **qNaN** and **sNaN**. 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:

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

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

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

The following signature is also valid

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

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.

$\mathrm{i}(I)$ (see clause 4.1.5) 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 4.1.5) 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:

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

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

NOTE – If one or both of the $result_I$ function (defined in part 2) 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. Similarly below (also for other exceptional values) for the specifications that do not use $result_{\mathrm{c}(I)}$ but specify the result parts directly.

5.1.2 Imaginary and complex integer operations

$itimes_{I\to \mathrm{i}(I)} : I \to \mathrm{i}(I)$
$itimes_{I\to \mathrm{i}(I)}(x) \ = \hat{\mathbf{i}}\cdot\, x$

$itimes_{\mathrm{i}(I)\to I} : \mathrm{i}(I) \to I \cup \{\mathbf{overflow}\}$
$itimes_{\mathrm{i}(I)\to I}(\hat{\mathbf{i}}\cdot\, y)$
$\qquad = neg_I(y)$

$itimes_{\mathrm{c}(I)} : \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\mathbf{overflow}\}$
$itimes_{\mathrm{c}(I)}(x \mathbf{+} \hat{\mathbf{i}}\cdot\, y)$
$\qquad = neg_I(y) \mathbf{+} \hat{\mathbf{i}}\cdot\, x$

$re_I : I \to I$
$re_I(x) \qquad = x \qquad \text{if } x \in I$

$re_{\mathrm{i}(I)} : \mathrm{i}(I) \to \{0\}$
$re_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y) \qquad = 0 \qquad \text{if } y \in I$

$re_{\mathrm{c}(I)} : \mathrm{c}(I) \to I$
$re_{\mathrm{c}(I)}(x \mathbf{+} \hat{\mathbf{i}}\cdot\, y) \quad = x \qquad \text{if } x \in I$

$im_I : I \to \{0\}$
$im_I(x) \qquad = 0 \qquad \text{if } x \in I$

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

$im_{\mathrm{c}(I)} : \mathrm{c}(I) \to I$
$im_{\mathrm{c}(I)}(x \mathbf{+} \hat{\mathbf{i}}\cdot\, y) \quad = y \qquad \text{if } y \in I$

$plusitimes_{\mathrm{c}(I)} : I \times I \to \mathrm{c}(I)$
$plusitimes_{\mathrm{c}(I)}(x, z)$
$\qquad = x \mathbf{+} \hat{\mathbf{i}}\cdot\, z$

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

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

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

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

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

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

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

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

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

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

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

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

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

$$sub_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \mathrm{i}(I) \cup \{\textbf{overflow}\}$$

$sub_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot y, \hat{\mathbf{i}}\cdot w) = \hat{\mathbf{i}}\cdot sub_I(y, w)$

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

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

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

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

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

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

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

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

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

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

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

$$mul_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \mathrm{c}(I) \cup \{\textbf{overflow}\}$$
$$mul_{\mathrm{c}(I),I}(x \mathbin{\textbf{+î·}} y, z) = mul_I(x, z) \mathbin{\textbf{+î·}} mul_I(y, z)$$

$$mul_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\textbf{overflow}\}$$
$$mul_{\mathrm{i}(I),\mathrm{c}(I)}(\textbf{î·}\, y, z \mathbin{\textbf{+î·}} w) = result_I(-(y \cdot w)) \mathbin{\textbf{+î·}} result_I(y \cdot z) \quad \text{if } y, z, w \in I$$

$$mul_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \mathrm{c}(I) \cup \{\textbf{overflow}\}$$
$$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x \mathbin{\textbf{+î·}} y, \textbf{î·}\, w) = result_I(-(y \cdot w)) \mathbin{\textbf{+î·}} result_I(x \cdot w) \quad \text{if } x, y, w \in I$$

$$mul_{\mathrm{c}(I)} : \mathrm{c}(I) \times \mathrm{c}(I) \to \mathrm{c}(I) \cup \{\textbf{overflow}\}$$
$$mul_{\mathrm{c}(I)}(x \mathbin{\textbf{+î·}} y, z \mathbin{\textbf{+î·}} w) = result_{\mathrm{c}(I)}((x + \tilde{\mathrm{i}} \cdot y) \cdot (z + \tilde{\mathrm{i}} \cdot w)) \quad \text{if } x, y, z, w \in I$$

$$eq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$$
$$eq_{\mathrm{i}(I)}(\textbf{î·}\, y, \textbf{î·}\, w) = eq_I(y, w)$$

$$eq_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \to \textbf{Boolean}$$
$$eq_{I,\mathrm{i}(I)}(x, \textbf{î·}\, w) = eq_{\mathrm{c}(I)}(x \mathbin{\textbf{+î·}} 0, 0 \mathbin{\textbf{+î·}} w)$$

$$eq_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \to \textbf{Boolean}$$
$$eq_{\mathrm{i}(I),I}(\textbf{î·}\, y, z) = eq_{\mathrm{c}(I)}(0 \mathbin{\textbf{+î·}} y, z \mathbin{\textbf{+î·}} 0)$$

$$eq_{I,\mathrm{c}(I)} : I \times \mathrm{c}(I) \to \textbf{Boolean}$$
$$eq_{I,\mathrm{c}(I)}(x, z \mathbin{\textbf{+î·}} w) = eq_{\mathrm{c}(I)}(x \mathbin{\textbf{+î·}} 0, z \mathbin{\textbf{+î·}} w)$$

$$eq_{\mathrm{c}(I),I} : \mathrm{c}(I) \times I \to \textbf{Boolean}$$
$$eq_{\mathrm{c}(I),I}(x \mathbin{\textbf{+î·}} y, z) = eq_{\mathrm{c}(I)}(x \mathbin{\textbf{+î·}} y, z \mathbin{\textbf{+î·}} 0)$$

$$eq_{\mathrm{i}(I),\mathrm{c}(I)} : \mathrm{i}(I) \times \mathrm{c}(I) \to \textbf{Boolean}$$
$$eq_{\mathrm{i}(I),\mathrm{c}(I)}(\textbf{î·}\, y, z \mathbin{\textbf{+î·}} w) = eq_{\mathrm{c}(I)}(0 \mathbin{\textbf{+î·}} y, z \mathbin{\textbf{+î·}} w)$$

$$eq_{\mathrm{c}(I),\mathrm{i}(I)} : \mathrm{c}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$$

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

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

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

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

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

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

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

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

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

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

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

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

$lss_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$
$lss_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y, \hat{\mathbf{i}}\cdot\, w) \;= lss_I(y, w)$

$leq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$
$leq_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y, \hat{\mathbf{i}}\cdot\, w) \;= leq_I(y, w)$

$gtr_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$
$gtr_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y, \hat{\mathbf{i}}\cdot\, w) \;= gtr_I(y, w)$

$geq_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$
$geq_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y, \hat{\mathbf{i}}\cdot\, w) = geq_I(y, w)$

$abs_{\mathrm{i}(I)} : \mathrm{i}(I) \to I$
$abs_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y) \quad = abs_I(y)$

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

$signum_I(x)$	$= 1$	if ($x \in I$ and $x \geqslant 0$) or $x = \boldsymbol{+\infty}$
	$= -1$	if ($x \in I$ and $x < 0$) or $x = \boldsymbol{-\infty}$
	$= \textbf{invalid}$	otherwise

$signum_{\mathrm{i}(I)} : \mathrm{i}(I) \to \{\hat{\mathbf{i}}\cdot\, (-1), \hat{\mathbf{i}}\cdot\, 1\}$
$signum_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y) \;= \hat{\mathbf{i}}\cdot\, signum_I(y)$

$divides_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \textbf{Boolean}$
$divides_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y, \hat{\mathbf{i}}\cdot\, w)$
$\qquad = divides_I(y, w)$

$divides_{I,\mathrm{i}(I)} : I \times \mathrm{i}(I) \to \textbf{Boolean}$
$divides_{I,\mathrm{i}(I)}(x, \hat{\mathbf{i}}\cdot\, w)$
$\qquad = divides_I(x, w)$

$divides_{\mathrm{i}(I),I} : \mathrm{i}(I) \times I \to \textbf{Boolean}$
$divides_{\mathrm{i}(I),I}(\hat{\mathbf{i}}\cdot\, y, z)$
$\qquad = divides_I(y, z)$

$max_{\mathrm{i}(I)} : \mathrm{i}(I) \times \mathrm{i}(I) \to \mathrm{i}(I)$
$max_{\mathrm{i}(I)}(\hat{\mathbf{i}}\cdot\, y, \hat{\mathbf{i}}\cdot\, w)$
$\qquad = \hat{\mathbf{i}}\cdot\, (max_I(y, w))$

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

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

$$max\_seq_{\mathrm{i}(I)} : [\mathrm{i}(I)] \to \mathrm{i}(I) \cup \{\mathbf{infinitary}\}$$
$$max\_seq_{\mathrm{i}(I)}([\hat{\mathbf{i}}\cdot y_1, ..., \hat{\mathbf{i}}\cdot y_n]) = \hat{\mathbf{i}}\cdot (max\_seq_I([y_1, ..., y_n]))$$

$$min\_seq_{\mathrm{i}(I)} : [\mathrm{i}(I)] \to \mathrm{i}(I) \cup \{\mathbf{infinitary}\}$$
$$min\_seq_{\mathrm{i}(I)}([\hat{\mathbf{i}}\cdot y_1, ..., \hat{\mathbf{i}}\cdot y_n]) = \hat{\mathbf{i}}\cdot (min\_seq_I([y_1, ..., y_n]))$$

5.2 Imaginary and complex floating point datatypes and operations

Clause 5.2 of part 1 and clause 5.2 of part 2 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.

A 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. 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 4.1.5) 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 4.1.5) 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. Some operations are approximate, with maximum error requirements implied by their exact relationships with operations defined in part 1 or part 2 (or, for the complex hyperbolic operations, 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 and non-zero results. The maximum error parameters also describe the maximum *absolute* error, in ulps, for $-fminN_F$, $fminN_F$, subnormal, or zero 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 $h_{c(F)}$, approximating f, is $max\_error\_op_{c(F)}$, then for all arguments $z, ... \in \mathcal{C}_F \times ...$ the following equations shall hold:

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

Some operations have a **Boolean** parameter $box\_error\_mode\_op_{c(F)}$. 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:

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

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 an 'rectangular' error bound, that can only span over a zero (in either dimension) very close to 0. The box error requirement results in a 'square-formed' error bound, that for cancellation cases can span over 0 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 requirement shall hold:

a) The approximation helper functions shall be zero exactly at the points where the approximated mathematical function is exactly zero.

b) At points where the approximation helper functions are not zero, they shall have the same sign as 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 part, each approximation helper function shall be a monotonic approximation 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; $big\_angle\_r_F$ is specified in part 2).

NOTES

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

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

5.2.4 The complex floating point *result* helper functions

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

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

NOTES

1 **overflow** and **underflow** can both occur for a single application of a complex operation.

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

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

$$no\_result_{\mathrm{c}(F)} : \mathrm{c}(F) \to \{\mathbf{invalid}\}$$

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

$$no\_result_{\mathrm{i}(F)} : \mathrm{i}(F) \to \{\mathbf{invalid}\}$$

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

$$no\_result_{F\to\mathrm{c}(F)} : F \to \{\mathbf{invalid}\}$$

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

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

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

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

NOTES

3 The handling of other special values, if available, is left unspecified by this part.

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

5.2.5 Basic arithmetic for complex floating point

$$
\begin{aligned}
&\mathit{itimes}_{F\to \mathrm{i}(F)} : F \to \mathrm{i}(F) \\
&\mathit{itimes}_{F\to \mathrm{i}(F)}(x) = \mathbf{\hat{\imath}\cdot}\ x
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{itimes}_{\mathrm{i}(F)\to F} : \mathrm{i}(F) \to F \cup \{-\mathbf{0}\} \\
&\mathit{itimes}_{\mathrm{i}(F)\to F}(\mathbf{\hat{\imath}\cdot}\ y) \\
&\qquad = \mathit{neg}_F(y)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{itimes}_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F \cup \{-\mathbf{0}\}) \\
&\mathit{itimes}_{\mathrm{c}(F)}(x \mathbin{\mathbf{+\hat{\imath}\cdot}} y) \\
&\qquad = \mathit{neg}_F(y) \mathbin{\mathbf{+\hat{\imath}\cdot}} x
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{re}_F : F \to F \\
&\mathit{re}_F(x) \qquad = x \qquad\qquad \text{if } x \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \\
&\qquad\qquad = \mathit{no\_result}_F(x) \qquad \text{otherwise}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{re}_{\mathrm{i}(F)} : \mathrm{i}(F) \to \{-\mathbf{0}, 0\} \\
&\mathit{re}_{\mathrm{i}(F)}(\mathbf{\hat{\imath}\cdot}\ y) \qquad = 0 \qquad\qquad \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = +\infty \\
&\qquad\qquad = -\mathbf{0} \qquad\qquad \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{-\infty, -\mathbf{0}\} \\
&\qquad\qquad = \mathit{no\_result}_F(y) \qquad \text{otherwise}
\end{aligned}
$$

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

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

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

$im_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) = y$ if $y \in F \cup \{-\infty, -\mathbf{0}, +\infty\}$

$= no\_result_F(y)$ otherwise

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

$plusitimes_{\mathrm{c}(F)}(x, z) = x \mathbin{+\hat{\imath}\cdot} z$

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

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

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

$neg_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) = neg_F(x) \mathbin{+\hat{\imath}\cdot} neg_F(y)$

$conj_F : F \to F$

$conj_F(x) = x$

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

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

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

$conj_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) = x \mathbin{+\hat{\imath}\cdot} neg_F(y)$

$add_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathrm{i}(F) \cup \{(\mathbf{underflow}), \mathbf{overflow}\}$

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

$add_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \to \mathrm{c}(F)$
$add_{F,\mathrm{i}(F)}(x, \hat{\mathbf{i}}\cdot\, w)$
$\qquad = x \mathbin{+\!\!\!\cdot} w$

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

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

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

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

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

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

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

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

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

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

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

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

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

$sub_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F) \cup \{(\mathbf{underflow}), \mathbf{overflow}\}$
$sub_{\mathrm{c}(F)}(x, z) \qquad = add_{\mathrm{c}(F)}(x, neg_{\mathrm{c}(F)}(z))$

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

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

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

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

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

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

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

$mul_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot\, w)$
$\qquad = neg_F(mul_F(y,w)) \mathbin{+\hat{\imath}\cdot} mul_F(x,w)$
NOTE 1 – $mul_{\mathrm{c}(F)}$ is specified in clause 5.2.6.

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

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

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

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

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

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

$div_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$div_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot\, w)$
$\qquad = div_F(y,w) \mathbin{+\hat{\imath}\cdot} neg_F(div_F(x,w))$
NOTE 2 – $div_{\mathrm{c}(F)}$ is specified in clause 5.2.6.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$abs_{\mathrm{c}(F)} : \mathrm{c}(F) \to F \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$abs_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y) = hypot_F(x, y)$

$phase_F : F \to F$
$phase_F(x) = arc_F(x, im_F(x))$

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 E 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.

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

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

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

$$phaseu_F : F \times F \to F \cup \{\mathbf{invalid}\}$$
$$phaseu_F(u, x) = arcu_F(u, x, im_F(x))$$

$$phaseu_{\mathrm{i}(F)} : F \times \mathrm{i}(F) \to F \cup \{\mathbf{invalid}\}$$
$$phaseu_{\mathrm{i}(F)}(u, \hat{\imath}\cdot y) = arcu_F(u, re_{\mathrm{i}(F)}(\hat{\imath}\cdot y), y)$$

$$phaseu_{\mathrm{c}(F)} : F \times \mathrm{c}(F) \to F \cup \{\mathbf{underflow}, \mathbf{invalid}\}$$
$$phaseu_{\mathrm{c}(F)}(u, x \mathbin{+\hat{\imath}\cdot} y) = arcu_F(u, x, y)$$

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

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

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

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

5.2.6 Complex sign, multiplication, and division

There shall be a box mode error parameter each for multiplication and division 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 $\leqslant 15$.

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

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

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 $x, y \in \mathcal{R}$.

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

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

$signum^*_{c(F)}(z)$ 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 $signum^*_{c(F)}$ approximation helper function are:

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

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

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

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

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

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

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

$signum_{c(F)}(x \mathbf{+\hat{\imath}\cdot} y)$

$= result^*_{c(F)}(signum^*_{c(F)}(x + \tilde{\imath} \cdot y), nearest_F)$ if $x, y \in F$ and $x + \tilde{\imath} \cdot y \neq 0$

$= conj_{c(F)}(signum_{c(F)}(conj_{c(F)}(x \mathbf{+\hat{\imath}\cdot} y)))$ if $y \in \{\mathbf{-\infty}, \mathbf{-0}\}$

$= neg_{c(F)}(signum_{c(F)}(neg_{c(F)}(x \mathbf{+\hat{\imath}\cdot} y)))$ if $x \in \{\mathbf{-\infty}, \mathbf{-0}\}$ and $y \notin \{\mathbf{-\infty}, \mathbf{-0}\}$

$= sin_F(arc_F(y, x)) \mathbf{+\hat{\imath}\cdot} sin_F(arc_F(x, y))$ otherwise

NOTE 3 – $signum_{c(F)}(x \mathbf{+\hat{\imath}\cdot} y) \approx cos_F(arc_F(x, y)) \mathbf{+\hat{\imath}\cdot} sin_F(arc_F(x, y))$. The specification is more complicated, in order to allow higher accuracy, including the sign of zero result parts.

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

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

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

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

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

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

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

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

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

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

NOTE 4 – NaN (and **invalid**) is not avoided in the "otherwise" case here. Note in particular cases like $mul_{\mathrm{c}(F)}(2 \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{-0}), 3 \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{+\infty}))$ which is **invalid** with a continuation value of $\mathbf{qNaN} \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{+\infty})$. However, $mul_{F,\mathrm{c}(F)}(2, 3 \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{+\infty}))$ returns $6 \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{+\infty})$, due to the strong implicit zero (see B.5.2).

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

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

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

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

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

5.2.7 Operations for conversion from polar to Cartesian

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

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

$polar^*_{\mathrm{c}(F)}(x, z)$ 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 requirement on the $polar^*_{\mathrm{c}(F)}$ approximation helper function are:

$$
\begin{aligned}
&polar^*_{\mathrm{c}(F)}(-x, z) = -polar^*_{\mathrm{c}(F)}(x, z) \\
&polar^*_{\mathrm{c}(F)}(x, -z) = \mathrm{conj}(polar^*_{\mathrm{c}(F)}(x, z))
\end{aligned}
$$

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

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

$$\begin{aligned} polar_{\mathrm{c}(F)}(x,z) &= result^*_{\mathrm{c}(F)}(polar^*_{\mathrm{c}(F)}(x,z), nearest_F) \\ &\qquad \text{if } x,z \in F \text{ and } |z| < big\_angle\_r_F \text{ and } x \neq 0 \text{ and } z \neq 0 \\ &= mul_F(x, cos_F(z)) \mathbin{\mathbf{+\hat{\imath}\cdot}} mul_F(x, sin_F(z)) \\ &\qquad \text{otherwise} \end{aligned}$$

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

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

$polaru^*_{\mathrm{c}(F)}(u,x,z)$ 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 requirement on the $polaru^*_{\mathrm{c}(F)}$ approximation helper function are:

$$\begin{aligned} polaru^*_{\mathrm{c}(F)}(u,-x,z) &= -polaru^*_{\mathrm{c}(F)}(u,x,z) && \text{if } u \neq 0 \\ polaru^*_{\mathrm{c}(F)}(u,x,-z) &= \mathrm{conj}(polaru^*_{\mathrm{c}(F)}(u,x,z)) && \text{if } u \neq 0 \\ polaru^*_{\mathrm{c}(F)}(-u,x,z) &= polaru^*_{\mathrm{c}(F)}(u,x,-z) && \text{if } u \neq 0 \end{aligned}$$

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

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

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

5.3 Elementary transcendental imaginary and complex floating point operations

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

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 $\leqslant 2{\cdot}rnd\_error_F$. The $max\_error\_power_{\mathrm{c}(F)}$ parameter shall have a value that is $\leqslant 7$.

5.3.1.1 Exponentiation of imaginary base to integer power

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

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

$$
\begin{aligned}
power_{\mathrm{i}(F),I}(\hat{\mathbf{i}}{\cdot}\, x, y) & \\
&= re_{\mathrm{i}(F)}(\hat{\mathbf{i}}{\cdot}\, x) \mathbin{+\hat{\mathbf{i}}{\cdot}}\, power_{F,I}(x, y) \\
&\qquad\qquad \text{if } y \in I \text{ and } 4|(y+3) \\
&= neg_F(power_{F,I}(abs_F(x), y)) \mathbin{+\hat{\mathbf{i}}{\cdot}}\, 0 \\
&\qquad\qquad \text{if } y \in I \text{ and } 4|(y+2) \\
&= neg_{\mathrm{c}(F)}(re_{\mathrm{i}(F)}(\hat{\mathbf{i}}{\cdot}\, x) \mathbin{+\hat{\mathbf{i}}{\cdot}}\, power_{F,I}(x, y)) \\
&\qquad\qquad \text{if } y \in I \text{ and } 4|(y+1) \\
&= power_{F,I}(abs_F(x), y) \mathbin{+\hat{\mathbf{i}}{\cdot}}\, (-\mathbf{0}) \\
&\qquad\qquad \text{if } y \in I \text{ and } 4|y
\end{aligned}
$$

5.3.1.2 Natural exponentiation

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

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

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

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

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:

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

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

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no cos_F, sin_F, or exp_F operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

NOTES

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

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

5.3.1.3 Complex exponentiation of argument base

$$power_{F\to\mathrm{c}(F)} : F \times F \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$
$$power_{F\to\mathrm{c}(F)}(x, z) = power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} im_F(x), z \mathbin{+\hat{\imath}\cdot} im_F(z))$$

$$power_{\mathrm{i}(F)} : \mathrm{i}(F) \times \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$
$$power_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) = power_{\mathrm{c}(F)}(im_F(y) \mathbin{+\hat{\imath}\cdot} y, im_F(w) \mathbin{+\hat{\imath}\cdot} w)$$

$$power_{\mathrm{i}(F),F} : \mathrm{i}(F) \times F \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$
$$power_{\mathrm{i}(F),F}(\hat{\imath}\cdot y, z) = power_{\mathrm{c}(F)}(im_F(y) \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} im_F(z))$$

$$power_{F,\mathrm{i}(F)} : F \times \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$$
$$power_{F,\mathrm{i}(F)}(x, \hat{\imath}\cdot w) = power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} im_F(x), im_F(w) \mathbin{+\hat{\imath}\cdot} w)$$

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

$power_{\mathrm{c}(F),F}(x \mathbin{+\hat{\imath}\cdot} y, z)$
$= power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} im_F(z))$

$power_{F,\mathrm{c}(F)} : F \times \mathrm{c}(F) \to \mathrm{c}(F) \cup$
$\{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$power_{F,\mathrm{c}(F)}(x, z \mathbin{+\hat{\imath}\cdot} w)$
$= power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} im_F(x), z \mathbin{+\hat{\imath}\cdot} w)$

$power_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \to \mathrm{c}(F) \cup$
$\{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$power_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot y, z \mathbin{+\hat{\imath}\cdot} w)$
$= power_{\mathrm{c}(F)}(im_F(y) \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$power_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \to \mathrm{c}(F) \cup$
$\{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$power_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$= power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, im_F(w) \mathbin{+\hat{\imath}\cdot} w)$

EDITOR'S NOTE – Most (all) of the above just refer to $power_{\mathrm{c}(F)}$ (as yet). Possible exceptions are $power_{F\to\mathrm{c}(F)}$, which maps the real axis to mirrored angled lines (+-0 dependent); $power_{\mathrm{c}(F),F}$ which maps the complex plane to a "folding fan"; $power_{\mathrm{i}(F),F}$ which maps the imaginary axis to a folded line; and $power_{F,\mathrm{i}(F)}$ which maps the real axis to the unit circle.

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

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

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

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

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

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

$power_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F) \cup$
$\{\mathbf{underflow}, \mathbf{overflow}, \mathbf{absolute\_precision\_underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$
$= result^*_{\mathrm{c}(F)}(power^*_{\mathrm{c}(F)}(x + \tilde{\imath}\cdot y, z + \tilde{\imath}\cdot w), nearest_F)$
if $x, y, z, w \in F$ and ($x \neq 0$ or $y \neq 0$) and
EDITOR'S NOTE – $|\ln(\sqrt{x^2+y^2})\cdot w + arc(x,y)\cdot z|$ is not too large...???
$= power_{\mathrm{c}(F)}(0 \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$
if $x = \mathbf{-0}$
$= conj_{\mathrm{c}(F)}(power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0, z \mathbin{+\hat{\imath}\cdot} neg_F(w)))$
if $y = \mathbf{-0}$ and $x \neq \mathbf{-0}$
$= power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, 0 \mathbin{+\hat{\imath}\cdot} w)$
if $z = \mathbf{-0}$ and $x \neq \mathbf{-0}$ and $y \neq \mathbf{-0}$
$= conj_{\mathrm{c}(F)}(power_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} neg_F(y), z \mathbin{+\hat{\imath}\cdot} 0))$

$$
\begin{array}{ll}
 & \text{if } w = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \text{ and } y \neq \mathbf{-0} \text{ and } z \neq \mathbf{-0} \\
= 0 \mathbf{+\hat{\imath}\cdot} 0 & \text{if } x = 0 \text{ and } y = 0 \text{ and } z \in F \text{ and } z > 0 \\
= \mathbf{infinitary}() & \text{if } x = 0 \text{ and } y = 0 \text{ and } z \in F \text{ and } z < 0 \\
= exp_{\mathrm{c}(F)}(mul_{\mathrm{c}(F)}(ln_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y), z \mathbf{+\hat{\imath}\cdot} w)) & \\
 & \text{otherwise}
\end{array}
$$

EDITOR'S NOTE – find a way to terminate the infinite recursion for the power(x+i(-0),z+i0) and power(x+i0,z+i(-0)) cases!!!

NOTE – Complex raising to a power is multi-valued. The principal result is given by $b^q = e^{q \cdot \ln(b)}$. The b^q function branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } x < 0\} \times \mathcal{C}$ (except when q is in $\mathcal{Z}$). Thus $power_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} 0, z) \neq power_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} (\mathbf{-0}), z)$.

5.3.1.4 Complex square root

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

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

$$
\begin{array}{ll}
sqrt_{F \to \mathrm{c}(F)}(x) & \\
\quad = 0 \mathbf{+\hat{\imath}\cdot} sqrt_F(neg_F(x)) & \text{if } (x \in F \text{ and } x < 0) \text{ or } x \in \{\mathbf{-\infty}, \mathbf{-0}\} \\
\quad = sqrt_F(x) \mathbf{+\hat{\imath}\cdot} (\mathbf{-0}) & \text{if } (x \in F \text{ and } x \geqslant 0) \text{ or } x = \mathbf{+\infty} \\
\quad = no\_result_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} im_F(x)) & \\
 & \text{otherwise}
\end{array}
$$

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

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

$sqrt_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot} y)$

$$= sqrt_{\mathrm{c}(F)}(re_{\mathrm{i}(F)}(y) \mathbf{+\hat{\imath}\cdot} y)$$

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:

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

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

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

$sqrt_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y)$

$$
\begin{array}{ll}
\quad = result^*_{\mathrm{c}(F)}(sqrt^*_{\mathrm{c}(F)}(x + \tilde{\imath} \cdot y), nearest_F) & \\
 & \text{if } x \mathbf{+\hat{\imath}\cdot} y \in \mathrm{c}(F) \\
\quad = sqrt_{\mathrm{c}(F)}(0 \mathbf{+\hat{\imath}\cdot} y) & \text{if } x = \mathbf{-0} \\
\quad = conj_{\mathrm{c}(F)}(sqrt_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} 0)) & \\
 & \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y = \mathbf{-0} \\
\quad = (\mathbf{+\infty}) \mathbf{+\hat{\imath}\cdot} (\mathbf{+\infty}) & \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \text{ and } y = \mathbf{+\infty} \\
\quad = (\mathbf{+\infty}) \mathbf{+\hat{\imath}\cdot} 0 & \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } x \geqslant 0 \\
\quad = (\mathbf{+\infty}) \mathbf{+\hat{\imath}\cdot} (\mathbf{-0}) & \text{if } x = \mathbf{+\infty} \text{ and } ((y \in F \text{ and } y < 0 \text{ or } y = \mathbf{-0})
\end{array}
$$

$= (+\infty) +\hat{\imath}\cdot (-\infty)$ if $x \in F \cup \{-\infty, -\mathbf{0}, +\infty\}$ and $y = -\infty$
$= 0 +\hat{\imath}\cdot (+\infty)$ if $x = -\infty$ and $y \in F$ and $x \geqslant 0$
$= 0 +\hat{\imath}\cdot (-\infty)$ if $x = -\infty$ and $((y \in F$ and $y < 0$ or $y = -\mathbf{0})$
$= no\_result_{\mathrm{c}(F)}(x +\hat{\imath}\cdot y)$ otherwise

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 $\{x \mid x \in \mathcal{R}$ and $x < 0\}$. Thus $sqrt_{\mathrm{c}(F)}(x+\hat{\imath}\cdot 0) \neq sqrt_{\mathrm{c}(F)}(x +\hat{\imath}\cdot (-\mathbf{0}))$ when $x < 0$.

5.3.1.5 Natural logarithm

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

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

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

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 E.

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

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

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

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

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

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

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

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

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

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

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no arc_F or ln_F operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

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

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

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 principle value for the inverse) branch cuts at $\{x \mid x \in \mathcal{R} \text{ and } x < 0\}$, is continuous on the rest of $\mathcal{C}$, and $\ln(z) \in \mathcal{R}$ if $x \in \mathcal{R}$ and $x > 0$. Thus $ln_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0) \neq ln_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} (-\mathbf{0}))$ when $x < 0$.

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

5.3.1.6 Argument base logarithm

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

$$
\begin{aligned}
&logbase_{F \to \mathrm{c}(F)}(x, z) \\
&\qquad = logbase_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} im_F(x), z \mathbin{+\hat{\imath}\cdot} im_F(z))
\end{aligned}
$$

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

$$
\begin{aligned}
&logbase_{\mathrm{i}(F)}(\hat{\imath}\cdot y, \hat{\imath}\cdot w) \\
&\qquad = logbase_{\mathrm{c}(F)}(im_F(y) \mathbin{+\hat{\imath}\cdot} y, im_F(w) \mathbin{+\hat{\imath}\cdot} w)
\end{aligned}
$$

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

$$
\begin{aligned}
&logbase_{\mathrm{i}(F),F}(\hat{\imath}\cdot y, z) \\
&\qquad = logbase_{\mathrm{c}(F)}((im_F(y)) \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} im_F(z))
\end{aligned}
$$

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

$$
\begin{aligned}
&logbase_{F,\mathrm{i}(F)}(x, \hat{\imath}\cdot w) \\
&\qquad = logbase_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} im_F(x), im_F(w) \mathbin{+\hat{\imath}\cdot} w)
\end{aligned}
$$

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

$$
\begin{aligned}
&logbase_{\mathrm{c}(F),F}(x \mathbin{+\hat{\imath}\cdot} y, z) \\
&\qquad = logbase_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} im_F(z))
\end{aligned}
$$

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

$logbase_{\mathrm{i}(F),\mathrm{c}(F)} : \mathrm{i}(F) \times \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(\hat{\imath}\cdot y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = logbase_{\mathrm{c}(F)}(im_F(y) \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$

$logbase_{\mathrm{c}(F),\mathrm{i}(F)} : \mathrm{c}(F) \times \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{invalid}\}$
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(x \mathbin{+\hat{\imath}\cdot} y, \hat{\imath}\cdot w)$
$\quad = logbase_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, im_F(w) \mathbin{+\hat{\imath}\cdot} w)$

EDITOR'S NOTE – Most (all?) of the above will just refer to $logbase_{\mathrm{c}(F)}$ as yet...

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

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

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

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

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

$logbase^*_{\mathrm{c}(F)}(x + \tilde{\imath}\cdot y, 0 + \tilde{\imath}\cdot w) = 0$
if $x, y, w \in F$ and $x \neq 0$

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

$logbase_{\mathrm{c}(F)} : \mathrm{c}(F) \times \mathrm{c}(F) \to \mathrm{c}(F \cup \{-\mathbf{0}\}) \cup \{\mathbf{infinitary}, \mathbf{invalid}\}$
$logbase_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w)$
$\quad = result^*_{\mathrm{c}(F)}(logbase^*_{\mathrm{c}(F)}(x + \tilde{\imath}\cdot y, z + \tilde{\imath}\cdot w), nearest_F)$
if $x \mathbin{+\hat{\imath}\cdot} y, z \mathbin{+\hat{\imath}\cdot} w \in \mathrm{c}(F)$ and $x \neq 0$
$\quad = div_{\mathrm{c}(F)}(ln_{\mathrm{c}(F)}(z \mathbin{+\hat{\imath}\cdot} w), ln_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y))$
otherwise

NOTE – Complex logarithm with argument base is multi-valued. The principal result is given by $\log_b(q) = \ln(q)/\ln(b)$. Apart from the poles, the $\log_b(q)$ function branch cuts at $(\{x \mid x \in \mathcal{R} \text{ and } x < 0\} \times \mathcal{C}) \cup (\mathcal{C} \times \{x \mid x \in \mathcal{R} \text{ and } x < 0\})$.

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

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

$$rad_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{absolute\_precision\_underflow}\}$$
$$\begin{aligned} rad_{\mathrm{c}(F)}(x + \hat{\mathbf{\imath}}\cdot y) &= rad_F(x) + \hat{\mathbf{\imath}}\cdot y && \text{if } y \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \text{ and } rad_F(x) \in F \cup \{-\mathbf{0}\} \\ &= \mathbf{absolute\_precision\_underflow}(\mathbf{qNaN} + \hat{\mathbf{\imath}}\cdot \mathbf{qNaN}) \\ & && \text{if } y \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \text{ and} \\ & && \quad rad_F(x) = \mathbf{absolute\_precision\_underflow} \\ &= no\_result_{\mathrm{c}(F)}(x + \hat{\mathbf{\imath}}\cdot y) && \text{otherwise} \end{aligned}$$

5.3.2.2 Radian sine

NOTE – TEMP
$\sin(-z) = -\sin(z)$
$\sin(\mathrm{conj}(z)) = \mathrm{conj}(\sin(z))$
$\sin(z + k \cdot 2 \cdot \pi) = \sin(z)$ if $k \in \mathcal{Z}$
$\sin(z) = \cos(\pi/2 - z)$

$\sin(x) = -\tilde{\imath} \cdot \sinh(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sinh(-\tilde{\imath} \cdot x)$
$\sin(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sinh(-y) = \tilde{\imath} \cdot \sinh(y)$
$\sin(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sinh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sinh(y - \tilde{\imath} \cdot x)$

$\sin(x + \tilde{\imath} \cdot y) = \sin(x) \cdot \cosh(y) + \tilde{\imath} \cdot \cos(x) \cdot \sinh(y)$

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

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

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:

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

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

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no sin_F or $sinh_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

5.3.2.3 Radian cosine

NOTE – TEMP
$\cos(-z) = \cos(z)$
$\cos(\mathrm{conj}(z)) = \mathrm{conj}(\cos(z))$
$\cos(z + k \cdot 2 \cdot \pi) = \cos(z)$ if $k \in \mathcal{Z}$
$\cos(z) = \sin(\pi/2 - z)$

$\cos(x) = \cosh(\tilde{\imath} \cdot x) = \cosh(-\tilde{\imath} \cdot x)$
$\cos(\tilde{\imath} \cdot y) = \cosh(y)$
$\cos(x + \tilde{\imath} \cdot y) = \cosh(-y + \tilde{\imath} \cdot x) = \cosh(y - \tilde{\imath} \cdot x)$

$\cos(x + \tilde{\imath} \cdot y) = \cos(x) \cdot \cosh(y) + \tilde{\imath} \cdot \sin(x) \cdot \sinh(y)$

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

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

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

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

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

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

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

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

The relationship 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:

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no cos_F or $cosh_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

5.3.2.4 Radian tangent

NOTE – TEMP
$\tan(-z) = -\tan(z)$
$\tan(\mathrm{conj}(z)) = \mathrm{conj}(\tan(z))$
$\tan(z + k \cdot 2 \cdot \pi) = \tan(z)$ if $k \in \mathcal{Z}$
$\tan(z) = \cot(\pi/2 - z)$

$\tan(x) = -\tilde{\imath} \cdot \tanh(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \tanh(-\tilde{\imath} \cdot x)$
$\tan(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \tanh(-y) = \tilde{\imath} \cdot \tanh(y)$
$\tan(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \tanh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \tanh(y - \tilde{\imath} \cdot x)$

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

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

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

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

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

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

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

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

The relationship 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:

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no tan_F or $tanh_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

$$
\begin{aligned}
tan_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) &= result^*_{\mathrm{c}(F)}(tan^*_{\mathrm{c}(F)}(x + \tilde{\imath} \cdot y), nearest_F) \\
&\qquad \text{if } x, y \in F \text{ and } |x| \leqslant big\_angle\_r_F \\
&= conj_{\mathrm{c}(F)}(tan_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0)) \\
&\qquad \text{if } y = \mathbf{-0}
\end{aligned}
$$

$$
\begin{aligned}
&= neg_F(tan_{\mathrm{c}(F)}(0 \mathbin{+\hat{\imath}\cdot} neg_F(y))) \\
&\qquad\qquad \text{if } x = \mathbf{-0} \text{ and } y \neq \mathbf{-0} \\
&= mul_F(tan_F(x), 0) \mathbin{+\hat{\imath}\cdot} tanh_F(y) \\
&\qquad\qquad \text{if } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \\
&= rad_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \qquad \text{otherwise}
\end{aligned}
$$

5.3.2.5 Radian cotangent

NOTE – TEMP

$\cot(-z) = -\cot(z)$
$\cot(\mathrm{conj}(z)) = \mathrm{conj}(\cot(z))$
$\cot(z + k \cdot 2 \cdot \pi) = \coth(z)$ if $k \in \mathcal{Z}$
$\cot(z) = \tan(\pi/2 - z)$
$\cot(z) = 1/\tan(z)$

$\cot(x) = \tilde{\imath} \cdot \coth(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \coth(-\tilde{\imath} \cdot x)$
$\cot(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \coth(-y) = -\tilde{\imath} \cdot \coth(y)$
$\cot(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \coth(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \coth(y - \tilde{\imath} \cdot x)$

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

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

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

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

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

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

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

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

The relationship 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:

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no cot_F or $coth_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

$$
\begin{aligned}
& & & \text{if } x = \mathbf{-0} \text{ and } y \neq \mathbf{-0} \\
&= mul_F(tan_F(x), 0) \mathbf{+\hat{\imath}\cdot} neg_F(tanh_F(y)) & & \\
& & & \text{if } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \\
&= rad_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y) & & \text{otherwise}
\end{aligned}
$$

5.3.2.6 Radian secant

NOTE – TEMP
$\mathrm{sec}(-z) = \mathrm{sec}(z)$
$\mathrm{sec}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{sec}(z))$
$\mathrm{sec}(z + k \cdot 2 \cdot \pi) = \mathrm{sec}(z)$ if $k \in \mathcal{Z}$
$\mathrm{sec}(z) = \mathrm{csc}(\pi/2 - z)$
$\mathrm{sec}(z) = 1/\cos(z)$

$\mathrm{sec}(x) = \mathrm{sech}(\tilde{\imath} \cdot x) = \mathrm{sech}(-\tilde{\imath} \cdot x)$
$\mathrm{sec}(\tilde{\imath} \cdot y) = \mathrm{sech}(-y) = \mathrm{sech}(y)$
$\mathrm{sec}(x + \tilde{\imath} \cdot y) = \mathrm{sech}(-y + \tilde{\imath} \cdot x) = \mathrm{sech}(y - \tilde{\imath} \cdot x)$

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

$sec_{\mathrm{i}(F)} : \mathrm{i}(F) \to F \cup \{\mathbf{underflow}\}$

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

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

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

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

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

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

The relationship 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:

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no sec_F or $sech_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

5.3.2.7 Radian cosecant

NOTE – TEMP
$\csc(-z) = -\csc(z)$
$\csc(\text{conj}(z)) = \text{conj}(\csc(z))$
$\csc(z + k \cdot 2 \cdot \pi) = \csc(z)$ if $k \in \mathcal{Z}$
$\csc(z) = \sec(\tilde{\imath} \cdot \pi/2 - z)$
$\csc(z) = 1/\sin(z)$

$\csc(x) = \tilde{\imath} \cdot \text{csch}(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \text{csch}(-\tilde{\imath} \cdot x)$
$\csc(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \text{csch}(-y) = -\tilde{\imath} \cdot \text{csch}(y)$
$\csc(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \text{csch}(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \text{csch}(y - \tilde{\imath} \cdot x)$

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

$csc_{\text{i}(F)} : \text{i}(F) \to \text{i}(F) \cup \{\textbf{underflow}, \textbf{overflow}, \textbf{infinitary}\}$
$csc_{\text{i}(F)}(\hat{\textbf{i}}\cdot\, y) \quad = neg_{\text{i}(F)}(\hat{\textbf{i}}\cdot\, csch_F(y))$

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

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

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

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

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

The relationship 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:

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

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no csc_F or $csch_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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

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

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

5.3.2.8 Radian arc sine

NOTE 1 – TEMP
$\arcsin(-z) = -\arcsin(z)$
$\arcsin(z) = \pi/2 - \arccos(z)$
$\arcsin(\mathrm{conj}(z)) = \mathrm{conj}(\arcsin(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \leqslant 1$
$Arcsin(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arcsinh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arcsinh(y - \tilde{\imath} \cdot x)$

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

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

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

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

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

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

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

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

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

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

$arcsin^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arcsin^*_{\mathrm{c}(F)}(z))$ if $z \in \mathcal{C}_F$ and ($\mathfrak{Re}(z) \leqslant 1$ or $\mathfrak{Im}(z) \neq 0$)
$arcsin^*_{\mathrm{c}(F)}(-z) = -arcsin^*_{\mathrm{c}(F)}(z)$ if $z \in \mathcal{C}_F$ and ($\mathfrak{Re}(z) \leqslant 1$ or $\mathfrak{Im}(z) \neq 0$)

The relationship 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:

$arcsin^*_{\mathrm{c}(F)}(x) = arcsin^*_F(x)$ if $x \in F$ and $|x| \leqslant 1$
$arcsin^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot arcsinh^*_F(y)$ if $y \in F$
$arcsin^*_{\mathrm{c}(F)}(x) = -\pi/2 + \tilde{\imath} \cdot arccosh^*_F(-x)$ if $x \in F$ and $x \leqslant -1$
$arcsin^*_{\mathrm{c}(F)}(x) = \pi/2 + \tilde{\imath} \cdot arccosh^*_F(x)$ if $x \in F$ and $x \geqslant 1$
$arcsin^*_{\mathrm{c}(F)}(x) = \pi/2 + \tilde{\imath} \cdot arccosh^*_F(x)$ if $x \in F$ and $x \geqslant 1$

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no $arcsin_F$, $arcsinh_F$, or $arccosh_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

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:

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

$$
\begin{aligned}
&arcsin_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y)\\
&\qquad = result^{*}_{\mathrm{c}(F)}(arcsin^{\#}_{\mathrm{c}(F)}(x + \tilde{\imath}\cdot y), nearest_F)\\
&\qquad\qquad \text{if } x, y \in F\\
&\qquad = neg_{\mathrm{c}(F)}(arcsin_{\mathrm{c}(F)}(0 \mathbin{+\hat{\imath}\cdot} neg_F(y))\\
&\qquad\qquad \text{if } x = \mathbf{-0}\\
&\qquad = conj_{\mathrm{c}(F)}(arcsin_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0))\\
&\qquad\qquad \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0}\\
&\qquad = arc_F(abs_F(y), x) \mathbin{+\hat{\imath}\cdot} mul_F(signum_F(y), \mathbf{+\infty}))\\
&\qquad\qquad \text{if } (x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\}) \text{ or}\\
&\qquad\qquad\quad (y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } x \in F)\\
&\qquad = no\_result_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \quad \text{otherwise}
\end{aligned}
$$

NOTE 2 – 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 $arcsin_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0) \neq arcsin_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}))$ when $|x| > 1$.

5.3.2.9 Radian arc cosine

NOTE 1 – TEMP
$\arccos(-z) = \tilde{\imath} \cdot \pi - \arccos(z)$
$\arccos(\mathrm{conj}(z)) = \mathrm{conj}(\arccos(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \leqslant 1$
$\arccos(z) = \pi/2 - \arcsin(z)$
$Arccos(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arccosh(x + \tilde{\imath} \cdot y)$

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

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

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

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

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

$$
\begin{aligned}
&arccos_{\mathrm{i}(F)\to\mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot} y)\\
&\qquad = up_F(\pi/2) \mathbin{+\hat{\imath}\cdot} neg_F(arcsinh_F(y))\\
&\qquad\qquad \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{\mathbf{-\infty}, \mathbf{-0}\}\\
&\qquad = down_F(\pi/2) \mathbin{+\hat{\imath}\cdot} neg_F(arcsinh_F(y))\\
&\qquad\qquad \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = \mathbf{+\infty}\\
&\qquad = no\_result_{\mathrm{i}(F)\to\mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot} y) \quad \text{otherwise}
\end{aligned}
$$

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:

$$
\begin{array}{ll}
arccos^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arccos^*_{\mathrm{c}(F)}(z)) & \text{if } z \in \mathcal{C}_F \text{ and } (|\mathfrak{Re}(z)| \leqslant 1 \text{ or } \mathfrak{Im}(z) \neq 0) \\
\mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(-z)) = -\mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z)) & \text{if } z \in \mathcal{C}_F \text{ and } (|\mathfrak{Re}(z)| \leqslant 1 \text{ or } \mathfrak{Im}(z) \neq 0) \\
\mathfrak{Re}(arccos^*_{\mathrm{c}(F)}(x)) = \pi & \text{if } x \in F \text{ and } x \leqslant -1 \\
\mathfrak{Re}(arccos^*_{\mathrm{c}(F)}(x)) = 0 & \text{if } x \in F \text{ and } x \geqslant 1 \\
\mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z)) \geqslant 0 & \text{if } z \in \mathcal{C}_F \text{ and } \mathfrak{Im}(z) \leqslant 0
\end{array}
$$

The relationship to the $arccos^*_F$ and $arccosh^*_F$ approximation helper functions for $arccos_F$ and $arccosh_F$ operations in an associated library for real-valued operations shall be:

$$
\begin{array}{ll}
arccos^*_{\mathrm{c}(F)}(x) = arccos^*_F(x) & \text{if } x \in F \text{ and } |x| \leqslant 1 \\
arccos^*_{\mathrm{c}(F)}(x) = -\tilde{\imath} \cdot arccosh^*_F(x) & \text{if } x \in F \text{ and } x \geqslant 1 \\
arccos^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y) = \pi/2 - \tilde{\imath} \cdot arcsinh^*_F(y) & \text{if } y \in F \\
arccos^*_{\mathrm{c}(F)}(x) = \pi - \tilde{\imath} \cdot arccosh^*_F(-x) & \text{if } x \in F \text{ and } x \leqslant -1
\end{array}
$$

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no $arccos_F$ or $arccosh_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

The $arccos^{\#}_{\mathrm{c}(F)}$ range limitation helper function:

$$
\begin{array}{lll}
arccos^{\#}_{\mathrm{c}(F)}(z) & = \max\{up_F(\pi/2), \min\{\mathfrak{Re}(arccos^*_{\mathrm{c}(F)}(z)), down_F(\pi)\}\} + \tilde{\imath} \cdot \mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z)) & \\
 & & \text{if } \mathfrak{Re}(z) < 0 \\
 & = \min\{\mathfrak{Re}(arccos^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\} + \tilde{\imath} \cdot \mathfrak{Im}(arccos^*_{\mathrm{c}(F)}(z)) & \\
 & & \text{if } \mathfrak{Re}(z) \geqslant 0
\end{array}
$$

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

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

$$
\begin{array}{lll}
arccos_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y) & & \\
 & = result^*_{\mathrm{c}(F)}(arccos^{\#}_{\mathrm{c}(F)}(x + \tilde{\imath} \cdot y), nearest_F) & \\
 & & \text{if } x, y \in F \text{ and } (y \neq 0 \text{ or } |x| > 1) \\
 & = arccos_F(x) \mathbf{+\hat{\imath}\cdot} (\mathbf{-0}) & \text{if } x \in F \text{ and } y = 0 \text{ and } |x| \leqslant 1 \\
 & = up_F(\pi/2) \mathbf{+\hat{\imath}\cdot} neg_F(arcsinh_F(y)) & \\
 & & \text{if } x = \mathbf{-0} \\
 & = conj_{\mathrm{c}(F)}(arccos_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} 0)) & \\
 & & \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\
 & = arc_F(x, y) \mathbf{+\hat{\imath}\cdot} (\mathbf{-\infty}) & \text{if } x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and} \\
 & & \quad ((y \in F \text{ and } y \geqslant 0) \text{ or } y = \mathbf{+\infty}) \\
 & = arc_F(x, neg_F(y)) \mathbf{+\hat{\imath}\cdot} (\mathbf{+\infty}) & \\
 & & \text{if } x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and} \\
 & & \quad ((y \in F \text{ and } y < 0) \text{ or } y = \mathbf{-\infty}) \\
 & = no\_result_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y) & \text{otherwise}
\end{array}
$$

NOTE 2 – 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 $arccos_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} 0) \neq arccos_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} (\mathbf{-0}))$ when $|x| > 1$.

5.3.2.10 Radian arc tangent

NOTE 1 – TEMP
$\arctan(-z) = -\arctan(z)$

$\arctan(\text{conj}(z)) = \text{conj}(\arctan(z))$ if $\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| \leqslant 1$
$\arctan(z) = \pm\pi/2 - \text{arccot}(z)$
$Arctan(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arctanh(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arctanh(y - \tilde{\imath} \cdot x)$

The $arctan_{\text{i}(F)}$ operation:

$arctan_{\text{i}(F)} : \text{i}(F) \to \text{i}(F) \cup \{\textbf{infinitary}, \textbf{invalid}\}$

$arctan_{\text{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot arctanh_F(y)$

The $arctan_{\text{i}(F)\to\text{c}(F)}$ operation:

$arctan_{\text{i}(F)\to\text{c}(F)} : \text{i}(F) \to \text{c}(F \cup \{-\mathbf{0}\}) \cup \{\textbf{infinitary}\}$

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

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

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

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

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

$$
\begin{aligned}
&arctan^*_{\text{c}(F)}(\text{conj}(z)) = \text{conj}(arctan^*_{\text{c}(F)}(z)) && \text{if } z \in \mathcal{C}_F \\
&arctan^*_{\text{c}(F)}(-z) = -arctan^*_{\text{c}(F)}(z) && \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Re}(z) < 1 \text{ or } \mathfrak{Im}(z) \neq 0)
\end{aligned}
$$

The relationship 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:

$$
\begin{aligned}
&arctan^*_{\text{c}(F)}(x) = arctan^*_F(x) && \text{if } x \in F \\
&arctan^*_{\text{c}(F)}(\tilde{\imath}\cdot y) = \tilde{\imath}\cdot arctanh^*_F(y) && \text{if } y \in F \text{ and } |y| < 1 \\
&arctan^*_{\text{c}(F)}(\tilde{\imath}\cdot y)) = \pi/2 + \tilde{\imath}\cdot arccoth^*_F(y) && \text{if } y \in F \text{ and } |y| > 1
\end{aligned}
$$

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no $arctan_F$, $arctanh_F$, or $arccoth_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

The $arctan^{\#}_{\text{c}(F)}$ range limitation helper function:

$arctan^{\#}_{\text{c}(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arctan^*_{\text{c}(F)}(z)), down_F(\pi/2)\}\} + \tilde{\imath}\cdot\mathfrak{Im}(arctan^*_{\text{c}(F)}(z))$

The $arctan_{\text{c}(F)}$ operation:

$arctan_{\text{c}(F)} : \text{c}(F) \to \text{c}(F) \cup \{\textbf{underflow}, \textbf{infinitary}\}$

$$
\begin{aligned}
&arctan_{\text{c}(F)}(x + \hat{\imath}\cdot y) \\
&\qquad = result^*_{\text{c}(F)}(arctan^{\#}_{\text{c}(F)}(x + \tilde{\imath}\cdot y), nearest_F) && \text{if } x, y \in F \text{ and } (x \neq 0 \text{ or } |y| \neq 1) \\
&\qquad = \textbf{infinitary}(0 + \hat{\imath}\cdot (mul_F(y, +\infty))) && \text{if } x = 0 \text{ and } |y| = 1 \\
&\qquad = neg_{\text{c}(F)}(arctan_{\text{c}(F)}(0 + \hat{\imath}\cdot neg_F(y)))
\end{aligned}
$$

$$
\begin{aligned}
& & & \text{if } x = \mathbf{-0} \\
&= conj_{\mathrm{c}(F)}(arctan_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} 0)) & & \\
& & & \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\
&= mul_F(signum_F(x), down_F(\pi/2)) \mathbf{+\hat{\imath}\cdot} mul_F(signum_F(y), 0) & & \\
& & & \text{if } (x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\}) \text{ or} \\
& & & \quad (x \in F \text{ and } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\}) \\
&= no\_result_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot} y) & & \text{otherwise}
\end{aligned}
$$

NOTE 2 – 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 $arctan_{\mathrm{c}(F)}(0 \mathbf{+\hat{\imath}\cdot} y) \neq arctan_{\mathrm{c}(F)}((\mathbf{-0}) \mathbf{+\hat{\imath}\cdot} y)$ when $|y| > 1$.

5.3.2.11 Radian arc cotangent

NOTE 1 – TEMP
$\operatorname{arccot}(-z) = -\operatorname{arccot}(z)$
$\operatorname{arccot}(\operatorname{conj}(z)) = \operatorname{conj}(\operatorname{arccot}(z))$ if $\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| \geqslant 1$ (> 1?)
$\operatorname{arccot}(z) = \pm\pi/2 - \arctan(z)$
$Arccot(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arccoth(-y + \tilde{\imath} \cdot x)$
$\operatorname{arccot}(z) = \arctan(1/z)$

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

$arccot_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}, \mathbf{invalid}\}$

$arccot_{\mathrm{i}(F)}(\mathbf{\hat{\imath}\cdot} y) = \mathbf{\hat{\imath}\cdot} arccoth_F(y)$

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

$arccot_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$
\begin{aligned}
& arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot} y) & & \\
&= up_F(-\pi/2) \mathbf{+\hat{\imath}\cdot} arctanh_F(y) & & \\
& & & \text{if } (y \in F \text{ and } -1 < y \text{ and } y < 0) \text{ or } y = \mathbf{-0} \\
&= down_F(\pi/2) \mathbf{+\hat{\imath}\cdot} arctanh_F(y) & & \\
& & & \text{if } y \in F \text{ and } 0 \leqslant y \text{ and } y < 1 \\
&= mul_F(0, y) \mathbf{+\hat{\imath}\cdot} arccoth_F(y) & & \\
& & & \text{if } y \in F \text{ and } |y| \geqslant 1 \\
&= div_F(1, y) \mathbf{+\hat{\imath}\cdot} arccoth_F(y) & & \\
& & & \text{if } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \\
&= no\_result_{\mathrm{i}(F)\to\mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot} y) & & \text{otherwise}
\end{aligned}
$$

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

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

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

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

$$
\begin{aligned}
& arccot^*_{\mathrm{c}(F)}(\operatorname{conj}(z)) = \operatorname{conj}(arccot^*_{\mathrm{c}(F)}(z)) & & \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Re}(z) \neq 0 \text{ or } |\mathfrak{Im}(z)| > 1) \\
& arccot^*_{\mathrm{c}(F)}(-z) = -arccot^*_{\mathrm{c}(F)}(z) & & \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Re}(z) \neq 0 \text{ or } |\mathfrak{Im}(z)| > 1) \\
& \mathfrak{Re}(arccot^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y)) = \pi/2 & & \text{if } y \in F \text{ and } |y| < 1
\end{aligned}
$$

The relationship to the $arccot^*_F$, $arccoth^*_F$ and $arctanh^*_F$ approximation helper functions $arccot_F$, $arccoth_F$ and $arctanh_F$ operations in an associated library for real-valued operations shall be:

$arccot^*_{\mathrm{c}(F)}(x) = arccot^*_F(x)$ if $x \in F$
$arccot^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot arccoth^*_F(-y)$ if $y \in F$ and $|y| > 1$
EDITOR'S NOTE – ...

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no $arccot_F$, $arccoth_F$, or $arctanh_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

The $arccot^{\#}_{\mathrm{c}(F)}$ range limitation helper function:

$$arccot^{\#}_{\mathrm{c}(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arccot^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\}\} + \tilde{\imath} \cdot \mathfrak{Im}(arccot^*_{\mathrm{c}(F)}(z))$$

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

$arccot_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned}
arccot_{\mathrm{c}(F)}(x \mathbin{\mathbf{+\hat{\imath}\cdot}} y) &= result^*_{\mathrm{c}(F)}(arccot^{\#}_{\mathrm{c}(F)}(x + \tilde{\imath} \cdot y), nearest_F) \\
&\qquad \text{if } x, y \in F \text{ and } (|y| \neq 1 \text{ or } x \neq 0) \text{ and } y \neq 0 \\
&= arccot_F(x) \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{-0}) \qquad \text{if } x \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \text{ and } y = 0 \\
&= neg_{\mathrm{c}(F)}(arccot_{\mathrm{c}(F)}(0 \mathbin{\mathbf{+\hat{\imath}\cdot}} neg_F(y))) \\
&\qquad \text{if } x = \mathbf{-0} \\
&= conj_{\mathrm{c}(F)}(arccot_{\mathrm{c}(F)}(x \mathbin{\mathbf{+\hat{\imath}\cdot}} 0)) \\
&\qquad \text{if } y = \mathbf{-0} \text{ and } x \neq \mathbf{-0} \\
&= mul_F(signum_F(x), 0) \mathbin{\mathbf{+\hat{\imath}\cdot}} mul_F(signum_F(y), 0) \\
&\qquad \text{if } (x \in \{\mathbf{-\infty}, \mathbf{+\infty}\} \text{ and } y \in F \cup \{\mathbf{-\infty}, \mathbf{+\infty}\}) \text{ or} \\
&\qquad\quad (x \in F \text{ and } y \in \{\mathbf{-\infty}, \mathbf{+\infty}\}) \\
&= \mathbf{infinitary}(0 \mathbin{\mathbf{+\hat{\imath}\cdot}} (mul_F(y, \mathbf{-\infty}))) \\
&\qquad \text{if } x = 0 \text{ and } |y| = 1 \\
&= no_result_{\mathrm{c}(F)}(x \mathbin{\mathbf{+\hat{\imath}\cdot}} y) \qquad \text{otherwise}
\end{aligned}$$

NOTE 2 – 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 $\{\tilde{\imath} \cdot y \mid y \in \mathcal{R} \text{ and } |y| < 1\}$. Thus $arccot_{\mathrm{c}(F)}(0 \mathbin{\mathbf{+\hat{\imath}\cdot}} y) \neq arccot_{\mathrm{c}(F)}((\mathbf{-0}) \mathbin{\mathbf{+\hat{\imath}\cdot}} y)$ when $|y| < 1$ or $y = \mathbf{-0}$.

5.3.2.12 Radian arc secant

NOTE 1 – TEMP
$\mathrm{arcsec}(-z) = \pi - \mathrm{arcsec}(z)$
$\mathrm{arcsec}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arcsec}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$
$\mathrm{arcsec}(z) = \pi/2 - \mathrm{arccsc}(z)$
$Arcsec(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arcsech(x + \tilde{\imath} \cdot y)$
$\mathrm{arcsec}(z) = \arccos(1/z)$

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

$arcsec_{F \to \mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$\begin{aligned}
arcsec_{F \to \mathrm{c}(F)}(x) &= down_F(\pi) \mathbin{\mathbf{+\hat{\imath}\cdot}} arcsech_F(neg_F(x)) \\
&\qquad \text{if } x \in F \text{ and } 0 \leqslant x < 1 \\
&= 0 \mathbin{\mathbf{+\hat{\imath}\cdot}} neg_F(arcsech_F(x)) \\
&\qquad \text{if } (x \in F \text{ and } -1 < x < 0) \text{ or } x = \mathbf{-0} \\
&= arcsec_F(x) \mathbin{\mathbf{+\hat{\imath}\cdot}} (\mathbf{-0}) \qquad \text{if } (x \in F \text{ and } x \geqslant 1) \text{ or } x = \mathbf{+\infty} \\
&= arcsec_F(x) \mathbin{\mathbf{+\hat{\imath}\cdot}} 0 \qquad \text{if } (x \in F \text{ and } x \leqslant -1) \text{ or } x = \mathbf{-\infty}
\end{aligned}$$

$= no\_result_{F \to \mathrm{c}(F)}(x)$ otherwise

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

$arcsec_{\mathrm{i}(F) \to \mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arcsec_{\mathrm{i}(F) \to \mathrm{c}(F)}(\hat{\mathbf{i}}\cdot y)$

$= up_F(\pi/2) \mathbin{+\hat{\mathbf{i}}\cdot} arccsch_F(y)$ if $(y \in F$ and $y \geqslant 0)$ or $y = +\infty$

$= down_F(\pi/2) \mathbin{+\hat{\mathbf{i}}\cdot} arccsch_F(y)$ if $(y \in F$ and $y < 0)$ or $y \in \{-\infty, -\mathbf{0}\}$

$= no\_result_{\mathrm{i}(F) \to \mathrm{c}(F)}(\hat{\mathbf{i}}\cdot y)$ otherwise

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

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

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

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

$arcsec^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arcsec^*_{\mathrm{c}(F)}(z))$ if $z \in \mathcal{C}_F$ and $(\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$

The relationship to the $arcsec^*_F$ and $arcsech^*_F$ approximation helper functions for $arcsec_F$ and $arcsech_F$ operations in an associated library for real-valued operations shall be:

$arcsec^*_{\mathrm{c}(F)}(x) = arcsec^*_F(x)$ if $x \in F$ and $|x| \geqslant 1$

EDITOR'S NOTE – ...

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no $arcsec_F$ or $arcsech_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

The $arcsec^\#_{\mathrm{c}(F)}$ range limitation helper function:

$arcsec^\#_{\mathrm{c}(F)}(z) = \min\{\mathfrak{Re}(arcsec^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\} + \tilde{\imath} \cdot \mathfrak{Im}(arcsec^*_{\mathrm{c}(F)}(z))$ if $\mathfrak{Re}(z) \geqslant 0$

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

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

$arcsec_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arcsec_{\mathrm{c}(F)}(x \mathbin{+\hat{\mathbf{i}}\cdot} y)$

$= result^*_{\mathrm{c}(F)}(arcsec^\#_{\mathrm{c}(F)}(x + \tilde{\imath} \cdot y), nearest_F)$ if $x, y \in F$ and $(x \neq 0$ or $y \neq 0)$

$= \mathbf{infinitary}(down(\pi/2) \mathbin{+\hat{\mathbf{i}}\cdot} (+\infty))$ if $y = 0$ and $x = 0$

$= arcsec_{\mathrm{c}(F)}(0 \mathbin{+\hat{\mathbf{i}}\cdot} neg_F(y))$ if $x = -\mathbf{0}$

$= conj_{\mathrm{c}(F)}(arcsec_{\mathrm{c}(F)}(x \mathbin{+\hat{\mathbf{i}}\cdot} 0))$ if $y = -\mathbf{0}$ and $x \neq -\mathbf{0}$

$= mul_F(signum_F(x), down_F(\pi/2)) \mathbin{+\hat{\mathbf{i}}\cdot} mul_F(signum_F(y), 0)$ if $(x \in \{-\infty, +\infty\}$ and $y \in F \cup \{-\infty, +\infty\})$ or $(y \in \{-\infty, +\infty\}$ and $x \in F)$

$= no\_result_{\mathrm{c}(F)}(x \mathbin{+\hat{\mathbf{i}}\cdot} y)$ otherwise

NOTE 2 – 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 $\{x \mid x \in \mathcal{R} \text{ and } -1 < x < 1\}$. Thus $arcsec_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0) \neq arcsec_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}))$ when $-1 < x < 1$ or $x = \mathbf{-0}$.

5.3.2.13 Radian arc cosecant

NOTE 1 – TEMP
$\mathrm{arccsc}(-z) = -\mathrm{arccsc}(z)$
$\mathrm{arccsc}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arccsc}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$
$\mathrm{arccsc}(z) = \pi/2 - \mathrm{arcsec}(z)$
$Arcsec(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arcsech(-y + \tilde{\imath} \cdot x)$
$\mathrm{arccsc}(z) = \arcsin(1/z)$

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

$arccsc_{F \to \mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$
\begin{aligned}
arccsc_{F \to \mathrm{c}(F)}(x) &= arccsc_F(x) \mathbin{+\hat{\imath}\cdot} 0 && \text{if } (x \in F \text{ and } x > 1) \text{ or } x = \mathbf{+\infty} \\
&= down_F(\pi/2) \mathbin{+\hat{\imath}\cdot} arcsech_F(x) && \\
& && \text{if } (x \in F \text{ and } 0 \leqslant x \leqslant 1) \\
&= up_F(-\pi/2) \mathbin{+\hat{\imath}\cdot} neg_F(arcsech_F(neg_F(x))) && \\
& && \text{if } (x \in F \text{ and } -1 \leqslant x < 0) \text{ or } x = \mathbf{-0} \\
&= arccsc_F(x) \mathbin{+\hat{\imath}\cdot} (\mathbf{-0}) && \text{if } (x \in F \text{ and } x < -1) \text{ or } x = \mathbf{-\infty} \\
&= no\_result_{F \to \mathrm{c}(F)}(x) && \text{otherwise}
\end{aligned}
$$

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

$arccsc_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccsc_{\mathrm{i}(F)}(\hat{\imath}\cdot y) = \hat{\imath}\cdot arccsch_F(neg_F(y))$

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

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

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

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

$$
\begin{aligned}
&arccsc^*_{\mathrm{c}(F)}(\mathrm{conj}(z)) = \mathrm{conj}(arccsc^*_{\mathrm{c}(F)}(z)) && \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Im}(z) \neq 0 \text{ or } |\mathfrak{Re}(z)| \geqslant 0) \\
&arccsc^*_{\mathrm{c}(F)}(-z) = -arccsc^*_{\mathrm{c}(F)}(z) && \text{if } z \in \mathcal{C}_F \text{ and } (\mathfrak{Im}(z) \neq 0 \text{ or } |\mathfrak{Re}(z)| \geqslant 0)
\end{aligned}
$$

The relationship to the $arccsc^*_F$ and $arccsch^*_F$ approximation helper functions for $arccsc_F$ and $arccsch_F$ operations in an associated library for real-valued operations shall be:

$$
\begin{aligned}
&arccsc^*_{\mathrm{c}(F)}(x) = arccsc^*_F(x) && \text{if } x \in F \text{ and } |x| \geqslant 1 \\
&arccsc^*_{\mathrm{c}(F)}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot arccsch^*_F(-y) && \text{if } y \in F
\end{aligned}
$$

The requirements implied by these relationships and the requirements from part 2 shall hold even if there is no $arccsc_F$ or $arccsch_F$ operations in any associated library for real-valued operations or there is no associated library for real-valued operations.

The $arccsc^{\#}_{\mathrm{c}(F)}$ range limitation helper function:

$arccsc^{\#}_{\mathrm{c}(F)}(z) = \max\{up_F(-\pi/2), \min\{\mathfrak{Re}(arccsc^*_{\mathrm{c}(F)}(z)), down_F(\pi/2)\}\} + \tilde{\imath} \cdot \mathfrak{Im}(arccsc^*_{\mathrm{c}(F)}(z))$

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

$arccsc_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$
\begin{aligned}
&arccsc_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \\
&\qquad = result^{*}_{\mathrm{c}(F)}(arccsc^{\#}_{\mathrm{c}(F)}(x + \tilde{\imath} \cdot y), nearest_F) \\
&\qquad\qquad \text{if } x, y \in F \text{ and } (y \neq 0 \text{ or } 0 < |x| < 1) \\
&\qquad = arccsc_F(x) \mathbin{+\hat{\imath}\cdot} (-\mathbf{0}) \qquad \text{if } y = 0 \text{ and } |x| \geqslant 1 \\
&\qquad = \mathbf{infinitary}(0 \mathbin{+\hat{\imath}\cdot} (-\boldsymbol{\infty})) \\
&\qquad\qquad \text{if } x = 0 \text{ and } y = 0 \\
&\qquad = neg_{\mathrm{c}(F)}(arccsc_{\mathrm{c}(F)}(0 \mathbin{+\hat{\imath}\cdot} neg_F(y))) \\
&\qquad\qquad \text{if } x = -\mathbf{0} \\
&\qquad = conj_{\mathrm{c}(F)}(arccsc_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0)) \\
&\qquad\qquad \text{if } y = -\mathbf{0} \text{ and } x \neq -\mathbf{0} \\
&\qquad = mul_F(signum_F(x), 0) \mathbin{+\hat{\imath}\cdot} mul_F(signum_F(y), -\mathbf{0}) \\
&\qquad\qquad \text{if } (x \in \{-\boldsymbol{\infty}, +\boldsymbol{\infty}\} \text{ and } y \in F \cup \{-\boldsymbol{\infty}, +\boldsymbol{\infty}\}) \text{ or} \\
&\qquad\qquad\quad y \in \{-\boldsymbol{\infty}, +\boldsymbol{\infty}\} \text{ and } x \in F \\
&\qquad = no\_result_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \qquad \text{otherwise}
\end{aligned}
$$

NOTE 2 – 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 $\{x \mid x \in \mathcal{R} \text{ and } -1 < x < 1\}$. Thus $arccsc_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} 0) \neq arccsc_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} (-\mathbf{0}))$ when $-1 < x < 1$ or $x = -\mathbf{0}$.

5.3.3 Operations for hyperbolic elementary functions

Note that the *correspondences* specified below to other ISO/IEC 10967 operations are exact, not approximate.

5.3.3.1 Hyperbolic normalisation

$$
\begin{aligned}
&radh_F : F \to F \\
&radh_F(x) \qquad = x
\end{aligned}
$$

$$
\begin{aligned}
&radh_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\} \\
&radh_{\mathrm{i}(F)}(\hat{\imath}\cdot y) \qquad = \hat{\imath}\cdot rad_F(y)
\end{aligned}
$$

$$
\begin{aligned}
&radh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{absolute\_precision\_underflow}\} \\
&radh_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y) \\
&\qquad = itimes_{\mathrm{c}(F)}(rad_{\mathrm{c}(F)}(y \mathbin{+\hat{\imath}\cdot} neg_F(x)))
\end{aligned}
$$

5.3.3.2 Hyperbolic sine

NOTE – TEMP

$\sinh(-z) = -\sinh(z)$
$\sinh(\mathrm{conj}(z)) = \mathrm{conj}(\sinh(z))$
$\sinh(z + \tilde{\imath} \cdot k \cdot 2 \cdot \pi) = \sinh(z)$ if $k \in \mathcal{Z}$
$\sinh(z) = \cosh(\tilde{\imath} \cdot \pi/2 - z)$

$\sinh(x) = -\tilde{\imath} \cdot \sin(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sin(-\tilde{\imath} \cdot x)$
$\sinh(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sin(-y) = \tilde{\imath} \cdot \sin(y)$

$\sinh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \sin(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \sin(y - \tilde{\imath} \cdot x)$

$\sinh(x + \tilde{\imath} \cdot y) = \sinh(x) \cdot \cos(y) + \tilde{\imath} \cdot \cosh(x) \cdot \sin(y)$

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

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

$sinh_{\mathrm{i}(F)}(\hat{\mathbf{\imath}} \cdot y) \quad = \hat{\mathbf{\imath}} \cdot (sin_F(y))$

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

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

$sinh_{\mathrm{c}(F)}(x + \hat{\mathbf{\imath}} \cdot y)$
$\quad = itimes_{\mathrm{c}(F)}(sin_{\mathrm{c}(F)}(y + \hat{\mathbf{\imath}} \cdot neg_F(x)))$

5.3.3.3 Hyperbolic cosine

NOTE – TEMP
$\cosh(-z) = \cosh(z)$
$\cosh(\mathrm{conj}(z)) = \mathrm{conj}(\cosh(z))$
$\cosh(z + \tilde{\imath} \cdot k \cdot 2 \cdot \pi) = \cosh(z)$ if $k \in \mathcal{Z}$
$\cosh(z) = \sinh(\tilde{\imath} \cdot \pi/2 - z)$

$\cosh(x) = \cos(\tilde{\imath} \cdot x) = \cos(-\tilde{\imath} \cdot x)$
$\cosh(\tilde{\imath} \cdot y) = \cos(y)$
$\cosh(x + \tilde{\imath} \cdot y) = \cos(-y + \tilde{\imath} \cdot x) = \cos(y - \tilde{\imath} \cdot x)$

$\cosh(x + \tilde{\imath} \cdot y) = \cosh(x) \cdot \cos(y) + \tilde{\imath} \cdot \sinh(x) \cdot \sin(y)$

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

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

$cosh_{\mathrm{i}(F)}(\hat{\mathbf{\imath}} \cdot y) \quad = cos_F(y)$

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

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

$cosh_{\mathrm{c}(F)}(x + \hat{\mathbf{\imath}} \cdot y)$
$\quad = cos_{\mathrm{c}(F)}(y + \hat{\mathbf{\imath}} \cdot neg_F(x))$

5.3.3.4 Hyperbolic tangent

NOTE – TEMP
$\tanh(-z) = -\tanh(z)$
$\tanh(\mathrm{conj}(z)) = \mathrm{conj}(\tanh(z))$
$\tanh(z + \tilde{\imath} \cdot k \cdot 2 \cdot \pi) = \tanh(z)$ if $k \in \mathcal{Z}$
$\tanh(z) = \coth(\tilde{\imath} \cdot \pi/2 - z)$

$\tanh(x) = -\tilde{\imath} \cdot \tan(\tilde{\imath} \cdot x) = \tilde{\imath} \cdot \tan(-\tilde{\imath} \cdot x)$
$\tanh(\tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \tan(-y) = \tilde{\imath} \cdot \tan(y)$
$\tanh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot \tan(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot \tan(y - \tilde{\imath} \cdot x)$

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

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

$tanh_{\mathrm{i}(F)}(\hat{\imath}\cdot y) \quad = \hat{\imath}\cdot (tan_F(y))$

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

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

$tanh_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y)$
$\qquad = itimes_{\mathrm{c}(F)}(tan_{\mathrm{c}(F)}(y \mathbin{+\hat{\imath}\cdot} neg_F(x)))$

5.3.3.5 Hyperbolic cotangent

NOTE – TEMP
$\coth(-z) = -\coth(z)$
$\coth(\mathrm{conj}(z)) = \mathrm{conj}(\coth(z))$
$\coth(z + \tilde{\imath}\cdot k \cdot 2\cdot\pi) = \coth(z)$ if $k \in \mathcal{Z}$
$\coth(z) = \tanh(\tilde{\imath}\cdot\pi/2 - z)$
$\coth(z) = 1/\tanh(z)$

$\coth(x) = \tilde{\imath}\cdot\cot(\tilde{\imath}\cdot x) = -\tilde{\imath}\cdot\cot(-\tilde{\imath}\cdot x)$
$\coth(\tilde{\imath}\cdot y) = \tilde{\imath}\cdot\cot(-y) = -\tilde{\imath}\cdot\cot(y)$
$\coth(x + \tilde{\imath}\cdot y) = \tilde{\imath}\cdot\cot(-y + \tilde{\imath}\cdot x) = -\tilde{\imath}\cdot\cot(y - \tilde{\imath}\cdot x)$

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

$coth_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$

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

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

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

$coth_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y)$
$\qquad = itimes_{\mathrm{c}(F)}(cot_{\mathrm{c}(F)}(neg_F(y) \mathbin{+\hat{\imath}\cdot} x))$

5.3.3.6 Hyperbolic secant

NOTE – TEMP
$\mathrm{sech}(-z) = \mathrm{sech}(z)$
$\mathrm{sech}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{sech}(z))$
$\mathrm{sech}(z + \tilde{\imath}\cdot k \cdot 2\cdot\pi) = \mathrm{sech}(z)$ if $k \in \mathcal{Z}$
$\mathrm{sech}(z) = \mathrm{csch}(\tilde{\imath}\cdot\pi/2 - z)$
$\mathrm{sech}(z) = 1/\cosh(z)$

$\mathrm{sech}(x) = \sec(\tilde{\imath}\cdot x) = \sec(-\tilde{\imath}\cdot x)$
$\mathrm{sech}(\tilde{\imath}\cdot y) = \sec(-y) = \sec(y)$
$\mathrm{sech}(x + \tilde{\imath}\cdot y) = \sec(-y + \tilde{\imath}\cdot x) = \sec(y - \tilde{\imath}\cdot x)$

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

$sech_{\mathrm{i}(F)} : \mathrm{i}(F) \to F \cup \{\mathbf{overflow}, \mathbf{absolute\_precision\_underflow}\}$

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

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

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

$sech_{\mathrm{c}(F)}(x \mathbin{+\hat{\imath}\cdot} y)$
$\qquad = sec_{\mathrm{c}(F)}(neg_F(y) \mathbin{+\hat{\imath}\cdot} x)$

5.3.3.7 Hyperbolic cosecant

NOTE – TEMP
$\mathrm{csch}(-z) = -\mathrm{csch}(z)$
$\mathrm{csch}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{csch}(z))$
$\mathrm{csch}(z + \tilde{\imath} \cdot k \cdot 2 \cdot \pi) = \mathrm{csch}(z)$ if $k \in \mathcal{Z}$
$\mathrm{csch}(z) = \mathrm{sech}(\tilde{\imath} \cdot \pi/2 - z)$
$\mathrm{csch}(z) = 1/\sinh(z)$

$\mathrm{csch}(x) = \tilde{\imath} \cdot \mathrm{csc}(\tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{csc}(-\tilde{\imath} \cdot x)$
$\mathrm{csch}(\tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{csc}(-y) = -\tilde{\imath} \cdot \mathrm{csc}(y)$
$\mathrm{csch}(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot \mathrm{csc}(-y + \tilde{\imath} \cdot x) = -\tilde{\imath} \cdot \mathrm{csc}(y - \tilde{\imath} \cdot x)$

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

$csch_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{overflow}, \mathbf{infinitary}, \mathbf{absolute\_precision\_underflow}\}$

$csch_{\mathrm{i}(F)}(\hat{\mathbf{\imath}}\cdot\, y) \quad = \hat{\mathbf{\imath}}\cdot\, (csc_F(neg_F(y)))$

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

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

$csch_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot\, y)$
$\quad = itimes_{\mathrm{c}(F)}(csc_{\mathrm{c}(F)}(neg_F(y) \mathbf{+}\hat{\mathbf{\imath}}\cdot\, x))$

5.3.3.8 Inverse hyperbolic sine

NOTE 1 – TEMP
$\mathrm{arcsinh}(-z) = -\mathrm{arcsinh}(z)$
$\mathrm{arcsinh}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arcsinh}(z))$ if $\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| \leqslant 1$
$\mathrm{arcsinh}(z) = \tilde{\imath} \cdot \pi/2 - \mathrm{arccosh}(z)$ if $\mathfrak{Re}(z) > 0$?
$Arcsinh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arcsin(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arcsin(y - \tilde{\imath} \cdot x)$

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

$arcsinh_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{invalid}\}$

$arcsinh_{\mathrm{i}(F)}(\hat{\mathbf{\imath}}\cdot\, y) = \hat{\mathbf{\imath}}\cdot\, arcsin_F(y)$

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

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

$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\mathbf{\imath}}\cdot\, y)$
$\quad = itimes_{\mathrm{i}(F)}(arcsin_{F\to\mathrm{c}(F)}(y))$

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

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

$arcsinh_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot\, y)$
$\quad = itimes_{\mathrm{c}(F)}(arcsin_{\mathrm{c}(F)}(y \mathbf{+}\hat{\mathbf{\imath}}\cdot\, neg_F(x)))$

NOTE 2 – 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 $arcsinh_{\mathrm{c}(F)}(0 \mathbf{+}\hat{\mathbf{\imath}}\cdot\, y) \neq arcsinh_{\mathrm{c}(F)}(\mathbf{-0} \mathbf{+}\hat{\mathbf{\imath}}\cdot\, y)$ when $|y| > 1$.

5.3.3.9 Inverse hyperbolic cosine

NOTE 1 – TEMP
$\operatorname{arccosh}(-z) = \tilde{\imath} \cdot \pi - \operatorname{arccosh}(z)$
$\operatorname{arccosh}(\operatorname{conj}(z)) = \operatorname{conj}(\operatorname{arccosh}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $\mathfrak{Re}(z) \geqslant 1$
$\operatorname{arccosh}(z) = \tilde{\imath} \cdot \pi/2 - \operatorname{arcsinh}(z)$ if $\mathfrak{Re}(z) > 0$?
$Arccosh(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arccos(x + \tilde{\imath} \cdot y)$

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

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

$$
\begin{aligned}
&arccosh_{F \to \mathrm{c}(F)}(x) \\
&\qquad = itimes_{\mathrm{c}(F)}(arccos_{F \to \mathrm{c}(F)}(x)) \\
&\qquad\qquad \text{if } (x \in F \text{ and } x < 0) \text{ or } x \in \{\mathbf{-\infty}, \mathbf{-0}\} \\
&\qquad = neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arccos_{F \to \mathrm{c}(F)}(x))) \\
&\qquad\qquad \text{if } (x \in F \text{ and } x \geqslant 0) \text{ or } x = \mathbf{+\infty} \\
&\qquad = no\_result_{F \to \mathrm{c}(F)}(x) \qquad \text{otherwise}
\end{aligned}
$$

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

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

$$
\begin{aligned}
&arccosh_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}} \cdot y) \\
&\qquad = itimes_{\mathrm{c}(F)}(arccos_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}} \cdot y)) \\
&\qquad\qquad \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = \mathbf{+\infty} \\
&\qquad = neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arccos_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}} \cdot y))) \\
&\qquad\qquad \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{\mathbf{-\infty}, \mathbf{-0}\} \\
&\qquad = no\_result_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}} \cdot y) \quad \text{otherwise}
\end{aligned}
$$

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

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

$$
\begin{aligned}
&arccosh_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}} \cdot y) \\
&\qquad = itimes_{\mathrm{c}(F)}(arccos_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}} \cdot y)) \\
&\qquad\qquad \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = \mathbf{+\infty} \\
&\qquad = neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arccos_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}} \cdot y))) \\
&\qquad\qquad \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{\mathbf{-\infty}, \mathbf{-0}\} \\
&\qquad = no\_result_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}} \cdot y) \quad \text{otherwise}
\end{aligned}
$$

NOTE 2 – 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 $arccosh_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}} \cdot 0) \neq arccosh_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}} \cdot (\mathbf{-0}))$ when $x < 1$ or $x = \mathbf{-0}$.

5.3.3.10 Inverse hyperbolic tangent

NOTE 1 – TEMP
$\operatorname{arctanh}(-z) = -\operatorname{arctanh}(z)$
$\operatorname{arctanh}(\operatorname{conj}(z)) = \operatorname{conj}(\operatorname{arctanh}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \leqslant 1$
$\operatorname{arctanh}(z) = \pm\tilde{\imath} \cdot \pi/2 - \operatorname{arccoth}(z)$
$Arctanh(x + \tilde{\imath} \cdot y) = -\tilde{\imath} \cdot Arctan(-y + \tilde{\imath} \cdot x) = \tilde{\imath} \cdot Arctan(y - \tilde{\imath} \cdot x)$

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

$arctanh_{F \to \mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arctanh_{F\to \mathrm{c}(F)}(x)$
$\quad = \hat{\mathbf{\imath}}\cdot (arctan_{\mathrm{i}(F)\to \mathrm{c}(F)}(\hat{\mathbf{\imath}}\cdot neg_F(x)))$

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

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

$arctanh_{\mathrm{i}(F)}(\hat{\mathbf{\imath}}\cdot y) = \hat{\mathbf{\imath}}\cdot arctan_F(y)$

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

$arctanh_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arctanh_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot y)$
$\quad = itimes_{\mathrm{c}(F)}(arctan_{\mathrm{c}(F)}(y \mathbf{+}\hat{\mathbf{\imath}}\cdot neg_F(x)))$

NOTE 2 – 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 $arctanh_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot 0) \neq arctanh_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot (\mathbf{-0}))$ when $|x| > 1$.

5.3.3.11 Inverse hyperbolic cotangent

NOTE 1 – TEMP
$\mathrm{arccoth}(-z) = -\mathrm{arccoth}(z)$
$\mathrm{arccoth}(\mathrm{conj}(z)) = \mathrm{conj}(\mathrm{arccoth}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $|\mathfrak{Re}(z)| \geqslant 1$
$\mathrm{arccoth}(z) = \pm\tilde{\imath}\cdot\pi/2 - \mathrm{arctanh}(z)$ if ...
$Arccoth(x + \tilde{\imath}\cdot y) = \tilde{\imath}\cdot Arccot(-y + \tilde{\imath}\cdot x)$
$\mathrm{arccoth}(z) = \mathrm{arctanh}(1/z)$

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

$arccoth_{F\to \mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccoth_{F\to \mathrm{c}(F)}(x)$
$\quad = \hat{\mathbf{\imath}}\cdot arccot_{\mathrm{i}(F)\to \mathrm{c}(F)}(\hat{\mathbf{\imath}}\cdot x)$

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

$arccoth_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}\}$

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

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

$arccoth_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccoth_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot y)$
$\quad = itimes_{\mathrm{c}(F)}(arccot_{\mathrm{c}(F)}(neg_F(y) \mathbf{+}\hat{\mathbf{\imath}}\cdot x))$

NOTE 2 – 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 $arccoth_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot 0) \neq arccoth_{\mathrm{c}(F)}(x \mathbf{+}\hat{\mathbf{\imath}}\cdot (\mathbf{-0}))$ when $|x| < 1$ or $x = \mathbf{-0}$.

5.3.3.12 Inverse hyperbolic secant

NOTE 1 – TEMP
$\text{arcsech}(-z) = \tilde{\imath} \cdot \pi - \text{arcsech}(z)$
$\text{arcsech}(\text{conj}(z)) = \text{conj}(\text{arcsech}(z))$ if $\mathfrak{Im}(z) \neq 0$ or $0 \leqslant \mathfrak{Re}(z) \leqslant 1$
$\text{arcsech}(z) = \tilde{\imath} \cdot \pi/2 - \text{arccsch}(z)$ if $\mathfrak{Re}(z) > 0$?
$Arcsech(x + \tilde{\imath} \cdot y) = \pm\tilde{\imath} \cdot Arcsec(x + \tilde{\imath} \cdot y)$
$\text{arcsech}(z) = \text{arccosh}(1/z)$

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

$arcsech_{F \to \mathrm{c}(F)} : F \to \mathrm{c}(F \cup \{\mathbf{-0}\}) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$arcsech_{F \to \mathrm{c}(F)}(x) = \begin{cases} itimes_{\mathrm{c}(F)}(arcsec_{F \to \mathrm{c}(F)}(x)) & \text{if } (x \in F \text{ and } x < 0) \text{ or } x \in \{\mathbf{-\infty}, \mathbf{-0}\} \\ neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arcsec_{F \to \mathrm{c}(F)}(x))) & \text{if } (x \in F \text{ and } x \geqslant 0) \text{ or } x = \mathbf{+\infty} \\ no\_result_{F \to \mathrm{c}(F)}(x) & \text{otherwise} \end{cases}$$

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

$arcsech_{\mathrm{i}(F) \to \mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$arcsech_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot}\, y) = \begin{cases} itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot}\, y)) & \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = \mathbf{+\infty} \\ neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot}\, y))) & \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{\mathbf{-\infty}, \mathbf{-0}\} \\ no\_result_{\mathrm{i}(F) \to \mathrm{c}(F)}(\mathbf{\hat{\imath}\cdot}\, y) & \text{otherwise} \end{cases}$$

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

$arcsech_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$$arcsech_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot}\, y) = \begin{cases} itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot}\, y)) & \text{if } (y \in F \text{ and } y \geqslant 0) \text{ or } y = \mathbf{+\infty} \\ neg_{\mathrm{c}(F)}(itimes_{\mathrm{c}(F)}(arcsec_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot}\, y))) & \text{if } (y \in F \text{ and } y < 0) \text{ or } y \in \{\mathbf{-\infty}, \mathbf{-0}\} \\ no\_result_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot}\, y) & \text{otherwise} \end{cases}$$

NOTE 2 – 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 $arcsech_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot}\, 0) \neq arcsech_{\mathrm{c}(F)}(x \mathbf{+\hat{\imath}\cdot}\, (\mathbf{-0}))$ when $x \leqslant 0$ or $x = \mathbf{-0}$ or $x > 1$.

5.3.3.13 Inverse hyperbolic cosecant

NOTE 1 – TEMP
$\text{arccsch}(-z) = -\text{arccsch}(z)$
$\text{arccsch}(\text{conj}(z)) = \text{conj}(\text{arccsch}(z))$ if $\mathfrak{Re}(z) \neq 0$ or $|\mathfrak{Im}(z)| \geqslant 1$
$\text{arccsch}(z) = \tilde{\imath} \cdot \pi/2 - \text{arcsech}(z)$ if $\mathfrak{Re}(z) > 0$?
$Arccsch(x + \tilde{\imath} \cdot y) = \tilde{\imath} \cdot Arccsc(-y + \tilde{\imath} \cdot x)$
$\text{arccsch}(z) = \text{arcsinh}(1/z)$

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

$arccsch_{\mathrm{i}(F)} : \mathrm{i}(F) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{invalid}\}$

$arccsch_{\mathrm{i}(F)}(\hat{\mathbf{i}}\cdot y)$
$= \hat{\mathbf{i}}\cdot arccsc_F(neg_F(y))$

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

$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)} : \mathrm{i}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\mathbf{i}}\cdot y)$
$= itimes_{\mathrm{c}(F)}(arccsc_{F\to\mathrm{c}(F)}(y))$

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

$arccsch_{\mathrm{c}(F)} : \mathrm{c}(F) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{infinitary}\}$

$arccsch_{\mathrm{c}(F)}(x \mathbin{+\!\!\hat{\mathbf{i}}\cdot} y)$
$= itimes_{\mathrm{c}(F)}(arccsc_{\mathrm{c}(F)}(neg_F(y) \mathbin{+\!\!\hat{\mathbf{i}}\cdot} x))$

NOTE 2 – 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 $\{\tilde{\mathrm{i}}\cdot y \mid y \in \mathcal{R} \text{ and } |y| < 1\}$. Thus $arccsch_{\mathrm{c}(F)}(0 \mathbin{+\!\!\hat{\mathbf{i}}\cdot} y) \neq arccsch_{\mathrm{c}(F)}(\mathbf{-0} \mathbin{+\!\!\hat{\mathbf{i}}\cdot} y)$ when $-1 < y < 1$ 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 conforms to part 1.

$convert_{I\to\mathrm{c}(I)} : I \to \mathrm{c}(I)$

$convert_{I\to\mathrm{c}(I)}(x)$
$= x \mathbin{+\!\!\hat{\mathbf{i}}\cdot} 0$

$convert_{\mathrm{i}(I)\to\mathrm{c}(I)} : \mathrm{i}(I) \to \mathrm{c}(I)$

$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(\hat{\mathbf{i}}\cdot y)$
$= 0 \mathbin{+\!\!\hat{\mathbf{i}}\cdot} y$

$convert_{\mathrm{i}(I)\to\mathrm{i}(I')} : \mathrm{i}(I) \to \mathrm{i}(I') \cup \{\mathbf{overflow}\}$

$convert_{\mathrm{i}(I)\to\mathrm{i}(I')}(\hat{\mathbf{i}}\cdot y)$
$= \hat{\mathbf{i}}\cdot convert_{I\to I'}(y)$

$convert_{\mathrm{c}(I)\to\mathrm{c}(I')} : \mathrm{c}(I) \to \mathrm{c}(I') \cup \{\mathbf{overflow}\}$

$convert_{\mathrm{c}(I)\to\mathrm{c}(I')}(x \mathbin{+\!\!\hat{\mathbf{i}}\cdot} y)$
$= convert_{I\to I'}(x) \mathbin{+\!\!\hat{\mathbf{i}}\cdot} convert_{I\to I'}(y)$

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

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

$$\begin{aligned} &convert_{F \to \mathrm{c}(F)}(x) \\ &\qquad = x \mathbin{+\hat{\imath}\cdot} im_F(x) && \text{if } x \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \\ &\qquad = no\_result_{F \to \mathrm{c}(F)}(x) && \text{otherwise} \end{aligned}$$

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

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

$$\begin{aligned} &convert_{\mathrm{i}(F) \to \mathrm{c}(F)}(\hat{\imath}\cdot y) \\ &\qquad = re_{\mathrm{i}(F)}(\hat{\imath}\cdot y) \mathbin{+\hat{\imath}\cdot} y && \text{if } y \in F \cup \{-\infty, -\mathbf{0}, +\infty\} \\ &\qquad = no\_result_{\mathrm{i}(F) \to \mathrm{c}(F)}(\hat{\imath}\cdot y) && \text{otherwise} \end{aligned}$$

The $convert_{\mathrm{i}(F) \to \mathrm{i}(F')}$ operation:

$convert_{\mathrm{i}(F) \to \mathrm{i}(F')} : \mathrm{i}(F) \to \mathrm{i}(F') \cup \{\mathbf{underflow}, \mathbf{overflow}\}$

$$\begin{aligned} &convert_{\mathrm{i}(F) \to \mathrm{i}(F')}(\hat{\imath}\cdot y) \\ &\qquad = \hat{\imath}\cdot convert_{F \to F'}(y) \end{aligned}$$

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

$convert_{\mathrm{c}(F) \to \mathrm{c}(F')} : \mathrm{c}(F) \to \mathrm{c}(F') \cup \{\mathbf{underflow}, \mathbf{overflow}\}$

$$\begin{aligned} &convert_{\mathrm{c}(F) \to \mathrm{c}(F')}(x \mathbin{+\hat{\imath}\cdot} y) \\ &\qquad = convert_{F \to F'}(x) \mathbin{+\hat{\imath}\cdot} convert_{F \to F'}(y) \end{aligned}$$

The $convert_{\mathrm{i}(F) \to \mathrm{i}(D)}$ operation:

$convert_{\mathrm{i}(F) \to \mathrm{i}(D)} : \mathrm{i}(F) \to \mathrm{i}(D) \cup \{\mathbf{overflow}\}$

$$\begin{aligned} &convert_{\mathrm{i}(F) \to \mathrm{i}(D)}(\hat{\imath}\cdot y) \\ &\qquad = \hat{\imath}\cdot convert_{F \to D}(y) \end{aligned}$$

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

$convert_{\mathrm{c}(F) \to \mathrm{c}(D)} : \mathrm{c}(F) \to \mathrm{c}(D) \cup \{\mathbf{overflow}\}$

$$\begin{aligned} &convert_{\mathrm{c}(F) \to \mathrm{c}(D)}(x \mathbin{+\hat{\imath}\cdot} y) \\ &\qquad = convert_{F \to D}(x) \mathbin{+\hat{\imath}\cdot} convert_{F \to D}(y) \end{aligned}$$

The $convert_{\mathrm{i}(D) \to \mathrm{i}(F)}$ operation:

$convert_{\mathrm{i}(D) \to \mathrm{i}(F)} : \mathrm{i}(D) \to \mathrm{i}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$

$convert_{\mathrm{i}(D)\to\mathrm{i}(F)}(\hat{\mathbf{i}}\cdot y)$
$= \hat{\mathbf{i}}\cdot convert_{D\to F}(y)$

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

$convert_{\mathrm{c}(D)\to\mathrm{c}(F)} : \mathrm{c}(D) \to \mathrm{c}(F) \cup \{\mathbf{underflow}, \mathbf{overflow}\}$
$convert_{\mathrm{c}(D)\to\mathrm{c}(F)}(x + \hat{\mathbf{i}}\cdot y)$
$= convert_{D\to F}(x) + \hat{\mathbf{i}}\cdot convert_{D\to F}(y)$

5.5 Support for imaginary and complex numerals

Rather than specify special numerals for imaginary and complex values, imaginary units are specified. These can be used as a basis for expressions that transform ordinary numerals to imaginary or complex numerals.

The imaginary and complex unit operations are as follows:

$imaginary\_unit_{\mathrm{i}(I)}$
$imaginary\_unit_{\mathrm{c}(I)}$
$imaginary\_unit_{\mathrm{i}(F)}$
$imaginary\_unit_{\mathrm{c}(F)}$

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.

An implementation shall not give notifications for operations conforming to this part, 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 $-\mathbf{0}$, $-\boldsymbol{\infty}$, $+\boldsymbol{\infty}$, 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: $-\mathbf{0}$, $+\boldsymbol{\infty}$, $-\boldsymbol{\infty}$, *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 $-\mathbf{0}$, $+\boldsymbol{\infty}$, $-\boldsymbol{\infty}$, or **NaN**.

7 Relationship with language standards

A computing system often provides some of the operations specified in this part within the context of a programming language. The requirements of the present standard shall be in addition to those imposed by the relevant programming language standards.

This part does not define the syntax of arithmetic expressions. However, programmers need to know how to reliably access the operations specified in this part.

> NOTE 1 – Providing the information required in this clause is properly the responsibility of programming language standards. 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 specified in this part and made available. An implementation should document the notation that should be used to invoke an operation specified in this part and that could be made available.

> NOTE 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 [7] |
> | `casin(`x`)` | in C [13] |
> | `asin(`x`)` | in Fortran [18] and C++ [14] |
> | `(asin` x`)` | in Common Lisp [38] |
>
> with a suitable expression of the argument (x).

An implementation shall document the semantics of arithmetic expressions in terms of compositions of the operations specified in clause 5 of this part and in clause 5 of part 1.

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 range 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.

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 part.

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 part 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 maximum error parameter, the notation to be used to access that parameter.

j) The notation to be used to access the parameters $big\_angle\_r_F$ and $big\_angle\_u_F$.

k) For each of the provided operations where this part specifies a relation to another operation specified in this International Standard, the binding for that other operation.

l) 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, the following information shall also be provided by any conforming implementation.

m) The means for selecting the modes of operation that ensure conformity.

n) 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.)

o) The methods used for notification, and the information made available about the notification. (See clause 6 of part 1.)

p) 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.)

q) 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.

$max\_error\_mul_{c(F)} \in [0.5, 5]$
$max\_error\_div_{c(F)} \in [0.5, 15]$

$max\_error\_exp_{c(F)} \in [0.5, 2 \cdot rnd\_error_F]$
$max\_error\_power_{c(F)} \in [0.5, 7]$

$max\_error\_sin_{c(F)} \in [0.5, 11]$
$max\_error\_tan_{c(F)} \in [0.5, 14]$

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 Partial conformity to part 1 or to part 2

...

Annex B
(informative)

Rationale

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* (LIA-3).

B.1 Scope

B.1.1 Inclusions

Integer complex datatypes are included in Common Lisp [38]. Integer complex numbers are sometimes referred to as Gaussian numbers. (Common Lisp also include complex datatypes that have rational datatype values in the parts.)

Imaginary datatypes are included in C99 (informatively; annex G of [13]) and in Ada (normatively; annex G of [7]).

B.1.2 Exclusions

LIA-3 only cover Cartesian complex datatypes. Polar complex datatypes are not included since no programming language provides them. Neither rational nor fixed point complex datatypes are covered by LIA-3 since (yet) no part of LIA cover rational or fixed point datatypes.

B.2 Conformity

Conformity to this standard is dependent on the existence of language binding standards. Each programming language committee (or other organisation 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.

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 is 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 $\mathbf{+\tilde{\imath}\cdot}$ 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 $\mathbf{\tilde{\imath}\cdot}$ 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 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 Datatypes and exceptional values

LIA-3 uses the same set of exceptional values as LIA-2.

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 [7] exception), while **invalid** with a specified continuation value 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.4.1.5 Complex value constructors and complex datatype constructors

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 `exception`s, 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

...

B.5.2 Imaginary and complex floating point datatypes and operations

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* [53], 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*[48] as justification. The report *Complex Floating-Point C Extensions* [54], also issued by the ANSI X3J11 committee, extends these recommendations to imaginary and complex datatypes. These reports have strongly influenced the C99 [13] 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 $\mathbf{00}$. Note that $\mathbf{00}$ is *not* represented in the datatype corresponding to F, nor in $\mathrm{i}(F)$ or $\mathrm{c}(F)$, not even as a component. $\mathbf{00}$ is strong in the sense that $\mathbf{00}$ overtrumps NaN and infinity values, which neither 0 nor $\mathbf{-0}$ does. I.e. the implicit zeroes ($\mathbf{00}$)

work *as if* the rules of the following table applies (plus commutativity):

$mul_F(0, +\infty) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(-\mathbf{0}, +\infty) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{00}, +\infty) = \mathbf{00}$
$mul_F(0, -\infty) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(-\mathbf{0}, -\infty) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{00}, -\infty) = \mathbf{00}$
$mul_F(0, \mathbf{qNaN}) = \mathbf{qNaN}$	$mul_F(-\mathbf{0}, \mathbf{qNaN}) = \mathbf{qNaN}$	$mul_F(\mathbf{00}, \mathbf{qNaN}) = \mathbf{00}$
$mul_F(0, \mathbf{sNaN}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(-\mathbf{0}, \mathbf{sNaN}) = \mathbf{invalid}(\mathbf{qNaN})$	$mul_F(\mathbf{00}, \mathbf{sNaN}) = \mathbf{00}$

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, i(F) and 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 i(F)) but return a value in 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 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) 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 require them.

EDITOR'S NOTE – include anyway?

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).

EDITOR'S NOTE – include anyway?

B.5.3.3 Operations for hyperbolic elementary functions

These 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 more rarely 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.

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.

EDITOR'S NOTE – include anyway?

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.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 by the binding.

Portability of programs can be improved if two conforming LIA-3 systems using the same 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 radian trigonometric operations are provided), but if one system provides it by means of the identifier `BigAngle` and another by the identifier `MaxAngle`, 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.

The following clauses are suggestions rather than requirements because the areas covered are the responsibility of the various 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
Haskell
Java
Common Lisp
ISLisp
Modula-2
PL/I
SML

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. Where parameters, constants, or operations are not provided by the language, names and syntax are suggested. (Already provided syntax is marked with a $\star$.)

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 examplified 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 examplified 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* [7].

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, and the notations *IINT* and *IFLT* are used to stand for the names of one of the corresponding imaginary datatypes (respectively, and the notations *CINT* and *CFLT* are used to stand for the names of one of the corresponding complex datatypes (respectively).

Ada has an overloading system, so that the same name can be used for different types of arguments to the operations.

The LIA-3 complex integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`iTimes(`x`)`	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`iTimes(`x`)`	†
$itimes_{\mathrm{c}(I)}(x)$	`iTimes(`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`)`	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	`Compose_From_Cartesian(`x`,`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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†

$max_{\mathrm{i}(I)}(x, y)$	$CINT$'`max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x, y)$	$CINT$'`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`MaxSeq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`MinSeq(`xs`)`	†

where x and y are expressions of type INT, $IINT$, or $CINT$ as appropriate, and xs is an expression of type array of $CINT$.

The LIA-3 basic complex floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\rightarrow\mathrm{i}(F)}(x)$	`i * `x or `j * `x	⋆
$itimes_{\mathrm{i}(F)\rightarrow F}(x)$	`i * `x or `j * `x	⋆
$itimes_{\mathrm{c}(F)}(x)$	`i * `x or `j * `x	⋆
$re_F(x)$	`Re(`x`)`	†
$re_{\mathrm{i}(F)}(x)$	`Re(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	`Re(`x`)`	⋆
$im_F(x)$	`Im(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	`Im(`x`)`	⋆
$im_{\mathrm{c}(F)}(x)$	`Im(`x`)`	⋆
$plusitimes_{\mathrm{c}(F)}(x, y)$	`Compose_From_Cartesian(`x`,`y`)`	⋆
$plusitimes_{\mathrm{c}(F)}(x, y)$	x `+ i * `y or x `+ j * `y	⋆
$neg_{\mathrm{i}(F)}(x)$	`-`x	⋆
$neg_{\mathrm{c}(F)}(x)$	`-`x	⋆
$conj_F(x)$	`Conjugate(`x`)`	†
$conj_{\mathrm{i}(F)}(x)$	`Conjugate(`x`)`	⋆
$conj_{\mathrm{c}(F)}(x)$	`Conjugate(`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	$\star$
$mul_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	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	$\star$
$eq_{\mathrm{i}(F)}(x,y)$	x `=` y	$\star$
$eq_{F,\mathrm{i}(F)}(x,y)$	x `=` y	$\star$
$eq_{\mathrm{i}(F),F}(x,y)$	x `=` y	$\star$
$eq_{F,\mathrm{c}(F)}(x,y)$	x `=` y	$\star$
$eq_{\mathrm{c}(F),F}(x,y)$	x `=` y	$\star$
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `=` y	$\star$
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `=` y	$\star$
$eq_{\mathrm{c}(F)}(x,y)$	x `=` y	$\star$
$neq_{\mathrm{i}(F)}(x,y)$	x `/=` y	$\star$
$neq_{F,\mathrm{i}(F)}(x,y)$	x `/=` y	$\star$
$neq_{\mathrm{i}(F),F}(x,y)$	x `/=` y	$\star$
$neq_{F,\mathrm{c}(F)}(x,y)$	x `/=` y	$\star$
$neq_{\mathrm{c}(F),F}(x,y)$	x `/=` y	$\star$
$neq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `/=` y	$\star$
$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `/=` y	$\star$
$neq_{\mathrm{c}(F)}(x,y)$	x `/=` y	$\star$
$lss_{\mathrm{i}(F)}(x,y)$	x `<` y	$\star$
$leq_{\mathrm{i}(F)}(x,y)$	x `<=` y	$\star$
$gtr_{\mathrm{i}(F)}(x,y)$	x `>` y	$\star$
$geq_{\mathrm{i}(F)}(x,y)$	x `>=` y	$\star$
$abs_{\mathrm{i}(F)}(x)$	`abs` x	$\star$
$abs_{\mathrm{c}(F)}(x)$	`abs` x or `Modulus(`x`)`	$\star$
$phase_F(x)$	`Argument(`x`)`	†
$phase_{\mathrm{i}(F)}(x)$	`Argument(`x`)`	†
$phase_{\mathrm{c}(F)}(x)$	`Argument(`x`)`	$\star$
$phaseu_F(u,x)$	`Argument(`x`,`u`)`	†
$phaseu_{\mathrm{i}(F)}(u,x)$	`Argument(`x`,`u`)`	†
$phaseu_{\mathrm{c}(F)}(u,x)$	`Argument(`x`,`u`)`	$\star$

$signum_F(x)$	`Signum(`x`)`	†
$signum_{\mathrm{i}(F)}(x)$	`Signum(`x`)`	†
$signum_{\mathrm{c}(F)}(x)$	`Signum(`x`)`	†
$polar_F(x, y)$	`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 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`)`	†
$max\_err\_tan_{\mathrm{c}(F)}$	`Err_Tan(`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 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`) ** Compose_From_Cartesian(`y`)`	⋆
$power_{\mathrm{i}(F)}(x, y)$	x `**` y	†
$power_{\mathrm{i}(F),F}(b, y)$	b `**` y	†
$power_{F,\mathrm{i}(F)}(b, x)$	b `**` x	†
$power_{\mathrm{c}(F),F}(b, y)$	b `**` y	⋆
$power_{F,\mathrm{c}(F)}(b, x)$	b `**` x	⋆
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b, y)$	b `**` y	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b, x)$	b `**` x	†
$power_{\mathrm{c}(F)}(x, y)$	x `**` y	⋆
$sqrt_{F\to\mathrm{c}(F)}(x)$	`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`)`	†
$ln_{F\to \mathrm{c}(F)}(x)$	`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`)`	⋆
$logbase_{F\to \mathrm{c}(F)}(b,x)$	`Logc(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F),F}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F),F}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{F,\mathrm{c}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$rad_{\mathrm{i}(F)}(x)$	`Rad(`x`)`	†
$rad_{\mathrm{c}(F)}(x)$	`Rad(`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`)`	†
$arcsin_{F\to \mathrm{c}(F)}(x)$	`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`)`	†
$arccos_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`ArcCosc(`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`)`	†
$arccosh_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`ArcCosH(`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`)`	†
$arcsech_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`ArcSecH(`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 b, x, and y are expressions of type *FLT*, *IFLT*, or *CFLT* as appropriate.

Arithmetic value conversions in Ada are always explicit.

(mockup so far!)

$convert_{I\to \mathrm{c}(I)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(I)\to \mathrm{c}(I)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(I)\to \mathrm{i}(I')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{c}(I)\to \mathrm{c}(I')}(x)$	`Compose_From_Cartesian(`x`)`	†

$convert_{F \to \mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	$\star$
$convert_{F \to \mathrm{c}(F)}(x)$	x `+ i * Im(`x`)` or x `+ j * Im(`x`)`	†
$convert_{\mathrm{i}(F) \to \mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	$\star$
$convert_{\mathrm{i}(F) \to \mathrm{c}(F)}(x)$	`Re(`x`) +` x	$\star$
$convert_{\mathrm{i}(F) \to \mathrm{i}(F')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{c}(F) \to \mathrm{c}(F')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(F) \to \mathrm{i}(D)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{c}(F) \to \mathrm{c}(D)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(D) \to \mathrm{i}(F)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{c}(D) \to \mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	†

EDITOR'S NOTE – complex I/O!!!

Numerals:

$imaginary\_unit_{\mathrm{i}(I)}$	`ii`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`-0+ii`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`i` or `j`	$\star$
$imaginary\_unit_{\mathrm{c}(F)}$	`-0+i` or `-0+j`	$\star$

C.2 C

The programming language C is defined by ISO/IEC 9899:1999, *Information technology – Programming languages – C* [13].

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.

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.

C99 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* is used below to designate one of the complex or imaginary (as appropriate for the operation) floating point datatypes. The operations marked "(⋆)" are available only when the implementation provides imaginary floating point datatypes.

The LIA-3 complex integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`II * `x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`II * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`II * `x	†
$re_{I}(x)$	`creali`$t(x)$	†
$re_{\mathrm{i}(I)}(x)$	`creali`$t(x)$	†
$re_{\mathrm{c}(I)}(x)$	`creali`$t(x)$	†
$im_{I}(x)$	`cimagi`$t(x)$	†
$im_{\mathrm{i}(I)}(x)$	`cimagi`$t(x)$	†
$im_{\mathrm{c}(I)}(x)$	`cimagi`$t(x)$	†
$plusitimes_{I\to \mathrm{c}(I)}(x,y)$	x `+ II * `y	†
$neg_{\mathrm{i}(I)}(x)$	`-`x	†
$neg_{\mathrm{c}(I)}(x)$	`-`x	†
$conj_{I}(x)$	`conji`$t(x)$	†
$conj_{\mathrm{i}(I)}(x)$	`conji`$t(x)$	†
$conj_{\mathrm{c}(I)}(x)$	`conji`$t(x)$	†
$add_{\mathrm{i}(I)}(x,y)$	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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†
$max_{\mathrm{i}(I)}(x,y)$	`max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x,y)$	`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†

where x and y are expressions of type *CINT*.,.,.,.,,

The LIA-3 non-transcendental complex floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\to\mathrm{i}(F)}(x)$	`I *` x	($\star$)
$itimes_{\mathrm{i}(F)\to F}(x)$	`I *` x	($\star$)
$itimes_{\mathrm{c}(F)}(x)$	`I *` x	$\star$
$re_F(x)$	`creal`t`(`x`)`	†
$re_{\mathrm{i}(F)}(x)$	`creal`t`(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	`creal`t`(`x`)` or `creal(`x`)`	$\star$
$im_F(x)$	`cimag`t`(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	`cimag`t`(`x`)`	†
$im_{\mathrm{c}(F)}(x)$	`cimag`t`(`x`)` or `cimag(`x`)`	$\star$
$plusitimes_{\mathrm{c}(F)}(x,y)$	x `+ I *` y	$\star$
$neg_{\mathrm{i}(F)}(x)$	`-` x	($\star$)
$neg_{\mathrm{c}(F)}(x)$	`-` x	$\star$
$conj_F(x)$	`conj`t`(`x`)`	†
$conj_{\mathrm{i}(F)}(x)$	`conj`t`(`x`)` or `conj(`x`)`	($\star$)
$conj_{\mathrm{c}(F)}(x)$	`conj`t`(`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			$\star$
$eq_{\mathrm{i}(F)}(x,y)$	x `==` y			$(\star)$
$eq_{F,\mathrm{i}(F)}(x,y)$	x `==` y			$\star$
$eq_{\mathrm{i}(F),F}(x,y)$	x `==` y			$(\star)$
$eq_{F,\mathrm{c}(F)}(x,y)$	x `==` y			$\star$
$eq_{\mathrm{c}(F),F}(x,y)$	x `==` y			$\star$
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `==` y			$(\star)$
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `==` y			$(\star)$
$eq_{\mathrm{c}(F)}(x,y)$	x `==` y			$\star$
$neq_{\mathrm{i}(F)}(x,y)$	x `!=` y			$(\star)$
$neq_{F,\mathrm{i}(F)}(x,y)$	x `!=` y			$\star$
$neq_{\mathrm{i}(F),F}(x,y)$	x `!=` y			$(\star)$
$neq_{F,\mathrm{c}(F)}(x,y)$	x `!=` y			$\star$
$neq_{\mathrm{c}(F),F}(x,y)$	x `!=` y			$\star$
$neq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `!=` y			$(\star)$
$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `!=` y			$(\star)$
$neq_{\mathrm{c}(F)}(x,y)$	x `!=` y			$\star$
$lss_{\mathrm{i}(F)}(x,y)$	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			†
$abs_{\mathrm{i}(F)}(x)$	`fabs(`x`)`			$(\star)$
$abs_{\mathrm{c}(F)}(x)$	`cabs`*t*`(`x`)`	or	`fabs(`x`)`	$\star$
$phase_F(x)$	`carg`*t*`(`x`)`			†
$phase_{\mathrm{i}(F)}(x)$	`carg`*t*`(`x`)`	or	`carg(`x`)`	$(\star)$
$phase_{\mathrm{c}(F)}(x)$	`carg`*t*`(`x`)`	or	`carg(`x`)`	$\star$
$phaseu_F(u,x)$	`cargu`*t*`(`x`,` u`)`			†
$phaseu_{\mathrm{i}(F)}(u,x)$	`cargu`*t*`(`x`,` u`)`			†
$phaseu_{\mathrm{c}(F)}(u,x)$	`cargu`*t*`(`x`,` u`)`			†

$signum_F(x)$	`signbit(`x`)`	$\star$
$signum_{\mathrm{i}(F)}(x)$	`I*signbit(cimag(`x`))`	$\star$
$signum_{\mathrm{c}(F)}(x)$	`csign`t`(`x`)`	†
$polar_F(x, y)$	`polar`t`(`x`,`y`)`	†
$polaru_F(u, x, y)$	`polaru`t`(`x`,`y`,`u`)`	†
$proj_F(x)$	`proj`t`(`x`)` or `cproj(`x`)`	† Not LIA-3
$proj_{\mathrm{i}(F)}(x)$	`cproj(`x`)`	† Not LIA-3
$proj_{\mathrm{c}(F)}(x)$	`cproj`t`(`x`)` or `cproj(`x`)`	$\star$ Not LIA-3

where x and y are expressions of the same complex floating point type.,.,.,.,.

The LIA-3 parameters for 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	†
$max\_err\_exp_{\mathrm{c}(F)}$	`err_cexp`t	†
$max\_err\_power_{\mathrm{c}(F)}$	`err_cpower`t	†
$max\_err\_sin_{\mathrm{c}(F)}$	`err_csin`t	†
$max\_err\_tan_{\mathrm{c}(F)}$	`err_ctan`t	†

where t ...

The LIA-3 elementary complex floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b, y)$	`cpoweri`t`(`b`, `y`)`	†
$exp_{\mathrm{i}(F)}(x)$	`exp(`x`)`	($\star$)
$exp_{\mathrm{c}(F)}(x)$	`cexp`t`(`x`)` or `exp(`x`)`	$\star$
$power_{F\to\mathrm{i}(F)}(x, y)$	`powc(`x`, `y`)`	†
$power_{\mathrm{i}(F)}(x, y)$	`pow(`x`, `y`)`	$\star$
$power_{\mathrm{i}(F),F}(b, y)$	`pow(`b`, `y`)`	$\star$
$power_{F,\mathrm{i}(F)}(b, x)$	`pow(`b`, `x`)`	$\star$
$power_{\mathrm{c}(F),F}(b, y)$	`pow(`b`, `y`)`	$\star$
$power_{F,\mathrm{c}(F)}(b, x)$	`pow(`b`, `x`)`	$\star$
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b, y)$	`pow(`b`, `y`)`	$\star$
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b, x)$	`pow(`b`, `x`)`	$\star$
$power_{\mathrm{c}(F)}(b, y)$	`cpow`t`(`b`, `y`)` or `pow(`b`, `y`)`	$\star$ (dev. at orig?)
$sqrt_{F\to\mathrm{c}(F)}(x)$	`sqrtc(`x`)`	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`sqrt(`x`)`	($\star$)
$sqrt_{\mathrm{c}(F)}(x)$	`csqrt`t`(`x`)` or `sqrt(`x`)`	$\star$

$ln_{F\to c(F)}(x)$	`logc`*t*`(`*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*`)`	⋆
$logbase_{F\to c(F)}(b,x)$	`Logc(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{i(F)}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{i(F),F}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{F,i(F)}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{c(F),F}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{F,c(F)}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{i(F),c(F)}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{c(F),i(F)}(b,x)$	`Log(`*x*`, `*b*`)` (note parameter order)	†
$logbase_{c(F)}(b,x)$	`clogbase`*t*`(`*b*`, `*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`*t*`(`*x*`)`	(⋆)
$tan_{c(F)}(x)$	`ctan`*t*`(`*x*`)` or `tan(`*x*`)`	⋆
$cot_{i(F)}(x)$	`cot(`*x*`)`	†
$cot_{c(F)}(x)$	`ccot`*t*`(`*x*`)`	†
$sec_{i(F)}(x)$	`sec(`*x*`)`	†
$sec_{c(F)}(x)$	`csec`*t*`(`*x*`)`	†
$csc_{i(F)}(x)$	`csc(`*x*`)`	†
$csc_{c(F)}(x)$	`ccsc`*t*`(`*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(`*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(`*x*`)`	†
$arctan_{c(F)}(x)$	`catan`*t*`(`*x*`)` or `atan(`*x*`)`	⋆
$arccot_{i(F)}(x)$	`acot(`*x*`)`	†
$arccot_{i(F)\to c(F)}(x)$	`acotc(`*x*`)`	†
$arccot_{c(F)}(x)$	`cacot`*t*`(`*x*`)`	†
$arcsec_{F\to c(F)}(x)$	`asecc(`*x*`)`	†
$arcsec_{i(F)\to c(F)}(x)$	`asecc(`*x*`)`	†
$arcsec_{c(F)}(x)$	`casec`*t*`(`*x*`)`	†

$arccsc_{F\to \mathrm{c}(F)}(x)$	`acscc`(x)	†
$arccsc_{\mathrm{i}(F)}(x)$	`acsc`(x)	†
$arccsc_{\mathrm{c}(F)}(x)$	`cacsc`$t(x)$	†
$radh_F(x)$	`radianh`$t(x)$	†
$radh_{\mathrm{i}(F)}(x)$	`radianh`$t(x)$	†
$radh_{\mathrm{c}(F)}(x)$	`radianh`$t(x)$	†
$sinh_{\mathrm{i}(F)}(x)$	`sinh`(x)	$(\star)$
$sinh_{\mathrm{c}(F)}(x)$	`csinh`$t(x)$ or `sinh`(x)	$\star$
$cosh_{\mathrm{i}(F)}(x)$	`cosh`(x)	$(\star)$
$cosh_{\mathrm{c}(F)}(x)$	`ccosh`$t(x)$ or `cosh`(x)	$\star$
$tanh_{\mathrm{i}(F)}(x)$	`tanh`(x)	$(\star)$
$tanh_{\mathrm{c}(F)}(x)$	`ctanh`$t(x)$ or `tanh`(x)	$\star$
$coth_{\mathrm{i}(F)}(x)$	`coth`(x)	†
$coth_{\mathrm{c}(F)}(x)$	`ccoth`$t(x)$	†
$sech_{\mathrm{i}(F)}(x)$	`sech`(x)	†
$sech_{\mathrm{c}(F)}(x)$	`csech`$t(x)$	†
$csch_{\mathrm{i}(F)}(x)$	`csch`(x)	†
$csch_{\mathrm{c}(F)}(x)$	`ccsch`$t(x)$	†
$arcsinh_{\mathrm{i}(F)}(x)$	`asinh`(x)	$(\star)$
$arcsinh_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`asinhc`(x)	†
$arcsinh_{\mathrm{c}(F)}(x)$	`casinh`$t(x)$ or `asinh`(x)	$\star$
$arccosh_{F\to \mathrm{c}(F)}(x)$	`acoshc`(x)	†
$arccosh_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`acosh`(x)	$(\star)$
$arccosh_{\mathrm{c}(F)}(x)$	`cacosh`$t(x)$ or `acosh`(x)	$\star$
$arctanh_{F\to \mathrm{c}(F)}(x)$	`atanhc`(x)	†
$arctanh_{\mathrm{i}(F)}(x)$	`atanh`(x)	$(\star)$
$arctanh_{\mathrm{c}(F)}(x)$	`catanh`$t(x)$ or `atanh`(x)	$\star$
$arccoth_{F\to \mathrm{c}(F)}(x)$	`acothc`(x)	†
$arccoth_{\mathrm{i}(F)}(x)$	`acoth`(x)	†
$arccoth_{\mathrm{c}(F)}(x)$	`cacoth`$t(x)$	†
$arcsech_{F\to \mathrm{c}(F)}(x)$	`asechc`(x)	†
$arcsech_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`asech`(x)	†
$arcsech_{\mathrm{c}(F)}(x)$	`casech`$t(x)$	†
$arccsch_{\mathrm{i}(F)}(x)$	`acsch`(x)	†
$arccsch_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`acschc`(x)	†
$arccsch_{\mathrm{c}(F)}(x)$	`cacsch`$t(x)$	†

where b, x, and y are expressions of the same complex floating point type.,.,., t is a string, part of the operation name, and is "`f`" for `float _Complex`, the empty string for `double _Complex`, and "`l`" for `long double _Complex`.

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.

$convert_{I\to \mathrm{c}(I)}(x)$	`(`INT `_Complex)`x	†
$convert_{\mathrm{i}(I)\to \mathrm{c}(I)}(x)$	`(`INT `_Complex)`x	†

$convert_{\mathrm{i}(I)\to\mathrm{i}(I')}(x)$	`(`$INT2$ `_Imaginary)`x	†
$convert_{\mathrm{c}(I)\to\mathrm{c}(I')}(x)$	`(`$INT2$ `_Complex)`x	†
$convert_{F\to\mathrm{c}(F)}(x)$	`(`FLT `_Complex)`x	$\star$
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(`FLT `_Complex)`x	($\star$)
$convert_{\mathrm{i}(F)\to\mathrm{i}(F')}(x)$	`(`$FLT2$ `_Imaginary)`x	($\star$)
$convert_{\mathrm{c}(F)\to\mathrm{c}(F')}(x)$	`(`$FLT2$ `_Complex)`x	$\star$
$convert_{\mathrm{i}(F'')\to\mathrm{i}(F)}(x)$	`sscanf...`x	($\star$)
$convert_{\mathrm{c}(F'')\to\mathrm{c}(F)}(x)$	`sscanf...`x	$\star$
$convert_{\mathrm{i}(F)\to\mathrm{i}(F'')}(x)$	`sprintf...`x	($\star$)
$convert_{\mathrm{c}(F)\to\mathrm{c}(F'')}(x)$	`sprintf...`x	$\star$
$convert_{\mathrm{i}(D)\to\mathrm{i}(F)}(x)$	`sscanf...(...`x`)`	($\star$)
$convert_{\mathrm{c}(D)\to\mathrm{c}(F)}(x)$	`sscanf...(...`x`)`	$\star$
$convert_{\mathrm{i}(F)\to\mathrm{i}(D)}(x)$	`sprintf...(...`x`)`	($\star$)
$convert_{\mathrm{c}(F)\to\mathrm{c}(D)}(x)$	`sprintf...(...`x`)`	$\star$

complex IO??

where x is an expression of type *INT*, y is an expression of type *FLT*, and z is an expression of type *FXD*, where *FXD* is a fixed point type. *INT2* is the integer datatype that corresponds to I'. A ? above indicates that the parameter is optional. e is greater than 0.,.,.,.,.,.,. Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`II` or `_Imaginary_II`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`((II))` or `_Complex_II`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`I` or `_Imaginary_I`	($\star$)
$imaginary\_unit_{\mathrm{c}(F)}$	`((I))` or `_Complex_I`	$\star$

C.3 C++

The programming language C++ is defined by ISO/IEC 14882:1998, *Programming languages – C++* [14].

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 three complex floating point datatypes: `complex<float>`, `complex<double>`, and `complex<long double>`. *CFLT* is used below to designate one of the floating point datatypes.

The LIA-3 complex integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`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`)`	†
$plusitimes_{\mathrm{c}}(I)(x,y)$	x `+ II * `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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†

$max_{\mathrm{i}(I)}(x, y)$	`max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x, y)$	`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`minseq(`xs`)`	†

where x and y are expressions of type *CINT*.

The LIA-3 non-transcendental complex floating point operations are listed below, along with the syntax used to invoke them:

$itimes_F(x)$	`I * `x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`I * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`I * `x	†
$re_F(x)$	x`.real()` or `real(`x`)`	†
$re_{\mathrm{i}(F)}(x)$	x`.real()` or `real(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	x`.real()` or `real(`x`)`	⋆
$im_F(x)$	x`.imag()` or `imag(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	x`.imag()` or `imag(`x`)`	†
$im_{\mathrm{c}(F)}(x)$	x`.imag()` or `imag(`x`)`	⋆
$plusitimes_F(x, y)$	`complex(`x`, `y`)`	⋆
$plusitimes_F(x, y)$	x `+ I * `y`)`	⋆
$neg_{\mathrm{i}(F)}(x)$	`- `x	†
$neg_{\mathrm{c}(F)}(x)$	`- `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	†
$add_{F,\mathrm{c}(F)}(x, y)$	x `+ `y	⋆
$add_{\mathrm{i}(F),F}(x, y)$	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	†
$sub_{F,\mathrm{c}(F)}(x, y)$	x `- `y	⋆
$sub_{\mathrm{i}(F),F}(x, y)$	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	†
$mul_{F,\mathrm{c}(F)}(x, y)$	x `* `y	⋆
$mul_{\mathrm{i}(F),F}(x, y)$	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	†
$div_{F,\mathrm{c}(F)}(x,y)$	x `/` y	⋆
$div_{\mathrm{i}(F),F}(x,y)$	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	⋆
$eq_{\mathrm{i}(F)}(x,y)$	x `==` y	†
$eq_{F,\mathrm{i}(F)}(x,y)$	x `==` y	†
$eq_{F,\mathrm{c}(F)}(x,y)$	x `==` y	⋆
$eq_{\mathrm{i}(F),F}(x,y)$	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	†
$neq_{F,\mathrm{c}(F)}(x,y)$	x `!=` y	⋆
$neq_{\mathrm{i}(F),F}(x,y)$	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	†
$abs_{\mathrm{i}(F)}(x)$	`abs(`x`)`	⋆
$abs_{\mathrm{c}(F)}(x)$	`abs(`x`)`	⋆
$sqrabs_{\mathrm{c}(F)}(x)$	`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`)`	†

$polar_F(x,y)$	`polar(`x`,`y`)`	$\star$
$polaru_F(u,x,y)$	`polaru(`x`, `y`, `u`)`	†
$signum_F(x)$	`signb(`x`)`	$\star$?
$signum_{\mathrm{i}(F)}(x)$	`sign(`x`)`	†
$signum_{\mathrm{c}(F)}(x)$	`sign(`x`)`	†
$proj_F(x)$	`proj`t`(`x`)` or `cproj(`x`)`	† Not LIA-3
$proj_{\mathrm{i}(F)}(x)$	`cproj(`x`)`	† Not LIA-3
$proj_{\mathrm{c}(F)}(x)$	`cproj`t`(`x`)` or `cproj(`x`)`	† Not LIA-3

where x, y and z are expressions of the same complex floating point type.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::box_err_cmul()`	†
$max\_err\_mul_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::err_cmul()`	†
$box\_error\_mode\_div_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::box_err_cdiv()`	†
$max\_err\_div_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::err_cdiv()`	†
$max\_err\_exp_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::err_exp()`	†
$max\_err\_power_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::err_power()`	†
$max\_err\_sin_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::err_sin()`	†
$max\_err\_tan_{\mathrm{c}(F)}$	`numeric_limits<CFLT>::err_tan()`	†

where u is an expression of a floating point type. Several of the parameter functions are constant for each type (and library).

The LIA-3 elementary complex floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b,y)$	`cpoweri`t`(`b`, `y`)`	†
$exp_{\mathrm{i}(F)}(x)$	`exp(`x`)`	†
$exp_{\mathrm{c}(F)}(x)$	`exp(`x`)`	$\star$
$power_{\mathrm{i}(F)}(x,y)$	`pow(`x`, `y`)`	$\star$
$power_{\mathrm{i}(F),F}(b,y)$	`pow(`b`, `y`)`	$\star$
$power_{F,\mathrm{i}(F)}(b,x)$	`pow(`b`, `x`)`	$\star$
$power_{\mathrm{c}(F),F}(b,y)$	`pow(`b`, `y`)`	$\star$
$power_{F,\mathrm{c}(F)}(b,x)$	`pow(`b`, `x`)`	$\star$
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b,y)$	`pow(`b`, `y`)`	$\star$
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`pow(`b`, `x`)`	$\star$
$power_{\mathrm{c}(F)}(b,y)$	`pow(`b`, `y`)`	$\star$?
$sqrt_{F\to\mathrm{c}(F)}(x)$	`sqrt(`x`)`	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`sqrt(`x`)`	†
$sqrt_{\mathrm{c}(F)}(x)$	`sqrt(`x`)`	$\star$

$ln_{F\to c(F)}(x)$	`log(abs(`x`))+I*atan2(imag(`x`),`x`))`	$\star$
$ln_{i(F)\to c(F)}(x)$	`log(abs(`x`))+I*atan2(imag(`x`),real(`x`))`	$\star$
$ln_{i(F)\to c(F)}(x)$	`log(`x`)`	†
$ln_{c(F)}(x)$	`log(`x`)`	$\star$
$logbase_{F\to c(F)}(b,x)$	`Logc(`x`, `b`)` (note parameter order)	†
$logbase_{i(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{i(F),F}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{F,i(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{c(F),F}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{F,c(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{i(F),c(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{c(F),i(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{c(F)}(b,x)$	`logbase(`b`, `x`)`	†
$rad_{i(F)}(x)$	`radian(`x`)`	†
$rad_{c(F)}(x)$	`radian(`x`)`	†
$sin_{i(F)}(x)$	`sin(`x`)`	†
$sin_{c(F)}(x)$	`sin(`x`)`	$\star$
$cos_{i(F)}(x)$	`cos(`x`)`	†
$cos_{c(F)}(x)$	`cos(`x`)`	$\star$
$tan_{i(F)}(x)$	`tan(`x`)`	†
$tan_{c(F)}(x)$	`tan(`x`)`	$\star$
$cot_{i(F)}(x)$	`cot(`x`)`	†
$cot_{c(F)}(x)$	`cot(`x`)`	†
$sec_{i(F)}(x)$	`sec(`x`)`	†
$sec_{c(F)}(x)$	`sec(`x`)`	†
$csc_{i(F)}(x)$	`csc(`x`)`	†
$csc_{c(F)}(x)$	`csc(`x`)`	†
$arcsin_{F\to c(F)}(x)$	`asin(`x`)`	†
$arcsin_{i(F)}(x)$	`asin(`x`)`	†
$arcsin_{c(F)}(x)$	`asin(`x`)`	$\star$
$arccos_{F\to c(F)}(x)$	`acos(`x`)`	†
$arccos_{c(F)}(x)$	`acos(`x`)`	$\star$
$arctan_{i(F)}(x)$	`atan(`x`)`	†
$arctan_{i(F)\to c(F)}(x)$	`atanc(`x`)`	†
$arctan_{c(F)}(x)$	`atan(`x`)`	$\star$
$arccot_{i(F)}(x)$	`acot(`x`)`	†
$arccot_{i(F)\to c(F)}(x)$	`acotc(`x`)`	†
$arccot_{c(F)}(x)$	`acot(`x`)`	†
$arcsec_{F\to c(F)}(x)$	`asec(`x`)`	†
$arcsec_{i(F)\to c(F)}(x)$	`asec(`x`)`	†
$arcsec_{c(F)}(x)$	`asec(`x`)`	†
$arccsc_{F\to c(F)}(x)$	`acsc(`x`)`	†
$arccsc_{i(F)}(x)$	`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)	†
$tanh_{\mathrm{c}(F)}(x)$	`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)	†
$arcsinh_{\mathrm{i}(F)}(x)$	`asinh`(x)	†
$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`asinhc`(x)	†
$arcsinh_{\mathrm{c}(F)}(x)$	`asinh`(x)	⋆
$arccosh_{F\to\mathrm{c}(F)}(x)$	`acoshc`(x)	†
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`acoshc`(x)	†
$arccosh_{\mathrm{c}(F)}(x)$	`acosh`(x)	⋆
$arctanh_{F\to\mathrm{c}(F)}(x)$	`atanhc`(x)	†
$arctanh_{\mathrm{i}(F)}(x)$	`atanh`(x)	†
$arctanh_{\mathrm{c}(F)}(x)$	`atanh`(x)	⋆
$arccoth_{F\to\mathrm{c}(F)}(x)$	`acothc`(x)	†
$arccoth_{\mathrm{i}(F)}(x)$	`acoth`(x)	†
$arccoth_{\mathrm{c}(F)}(x)$	`acoth`(x)	†
$arcsech_{F\to\mathrm{c}(F)}(x)$	`asech`(x)	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`asech`(x)	†
$arcsech_{\mathrm{c}(F)}(x)$	`asech`(x)	†
$arccsch_{\mathrm{i}(F)}(x)$	`acsch`(x)	†
$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`acschc`(x)	†
$arccsch_{\mathrm{c}(F)}(x)$	`acsch`(x)	†

where b, x, and y are expressions of the same complex floating point type. t is a string, part of the operation name, and is "`f`" for `_Complex float`, the empty string for `_Complex double`, and "`l`" for `_Complex long float`.

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.

$convert_{I\to\mathrm{c}(I)}(x)$	`(`INT `_Complex)`x	†
$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`(`INT `_Complex)`x	†
$convert_{\mathrm{i}(I)\to\mathrm{i}(I')}(x)$	`(`$INT2$ `_Imaginary)`x	†
$convert_{\mathrm{c}(I)\to\mathrm{c}(I')}(x)$	`(`$INT2$ `_Complex)`x	†

$convert_{F \to \mathrm{c}(F)}(x)$	(*FLT* _Complex)x	$\star$
$convert_{\mathrm{i}(F) \to \mathrm{c}(F)}(x)$	(*FLT* _Complex)x	$(\star)$
$convert_{\mathrm{i}(F) \to \mathrm{i}(F')}(x)$	(*FLT*2 _Imaginary)x	$(\star)$
$convert_{\mathrm{c}(F) \to \mathrm{c}(F')}(x)$	(*FLT*2 _Complex)x	$\star$
$convert_{\mathrm{i}(F'') \to \mathrm{i}(F)}(x)$	sscanf...x	$(\star)$
$convert_{\mathrm{c}(F'') \to \mathrm{c}(F)}(x)$	sscanf...x	$\star$
$convert_{\mathrm{i}(F) \to \mathrm{i}(F'')}(x)$	sprintf...x	$(\star)$
$convert_{\mathrm{c}(F) \to \mathrm{c}(F'')}(x)$	sprintf...x	$\star$
$convert_{\mathrm{i}(D) \to \mathrm{i}(F)}(x)$	sscanf...(...x)	$(\star)$
$convert_{\mathrm{c}(D) \to \mathrm{c}(F)}(x)$	sscanf...(...x)	$\star$
$convert_{\mathrm{i}(F) \to \mathrm{i}(D)}(x)$	sprintf...(...x)	$(\star)$
$convert_{\mathrm{c}(F) \to \mathrm{c}(D)}(x)$	sprintf...(...x)	$\star$

complex IO...??

Numerals...:

$imaginary\_unit_{(I)}$	II or _Imaginary_II	†
$imaginary\_unit_{\mathrm{c}(I)}$	((II)) or _Complex_II	†
$imaginary\_unit_{\mathrm{i}(F)}$	I or _Imaginary_I	†
$imaginary\_unit_{\mathrm{c}(F)}$	_Complex_I	†

C.4 Fortran

The programming language Fortran is defined by ISO/IEC 1539-1:1997, *Information technology – Programming languages – Fortran – Part 1: Base language* [18].

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 one integer datatype, denoted as `INTEGER`, and two floating point data type denoted as `REAL` (single precision) and `DOUBLE PRECISION`.

An implementation is permitted to offer additional `INTEGER` types with a different range and additional `REAL` 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:

$itimes_{I\rightarrow \mathrm{i}(I)}(x)$	`II * `x	†
$itimes_{\mathrm{i}(I)\rightarrow I}(x)$	`II * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`II * `x	†
$re_I(x)$	`REALPART(`x`)`	†
$re_{\mathrm{i}(I)}(x)$	`REALPART(`x`)`	†
$re_{\mathrm{c}(I)}(x)$	`REALPART(`x`)`	†
$im_I(x)$	`IMAGPART(`x`)`	†
$im_{\mathrm{i}(I)}(x)$	`IMAGPART(`x`)`	†
$im_{\mathrm{c}(I)}(x)$	`IMAGPART(`x`)`	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	x `+ II * `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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	ABS(x)	†
$signum_I(x)$	SIGN(x)	†
$signum_{\mathrm{i}(I)}(x)$	SIGN(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)	†
$max_{\mathrm{i}(I)}(x, y)$	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($x_1, ..., x_n$)	†
$max\_seq_{\mathrm{i}(I)}(xs)$	MAX(xs)	†

$min\_seq_{\mathrm{i}(I)}(xs)$	`MIN(`xs`)`	†

where x and y are expressions of type `CINTEGER` and where xs is an expression of type array of `CINTEGER`.

The LIA-3 non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\to\mathrm{i}(F)}(x)$	`IU * `x	†
$itimes_{\mathrm{i}(F)\to F}(x)$	`IU * `x	†
$itimes_{\mathrm{c}(F)}(x)$	`IU * `x	†
$re_F(x)$	`REALPART(`x`)`	†
$re_{\mathrm{i}(F)}(x)$	`REALPART(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	`REALPART(`x`)`	⋆
$im_F(x)$	`IMAGPART(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	`IMAGPART(`x`)`	⋆
$im_{\mathrm{c}(F)}(x)$	`IMAGPART(`x`)`	⋆
$plusitimes_{\mathrm{c}(F)}(x,y)$	x `+ IU * `y`)`	⋆
$neg_{\mathrm{i}(F)}(x)$	`-`x	†
$neg_{\mathrm{c}(F)}(x)$	`-`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	⋆
$eq_{\mathrm{i}(F)}(x, y)$	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	†
$abs_{\mathrm{i}(F)}(x)$	`ABS(`x`)`	†
$abs_{\mathrm{c}(F)}(x)$	`ABS(`x`)`	⋆
$phase_F(x)$	`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`)`	†
$polar_F(x, y)$	`POLAR(`x`,` y`)`	†

$polaru_F(u,x,y)$	`POLARU(`x`, `y`, `u`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type array of *FLT*.

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 b, x and u are expressions of type *CFLT*. 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 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`)`	⋆
$power_{F\to\mathrm{c}(F)}(x,y)$	x `***` y	†
$power_{\mathrm{i}(F)}(x,y)$	x `**` y	†
$power_{\mathrm{i}(F),F}(b,y)$	b `**` y	†
$power_{F,\mathrm{i}(F)}(b,x)$	b `**` x	†
$power_{\mathrm{c}(F),F}(b,y)$	b `**` y	⋆?
$power_{F,\mathrm{c}(F)}(b,x)$	b `**` x	⋆?
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b,y)$	b `**` y	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	b `**` x	†
$power_{\mathrm{c}(F)}(x,y)$	x `**` y	⋆
$sqrt_{F\to\mathrm{c}(F)}(x)$	`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`)`	†
$ln_{F\to\mathrm{c}(F)}(x)$	`LOG(ABS(`x`))+I*ATAN2(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*ATAN2(Im(`x`),Re(`x`))`	†
$ln_{\mathrm{c}(F)}(x)$	`LOG(`x`)`	⋆

$logbase_{F\to c(F)}(b,x)$	`LOGC(`x`, `b`)` (note parameter order)	†
$logbase_{i(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{i(F),F}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{F,i(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{c(F),F}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{F,c(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{i(F),c(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{c(F),i(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{c(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$rad_{i(F)}(x)$	`RAD(`x`)`	†
$rad_{c(F)}(x)$	`RAD(`x`)`	†
$sin_{i(F)}(x)$	`SIN(`x`)`	†
$sin_{c(F)}(x)$	`SIN(`x`)`	⋆
$cos_{i(F)}(x)$	`COS(`x`)`	†
$cos_{c(F)}(x)$	`COS(`x`)`	⋆
$tan_{i(F)}(x)$	`TAN(`x`)`	†
$tan_{c(F)}(x)$	`TAN(`x`)`	⋆
$cot_{i(F)}(x)$	`COT(`x`)`	†
$cot_{c(F)}(x)$	`COT(`x`)`	⋆
$sec_{i(F)}(x)$	`SEC(`x`)`	†
$sec_{c(F)}(x)$	`SEC(`x`)`	†
$csc_{i(F)}(x)$	`CSC(`x`)`	†
$csc_{c(F)}(x)$	`CSC(`x`)`	†
$arcsin_{F\to c(F)}(x)$	`ASINC(`x`)`	†
$arcsin_{i(F)}(x)$	`ASIN(`x`)`	†
$arcsin_{c(F)}(x)$	`ASIN(`x`)`	⋆
$arccos_{F\to c(F)}(x)$	`ACOSC(`x`)`	†
$arccos_{i(F)\to c(F)}(x)$	`ACOS(`x`)`	†
$arccos_{c(F)}(x)$	`ACOS(`x`)`	⋆
$arctan_{i(F)}(x)$	`ATAN(`x`)`	†
$arctan_{i(F)\to c(F)}(x)$	`ATAN(`x`)`	†
$arctan_{c(F)}(x)$	`ATAN(`x`)`	⋆
$arccot_{i(F)}(x)$	`ACOT(`x`)`	†
$arccot_{i(F)\to c(F)}(x)$	`ACOT(`x`)`	†
$arccot_{c(F)}(x)$	`ACOT(`x`)`	⋆
$arcsec_{F\to c(F)}(x)$	`ASECC(`x`)`	†
$arcsec_{i(F)\to c(F)}(x)$	`ASEC(`x`)`	†
$arcsec_{c(F)}(x)$	`ASEC(`x`)`	†
$arccsc_{F\to c(F)}(x)$	`ACSCC(`x`)`	†
$arccsc_{i(F)}(x)$	`ACSC(`x`)`	†
$arccsc_{c(F)}(x)$	`ACSC(`x`)`	†
$radh_F(x)$	`RADH(`x`)`	†
$radh_{i(F)}(x)$	`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`)`	†
$arcsinh_{\mathrm{i}(F)}(x)$	`ASINH(`x`)`	†
$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`ASINHC(`x`)`	†
$arcsinh_{\mathrm{c}(F)}(x)$	`ASINH(`x`)`	⋆
$arccosh_{F\to\mathrm{c}(F)}(x)$	`ACOSHC(`x`)`	†
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`ACOSH(`x`)`	†
$arccosh_{\mathrm{c}(F)}(x)$	`ACOSH(`x`)`	⋆
$arctanh_{F\to\mathrm{c}(F)}(x)$	`ATANHC(`x`)`	†
$arctanh_{\mathrm{i}(F)}(x)$	`ATANH(`x`)`	†
$arctanh_{\mathrm{c}(F)}(x)$	`ATANH(`x`)`	⋆
$arccoth_{F\to\mathrm{c}(F)}(x)$	`ACOTHC(`x`)`	†
$arccoth_{\mathrm{i}(F)}(x)$	`ACOTH(`x`)`	†
$arccoth_{\mathrm{c}(F)}(x)$	`ACOTH(`x`)`	⋆
$arcsech_{F\to\mathrm{c}(F)}(x)$	`ASECHC(`x`)`	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`ASECH(`x`)`	†
$arcsech_{\mathrm{c}(F)}(x)$	`ASECH(`x`)`	†
$arccsch_{\mathrm{i}(F)}(x)$	`ACSCH(`x`)`	†
$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`ACSCHC(`x`)`	†
$arccsch_{\mathrm{c}(F)}(x)$	`ACSCH(`x`)`	†

where b, x, y, u, and v are expressions of type *CFLT*.

Arithmetic value conversions in Fortran are always explicit, and the conversion function is named like the target type, except when converting to/from strings.

(mockup so far!)		
$convert_{I\to\mathrm{c}(I)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(I)\to\mathrm{i}(I')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{c}(I)\to\mathrm{c}(I')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{F\to\mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	⋆
$convert_{F\to\mathrm{c}(F)}(x)$	x `+ i * Im(`x`)` or x `+ j * Im(`x`)`	†
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	⋆
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Re(`x`) +` x	⋆

$convert_{\mathrm{i}(F)\to\mathrm{i}(F')}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{c}(F)\to\mathrm{c}(F')}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{i}(F)\to\mathrm{i}(D)}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{c}(F)\to\mathrm{c}(D)}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{i}(D)\to\mathrm{i}(F)}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{c}(D)\to\mathrm{c}(F)}(x)$	`Compose_From_Cartesian`(x)	†

complex I/O!!!

Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`...`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`...`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`i`	†
$imaginary\_unit_{\mathrm{c}(F)}$	`-0+i`	†

C.5 Haskell

The programming language Haskell is defined by *Report on the programming language Haskell 98* [57], together with *Standard libraries for the Haskell 98 programming language* [58].

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 Haskell datatype `Bool` corresponds to the LIA datatype **Boolean**.

Every implementation of Haskell has at least two integer datatypes `Integer`, which is unlimited, and `Int`, and at least two floating point datatypes, `Float`, and `Double`. The notation *INT* is used to stand for the name of one of the integer datatypes, and *FLT* is used to stand for the name of one of the floating point datatypes in what follows.

Haskell has a type class system that allows for overloading, and allowing static type checking of dynamic overloading. But in contrast to object oriented programming languages, type classes are not types. E.g. + has the type `(Num a) => a -> a -> a`, where `Num` is a type class and `a` is a type variable. Datatypes for complex arithmetic are constructed via a type constructor, `Complex`, that takes a class restricted type parameter. The restriction, in Haskell 98, is to `RealFloat` (instead of `Real` which would have been more in line with Common Lisp), so complex integer datatypes cannot, in Haskell 98, be constructed via the standard type constructor `Complex`. Further, there is no standard `Imaginary` type constructor. The overloading system does not allow for "mixed" operand types.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\rightarrow \mathrm{i}(I)}(x)$	`I *` x	†
$itimes_{\mathrm{i}(I)\rightarrow I}(x)$	`I *`x	†
$itimes_{\mathrm{c}(I)}(x)$	`I *` x	†
$re_I(x)$	`realPart` x	†
$re_{\mathrm{i}(I)}(x)$	`realPart` x	†
$re_{\mathrm{c}(I)}(x)$	`realPart` x	†
$im_I(x)$	`imagPart` x	†
$im_{\mathrm{i}(I)}(x)$	`imagPart` x	†
$im_{\mathrm{c}(I)}(x)$	`imagPart` x	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	x `+ I *` y	†
$neg_{\mathrm{i}(I)}(x)$	`-` x or `negate` x	†
$neg_{\mathrm{c}(I)}(x)$	`-` x or `negate` 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 or `(+)` x y	†
$add_{I,\mathrm{i}(I)}(x,y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{i}(I),I}(x,y)$	x `+` y or `(+)` x y	†
$add_{I,\mathrm{c}(I)}(x,y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{c}(I),I}(x,y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x `+` y or `(+)` x y	†

$add_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{c}(I)}(x,y)$	x `+` y or `(+)` x y	†
$sub_{\mathrm{i}(I)}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{I,\mathrm{i}(I)}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{i}(I),I}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{I,\mathrm{c}(I)}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{c}(I),I}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{c}(I)}(x,y)$	x `-` y or `(-)` x y or `subtract` y x	†
$mul_{\mathrm{i}(I)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{I,\mathrm{i}(I)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{i}(I),I}(x,y)$	x `*` y or `(*)` x y	†
$mul_{I,\mathrm{c}(I)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{c}(I),I}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{c}(I)}(x,y)$	x `*` y or `(*)` x y	†
$eq_{\mathrm{i}(I)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{I,\mathrm{i}(I)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{i}(I),I}(x,y)$	x `==` y or `(==)` x y	†
$eq_{I,\mathrm{c}(I)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{c}(I),I}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{c}(I)}(x,y)$	x `==` y or `(==)` x y	†
$neq_{\mathrm{i}(I)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{I,\mathrm{i}(I)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{\mathrm{i}(I),I}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{I,\mathrm{c}(I)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{\mathrm{c}(I),I}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{\mathrm{c}(I)}(x,y)$	x `/=` y or `(/=)` 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	†
$abs_{\mathrm{i}(I)}(x)$	`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	†
$max_{\mathrm{i}(I)}(x, y)$	`max` x y	†
$min_{\mathrm{i}(I)}(x, y)$	`min` x y	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq` xs	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`maxseq` xs	†

where x and y are expressions of type *complex INT* [not in Haskell98].

The LIA-3 non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$itimes_F(x)$	`I *` x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`I *` x	†
$itimes_{\mathrm{c}(I)}(x)$	`I *` x	†
(The real and imaginary parts are also extractable through pattern matching.)		
$re_F(x)$	`realPart` x	†
$re_{\mathrm{i}(F)}(x)$	`realPart` x	†
$re_{\mathrm{c}(F)}(x)$	`realPart` x	⋆
$im_F(x)$	`imagPart` x	†
$im_{\mathrm{i}(F)}(x)$	`imagPart` x	†
$im_{\mathrm{c}(F)}(x)$	`imagPart` x	⋆
$plusitimes_{\mathrm{c}(F)}(x, y)$	x `:+` y	⋆
$neg_{\mathrm{i}(F)}(x)$	`-` x or `negate` x	†
$neg_{\mathrm{c}(F)}(x)$	`-` x or `negate` x	⋆
$conj_F(x)$	`conjugate` x	†
$conj_{\mathrm{i}(F)}(x)$	`conjugate` x	†
$conj_{\mathrm{c}(F)}(x)$	`conjugate` x	⋆
$add_{\mathrm{i}(F)}(x, y)$	x `+` y or `(+)` x y	†
$add_{F,\mathrm{i}(F)}(x, y)$	x `+` y or `(+)` x y	†
$add_{F,\mathrm{c}(F)}(x, y)$	x `+` y or `(+)` x y	⋆
$add_{\mathrm{i}(F),F}(x, y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{c}(F),F}(x, y)$	x `+` y or `(+)` x y	⋆
$add_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `+` y or `(+)` x y	†
$add_{\mathrm{c}(F)}(x, y)$	x `+` y or `(+)` x y	⋆
$sub_{\mathrm{i}(F)}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{F,\mathrm{i}(F)}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{F,\mathrm{c}(F)}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	⋆
$sub_{\mathrm{i}(F),F}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{c}(F),F}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	⋆
$sub_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	†
$sub_{\mathrm{c}(F)}(x, y)$	x `-` y or `(-)` x y or `subtract` y x	⋆

$mul_{\mathrm{i}(F)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{F,\mathrm{i}(F)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{F,\mathrm{c}(F)}(x,y)$	x `*` y or `(*)` x y	⋆
$mul_{\mathrm{i}(F),F}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{c}(F),F}(x,y)$	x `*` y or `(*)` x y	⋆
$mul_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `*` y or `(*)` x y	†
$mul_{\mathrm{c}(F)}(x,y)$	x `*` y or `(*)` x y	⋆
$div_{\mathrm{i}(F)}(x,y)$	x `/` y or `(/)` x y	†
$div_{F,\mathrm{i}(F)}(x,y)$	x `/` y or `(/)` x y	†
$div_{F,\mathrm{c}(F)}(x,y)$	x `/` y or `(/)` x y	⋆
$div_{\mathrm{i}(F),F}(x,y)$	x `/` y or `(/)` x y	†
$div_{\mathrm{c}(F),F}(x,y)$	x `/` y or `(/)` x y	⋆
$div_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `/` y or `(/)` x y	†
$div_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `/` y or `(/)` x y	†
$div_{\mathrm{c}(F)}(x,y)$	x `/` y or `(/)` x y	⋆
$eq_{\mathrm{i}(F)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{F,\mathrm{i}(F)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{F,\mathrm{c}(F)}(x,y)$	x `==` y or `(==)` x y	⋆
$eq_{\mathrm{i}(F),F}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{c}(F),F}(x,y)$	x `==` y or `(==)` x y	⋆
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `==` y or `(==)` x y	†
$eq_{\mathrm{c}(F)}(x,y)$	x `==` y or `(==)` x y	⋆
$neq_{\mathrm{i}(F)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{F,\mathrm{i}(F)}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{F,\mathrm{c}(F)}(x,y)$	x `/=` y or `(/=)` x y	⋆
$neq_{\mathrm{i}(F),F}(x,y)$	x `/=` y or `(/=)` x y	†
$neq_{\mathrm{c}(F),F}(x,y)$	x `/=` y or `(/=)` x y	⋆
$neq_{\mathrm{i}(F),\mathrm{c}(F)}(x,y)$	x `==` y or `(/=)` x y	†
$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x,y)$	x `==` y or `(/=)` x y	†
$neq_{\mathrm{c}(F)}(x,y)$	x `/=` y or `(/=)` 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	†
$abs_{\mathrm{i}(F)}(x)$	`magnitude` x	†
$abs_{\mathrm{c}(F)}(x)$	`magnitude` x	⋆
$abs_{\mathrm{c}(F)\to\mathrm{c}(F)}(x)$	`abs` x	⋆ Not LIA-3
$phase_F(x)$	`phase` x	†
$phase_{\mathrm{i}(F)}(x)$	`phase` x	†

$phase_{\mathrm{c}(F)}(x)$	`phase` x	$\star$
$phaseu_F(u,x)$	`phaseu(`x`,` u`)`	†
$phaseu_{\mathrm{i}(F)}(u,x)$	`phaseu(`x`,` u`)`	†
$phaseu_{\mathrm{c}(F)}(u,x)$	`phaseu(`x`,` u`)`	†
$signum_F(x)$	`sign` x	$\star$
$signum_{\mathrm{i}(F)}(x)$	`sign` x	†
$signum_{\mathrm{c}(F)}(x)$	`sign` x	$\star$
$polar_F(x,y)$	`mkPolar` x y	$\star$
$polaru_F(u,x,y)$	`mkPolaru` u x y	†
$to\_polar_F(z)$	`polar` z	$\star$Not LIA-3

where x, y and z are expressions of type `complex` *FLT*.

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` *FLT*. 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 elementary floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b,z)$	b `^^` z or `(^^)` b z	†
$power_{\mathrm{c}(F),I}(b,z)$	b `^^` z or `(^^)` b z	$\star$ Not LIA-3
$exp_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`exp` x	†
$exp_{\mathrm{i}(F)\to\mathrm{c}(F)}(\hat{\imath}\cdot x)$	`cis` x	$\star$
$exp_{\mathrm{c}(F)}(x)$	`exp` x	$\star$
$power_{\mathrm{i}(F)}(x,y)$	`pow(`x`,` y`)`	†
$power_{\mathrm{i}(F),F}(b,y)$	`pow(`b`,` y`)`	†
$power_{F,\mathrm{i}(F)}(b,x)$	`pow(`b`,` x`)`	†
$power_{\mathrm{c}(F),F}(b,y)$	`pow(`b`,` y`)`	†
$power_{F,\mathrm{c}(F)}(b,x)$	`pow(`b`,` x`)`	†
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b,y)$	`pow(`b`,` y`)`	$\star$
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`pow(`b`,` x`)`	$\star$
$power_{\mathrm{c}(F)}(b,y)$	`pow(`b`,` y`)`	$\star$

$sqrt_{F\to \mathrm{c}(F)}(x)$	`sqrtc` x	†
$sqrt_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`sqrt` x	†
$sqrt_{\mathrm{c}(F)}(x)$	`sqrt` x	⋆
$ln_{F\to \mathrm{c}(F)}(x)$	`log (abs` x`) :+ atan2 (imagPart` x`)` x	⋆
$ln_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`log (magnitude` x`) :+ atan2 (imagPart` x`) (realPart` x`)`	⋆
$ln_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`log` x	†
$ln_{\mathrm{c}(F)}(x)$	`log` x	⋆
$logbase_{F\to \mathrm{c}(F)}(b,x)$	`logBasec` b x	†
$logbase_{\mathrm{i}(F)}(b,x)$	`logBase` b x	†
$logbase_{\mathrm{i}(F),F}(b,x)$	`logBase` b x	†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`logBase` b x	†
$logbase_{\mathrm{c}(F),F}(b,x)$	`logBase` b x	†
$logbase_{F,\mathrm{c}(F)}(b,x)$	`logBase` b x	†
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`logBase` b x	†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`logBase` b x	†
$logbase_{\mathrm{c}(F)}(b,x)$	`logBase` b x	⋆
$rad_{\mathrm{i}(F)}(x)$	`radian` x	†
$rad_{\mathrm{c}(F)}(x)$	`radian` x	†
$sin_{\mathrm{i}(F)}(x)$	`s in` 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	†
$arccos_{\mathrm{c}(F)}(x)$	`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	†
$arcsec_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`asecc` 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	†
$tanh_{\mathrm{c}(F)}(x)$	`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	†
$arcsinh_{\mathrm{i}(F)}(x)$	`asinh` x	†
$arcsinh_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`asinhc` x	†
$arcsinh_{\mathrm{c}(F)}(x)$	`asinh` x	⋆
$arccosh_{F\to \mathrm{c}(F)}(x)$	`acoshc` x	†
$arccosh_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`acosh` x	†
$arccosh_{\mathrm{c}(F)}(x)$	`acosh` x	⋆
$arctanh_{F\to \mathrm{c}(F)}(x)$	`atanhc` x	†
$arctanh_{\mathrm{i}(F)}(x)$	`atanh` x	†
$arctanh_{\mathrm{c}(F)}(x)$	`atanh` x	⋆
$arccoth_{F\to \mathrm{c}(F)}(x)$	`acothc` x	†
$arccoth_{\mathrm{i}(F)}(x)$	`acoth` x	†
$arccoth_{\mathrm{c}(F)}(x)$	`acoth` x	†
$arcsech_{F\to \mathrm{c}(F)}(x)$	`asechc` x	†
$arcsech_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`asech` x	†
$arcsech_{\mathrm{c}(F)}(x)$	`asech` x	†
$arccsch_{\mathrm{i}(F)}(x)$	`acsch` x	†
$arccsch_{\mathrm{i}\to \mathrm{c}(F)}(x)$	`acschc` x	†
$arccsch_{\mathrm{c}(F)}(x)$	`acsch` x	†

where b, x, y, u, and v are expressions of type *complex FLT*.

Arithmetic value conversions in Haskell are always explicit. They are done with the overloaded `fromIntegral` and `fromFractional` operations.

$convert_{I\to \mathrm{c}(I)}(x)$	`..` x	†

$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`..` x	†
$convert_{F\to\mathrm{c}(F)}(x)$	x `- I * 0` or x `- _Imaginary_I * 0`	†
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`-0 +` x	†

...

complex IO...(show, read,)

where x is an expression of type *INT*, y is an expression of type *FLT*, and z is an expression of type *FXD*, where *FXD* is a fixed point type.

Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`II`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`0+II`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`I`	†
$imaginary\_unit_{\mathrm{c}(F)}$	`0+I`	†

C.6 Java

The programming language Java is defined by *The Java Language Specification* [56].

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. None of the LIA-3 operations are provided in the Java 2/SE (marked "⋆" below) library directly.

The Java datatype `boolean` corresponds to the LIA datatype **Boolean**.

Every implementation of Java has the integral datatypes `int`, and `long`.

Java has two floating point datatypes, `float` and `double`, which must conform to IEC 60559.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`I * `x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`I * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`I * `x	†
$re_I(x)$	`real(`x`)`	†
$re_{\mathrm{i}(I)}(x)$	`real(`x`)`	†
$re_{\mathrm{c}(I)}(x)$	`real(`x`)`	†
$im_I(x)$	`imag(`x`)`	†
$im_{\mathrm{i}(I)}(x)$	`imag(`x`)`	†
$im_{\mathrm{c}(I)}(x)$	`imag(`x`)`	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	x `+ I * `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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†
$max_{\mathrm{i}(I)}(x, y)$	`max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x, y)$	`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†

where x and y are expressions of type *CINT*.

The LIA-3 non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$itimes_F(x)$	`I * `x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`I * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`I * `x	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array of` *FLT*.

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`)`	†
$max\_err\_power_{\mathrm{c}(F)}(b, x)$	`err_power(`b`, `x`)`	†
$max\_err\_sin_{\mathrm{c}(F)}$	`err_sin(`x`)`	†
$max\_err\_tan_{\mathrm{c}(F)}$	`err_tan(`x`)`	†

where b, x and u are expressions of type *FLT*. 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 elementary floating point operations are listed below, along with the syntax used to invoke them. These are defined only for `double` not for `float`.

$power_{\mathrm{i}(F),I}(b, z)$	`pow(`b`, `z`)`	†
$exp_{\mathrm{i}(F)}(x)$	`exp(`x`)`	†
$exp_{\mathrm{c}(F)}(x)$	`exp(`x`)`	†
$power_{\mathrm{i}(F)}(x, y)$	`pow(`x`, `y`)`	†
$power_{\mathrm{i}(F),F}(b, y)$	`pow(`b`, `y`)`	†
$power_{F,\mathrm{i}(F)}(b, x)$	`pow(`b`, `x`)`	†
$power_{\mathrm{c}(F),F}(b, y)$	`pow(`b`, `y`)`	†
$power_{F,\mathrm{c}(F)}(b, x)$	`pow(`b`, `x`)`	†
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b, y)$	`pow(`b`, `y`)`	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b, x)$	`pow(`b`, `x`)`	†
$power_{\mathrm{c}(F)}(b, y)$	`pow(`b`, `y`)`	†
$sqrt_{F\to\mathrm{c}(F)}(x)$	`sqrt(`x`)`	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`sqrt(`x`)`	†

where b, x, y, u, and v are expressions of type *CFLT*.

Arithmetic value conversions in Java can be explicit or implicit. The rules for when implicit conversions are applied is not repeated here. The explicit arithmetic value conversions are usually expressed as 'casts', except when converting to/from strings.

$convert_{I \to \mathrm{c}(I)}(x)$	`..(`x`)`	†
$convert_{\mathrm{i}(I) \to \mathrm{c}(I)}(x)$	`..(`x`)`	†
$convert_{F \to \mathrm{c}(F)}(x)$	x `- I * 0` or x `- _Imaginary_I * 0`	†
$convert_{\mathrm{i}(F) \to \mathrm{c}(F)}(x)$	`-0 +` x	†

...

complex IO...??

where x is an expression of type *INT*, y is an expression of type *FLT*, and z is an expression of type *FXD*, where *FXD* is a fixed point type. *INT2* is the integer datatype that corresponds to I'. A ? above indicates that the parameter is optional. e is greater than 0.

Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`II`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`0 + II`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`I`	†
$imaginary\_unit_{\mathrm{c}(F)}$	`0 + I`	†

C.7 Common Lisp

The programming language Common Lisp is defined by ANSI X3.226-1994, *Information Technology – Programming Language – Common Lisp* [38].

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-2 for that operation. For each of the marked items a suggested identifier is provided.

Common Lisp does not have a single datatype that corresponds to the LIA-1 datatype **Boolean**. Rather, `NIL` corresponds to **false** and `T` corresponds to **true**.

Every implementation of Common Lisp has one unbounded integer datatype. Any mathematical integer is assumed to have a representation as a Common Lisp data object, subject only to total memory limitations.

Common Lisp has four floating point types: `short-float`, `single-float`, `double-float`, and `long-float`. Not all of these floating point types must be distinct.

The complex integer (Gaussian integer) operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`(i` x`)`	†
$itimes_{\mathrm{i}(I)}(x)$	`(i` x`)`	†
$itimes_{\mathrm{c}(I)}(x)$	`(i` x`)`	†
$re_I(x)$	`(realpart` x`)`	⋆
$re_{\mathrm{i}(I)}(x)$	`(realpart` x`)`	†
$re_{\mathrm{c}(I)}(x)$	`(realpart` x`)`	⋆
$im_I(x)$	`(imagpart` x`)`	⋆
$im_{\mathrm{i}(I)}(x)$	`(imagpart` x`)`	†
$im_{\mathrm{c}(I)}(x)$	`(imagpart` x`)`	⋆
$plusitimes_{\mathrm{c}(I)}(x,y)$	`(complex` x 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)`	$\star$
$sub_{\mathrm{c}(I),I}(x,y)$	`(- x y)`	$\star$
$sub_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`(- x y)`	†
$sub_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`(- x y)`	†
$sub_{\mathrm{c}(I)}(x,y)$	`(- x y)`	$\star$
$mul_{\mathrm{i}(I)}(x,y)$	`(* 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)`	$\star$
$mul_{\mathrm{c}(I),I}(x,y)$	`(* x y)`	$\star$
$mul_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`(* x y)`	†
$mul_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`(* x y)`	†
$mul_{\mathrm{c}(I)}(x,y)$	`(* x y)`	$\star$
$eq_{\mathrm{i}(I)}(x,y)$	`(= 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)`	$\star$
$eq_{\mathrm{c}(I),I}(x,y)$	`(= x y)`	$\star$
$eq_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`(= x y)`	†
$eq_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`(= x y)`	†
$eq_{\mathrm{c}(I)}(x,y)$	`(= x y)`	$\star$
$neq_{\mathrm{i}(I)}(x,y)$	`(/= 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)`	$\star$
$neq_{\mathrm{c}(I),I}(x,y)$	`(/= x y)`	$\star$
$neq_{\mathrm{i}(I),\mathrm{c}(I)}(x,y)$	`(/= x y)`	†
$neq_{\mathrm{c}(I),\mathrm{i}(I)}(x,y)$	`(/= x y)`	†
$neq_{\mathrm{c}(I)}(x,y)$	`(/= x y)`	$\star$
$lss_{\mathrm{i}(I)}(x,y)$	`(< 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)`	†
$abs_{\mathrm{i}(I)}(x)$	`(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)`	†
$max_{\mathrm{i}(I)}(x,y)$	`(max x y)`	†
$min_{\mathrm{i}(I)}(x,y)$	`(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 *CINT*.

The LIA-3 non-transcendental complex floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\to\mathrm{i}(F)}(x)$	`(i` x`)`	⋆
$itimes_{\mathrm{i}(F)\to F}(x)$	`(i` x`)`	⋆
$itimes_{\mathrm{c}(F)}(x)$	`(i` x`)`	⋆
$re_F(x)$	`(realpart` x`)`	⋆
$re_{\mathrm{i}(F)}(x)$	`(realpart` x`)`	†
$re_{\mathrm{c}(F)}(x)$	`(realpart` x`)`	⋆
$im_F(x)$	`(imagpart` x`)`	⋆
$im_{\mathrm{i}(F)}(x)$	`(imagpart` x`)`	†
$im_{\mathrm{c}(F)}(x)$	`(imagpart` x`)`	⋆
$plusitimes_{\mathrm{c}(F)}(x,y)$	`(complex` x y`)`	⋆
$neg_{\mathrm{i}(F)}(x)$	`(-` x`)`	†
$neg_{\mathrm{c}(F)}(x)$	`(-` x`)`	⋆
$conj_F(x)$	`(conjugate` x`)`	⋆
$conj_{\mathrm{i}(F)}(x)$	`(conjugate` x`)`	†
$conj_{\mathrm{c}(F)}(x)$	`(conjugate` 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`)`	⋆
$eq_{\mathrm{i}(F)}(x, y)$	`(=` x y`)`	†
$eq_{F,\mathrm{i}(F)}(x, y)$	`(=` x y`)`	†
$eq_{F,\mathrm{c}(F)}(x, y)$	`(=` x y`)`	⋆
$eq_{\mathrm{i}(F),F}(x, y)$	`(=` 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`)`	†
$neq_{F,\mathrm{c}(F)}(x, y)$	`(/=` x y`)`	⋆
$neq_{\mathrm{i}(F),F}(x, y)$	`(/=` 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`)`	†
$abs_{\mathrm{i}(F)}(x)$	`(abs` x`)`	†
$abs_{\mathrm{c}(F)}(x)$	`(abs` x`)`	⋆
$phase_F(x)$	`(phase` x`)`	⋆
$phase_{\mathrm{i}(F)}(x)$	`(phase` x`)`	†
$phase_{\mathrm{c}(F)}(x)$	`(phase` x`)`	⋆
$phaseu_F(u, x)$	`(phase` u x`)`	†
$phaseu_{\mathrm{i}(F)}(u, x)$	`(phase` u x`)`	†
$phaseu_{\mathrm{c}(F)}(u, x)$	`(phase` u x`)`	†
$signum_F(x)$	`(signum` x`)`	⋆
$signum_{\mathrm{i}(F)}(x)$	`(signum` x`)`	†
$signum_{\mathrm{c}(F)}(x)$	`(signum` x`)`	⋆

$polar_F(x, y)$	`(polar` x y`)`	†
$polaru_F(u, x, y)$	`(polaru` u x y`)`	†

where x, y and z are data objects of the same floating point type, and where xs is a data objects that is a list of data objects of (the same, in this binding) floating point type. Note that Common Lisp allows mixed number types in many of its operations. This example binding does not explain that in detail.

The LIA-3 parameters for operations approximating complex real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{\mathrm{c}(F)}$	`(err-mode-mul` x`)`	†
$max\_err\_mul_{\mathrm{c}(F)}$	`(err-mul` x`)`	†
$box\_error\_mode\_div_{\mathrm{c}(F)}$	`(err-mode-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 b, x and u are expressions of type *CFLT*. 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 elementary complex floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b, a)$	`(expt` b a`)`	†
$power_{\mathrm{c}(F),I}(x, a)$	`(expt` x a`)`	⋆ Not LIA-3
$exp_{\mathrm{i}(F)}(x)$	`(exp` x`)`	†
$exp_{\mathrm{i}(F)}(\hat{\mathbf{1}}\cdot v)$	`(cis` v`)`	⋆
$exp_{\mathrm{c}(F)}(x)$	`(exp` x`)`	⋆
$power_{F\to\mathrm{c}(F)}(x, y)$	`(exptc` x y`)`	†
$power_{\mathrm{i}(F)}(x, y)$	`(expt` x y`)`	†
$power_{\mathrm{i}(F),F}(b, y)$	`(expt` b y`)`	†
$power_{F,\mathrm{i}(F)}(b, x)$	`(expt` b x`)`	†
$power_{\mathrm{c}(F),F}(b, y)$	`(expt` b y`)`	⋆
$power_{F,\mathrm{c}(F)}(b, x)$	`(expt` b x`)`	⋆
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b, y)$	`(expt` b y`)`	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b, x)$	`(expt` b x`)`	†
$power_{\mathrm{c}(F)}(b, y)$	`(expt` b y`)` (deviation: `(expt 0.0 0.0)` is 1)	⋆
$sqrt_{F\to\mathrm{c}(F)}(x)$	`(sqrtc` x`)`	⋆
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(sqrt` x`)`	†
$sqrt_{\mathrm{c}(F)}(x)$	`(sqrt` x`)`	⋆
$ln_{F\to\mathrm{c}(F)}(x)$	`(complex (log (abs` x`)) (atan (imag` x`) *x))`	

$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(complex (log (abs` x`)) (atan (imag` x`)`	†`(real` x`))`
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(log` x`)`	†
$ln_{\mathrm{c}(F)}(x)$	`(log` x`)`	⋆
$logbase_{F\to\mathrm{c}(F)}(b,x)$	`(logc` x b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F)}(b,x)$	`(log` x b`)`	†
$logbase_{\mathrm{i}(F),F}(b,x)$	`(log` x b`)`	†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`(log` x b`)`	†
$logbase_{\mathrm{c}(F),F}(b,x)$	`(log` x b`)`	⋆
$logbase_{F,\mathrm{c}(F)}(b,x)$	`(log` x b`)`	⋆
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`(log` x b`)`	†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`(log` x b`)`	†
$logbase_{\mathrm{c}(F)}(b,x)$	`(log` x b`)`	⋆
$rad_{\mathrm{i}(F)}(x)$	`(radians` x`)`	†
$rad_{\mathrm{c}(F)}(x)$	`(radians` 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`)`	†
$arcsin_{F\to\mathrm{c}(F)}(x)$	`(asin` x`)`	⋆
$arcsin_{\mathrm{i}(F)}(x)$	`(asin` x`)`	†
$arcsin_{\mathrm{c}(F)}(x)$	`(asin` x`)`	⋆
$arccos_{F\to\mathrm{c}(F)}(x)$	`(acos` x`)`	⋆
$arccos_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(acos` x`)`	(⋆)
$arccos_{\mathrm{c}(F)}(x)$	`(acos` x`)`	⋆
$arctan_{\mathrm{i}(F)}(x)$	`(atanx` x`)`	†
$arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(atan` x`)`	†
$arctan_{\mathrm{c}(F)}(x)$	`(atan` x`)`	⋆
$arccot_{\mathrm{i}(F)}(x)$	`(acot` x`)`	†
$arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(acot` x`)`	†
$arccot_{\mathrm{c}(F)}(x)$	`(acot` x`)`	†
$arcsec_{F\to\mathrm{c}(F)}(x)$	`(asecc` x`)`	†
$arcsec_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(asecc` 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`)`	†
$radh_F(x)$	`(hypradians` x`)`	†
$radh_{\mathrm{i}(F)}(x)$	`(hypradians` x`)`	†
$radh_{\mathrm{c}(F)}(x)$	`(hypradians` 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`)`	†
$arcsinh_{\mathrm{i}(F)}(x)$	`(asinhx` x`)`	†
$arcsinh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(asinh` x`)`	†
$arcsinh_{\mathrm{c}(F)}(x)$	`(asinh` x`)`	⋆
$arccosh_{F\to\mathrm{c}(F)}(x)$	`(acosh` x`)`	⋆
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(acosh` x`)`	†
$arccosh_{\mathrm{c}(F)}(x)$	`(acosh` x`)`	⋆
$arctanh_{F\to\mathrm{c}(F)}(x)$	`(atanh` x`)`	⋆
$arctanh_{\mathrm{i}(F)}(x)$	`(atanh` x`)`	†
$arctanh_{\mathrm{c}(F)}(x)$	`(atanh` x`)`	⋆
$arccoth_{F\to\mathrm{c}(F)}(x)$	`(acoth` x`)`	†
$arccoth_{\mathrm{i}(F)}(x)$	`(acoth` x`)`	†
$arccoth_{\mathrm{c}(F)}(x)$	`(acoth` x`)`	†
$arcsech_{F\to\mathrm{c}(F)}(x)$	`(asech` x`)`	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(asech` x`)`	†
$arcsech_{\mathrm{c}(F)}(x)$	`(asech` x`)`	†
$arccsch_{\mathrm{i}(F)}(x)$	`(acschx` x`)`	†
$arccsch_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(acsch` x`)`	†
$arccsch_{\mathrm{c}(F)}(x)$	`(acsch` x`)`	†

where b, x, y, u, and v are expressions of type *CFLT*.

Arithmetic value conversions in Common Lisp are can be explicit or implicit. The rules for when implicit conversions are done is implementation defined.

$convert_{I\to\mathrm{c}(I)}(x)$	`(complex` x`)`	⋆
$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`(complex` x`)`	†
$convert_{F\to\mathrm{c}(F)}(x)$	`(complex` x`)`	⋆
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(complex` x`)`	†

complex I/O!!!

Numerals...

$imaginary\_unit_{\mathrm{i}(I)}$	`...`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`#C(0 1)`	⋆
$imaginary\_unit_{\mathrm{i}(F)}$	`...`	†
$imaginary\_unit_{\mathrm{c}(F)}$	`#C(0.0 1.0)`	⋆

In general two non-complex numerals, a and b, can be composed to a complex numeral `#C(`a b`)`.

C.8 ISLisp

The programming language ISLisp is defined by ISO/IEC 13816:1997, *Information technology – Programming languages, their environments and system software interfaces – Programming language ISLISP* [20].

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.

ISLisp does not have a single datatype that corresponds to the LIA datatype **Boolean**. Rather, `NIL` corresponds to **false** and `T` corresponds to **true**.

Every implementation of ISLisp has one unbounded integer datatype. Any mathematical integer is assumed to have a representation as a ISLisp data object, subject only to total memory limitations.

ISLisp has one floating point type required to conform to IEC 60559.

The complex integer (Gaussian integer) operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`(i` x`)`	†
$itimes_{\mathrm{i}(I)}(x)$	`(i` x`)`	†
$itimes_{\mathrm{c}(I)}(x)$	`(i` x`)`	†
$re_I(x)$	`(realpart` x`)`	†
$re_{\mathrm{i}(I)}(x)$	`(realpart` x`)`	†
$re_{\mathrm{c}(I)}(x)$	`(realpart` x`)`	†
$im_I(x)$	`(imagpart` x`)`	†
$im_{\mathrm{i}(I)}(x)$	`(imagpart` x`)`	†
$im_{\mathrm{c}(I)}(x)$	`(imagpart` x`)`	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	`(complex` x 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`)`	†
$eq_{\mathrm{i}(I)}(x,y)$	`(=` 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`)`	†
$abs_{\mathrm{i}(I)}(x)$	`(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`)`	†
$max_{\mathrm{i}(I)}(x,y)$	`(max` x y`)`	†
$min_{\mathrm{i}(I)}(x,y)$	`(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 *CINT*.

The LIA-3 non-transcendental complex floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\rightarrow\mathrm{i}(F)}(x)$	`(i` x`)`	†
$itimes_{\mathrm{i}(F)\rightarrow F}(x)$	`(i` x`)`	†
$itimes_{\mathrm{c}(F)}(x)$	`(i` x`)`	†
$re_F(x)$	`(realpart` x`)`	†
$re_{\mathrm{i}(F)}(x)$	`(realpart` x`)`	†
$re_{\mathrm{c}(F)}(x)$	`(realpart` x`)`	†
$im_F(x)$	`(imagpart` x`)`	†
$im_{\mathrm{i}(F)}(x)$	`(imagpart` x`)`	†
$im_{\mathrm{c}(F)}(x)$	`(imagpart` x`)`	†
$plusitimes_{\mathrm{c}(F)}(x,y)$	`(complex` x y`)`	†
$neg_{\mathrm{i}(F)}(x)$	`(-` x`)`	†
$neg_{\mathrm{c}(F)}(x)$	`(-` x`)`	†
$conj_F(x)$	`(conjugate` x`)`	†
$conj_{\mathrm{i}(F)}(x)$	`(conjugate` x`)`	†
$conj_{\mathrm{c}(F)}(x)$	`(conjugate` 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`)`	†
$eq_{\mathrm{i}(F)}(x, y)$	`(=` x y`)`	†
$eq_{F,\mathrm{i}(F)}(x, y)$	`(=` x y`)`	†
$eq_{F,\mathrm{c}(F)}(x, y)$	`(=` x y`)`	†
$eq_{\mathrm{i}(F),F}(x, y)$	`(=` 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`)`	†
$neq_{F,\mathrm{c}(F)}(x, y)$	`(/=` x y`)`	†
$neq_{\mathrm{i}(F),F}(x, y)$	`(/=` 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`)`	†
$abs_{\mathrm{i}(F)}(x)$	`(abs` x`)`	†
$abs_{\mathrm{c}(F)}(x)$	`(abs` x`)`	†
$phase_F(x)$	`(phase` x`)`	†
$phase_{\mathrm{i}(F)}(x)$	`(phase` x`)`	†
$phase_{\mathrm{c}(F)}(x)$	`(phase` x`)`	†
$phaseu_F(u, x)$	`(phase` u x`)`	†
$phaseu_{\mathrm{i}(F)}(u, x)$	`(phase` u x`)`	†
$phaseu_{\mathrm{c}(F)}(u, x)$	`(phase` u x`)`	†
$signum_F(x)$	`(signum` x`)`	†
$signum_{\mathrm{i}(F)}(x)$	`(signum` x`)`	†
$signum_{\mathrm{c}(F)}(x)$	`(signum` x`)`	†

$polar_F(x, y)$	`(polar` x y`)`	†
$polaru_F(u, x, y)$	`(polaru` u x y`)`	†

where x, y and z are data objects of the same floating point type, and where xs is a data objects that is a list of data objects of (the same, in this binding) floating point type.

The LIA-3 parameters for operations approximating complex real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{\mathrm{c}(F)}$	`(err-mode-mul` x`)`	†
$max\_err\_mul_{\mathrm{c}(F)}$	`(err-mul` x`)`	†
$box\_error\_mode\_div_{\mathrm{c}(F)}$	`(err-mode-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 b, x and u are expressions of type *CFLT*. 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 elementary complex floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b, a)$	`(expt` b a`)`	†
$power_{\mathrm{c}(F),I}(x, a)$	`(expt` x a`)`	† Not LIA-3
$exp_{\mathrm{i}(F)}(x)$	`(exp` x`)`	†
$exp_{\mathrm{i}(F)}(\hat{\mathbf{i}}\cdot v)$	`(cis` v`)`	†
$exp_{\mathrm{c}(F)}(x)$	`(exp` x`)`	†
$power_{F\to\mathrm{c}(F)}(x, y)$	`(exptc` x y`)`	†
$power_{\mathrm{i}(F)}(x, y)$	`(expt` x y`)`	†
$power_{\mathrm{i}(F),F}(b, y)$	`(expt` b y`)`	†
$power_{F,\mathrm{i}(F)}(b, x)$	`(expt` b x`)`	†
$power_{\mathrm{c}(F),F}(b, y)$	`(expt` b y`)`	†
$power_{F,\mathrm{c}(F)}(b, x)$	`(expt` b x`)`	†
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b, y)$	`(expt` b y`)`	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b, x)$	`(expt` b x`)`	†
$power_{\mathrm{c}(F)}(b, y)$	`(expt` b y`)` (deviation: `(expt 0.0 0.0)` is 1)	†
$sqrt_{F\to\mathrm{c}(F)}(x)$	`(sqrtc` x`)`	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(sqrt` x`)`	†
$sqrt_{\mathrm{c}(F)}(x)$	`(sqrt` x`)`	†
$ln_{F\to\mathrm{c}(F)}(x)$	`(complex (log (abs` x`)) (atan (imag` x`) `x`))`	†
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(complex (log (abs` x`)) (atan (imag` x`) (real` x`))`	†
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(log` x`)`	†

$ln_{\mathrm{c}(F)}(x)$	`(log` x`)`		†
$logbase_{F\rightarrow\mathrm{c}(F)}(b,x)$	`(logc` x b`)`	(note parameter order)	†
$logbase_{\mathrm{i}(F)}(b,x)$	`(log` x b`)`		†
$logbase_{\mathrm{i}(F),F}(b,x)$	`(log` x b`)`		†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`(log` x b`)`		†
$logbase_{\mathrm{c}(F),F}(b,x)$	`(log` x b`)`		†
$logbase_{F,\mathrm{c}(F)}(b,x)$	`(log` x b`)`		†
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`(log` x b`)`		†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`(log` x b`)`		†
$logbase_{\mathrm{c}(F)}(b,x)$	`(log` x b`)`		†
$rad_{\mathrm{i}(F)}(x)$	`(radians` x`)`		†
$rad_{\mathrm{c}(F)}(x)$	`(radians` 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`)`		†
$arcsin_{F\rightarrow\mathrm{c}(F)}(x)$	`(asinc` x`)`		†
$arcsin_{\mathrm{i}(F)}(x)$	`(asin` x`)`		†
$arcsin_{\mathrm{c}(F)}(x)$	`(asin` x`)`		†
$arccos_{F\rightarrow\mathrm{c}(F)}(x)$	`(acosc` x`)`		†
$arccos_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(acos` x`)`		†
$arccos_{\mathrm{c}(F)}(x)$	`(acos` x`)`		†
$arctan_{\mathrm{i}(F)}(x)$	`(atan` x`)`		†
$arctan_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(atanc` x`)`		†
$arctan_{\mathrm{c}(F)}(x)$	`(atan` x`)`		†
$arccot_{\mathrm{i}(F)}(x)$	`(acot` x`)`		†
$arccot_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(acotc` x`)`		†
$arccot_{\mathrm{c}(F)}(x)$	`(acot` x`)`		†
$arcsec_{F\rightarrow\mathrm{c}(F)}(x)$	`(asecc` x`)`		†
$arcsec_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(asecc` x`)`		†
$arcsec_{\mathrm{c}(F)}(x)$	`(asec` x`)`		†
$arccsc_{F\rightarrow\mathrm{c}(F)}(x)$	`(acscc` x`)`		†
$arccsc_{\mathrm{i}(F)}(x)$	`(acsc` x`)`		†
$arccsc_{\mathrm{c}(F)}(x)$	`(acsc` x`)`		†

$radh_{F}(x)$	`(hypradians` x`)`	†
$radh_{\mathrm{i}(F)}(x)$	`(hypradians` x`)`	†
$radh_{\mathrm{c}(F)}(x)$	`(hypradians` 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`)`	†
$arcsinh_{\mathrm{i}(F)}(x)$	`(asinhx` x`)`	†
$arcsinh_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(asinh` x`)`	†
$arcsinh_{\mathrm{c}(F)}(x)$	`(asinh` x`)`	†
$arccosh_{F\rightarrow\mathrm{c}(F)}(x)$	`(acosh` x`)`	†
$arccosh_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(acosh` x`)`	†
$arccosh_{\mathrm{c}(F)}(x)$	`(acosh` x`)`	†
$arctanh_{F\rightarrow\mathrm{c}(F)}(x)$	`(atanh` x`)`	†
$arctanh_{\mathrm{i}(F)}(x)$	`(atanh` x`)`	†
$arctanh_{\mathrm{c}(F)}(x)$	`(atanh` x`)`	†
$arccoth_{F\rightarrow\mathrm{c}(F)}(x)$	`(acoth` x`)`	†
$arccoth_{\mathrm{i}(F)}(x)$	`(acoth` x`)`	†
$arccoth_{\mathrm{c}(F)}(x)$	`(acoth` x`)`	†
$arcsech_{F\rightarrow\mathrm{c}(F)}(x)$	`(asech` x`)`	†
$arcsech_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(asech` x`)`	†
$arcsech_{\mathrm{c}(F)}(x)$	`(asech` x`)`	†
$arccsch_{\mathrm{i}(F)}(x)$	`(acschx` x`)`	†
$arccsch_{\mathrm{i}(F)\rightarrow\mathrm{c}(F)}(x)$	`(acsch` x`)`	†
$arccsch_{\mathrm{c}(F)}(x)$	`(acsch` x`)`	†

where b, x, y, u, and v are expressions of type *CFLT*.

Arithmetic value conversions in ISLisp are can be explicit or implicit. The rules for when implicit conversions are done is implementation defined.

$convert_{I\rightarrow c(I)}(x)$	`(complex` x`)`	†
$convert_{i(I)\rightarrow c(I)}(x)$	`...`x`)`	†
$convert_{F\rightarrow c(F)}(x)$	`(complex` x`)`	†
$convert_{i(F)\rightarrow c(F)}(x)$	`...`x`)`	†

Numerals...

$imaginary\_unit_{i(I)}$	`...`	†
$imaginary\_unit_{c(I)}$	`#C(0 1)`	†
$imaginary\_unit_{i(F)}$	`...`	†

$imaginary\_unit_{c(F)}$	`#C(0.0 1.0)`	†

In general two non-complex numerals, a and b, can be composed to a complex numeral `#C(`a b`)`.

C.9 Modula-2

The programming language Modula-2 is defined by ISO/IEC 10514-1:1996, *Information technology – Programming languages - Part 1: Modula-2, Base Language* [21]. 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 Modula-2 datatype `Boolean` corresponds to the LIA datatype **Boolean**.

Modula-2 has two floating point datatypes, `REAL` (and `LONGREAL`, and two complex datatypes based on these floating point datatypes, `COMPLEX` and `LONGCOMPLEX`.

The LIA-3 complex integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`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`)`	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	x `+ II * `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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†
$max_{\mathrm{i}(I)}(x, y)$	`max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x, y)$	`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†

where x and y are expressions of type *INT* and where xs is an expression of type `array [] of` *INT*.

The LIA-3 basic complex non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\to \mathrm{i}(F)}(x)$	`i * `x	†
$itimes_{\mathrm{i}(F)\to F}(x)$	`i * `x	†
$itimes_{\mathrm{c}(F)}(x)$	`i * `x	†
$re_{F}(x)$	`Re(`x`)`	†
$re_{\mathrm{i}(F)}(x)$	`Re(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	`Re(`x`)`	⋆
$im_{F}(x)$	`Im(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	`Im(`x`)`	†
$im_{\mathrm{c}(F)}(x)$	`Im(`x`)`	⋆
$plusitimes_{\mathrm{c}(F)}(x,y)$	x `+ I * `y	†
$neg_{\mathrm{i}(F)}(x)$	`- `x	†
$neg_{\mathrm{c}(F)}(x)$	`- `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	†
$mul_{F,\mathrm{c}(F)}(x,y)$	`ScalarMult(`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	⋆
$eq_{\mathrm{i}(F)}(x,y)$	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	†
$abs_{\mathrm{i}(F)}(x)$	`abs(`x`)`	†
$abs_{\mathrm{c}(F)}(x)$	`abs(`x`)`	⋆
$phase_F(x)$	`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`)`	†
$signum_F(x)$	`Signum(`x`)`	†
$signum_{\mathrm{i}(F)}(x)$	`Signum(`x`)`	†
$signum_{\mathrm{c}(F)}(x)$	`Signum(`x`)`	†
$polar_F(x,y)$	`polarToComplex(`x`,` y`)`	⋆
$polaru_F(u,x,y)$	`polarToComplex(`x`,` y`,` u`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array []`

`of` *FLT*.

The LIA-3 parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{c(F)}$	`box_err_mul(`x`)`	†
$max\_err\_mul_{c(F)}$	`err_mul(`x`)`	†
$box\_error\_mode\_div_{c(F)}$	`box_err_div(`x`)`	†
$max\_err\_div_{c(F)}$	`err_div(`x`)`	†
$max\_err\_exp_{c(F)}$	`err_exp(`x`)`	†
$max\_err\_power_{c(F)}$	`err_power(`x`)`	†
$max\_err\_sin_{c(F)}$	`err_sin(`x`)`	†
$max\_err\_tan_{c(F)}$	`err_tan(`x`)`	†

where x and u are expressions of type *FLT*. 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 elementary floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(x,y)$	x `**` y	†
$exp_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`exp(`x`)`	†
$exp_{\mathrm{c}(F)}(x)$	`exp(`x`)`	⋆
$power_{F\to\mathrm{c}(F)}(x,y)$	`powerc(`x`, `y`)`	†
$power_{\mathrm{i}(F)}(x,y)$	`power(`x`, `y`)`	†
$power_{\mathrm{i}(F),F}(b,y)$	`power(`b`, `y`)`	†
$power_{F,\mathrm{i}(F)}(b,x)$	`power(`b`, `x`)`	†
$power_{\mathrm{c}(F),F}(b,y)$	`power(`b`, `y`)`	⋆
$power_{F,\mathrm{c}(F)}(b,x)$	`power(`b`, `x`)`	†
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b,y)$	`power(`b`, `y`)`	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`power(`b`, `x`)`	†
$power_{\mathrm{c}(F)}(x,y)$	`power(`x`, `y`)`	†
$sqrt_{F\to\mathrm{c}(F)}(x)$	`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`)`	†
$ln_{F\to\mathrm{c}(F)}(x)$	`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`))`	†
$ln_{\mathrm{c}(F)}(x)$	`ln(`x`)`	⋆
$logbase_{F\to\mathrm{c}(F)}(b,x)$	`logc(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F)}(b,x)$	`log(`x`, `b`)` (note parameter order)	†

$logbase_{\mathrm{i}(F),F}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$logbase_{\mathrm{c}(F),F}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$logbase_{F,\mathrm{c}(F)}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$logbase_{\mathrm{c}(F)}(b,x)$	`log(`x`, `b`)`	(note parameter order)	†
$rad_{\mathrm{i}(F)}(x)$	`rad(`x`)`		†
$rad_{\mathrm{c}(F)}(x)$	`rad(`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`)`		†
$arcsin_{F\to\mathrm{c}(F)}(x)$	`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`)`		†
$arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arctan(`x`)`		†
$arctan_{\mathrm{c}(F)}(x)$	`arctan(`x`)`		⋆
$arccot_{\mathrm{i}(F)}(x)$	`arccot(`x`)`		†
$arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arccot(`x`)`		†
$arccot_{\mathrm{c}(F)}(x)$	`arccot(`x`)`		†
$arcsec_{F\to\mathrm{c}(F)}(x)$	`arcsecc(`x`)`		†
$arcsec_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arcsec(`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)	†
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arccosh`(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)	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arcsech`(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 b, x, y, u, and v are expressions of type *CFLT*.

Arithmetic value conversions in Modula-2 are can be explicit or implicit. The rules for when implicit conversions are applied is not repeated here. The explicit arithmetic value conversions are usually expressed as 'casts', except when converting to/from strings.

$convert_{I\to\mathrm{c}(I)}(x)$	`(`INT `_Complex)`x	†
$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`(`INT `_Complex)`x	†
$convert_{\mathrm{i}(I)\to\mathrm{i}(I')}(x)$	`(`$INT2$ `_Imaginary)`x	†
$convert_{\mathrm{c}(I)\to\mathrm{c}(I')}(x)$	`(`$INT2$ `_Complex)`x	†
$convert_{F\to\mathrm{c}(F)}(x)$	`(`FLT `_Complex)`x	$\star$
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`(`FLT `_Complex)`x	$(\star)$
$convert_{\mathrm{i}(F)\to\mathrm{i}(F')}(x)$	`(`$FLT2$ `_Imaginary)`x	$(\star)$
$convert_{\mathrm{c}(F)\to\mathrm{c}(F')}(x)$	`(`$FLT2$ `_Complex)`x	$\star$
$convert_{\mathrm{i}(F'')\to\mathrm{i}(F)}(x)$	`sscanf...`x	$(\star)$
$convert_{\mathrm{c}(F'')\to\mathrm{c}(F)}(x)$	`sscanf...`x	$\star$

$convert_{\mathrm{i}(F)\to\mathrm{i}(F'')}(x)$	`sprintf...`x	$(\star)$
$convert_{\mathrm{c}(F)\to\mathrm{c}(F'')}(x)$	`sprintf...`x	$\star$
$convert_{\mathrm{i}(D)\to\mathrm{i}(F)}(x)$	`sscanf...(...`x`)`	$(\star)$
$convert_{\mathrm{c}(D)\to\mathrm{c}(F)}(x)$	`sscanf...(...`x`)`	$\star$
$convert_{\mathrm{i}(F)\to\mathrm{i}(D)}(x)$	`sprintf...(...`x`)`	$(\star)$
$convert_{\mathrm{c}(F)\to\mathrm{c}(D)}(x)$	`sprintf...(...`x`)`	$\star$

complex IO??

where x is an expression of type *INT*, y is an expression of type *FLT*, and z is an expression of type *FXD*, where *FXD* is a fixed point type. *INT2* is the integer datatype that corresponds to I'. A ? above indicates that the parameter is optional. e is greater than 0. Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`II` or `_Imaginary_II`	†
$imaginary\_unit_{\mathrm{c}(I)}$	((`II`)) or `_Complex_II`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`I` or `_Imaginary_I`	$(\star)$
$imaginary\_unit_{\mathrm{c}(F)}$	((`I`)) or `_Complex_I`	$\star$

C.10 PL/I

The programming language PL/I is defined by ANSI X3.53-1976 (R1998), *Programming languages – PL/I* [39], and endorsed by ISO 6160:1979, *Programming languages – PL/I* [25]. The programming language General Purpose PL/I is defined by ISO/IEC 6522:1992, *Information technology – Programming languages – PL/I general-purpose subset* [26], also: ANSI X3.74-1987 (R1998).

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 in the PL/I datatype `BIT(1)` (1 = **true** and 0 = **false**).

An implementation of PL/I provides at least one integer data type, and at least one floating point data type. The attribute `FIXED(`n`,0)` selects a signed integer datatype with at least n (decimal or binary) digits of storage. The attribute `FLOAT(`k`)` selects a floating point datatype with at least n (decimal or binary) digits of precision.

The LIA-3 integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`II * `x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`II * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`II * `x	†
$re_{I}(x)$	`REALPART(`x`)`	†
$re_{\mathrm{i}(I)}(x)$	`REALPART(`x`)`	†
$re_{\mathrm{c}(I)}(x)$	`REALPART(`x`)`	†
$im_{I}(x)$	`IMAGPART(`x`)`	†
$im_{\mathrm{i}(I)}(x)$	`IMAGPART(`x`)`	†
$im_{\mathrm{c}(I)}(x)$	`IMAGPART(`x`)`	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	x `+ II * `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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†
$max_{\mathrm{i}(I)}(x, y)$	`max(`x`, `y`)`	†

$min_{\mathrm{i}(I)}(x, y)$	`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†

where x and y are expressions of type *CINT*.

The LIA-3 non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$itimes_{F\rightarrow\mathrm{i}(F)}(x)$	`i * `x or `j * `x	†
$itimes_{\mathrm{i}(F)\rightarrow F}(x)$	`i * `x or `j * `x	†
$itimes_{\mathrm{c}(F)}(x)$	`i * `x or `j * `x	†
$re_F(x)$	`REALPART(`x`)`	†
$re_{\mathrm{i}(F)}(x)$	`REALPART(`x`)`	†
$re_{\mathrm{c}(F)}(x)$	`REALPART(`x`)`	⋆
$im_F(x)$	`IMAGPART(`x`)`	†
$im_{\mathrm{i}(F)}(x)$	`IMAGPART(`x`)`	⋆
$im_{\mathrm{c}(F)}(x)$	`IMAGPART(`x`)`	⋆
$plusitimes_{\mathrm{c}(F)}(x, y)$	`COMPLEX(`x`,`y`)`	⋆
$neg_{\mathrm{i}(F)}(x)$	`-`x	†
$neg_{\mathrm{c}(F)}(x)$	`-`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	$\star$
$div_{\mathrm{i}(F)}(x, y)$	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	$\star$
$div_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `/` y	†
$div_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `/` y	†
$div_{\mathrm{c}(F)}(x, y)$	x `/` y	$\star$
$eq_{\mathrm{i}(F)}(x, y)$	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	$\star$
$eq_{\mathrm{c}(F),F}(x, y)$	x `=` y	$\star$
$eq_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `=` y	†
$eq_{\mathrm{c}(F)}(x, y)$	x `=` y	$\star$
$neq_{\mathrm{i}(F)}(x, y)$	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	$\star$
$neq_{\mathrm{c}(F),F}(x, y)$	x `/=` y	$\star$
$neq_{\mathrm{i}(F),\mathrm{c}(F)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(F),\mathrm{i}(F)}(x, y)$	x `/=` y	†
$neq_{\mathrm{c}(F)}(x, y)$	x `/=` y	$\star$
$lss_{\mathrm{i}(F)}(x, y)$	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	†
$abs_{\mathrm{i}(F)}(x)$	`abs(`x`)`	†
$abs_{\mathrm{c}(F)}(x)$	`abs(`x`)`	$\star$
$phase_F(x)$	`ARG(`x`)`	†
$phase_{\mathrm{i}(F)}(x)$	`ARG(`x`)`	†
$phase_{\mathrm{c}(F)}(x)$	`ARG(`x`)`	$\star$
$phaseu_F(u, x)$	`ARG(`x`,` u`)`	†
$phaseu_{\mathrm{i}(F)}(u, x)$	`ARG(`x`,` u`)`	†
$phaseu_{\mathrm{c}(F)}(u, x)$	`ARG(`x`,` u`)`	†
$signum_F(x)$	`SIGN(`x`)`	†
$signum_{\mathrm{i}(F)}(x)$	`SIGN(`x`)`	†
$signum_{\mathrm{c}(F)}(x)$	`SIGN(`x`)`	†

where x, y and z are expressions of type *FLT*, and where xs is an expression of type `array of` *FLT*.

The parameters for operations approximating real valued transcendental functions can be accessed by the following syntax:

$box\_error\_mode\_mul_{c(F)}$	`box_err_mul(`x`)`	†
$max\_err\_mul_{c(F)}$	`err_mul(`x`)`	†
$box\_error\_mode\_div_{c(F)}$	`box_err_div(`x`)`	†
$max\_err\_div_{c(F)}$	`err_div(`x`)`	†
$max\_err\_exp_{c(F)}$	`err_exp(`x`)`	†
$max\_err\_power_{c(F)}$	`err_power(`x`)`	†
$max\_err\_sin_{c(F)}$	`err_sin(`x`)`	†
$max\_err\_tan_{c(F)}$	`err_tan(`x`)`	†

where x and u are expressions of type *CFLT*. 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 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`)`	⋆
$power_{F\to\mathrm{c}(F)}(x,y)$	x `***` y	†
$power_{\mathrm{i}(F)}(x,y)$	x `**` y	†
$power_{\mathrm{i}(F),F}(b,y)$	b `**` y	†
$power_{F,\mathrm{i}(F)}(b,x)$	b `**` x	†
$power_{\mathrm{c}(F),F}(b,y)$	b `**` y	⋆?
$power_{F,\mathrm{c}(F)}(b,x)$	b `**` x	⋆?
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b,y)$	b `**` y	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	b `**` x	†
$power_{\mathrm{c}(F)}(x,y)$	x `**` y	⋆
$sqrt_{F\to\mathrm{c}(F)}(x)$	`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`)`	†
$ln_{F\to\mathrm{c}(F)}(x)$	`LOG(ABS(`x`))+I*arctan2(IMAGPART(`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*arctan2(IMAGPART(`x`),REALPART(`x`))`	†
$ln_{\mathrm{c}(F)}(x)$	`LOG(`x`)`	⋆

$logbase_{F\to \mathrm{c}(F)}(b,x)$	`LOGC(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F),F}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F),F}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{F,\mathrm{c}(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F)}(b,x)$	`LOG(`x`, `b`)` (note parameter order)	†
$rad_{\mathrm{i}(F)}(x)$	`RAD(`x`)`	†
$rad_{\mathrm{c}(F)}(x)$	`RAD(`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`)`	†
$arcsin_{F\to \mathrm{c}(F)}(x)$	`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`)`	†
$arctan_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`ARCTAN(`x`)`	†
$arctan_{\mathrm{c}(F)}(x)$	`ARCTAN(`x`)`	⋆
$arccot_{\mathrm{i}(F)}(x)$	`ARCCOT(`x`)`	†
$arccot_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`ARCCOT(`x`)`	†
$arccot_{\mathrm{c}(F)}(x)$	`ARCCOT(`x`)`	⋆
$arcsec_{F\to \mathrm{c}(F)}(x)$	`ARCSECC(`x`)`	†
$arcsec_{\mathrm{i}(F)\to \mathrm{c}(F)}(x)$	`ARCSEC(`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`)`	†
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`ARCCOSH(`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`)`	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`ARCSECH(`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 b, x, y, u, and v are expressions of type *CFLT*.

Arithmetic value conversions in PL/I are can be explicit or implicit. The rules for when implicit conversions are applied is not repeated here. The explicit arithmetic value conversions are usually expressed as 'casts', except when converting to/from strings.

(mockup so far!)		
$convert_{I\to\mathrm{c}(I)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{i}(I)\to\mathrm{i}(I')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{\mathrm{c}(I)\to\mathrm{c}(I')}(x)$	`Compose_From_Cartesian(`x`)`	†
$convert_{F\to\mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	⋆
$convert_{F\to\mathrm{c}(F)}(x)$	x `+ i * Im(`x`)` or x `+ j * Im(`x`)`	†
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`Compose_From_Cartesian(`x`)`	⋆
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`REALPART(`x`) +` x	⋆

$convert_{\mathrm{i}(F)\to\mathrm{i}(F')}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{c}(F)\to\mathrm{c}(F')}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{i}(F)\to\mathrm{i}(D)}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{c}(F)\to\mathrm{c}(D)}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{i}(D)\to\mathrm{i}(F)}(x)$	`Compose_From_Cartesian`(x)	†
$convert_{\mathrm{c}(D)\to\mathrm{c}(F)}(x)$	`Compose_From_Cartesian`(x)	†

complex I/O!!!

where x is an expression of type *INT*, y is an expression of type *FLT*, and z is an expression of type *FXD*, where *FXD* is a fixed point type. *INT2* is the integer datatype that corresponds to I'. A ? above indicates that the parameter is optional. a is greater than 0.

Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`...`	†
$imaginary\_unit_{\mathrm{c}(I)}$	`...`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`i` or `j`	⋆
$imaginary\_unit_{\mathrm{c}(F)}$	`-0+i` or `-0+j`	⋆

C.11 SML

The programming language SML is defined by *The Definition of Standard ML (Revised)* [59].

An implementation should follow all the requirements of LIA-2 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 SML datatype `Boolean` corresponds to the LIA datatype **Boolean**.

Every implementation of SML has at least one integer datatype, `int`, and at least one floating point datatype, `real`. The notation *INT* is used to stand for the name of one of the integer datatypes, and *FLT* is used to stand for the name of one of the floating point datatypes in what follows.

The LIA-2 integer operations are listed below, along with the syntax used to invoke them:

$itimes_{I\to \mathrm{i}(I)}(x)$	`I * `x	†
$itimes_{\mathrm{i}(I)\to I}(x)$	`I * `x	†
$itimes_{\mathrm{c}(I)}(x)$	`I * `x	†
$re_{I}(x)$	`real `x	†
$re_{\mathrm{i}(I)}(x)$	`real `x	†
$re_{\mathrm{c}(I)}(x)$	`real `x	†
$im_{I}(x)$	`imag `x	†
$im_{\mathrm{i}(I)}(x)$	`imag `x	†
$im_{\mathrm{c}(I)}(x)$	`imag `x	†
$plusitimes_{\mathrm{c}(I)}(x,y)$	x `+ I * `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	†
$eq_{\mathrm{i}(I)}(x, y)$	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	†
$abs_{\mathrm{i}(I)}(x)$	`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`)`	†
$max_{\mathrm{i}(I)}(x, y)$	`max(`x`, `y`)`	†
$min_{\mathrm{i}(I)}(x, y)$	`min(`x`, `y`)`	†
$max\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†
$min\_seq_{\mathrm{i}(I)}(xs)$	`maxseq(`xs`)`	†

where x and y are expressions of type *CINT*.

The LIA-3 non-transcendental floating point operations are listed below, along with the syntax used to invoke them:

$itimes_F(x)$	`I *` x	†
$itimes_{\mathrm{i}(F)\to F}(x)$	`I *` x	†
$itimes_{\mathrm{c}(F)}(x)$	`I *` x	†
$re_F(x)$	`real` x	†
$re_{\mathrm{i}(F)}(x)$	`real` x	†
$re_{\mathrm{c}(F)}(x)$	`real` x	†
$im_F(x)$	`imag` x	†
$im_{\mathrm{i}(F)}(x)$	`imag` x	†
$im_{\mathrm{c}(F)}(x)$	`imag` x	†
$plusitimes_F(x,y)$	`complex (`x`,` y`)`	†
$neg_{\mathrm{i}(F)}(x)$	`-` x	†
$neg_{\mathrm{c}(F)}(x)$	`-` 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	†
$add_{F,\mathrm{c}(F)}(x,y)$	x `+` y	†
$add_{\mathrm{i}(F),F}(x,y)$	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	†
$sub_{F,\mathrm{c}(F)}(x,y)$	x `-` y	†
$sub_{\mathrm{i}(F),F}(x,y)$	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	†
$mul_{F,\mathrm{c}(F)}(x,y)$	x `*` y	†
$mul_{\mathrm{i}(F),F}(x,y)$	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	†

$div_{F,\mathrm{c}(F)}(x,y)$	x `/` y	†
$div_{\mathrm{i}(F),F}(x,y)$	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	†
$eq_{\mathrm{i}(F)}(x,y)$	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	†
$abs_{\mathrm{i}(F)}(x)$	`abs` x	†
$abs_{\mathrm{c}(F)}(x)$	`abs` x	†
$phase_F(x)$	`arg` x	†
$phase_{\mathrm{i}(F)}(x)$	`arg` x	†
$phase_{\mathrm{c}(F)}(x)$	`arg` x	†
$phaseu_F(u,x)$	`argu (`u`, `x`)`	†
$phaseu_{\mathrm{i}(F)}(u,x)$	`argu (`u`, `x`)`	†
$phaseu_{\mathrm{c}(F)}(u,x)$	`argu (`u`, `x`)`	†
$signum_F(x)$	`sign` x	†
$signum_{\mathrm{i}(F)}(x)$	`sign` x	†
$signum_{\mathrm{c}(F)}(x)$	`sign` x	†
$polar_F(x,y)$	`polar (`x`,`y`)`	†
$polaru_F(u,x,y)$	`polaru (`u`, `x`,`y`)`	†

where x, y and z are expressions of type *CFLT*.

The binding for the floor, round, and ceiling operations here take advantage of the unlimited `Integer` type in SML, and that `Integer` is the default integer type.

The 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 and u are expressions of type *FLT*. 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 elementary floating point operations are listed below, along with the syntax used to invoke them:

$power_{\mathrm{i}(F),I}(b, y)$	`poweri (`b`, `y`)`	†
$exp_{\mathrm{i}(F)}(x)$	`exp` x	†
$exp_{\mathrm{c}(F)}(x)$	`exp` x	†
$power_{F\to\mathrm{i}(F)}(x, y)$	`powc (`x`, `y`)`	⋆
$power_{\mathrm{i}(F)}(x, y)$	`pow (`x`, `y`)`	†
$power_{\mathrm{i}(F),F}(b, y)$	`pow (`b`, `y`)`	†
$power_{F,\mathrm{i}(F)}(b, x)$	`pow (`b`, `x`)`	†
$power_{\mathrm{c}(F),F}(b, y)$	`pow (`b`, `y`)`	†
$power_{F,\mathrm{c}(F)}(b, x)$	`pow (`b`, `x`)`	†
$power_{\mathrm{i}(F),\mathrm{c}(F)}(b, y)$	`pow (`b`, `y`)`	†
$power_{\mathrm{c}(F),\mathrm{i}(F)}(b, x)$	`pow (`b`, `x`)`	†
$power_{\mathrm{c}(F)}(b, y)$	b `pow` y	†
$sqrt_{F\to\mathrm{c}(F)}(x)$	`sqrtc` x	†
$sqrt_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`sqrt` x	†
$sqrt_{\mathrm{c}(F)}(x)$	`sqrt` x	†
$ln_{F\to\mathrm{c}(F)}(x)$	`log (abs` x`) +: arctan2 (imag` x`, `x`)`	†
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`log (abs` x`) +: arctan2 (imag` x`, real `x`)`	†
$ln_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`log` x	†
$ln_{\mathrm{c}(F)}(x)$	`log` x	†
$logbase_{F\to\mathrm{c}(F)}(b, x)$	`Logc(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F)}(b, x)$	`Log(`x`, `b`)` (note parameter order)	†

$logbase_{\mathrm{i}(F),F}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{F,\mathrm{i}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F),F}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{F,\mathrm{c}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{i}(F),\mathrm{c}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F),\mathrm{i}(F)}(b,x)$	`Log(`x`, `b`)` (note parameter order)	†
$logbase_{\mathrm{c}(F)}(b,x)$	`logbase` b 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	†
$arcsin_{F\to\mathrm{c}(F)}(x)$	`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	†
$arccos_{\mathrm{c}(F)}(x)$	`arccos` x	†
$arctan_{\mathrm{i}(F)}(x)$	`arctan` x	†
$arctan_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`atanc(`x`)`	†
$arctan_{\mathrm{c}(F)}(x)$	`arctan` x	†
$arccot_{\mathrm{i}(F)}(x)$	`arccot` x	†
$arccot_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`acotc(`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	†
$radh_F(x)$	`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	†
$tanh_{\mathrm{c}(F)}(x)$	`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	†
$arccosh_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arccosh` 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	†
$arcsech_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`arcsech` x	†
$arcsech_{\mathrm{c}(F)}(x)$	`arcsech` x	†
$arccsch_{\mathrm{i}(F)}(x)$	`arccsch` x	†
$arccsch_{\mathrm{i}\to\mathrm{c}(F)}(x)$	`arccschc` x	†
$arccsch_{\mathrm{c}(F)}(x)$	`arccsch` x	†

where b, x, y, u, and v are expressions of type *FLT*, and z is an expressions of type *INT*

Type conversions in SML are always explicit.

$convert_{I\to\mathrm{c}(I)}(x)$	`..` x	†
$convert_{\mathrm{i}(I)\to\mathrm{c}(I)}(x)$	`..` x	†
$convert_{F\to\mathrm{c}(F)}(x)$	x `- I * 0` or x `- _Imaginary_I * 0`	†
$convert_{\mathrm{i}(F)\to\mathrm{c}(F)}(x)$	`-0 +` x	†

...
complex IO...??

where x is an expression of type *INT*, y is an expression of type *FLT*, and z is an expression of type *FXD*, where *FXD* is a fixed point type. *INT2* is the integer datatype that corresponds to I'. Numerals...:

$imaginary\_unit_{\mathrm{i}(I)}$	`II` or `_Imaginary_II`	†
$imaginary\_unit_{\mathrm{c}(I)}$	((`II`)) or `_Complex_II`	†
$imaginary\_unit_{\mathrm{i}(F)}$	`I` or `_Imaginary_I`	($\star$)
$imaginary\_unit_{\mathrm{c}(F)}$	((`I`)) or `_Complex_I`	$\star$

Annex D
(informative)

Bibliography

This annex gives references to publications relevant to LIA-3.

International standards documents

[1] ISO/IEC JTC1 Directives, Part 3: *Drafting and presentation of International Standards*, 1989.

[2] ISO 6093:1985, *Information processing – Representation of numerical values in character strings for information interchange.*

[3] ISO/IEC TR 10176:1998, *Information technology – Guidelines for the preparation of programming language standards.*

[4] ISO/IEC TR 10182:1993, *Information technology – Programming languages, their environments and system software interfaces – Guidelines for language bindings.*

[5] ISO/IEC 13886:1996, *Information technology – Language-Independent Procedure Calling*, (LIPC).

[6] ISO/IEC 11404:1996, *Information technology – Programming languages, their environments and system software interfaces – Language-independent datatypes*, (LID).

[7] ISO/IEC 8652:1995, *Information technology – Programming languages – Ada.*

[8] ISO/IEC 13813:1998, *Information technology – Programming languages – Generic packages of real and complex type declarations and basic operations for Ada (including vector and matrix types).*

[9] ISO/IEC 13814:1998, *Information technology – Programming languages – Generic package of complex elementary functions for Ada.*

[10] ISO 8485:1989, *Programming languages – APL.*

[11] ISO/IEC DIS 13751, *Information technology – Programming languages, their environments and system software interfaces – Programming language APL, extended*, 1999.

[12] ISO/IEC 10279:1991, *Information technology – Programming languages – Full BASIC.* (Essentially an endorsement of ANSI X3.113-1987 (R1998) [36].)

[13] ISO/IEC 9899:1999, *Programming languages – C.*

[14] ISO/IEC 14882:1998, *Programming languages – C++.*

[15] ISO 1989:1985, *Programming languages – COBOL.* (Endorsement of ANSI X3.23-1985 (R1991) [37].) Currently under revision (1998).

[16] ISO/IEC 16262:1998, *Information technology - ECMAScript language specification.*

[17] ISO/IEC 15145:1997, *Information technology – Programming languages – FORTH.* (Also: ANSI X3.215-1994.)

[18] ISO/IEC 1539-1:1997, *Information technology – Programming languages – Fortran - Part 1: Base language.*

[19] ISO/IEC TR 15580:1998, *Information technology – Programming languages – Fortran – Floating-point exception handling.*

[20] ISO/IEC 13816:1997, *Information technology – Programming languages, their environments and system software interfaces – Programming language ISLISP.*

[21] ISO/IEC 10514-1:1996, *Information technology – Programming languages – Part 1: Modula-2, Base Language.*

[22] ISO/IEC 10514-2:1998, *Information technology – Programming languages – Part 2: Generics Modula-2.*

[23] ISO 7185:1990, *Information technology – Programming languages – Pascal.*

[24] ISO/IEC 10206:1991, *Information technology – Programming languages – Extended Pascal.*

[25] ISO 6160:1979, *Programming languages – PL/I.* (Endorsement of ANSI X3.53-1976 (R1998) [39].)

[26] ISO/IEC 6522:1992, *Information technology – Programming languages – PL/I general-purpose subset.* (Also: ANSI X3.74-1987 (R1998).)

[27] ISO/IEC 13211-1:1995, *Information technology – Programming languages – Prolog – Part 1: General core.*

[28] ISO/IEC 9075:1992, *Information technology – Database languages – SQL.*

[29] ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1) – Part 1: Specification of basic notation.*

[30] ISO/IEC 9001:1994, *Quality systems – Model for quality assurance in design, development, production, installation and servicing.*

[31] ISO/IEC 9126:1991, *Information technology – Software product evaluation – Quality characteristics and guidelines for their use.*

[32] ISO/IEC 12119:1994, *Information technology – Software packages – Quality requirements and testing.*

[33] ISO/IEC 14598-1:1999, *Information technology – Software product evaluation – Part 1: General overview.*

National standards documents

[34] ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic.*

[35] ANSI/IEEE Standard 854-1987, *IEEE Standard for Radix-Independent Floating-Point Arithmetic.*

[36] ANSI X3.113-1987 (R1998), *Information technology – Programming languages – Full BASIC.*

[37] ANSI X3.23-1985 (R1991), *Programming languages – COBOL.*

[38] ANSI X3.226-1994, *Information Technology – Programming Language – Common Lisp.*

[39] ANSI X3.53-1976 (R1998), *Programming languages – PL/I.*

[40] ANSI/IEEE 1178-1990, *IEEE Standard for the Scheme Programming Language.*

[41] ANSI/NCITS 319-1998, *Information Technology – Programming Languages – Smalltalk.*

Books, articles, and other documents

[42] J. S. Squire (ed), *Ada Letters*, vol. XI, No. 7, ACM Press (1991).

[43] M. Abramowitz and I. Stegun (eds), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Tenth Printing, 1972, Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

[44] J. Du Croz and M. Pont, *The Development of a Floating-Point Validation Package*, *NAG Newsletter*, No. 3, 1984.

[45] J. W. Demmel and X. Li, *Faster Numerical Algorithms via Exception Handling*, 11th International Symposium on Computer Arithmetic, Winsor, Ontario, June 29 - July 2, 1993.

[46] D. Goldberg, *What Every Computer Scientist Should Know about Floating-Point Arithmetic.* ACM Computing Surveys, Vol. 23, No. 1, March 1991.

[47] J. R. Hauser, *Handling Floating-Point Exceptions in Numeric Programs.* ACM Transactions on Programming Languages and Systems, Vol. 18, No. 2, March 1986, Pages 139-174.

[48] W. Kahan, *Branch Cuts for Complex Elementary Functions, or Much Ado about Nothing's Sign Bit*, Chapter 7 in *The State of the Art in Numerical Analysis* ed. by M. Powell and A. Iserles (1987) Oxford.

[49] W. Kahan, *Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic*, Panel Discussion of *Floating-Point Past, Present and Future*, May 23, 1995, in a series of San Francisco Bay Area Computer Historical Perspectives, sponsored by SUN Microsystems Inc.

[50] U. Kulisch and W. L. Miranker, *Computer Arithmetic in Theory and Practice*, Academic Press, 1981.

[51] U. Kulisch and W. L. Miranker (eds), *A New Approach to Scientific Computation*, Academic Press, 1983.

[52] D. C. Sorenson and P. T. P. Tang, *On the Orthogonality of Eigenvectors Computed by Divide-and-Conquer Techniques*, SIAM Journal of Numerical Analysis, Vol. 28, No. 6, p. 1760, algorithm 5.3.

[53] *C9x Floating-Point Proposal*, December 1995, SC22/WG14 N510, N511, N512, N513, N314.

[54] *C9x Complex Arithmetic Proposal*, December 1995, SC22/WG14 N515, N516, N517, N518.

[55] David M. Gay, *Correctly Rounded Binary-Decimal and Decimal-Binary Conversions*, AT&T Bell Laboratories, Numerical Analysis Manuscript 90-10, November 1990.

[56] James Gosling, Bill Joy, Guy Steele, *The Java Language Specification*.

[57] Simon Peyton Jones et al., *Report on the programming language Haskell 98*, February 1999.

[58] Simon Peyton Jones et al., *Standard libraries for the Haskell 98 programming language*, February 1999.

[59] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen, *The Definition of Standard ML (Revised)*, The MIT Press, 1997, ISBN: 0-262-63181-4.

Annex E
(informative)

Possible changes to part 2

This annex indicates possible changes to ISO/IEC 10967-2:2001. A revision of part 2 is not limited to the items listed below, nor guaranteed to be done as described below.

a) For the arc_F operation, clause 5.3.8.15, the range limitation function should be redefined as follows, so that the returned angle in for the proper quadrant:

$$\begin{aligned} arc_F^{\#}(x,y) &= \max\{up_F(-\pi), \min\{arc_F^*(x,y), down_F(-\pi/2)\}\} \\ &\qquad \text{if } x < 0 \text{ and } y < 0 \\ &= \max\{up_F(-\pi/2), \min\{arc_F^*(x,y), down_F(\pi/2)\}\} \\ &\qquad \text{if } x \geqslant 0 \\ &= \max\{up_F(\pi/2), \min\{arc_F^*(x,y), down_F(\pi)\}\} \\ &\qquad \text{if } x < 0 \text{ and } y \geqslant 0 \end{aligned}$$

and the arc_F specification should say (correcting the range limitation):

$$\begin{aligned} arc_F(x,y) &= result_F^*(arc_F^{\#}(x,y), nearest_F) \\ &\qquad \text{if } x, y \in F \text{ and } (x \neq 0 \text{ or } y \neq 0) \\ &= 0 && \text{if } x = 0 \text{ and } y = 0 \\ &= down_F(\pi) && \text{if } x = \mathbf{-0} \text{ and } y = 0 \\ &= up_F(\pi/2) && \text{if } x = \mathbf{-0} \text{ and } ((y \in F \text{ and } y > 0) \text{ or } y = \mathbf{+\infty}) \\ &= down_F(-\pi/2) && \text{if } x = \mathbf{-0} \text{ and } ((y \in F \text{ and } y < 0) \text{ or } y = \mathbf{-\infty}) \\ &= neg_F(arc_F(x,0)) && \text{if } y = \mathbf{-0} \text{ and } x \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \\ &= 0 && \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y \geqslant 0 \\ &= \mathbf{-0} && \text{if } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y < 0 \\ &= nearest_F(\pi/4) && \text{if } x = \mathbf{+\infty} \text{ and } y = \mathbf{+\infty} \\ &= up_F(\pi/2) && \text{if } x \in F \text{ and } y = \mathbf{+\infty} \text{ and } x < 0 \\ &= down_F(\pi/2) && \text{if } x \in F \text{ and } y = \mathbf{+\infty} \text{ and } x \geqslant 0 \\ &= nearest_F(3 \cdot \pi/4) && \text{if } x = \mathbf{-\infty} \text{ and } y = \mathbf{+\infty} \\ &= down_F(\pi) && \text{if } x = \mathbf{-\infty} \text{ and } y \in F \text{ and } y \geqslant 0 \\ &= up_F(-\pi) && \text{if } x = -\infty \text{ and } y \in F \text{ and } y < 0 \\ &= nearest_F(-3 \cdot \pi/4) && \text{if } x = -\infty \text{ and } y = -\infty \\ &= down_F(-\pi/2) && \text{if } x \in F \text{ and } y = \mathbf{-\infty} \text{ and } x < 0 \\ &= up_F(-\pi/2) && \text{if } x \in F \text{ and } y = \mathbf{-\infty} \text{ and } x \geqslant 0 \\ &= nearest_F(-\pi/4) && \text{if } x = \mathbf{+\infty} \text{ and } y = \mathbf{-\infty} \\ &= no\_result2_F(x,y) && \text{otherwise} \end{aligned}$$

b) For the $arcu_F$ operation, clause 5.3.9.15, the range limitation function should be redefined as follows, so that the returned angle in for the proper quadrant:

$$\begin{aligned} arcu_F^{\#}(u,x,y) &= \max\{up_F(-u/2), \min\{arcu_F^*(u,x,y), down_F(-u/4)\}\} \\ &\qquad \text{if } x < 0 \text{ and } y < 0 \\ &= \max\{up_F(-u/4), \min\{arcu_F^*(u,x,y), down_F(u/4)\}\} \\ &\qquad \text{if } x \geqslant 0 \\ &= \max\{up_F(u/4), \min\{arcu_F^*(u,x,y), down_F(u/2)\}\} \\ &\qquad \text{if } x < 0 \text{ and } y \geqslant 0 \end{aligned}$$

and the $arcu_F$ specification should say (correcting the $<$ vs. $\neq$ typo as well as the range limitation):

$$\begin{aligned} arcu_F(u,x,y) &= result_F^*(arcu_F^{\#}(u,x,y), nearest_F) \\ &\qquad \text{if } u \in G_F \text{ and } x,y \in F \text{ and } (x \neq 0 \text{ or } y \neq 0) \\ &= mul_F(u,0) \qquad \text{if } u \in G_F \text{ and } x \in F \text{ and } x \geqslant 0 \text{ and } y = 0 \\ &= down_F(u/2) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-0} \text{ and } y = 0 \text{ and } u > 0 \\ &= up_F(u/2) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-0} \text{ and } y = 0 \text{ and } u < 0 \\ &= up_F(u/4) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-0} \text{ and} \\ &\qquad ((y \in F \text{ and } y > 0) \text{ or } y = \mathbf{+\infty}) \text{ and } u > 0 \\ &= down_F(-u/4) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-0} \text{ and} \\ &\qquad ((y \in F \text{ and } y < 0) \text{ or } y = \mathbf{-\infty}) \text{ and } u > 0 \\ &= down_F(u/4) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-0} \text{ and} \\ &\qquad ((y \in F \text{ and } y > 0) \text{ or } y = \mathbf{+\infty}) \text{ and } u < 0 \\ &= up_F(-u/4) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-0} \text{ and} \\ &\qquad ((y \in F \text{ and } y < 0) \text{ or } y = \mathbf{-\infty}) \text{ and } u < 0 \\ &= neg_F(arcu_F(u,x,0)) \qquad \text{if } u \in G_F \text{ and } y = \mathbf{-0} \text{ and } x \in F \cup \{\mathbf{-\infty}, \mathbf{-0}, \mathbf{+\infty}\} \\ \\ &= mul_F(0,u) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y \geqslant 0 \\ &= mul_F(0,-u) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{+\infty} \text{ and } y \in F \text{ and } y < 0 \\ &= nearest_F(u/8) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{+\infty} \text{ and } y = \mathbf{+\infty} \\ &= up_F(u/4) \qquad \text{if } u \in G_F \text{ and } x \in F \text{ and } y = \mathbf{+\infty} \text{ and } x \cdot u < 0 \\ &= down_F(u/4) \qquad \text{if } u \in G_F \text{ and } x \in F \text{ and } y = \mathbf{+\infty} \text{ and } x \cdot u \geqslant 0 \\ &= nearest_F(3 \cdot u/8) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-\infty} \text{ and } y = \mathbf{+\infty} \\ &= down_F(u/2) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-\infty} \text{ and } y \in F \text{ and} \\ &\qquad y \geqslant 0 \text{ and } u > 0 \\ &= up_F(-u/2) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-\infty} \text{ and } y \in F \text{ and} \\ &\qquad y < 0 \text{ and } u > 0 \\ &= up_F(u/2) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-\infty} \text{ and } y \in F \text{ and} \\ &\qquad y > 0 \text{ and } u < 0 \\ &= down_F(-u/2) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-\infty} \text{ and } y \in F \text{ and} \\ &\qquad y \leqslant 0 \text{ and } u < 0 \\ &= nearest_F(-3 \cdot u/8) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{-\infty} \text{ and } y = \mathbf{-\infty} \\ &= down_F(-u/4) \qquad \text{if } u \in G_F \text{ and } x \in F \text{ and } y = \mathbf{-\infty} \text{ and } x \cdot u < 0 \\ &= up_F(-u/4) \qquad \text{if } u \in G_F \text{ and } x \in F \text{ and } y = \mathbf{-\infty} \text{ and } x \cdot u \geqslant 0 \\ &= nearest_F(-u/8) \qquad \text{if } u \in G_F \text{ and } x = \mathbf{+\infty} \text{ and } y = \mathbf{-\infty} \\ \\ &= no\_result3_F(u,x,y) \qquad \text{otherwise} \end{aligned}$$