Disposition of COMMENTS ON ISO/IEC FCD 10967-3 given in ISO/IEC JTC1 SC22/WG11 N492

Drafted by Kent Karlsson, editor
Comments discussed and updated by WG11 2005-04-27

ISO CS

General

Text should be in alignment with ISO template. Please use the latest template, available from http://isotc.iso.ch/livelink/livelink/fetch/2000/2123/SDS_WEB/sds_temp.htm.

Accepted in principle.
The link given no longer works, or at least now requires a login. Note also that the 10967 series of documents are written using the LaTeX text formatting tool. It is not at all compatible with the Microsoft Word tool.

Footers

Should read "© 2004 ISO/IEC — All rights reserved". Please add.

Accepted in principle.
The line is placed at the page header, for editing consistency with parts 1 and 2. In addition, the footer space is better used to indicate running section number and title (as is done also for part 2).

Foreword

Outdated Foreword is used. Please use the latest Foreword, available from the ISO/IEC template.

See above.
The link given no longer works...; so the referenced template is not available to the editor...

1 Scope

The scope is rather long. According to 6.2.1 of the ISO/IEC Directives Part 2, a scope "shall be succinct so that it can be used as a summary for bibliographic purposes". Please consider shortening the scope. Perhaps some of the text could be moved to the Introduction.

Rejected.

The scope is about as long as it is for parts 1 and 2. The suggested change may be considered for a concerted revision of the three (so far) parts.

3 Normative references

Outdated text is used. Please replace with the text found in the ISO/IEC template or in 6.2.2 of the ISO/IEC Directives Part 2.

Accepted.

The text in the referenced section of the directives is used.

3 Normative references

If the reference in Note 1 is a Normative reference, it should be listed as such. If it is not, it should be moved to the Bibliography. Please change accordingly.

Accepted in principle.

The document mentioned in note 1 was already listed as a normative reference (just after note 1). The notes in this clause are moved to the rationale annex.

4.2 Definition of terms

Incorrect format. Please draft in accordance with Annex C of the ISO/IEC Directives Part 2.

Accepted.

Japan

Foreword, 2nd paragraph
The name of the document "ISO/IEC Directives, part 3" seems incorrect. "JTC1" should be inserted after "ISO/IEC" to be consistent with the reference [1].

Accepted in principle.

ISO secretariat points to ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. That document will be referenced instead.

Introduction, "The content", 1st paragraph, 2nd line
There are two contiguous "specifications" in "specifications to specifications for operations". One of them should be deleted.

Accepted in principle.

"specifications to specifications" replaced by "specifications to also cover".

1.2(c)(d)(e)
The second sentence of these items are "This part neither requires nor excludes such data or operations.". The word "data" should be changed to "datatypes" in these sentences.

Accepted.

(Also for parts 1 and 2, when revised.)

1.2(f)
This clause only speaks about "operations". The phrase "such data or operations" in the second sentence seems incorrect. It should be "such operations".

Rejected.

Matrix (etc.) values can have a datatype, so the word "data" is replaced by datatypes (also for parts 1 and 2, when revised). Note also that for symbolic operations, there must be a datatype that allows not just (e.g.) floating point values, but also symbolic values.

2, Note 1
This Note says that the Clause 8 requires a binding provided by an implementation. It is not easy to see this, since the word "binding" never appears in Clause 8. We suggest to

give some explanations in this Note on the relationship between "binding" and "documentation".

Accepted in principle.

The wording "to supply" is replaced by "to document".

3, Note 1
Here the word "annex" is capitalized, while many other appearances of "annex" are not capitalized. Please unify them to the appropriate form according to the convention for ISO/IEC standards.

Accepted.

The word "annex" will be written in titlecase, when it is part of a reference to a standards annex. This is in accordance with the ISO/IEC drafting directives.

4.1.2, set operators
The set operator "\times" appears twice; as the third operator in the list of set operators, and as an independent operator just following the list.

Accepted.

The more explanatory line is kept, and \times is removed from the list preceding it.

4.1.4, "overflow"
It seems that a noun is necessary just after "than".

Accepted in principle.

The formulation "has a magnitude larger than the maximum value representable in the result datatype" is used instead.

4.1.4, "infinitary"
We cannot understand the meaning of the second half of the sentence. What is referred to by "otherwise"? The phrase "from finite arguments" should be changed to "at the finite argument point" for consistency.

Accepted.

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

4.1.4, 2nd last paragraph
The last sentence is very hard to understand for us. Something seems necessary after "specifications for". What is the relationship between "when it is appropriate ..." and "an appropriate continuation ..."? The term "continuation value" is defined in 4.2, thus this is a forward reference. A section reference will help.

Accepted.

The explanation is clarified and a reference to the definition has been added.
This part of \LIA does not specify when it is appropriate to return this exceptional value, but does specify an appropriate continuation value (see 4.2.x).

4.1.5, paragraph just after Note 2
One of two occurrences of "or part 2:" should be deleted.

Accepted.

4.1.5, "Floating point operations from part 1"
The "and" between "gtr_F" and "geq_F" should be deleted.

Accepted.

4.1.5, paragraph just after Note 5
An "are" should be inserted between "IEC 60559 and" and "used in this part".

Accepted.

4.2, "error"(2)
The word "context" should be changed to "contexts".

Accepted.

4.2, "normalised"
A section reference is written as "5.2 of part 1". This is inconsistent with a similar reference in "precision" where the same section is written as "clause 5.2 of part 1". We do not know whether "clause" should be or should not be written in such places. Please unify them according to the convention for ISO/IEC standards.

Accepted.

References will be written according to the fourth edition of the ISO/IEC drafting directives.

4.2, "signature"
We do not understand the naming convention of operations. For example, the name of the "cos" function taking an imaginary argument is "cos_i(F)". Although this operation returns a real result of type F, this fact is not reflected in the operation name. On the other hand, the name of the "exp" function taking an imaginary argument is "exp_i(F)->c(F)" which contains the result type in its name. What is the principle under these namings?

The naming may sometimes be a bit haphazard, but the intent is to take as subscript enough parts of the signature to make the LIA operation name unique. However, exp, "i(F)->c(F)" is more than needed, and "i(F)" suffices.

5.1.2, "re_I" (for example)
The definition of "re_I" has a condition "if x \in I". We do not understand why such a condition is necessary. The intent might be to exclude infinities, but is it really necessary to exclude infinities from this definition?

Accepted in principle.

Explain in rationale: The intent is to exclude NaNs, esp. signalling NaNs, even though such usually aren't implemented, that is a theoretical possibility. To make that clear, the condition now includes infinites explicitly, rather than leaving that to implementations.

5.1.2, "signum_I"
The values returned for infinity arguments are defined here. This is a violation to the following sentence in 5.1: "Integer datatypes conforming to part 1 often do not contain any NaN or infinity values, ... this clause has no specifications for such values as arguments or results.".

Accepted.

The otherwise case as well as the infinity cases are removed.

5.1.2, "quot_c(I)"
In this definition, a complex value is written as "x + (i.y)". These parentheses seem redundant. For example, see the definition of "mul_c(I)", where the same complex value is written as "x + i.y".

Rejected.

All LIA parts use full parenthetisation on all mathematical expressions. Those where that was missed in the LIA-3 draft have been given parentheses.

5.1.2, "mod_c(I),i(I)"
The definition given here is wrong. In the modulus, the real part of the argument "x" never goes to the imaginary part of the result. It should go to the real part of the result.

Accepted.

All of the division and remainder operation specifications have been reviewed for correctness.

5.1.2, "residue_c(I),i(I)"
This definition is also wrong, just like "mod_c(I),i(I)".

Accepted.

All of the division and remainder operation specifications have been reviewed for correctness.

5.1.2, "pad_I,i(I)"
This function should be similar to "mod_I,i(I)" and "residue_I,i(I)" but there is a big difference from these two functions; the real part "x" is negated before passed to "pad_I" function. Is this intentional?

Accepted.

All of the division and remainder operation specifications have been reviewed for correctness.

5,1,2, "pad_c(I),i(I)"
This definition is also wrong, just like "mod_c(I),i(I)" and "residue_c(I),i(I)".

Accepted.

All of the division and remainder operation specifications have been reviewed for correctness.

5.1.2, "pad_c(I)"

The definition seems to have several errors. The second term should be subtracted from "x + i.y" not added to it. The function "round" is used in the definition, but shouldn't this be changed to "ceiling"?

Partially accepted.

Round is changed to ceiling. However, the total result is negated compared to the sibling operations mod and residue. This is done for consistency with pad in LIA-2, where it is done this way to avoid problems if the integer datatype is unsigned (so that the returned value always is non-negative), even though that is unlikely for LIA-3.

5.2.5, "plusitimes_c(F)"
This function takes two arguments of type F (real values). Why is the suffix of the operation name "c(F)"?

Accepted.

That is because the return type is c(F). However, there is no problem in renaming this operation to just use F in the index, and it is now so renamed.

5.2.5, "add_i(F)" (for example)
The range of this function is "i(F) \union { (underflow), overflow }". There must be some meaning conveyed by parentheses surrounding "underflow", but it is not explained anywhere.

Accepted.

The parentheses around "underflow" will be removed. Add text to rationale (or just note) that these operations will never underflow for IEC 60559 based implementations.

5.2.5, "sub_F,i(F)" (for example)
The range of this function is defined to be "c(F)", but we think that a negative zero may result for a zero argument. In the definition of "conj_i(F)", the range of a similar result is written as "i(F \union {-0})". Shouldn't the range of "sub_F,i(F)" also include "-0"?

Accepted.

5.2.5, "rounding_i(F)"
There is a TeX formatting error.

Accepted.

5.2.6, "signum*_c(F)"
In the requirements for this function, a special condition for "x + i.x" is given requiring 1/sqrt(2) for both real and imaginary parts of the result. Is this condition consistent with similar conditions for other functions? For example, conditions for "polar*_c(F)" function are given in 5.2.7, for 30 degrees and 60 degrees cases, but not for 45 degrees case.

Noted.

*Polar takes an angle argument, while signum takes a complex value argument. There are similar differences for operations in LIA-2, compare arc*_F. signum_c(F) is similar to arc_F, where the argument is a complex value instead of two real values. The editor would appreciate, however, precise suggestions on extra requirements to add, remove, or change.*

5.2.7
Some expressions are too wide for printing on A4 papers.

Accepted.

5.3.1.2 (for example)
The following sentence appears repeatedly in the definitions of functions: "The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no ... operations in any associated library for real-valued operations or there is no associated library for real-valued operations.". It is painful to read the same sentence several times. Isn't it possible to give this meta statement only once before giving definitions of specific functions?

Accepted.

The requirement formulation is generalised (to fit all operations), and moved to the introduction of 5.3.

5.3.1.3, "power_F->c(F)"
In the definition of "power_F->c(F)" function, a condition "|z/2| <= big_angle_u_F" is given several times. We think that the reference to "big_angle_u" in this part is not acceptable, since only radian-based trigonometrics are considered for complex cases. We suggest to change the condition to "|\pi.z| <= big_angle_r_F".

Rejected.

Compare the phaseu and polaru operations in LIA-3. The limit for power_F->c(F) is best expressed using big_angle_u. Text has been added to the rationale, saying that |z/2| <=

big_angle_u_F is easier to implement (no multiplication by \pi). However, the specification of power_F->c(F) has been corrected in other respects.

5.3.1.3, "power_i(F)" (for example)
In the definition, the real part is computed by "re_F(i.y)", but this function reference is not correct. Since the argument has the type "i(F)", the function name should be "re_i(F)" not "re_F". There are many occurrences of this error in the definitions of several functions.

Accepted in principle.

HOWEVER, since the power_c(F) operation is very intricate, it and the operations referring to it have been deleted from the draft. A future revision of LIA-3 may reinstate it/them, if the intricacies have been resolved, and a useful specification is available.

5.3.1.3, "power*_c(F)"
A condition "y >= 0" is given for the cases "(z-0.5)/2 \in Z" and "(z-1.5)/2 \in Z" but not for the case "(z-1)/2 \in Z". Is this intentional?

Accepted.

The two missing "y>=0" was an oversight.

HOWEVER, since the power_c(F) operation is very intricate, it and the operations referring to it have been deleted from the draft. A future revision of LIA-3 may reinstate it/them, if the intricacies have been resolved, and a useful specification is available.

5.3.1.3, "power_c(F)"
The condition given in the seventh alternative is wrong, or at the best redundant. It says "or x = -0" in its second term, but this is impossible since the fourth term is "x /= 0". The case "x = -0" is covered by the second alternative.

Accepted in principle.

The editor (this or a future one) will (may) work more on this specification.

HOWEVER, since the power_c(F) operation is very intricate, it and the operations referring to it have been deleted from the draft. A future revision of LIA-3 may reinstate it/them, if the intricacies have been resolved, and a useful specification is available.

5.3.1.4, "sqrt_c(F)"
Certain cases are covered by more than one alternative. In other words, there are several overlappings in the conditions given for alternatives. The case "x = +infinity and y = -0" is covered by the third and the sixth alternatives. The cases "x = -infinity and y = -0", "x = -0 and y = +infinity", and "x = -0 and y = -infinity" are also covered by two alternatives. The condition "x >= 0" in the fifth and the eighth alternatives should be "y >= 0".

Accepted.

5.3.1.5, "ln_c(F)"
The case "x \in {-0, 0} and y = -0" is doubly covered by the second and the third alternatives.

Accepted.

5.3.1.6, "logbase_i(F)"
One of two left parentheses before "re_F" should be deleted.

Accepted in principle.

But for logbase_i(F),F. Note also that several occurrences of re_F are now replaced by re_i(F).

HOWEVER, since the logbase_c(F) operation is very intricate, it and the operations referring to it have been deleted from the draft. A future revision of LIA-3 may reinstate it/them, if the intricacies have been resolved, and a useful specification is available.

5.3.1.6
In the "associated library for real-valued operations" requirements, the term "this relationship" is singular. This should be changed to plural "these relationships", since there are at least two equations given.

Accepted more generally.

The plural/singular has been reviewed for similar sentences in LIA-3.

5.3.2.5, Note
A space should be inserted between "when" and "x". The "a" should be deleted in "a zero". The alternative referred to in the Note says "x /= -0" contrary to the condition mentioned in the Note.

Accepted.

5.3.3
We consider that a sentence of the form "Note that ..." should not be used in the normative part of a standard.

Accepted.

"Note that the" -> "The", and the entire sentence is put in a Note.

7, 1st paragraph
The term "the present standard" looks strange. The term "this part" is often used in this document and refers to the same thing.

Accepted in principle.

The fourth edition of the drafting directives says that "this document" is the now preferred formulation. The text has been changed accordingly.

7, Note 1
In the first sentence, there is no mention to "binding standards". We think that this sentence should also refer to "language or binding standards".

Accepted.

7, 3rd paragraph
We think that "to invoke an operation or access a parameter" should be changed to "to invoke an operation or {to} access a parameter" (two occurrences).

Accepted.

Further, a note has been added, saying: "The requirement that the notation that should be used for accessing operations or parameters that are not made available should also be documented is a recommendation to reserve that notation for future use (or present use by some implementations). Thus, doing so would be a way of saying that that notation should not be used for something else, nor, e.g., for a less accurate but otherwise similar operation."

7, last paragraph
What is the meaning of "textual scope"?

Accepted in principle.

The term textual scope is used in its conventional meaning of "part of source code". Especially in reference to a control mechanism, which may cover "all of a programming project", "a source file", "a block", "between on/off pragmas" or some such mechanism. The note from that was in LIA-1 here is now reinstated for LIA-3.

B.1.1
The term "C99" is an informal one and seems inappropriate in this document.

Accepted.

B.1.2, 2nd paragraph
The first sentence is not grammatically correct. Change "no part" to "no parts" or "cover" to "covers".

Accepted.

"cover" -> "covers"

B.2, 1st paragraph
The word "organisation" should be spelled as "organization".

Accepted

B.4.1.2, 1st paragraph
In the first sentence, "is" should be changed to "are".

Accepted.

B.4.1.5
Is the position of the section header B.4.1.5 correct? The first sentence seems to belong to B.4.1.4.

Accepted.

Sentence moved to B.4.1.4.

B.4.1.5, 3rd paragraph
A "to" should be inserted after "while invalid with a specified continuation values".

Accepted (for the 2nd paragraph)

B.4.2
The font of "Ada exceptions" should be Roman, not typeface, since it refers to a general notion in the Ada language, not a specific occurrence of a keyword.

Accepted.

B.5.3.1 (for example)
The phrase "because no programming language require them" is not grammatically correct. Change "language" to "languages", or "require" to "requires". There are many occurrences of the same error.

Accepted.

B.5.3.2.4
The cycle of "tan" function is not 2.\pi, but is \pi.

The cycles of tan is every (non-zero) multiple of \pi, including 2·\pi. The smallest positive cycle that is common to all the trigonometric functions is 2·\pi, and that is the cycle value used for all of the trigonometric helper functions in LIA. In effect, LIA allows all of the trigonometric operations to use the same argument reduction calculation; no need for special (maybe slightly more accurate) argument reduction calculations for tan and cot operations. Since the equations listed are used as a basis for the LIA requirements, the mathematical equations listed are formulated for that purpose. Text has been added to the rationale on this.

B.5.3.2.5
The cycle is also \pi. The right-hand side of the cycle equation should be "cot" not "coth".

See above. The second part of the comment is accepted.

B.5.3.2.7
In the fourth equation, \pi/2 is multiplied by "i", but we do not understand this. We think that the "i." should be deleted.

Accepted.

B.5.3.2.9 (for example)

In the last equation, both plus and minus signs are allowed. Is this acceptable? We think that we only handle single-valued functions, and there would be no such freedom of sign. Are we wrong? By the way, what is the status of "Arccos" function (capitalized, Italic) in this document? Are they functions in the mathematics world?

The notation is taken from "Handbook of mathematical functions with graphs, formulas, and mathematical tables". The equalities that involve an uppercased elementary function name refer the multi-valued inverses. The equalities then say the values on the left side (seen as a set) is equal to the values on the right side (seen as a set). This could be expressed more conventionally, using set notation, but this notation is borrowed since it is convenient. The lowercase names for inverses are the corresponding functions, denoting the principal value for the multi-valued inverse. This explanation has been added to the rationale.

B.5.3.3.2 (for example)
Since hyperbolic functions are only defined as references to trigonometric functions, we think that these many equations are not necessary for them. Only the basic equivalences with trigonometric functions would suffice.

True, but that was not entirely clear from the outset. Recall for instance that the C standard (of 1999) does the relationships the other way around. But doing then that way around is less convenient when generalising to cover also cot, sec, and csc. The lists of equations have been trimmed.

B.5.3.3.3
In the sixth equation, "= cos(-y)" also holds but is not mentioned. This seems inconsistent with similar equalities in "sinh", "tanh", ...

Accepted.

B.5.3.3.4
The cy[c]le of "tanh" function is not i.2.\pi but is i.\pi.

See above, on cycles.

B.5.3.3.5
The cycle is also i.\pi.

See above, on cycles.

B.5.3.3.11

The condition given for the third equation is "if ...". This is obviously incomplete.

Accepted.

Since that relationship is not used as the basis of any LIA specification, it is removed.

C, 1st paragraph of p.90
The word "examplified" is a typo. It should be "exemplified". The same error can also be found in the 2nd paragraph.

Accepted.

C.1, "plusitimes_c(I)"
The right parenthesis at the end of the expression should be deleted.

Accepted.

C.1, paragraph just after "min_seq_i(I)"
Defining the type of "xs" as "array of IINT" is not sufficient in Ada.
The index type of the array should also be defined.

Accepted.

C.1, 1st paragraph of p.94
The sentence beginning with "The parameter functions are constant ..."
is grammatically incorrect. Something should be inserted after the comma, or the comma should be replaced by a semicolon.

Accepted.

The comma is replaced by a semicolon.

C.1, "eq_F,i(F)" (for example)
We do not think that these functions are defined by the Ada reference manual. Equality operators (equal and not-equal) are defined for two operands of the same type.

Accepted.

C.1, "power_c(F),I"

This function has a note "Not LIA-3". What is the intent to list such functions in the binding examples? There are many more occurrences of the same note in sections C.1 - C.5.

Accepted in principle.

The listed function is part of the standard library for Ada, and is specified in the section on complex arithmetic in the Ada standard. However, LIA-3 does not specify that operation. It is still relevant to list, and a real binding is likely to list it too, together with a full specification of that operation. An explanation for the "Not LIA-3" marks has been added to the introduction of annex C.

C.1, "ln_F->c(F)"
This function has two bindings. Readers may be unable to note this fact, since many functions with similar names are listed. Isn't it possible to show two expressions in an easily recognizable form? If the expressions are short, two expressions are shown in a line with an "or" delimiting them. Some display form similar to this one would be desirable for long expressions.

Accepted.

C.1, "ln_F->c(F)"
The right parenthesis at the end of the second expression should be deleted.

Accepted.

C.1, "arcsinh_i(F)" (for example)
The function name is written as "ArcSinH". The same function in the Ada reference manual is "Arcsinh". The method of capitalization is different. We think that those names defined in the Ada reference manual should be used.

Accepted.

C.1, "arccosh_i(F)->c(F)"
The corresponding Ada function name is defined to be "ArcCosH", but why isn't this "ArcCosHc"? "arcsinh_i(F)->c(F)" is a quite similar function, and its corresponding Ada function is "ArcSinHc". The same comment also applies to "arcsech_i(F)->c(F)".

Accepted.

C.1, "convert_i(F)->i(F')"

The constant "i" is defined in a generic package. If there is more than one imaginary type, separate "i"s exist for each of these types. Therefore, "i" cannot be used without a qualifying package name.

Accepted.

An introductory comment in C.1 on this has been added.

C.1, "convert_c(F)->c(F')"
There is a TeX formatting error (position of star).

Accepted.

C.1, "convert_c(F'')->c(F)"
The description of Ada wide characters as UCS-2/UTF-16 is not appropriate. The Ada reference manual does not impose such strong constraints on the external representation of wide characters.

Accepted in principle.

There is an implicit assumption about no data loss when doing input/output. Since the Ada standard is not explicit about it, the only reasonable option is to represent wide characters strings externally as UCS-2/UTF-16 big endian (i.e., network byte order, so that the endianness is not platform dependent). Text has been added to the binding example to note this.

C.2, 6th paragraph of p.100
An "and" would be necessary between "CFLT" and "IFLT".

Accepted.

C.2, 7th paragraph of p.100
A sentence is erroneously split into two paragraphs.

Accepted.

C.2, "eq_c(I)" (for example)
How can these operators be defined? We think that introducing such operators in C is difficult. Since C does not have operator definitions, users cannot define operators by themselves. A compiler can introduce such operators, but extending the language in such a way is hard to justify.

Correct. But the suggested syntax is for future standards primarily. If a user or library author wants to define these operations, they will need to use some other syntax.

C.2, "itimes_I->i(I)"
What is the type of "II"? "II" can be used for any imaginary types and thus its type is universal. Introducing such universal object in C would be difficult.

Accepted in principle.

The type of II is int _Imaginary, which will be automatically widened in expressions and parameter passing to a larger integer datatype (a compiler may still warn about mixed signed/unsigned, but should not do so if II is a language extension rather than library defined). Explaining the types to be "int _Imaginary" and "float _Imaginary" has been added in text just before the II and I bindings (given towards the end of the binding example), and a forward reference is added just before first use..

C.2, "eq_c(F)"
We think that the operator "==" is not language defined for two complex operands. Equality operators in C are defined for arithmetic or pointer types. Complex types are not considered arithmetic in C.

Rejected.

Arithmetic types are integer types and floating types. Floating types are real floating types and complex floating types. Imaginary types (annex G) are also floating types, and hence arithmetic types.

C.2, "exp_i(F)"
What is the meaning of a "dagger in parentheses"?

Accepted.

The dagger should be an asterisk (two occurrences).

C.2, "ln_F->c(F)"
The right parenthesis at the end of the expression should be deleted.

Accepted.

C.2, "logbase_F->c(F)"

Is it intentional to use the function name "logbasec", while other similar functions are "clogbase"?

Accepted, but for the time being moot.

logbasec takes a float argument and returns a complex. There need to be a name difference from the one returning a float. That name may well be clogbase instead. HOWEVER, due to the intricacy of power and logbase, most (except one) of those operations have been deleted from the draft. They may be reinstated later, when (and if) a usable specification for each of them is available.

C.2, "arccos_i(F)->c(F)"
The function name is defined to be "acos". However, the corresponding function name for Ada is "ArcCosc". Why isn't the C function name "acosc"?

Accepted in principle.

The suggested name for Ada is changed to Arccos.

C.2, 2nd last paragraph of p.108
Some parts of the type descriptions are redundant. In the list of syntax, none of "y", "z" and "FXD" appears.

Accepted.

C.2, "imaginary_unit_c(I)"
Why are the parentheses doubled?

Accepted.

The parentheses are removed, and the explanatory sentence after is expanded.

C.3, 4th paragraph
The hyphenation of "namespace" between "s" and "p" is not considered acceptable.

Accepted.

C.3, 7th paragraph
The sentence "CFLT is used below to designate one of the floating point datatypes." is not correct. "floating point datatypes" should be changed to "complex floating point datatypes".

Accepted.

C.3, "max_err_mul_c(F)"
The font of "<CFLT>" is not appropriate.

Accepted; also for max_err_div_c(F).

C.3, "signum_F"
What is the meaning of the question mark?

Accepted.

The C++ standard does not mention signb.

C.3, just after "polaru_F"
The phrase "where x and y are expressions of an imaginary or complex floating point type" is not quite precise. For example, the type of "u" is not defined. We think that the names "FLT", "IFLT" and "CFLT" should also be defined for C++, and they should be used in such type descriptions.

Accepted.

C.3, "ln_i(F)->c(F)"
There is a TeX formatting error.

Accepted.

C.3, "logbase_c(F)"
This function should also have the comment "(note parameter order)".

Accepted.

C.3, "arccos_c(F)"
There are three "arccos" functions defined in 5.3.2.9. But, here only two of them are given bindings.

Accepted.

The missing one is added.

C.3, 2nd paragraph of p.117
Shouldn't "C print/scan family" be changed to "C printf/scanf family"?

Accepted.

C.4, next to "signum_i(I)"
Here, a binding for a complicated expression "mul_I(abs_I(y), signum_I(x))" is given. Why is a binding necessary for such an expression?

That is an explanation of the two-argument SIGN operation in Fortran in terms of LIA operations. It is equivalent to the copysign operations of IEC 60559.

C.4, 1st paragraph of p.124
A sentence is erroneously split into two paragraphs.

Accepted.

C.4, "power_c(F),I"
What is the meaning of the question mark?

Accepted.

*The ** operator covers integer type power of a complex value. The question mark is removed.*

C.4, "ln_F->c(F)"
The right parenthesis at the end of the expression should be deleted.

Accepted.

C.4, just after "arccsch_c(F)"
The phrase "and i of integer datatype" is not appropriate, since "i" is not used anywhere.

Accepted.

C.5, "exp_i(F)"
The type of the operand in the reference notation is strange. Why is "i" multiplied in the operand position?

Accepted in principle.

This is an explanation of the cis function in Common Lisp. If the binding is written as exp_i(F)(x) --- cis(im(x)), it would have to be marked as unsupported, since there is no imaginary type in Common Lisp. A “note argument type” note has been added.

C.5, "ln_F->c(F)"
There is a TeX formatting error.

Accepted.

C.5, "ln_i(F)->c(F)"
There is a TeX formatting error. A right parenthesis should be inserted at the end of the expression.

Accepted.

C.5, "logbase_i(F)"
The comment "(note parameter order)" should also given to these functions.

Accepted.

C.5, "arcsin_F->c(F)"
The function name is defined to be "asin", but the corresponding name in Ada or C has "c" at the end of the name. What is the reason for this difference?

Accepted in principle.

Common Lisp had dynamic typing. asin can be given a floating point or complex float point argument. When given a floating point argument with absolute value larger than 1, a complex valued result is returned, not an error. This is in contrast to Ada and C, for instance. An explanation has been added in the introduction of the example binding for Common Lisp..

C.5, "arctan_i(F)"
The function name is "atanx". Why isn't it "atanc" like in other languages?

Rejected.

That is because Common Lisp is a dynamically typed language, and the common practice is for functions to return a complex value if the true result is complex, even though none of the arguments is a complex. So arctanx is a restricted version of arctan in this language.

C.5, last paragraph of p.135
"but n1 and n1 are ..." should be changed to "but n1 and n2 are ...".

Accepted.

D
Is there any policy in selecting references? If the intent is to list every major language in the reference, C# which is now an international standard should also be listed.

Accepted in principle.

The list has been updated and trimmed to a smaller number of references (essentially only references used, but also closely related references are kept).

D
The note in [15] should also be given to [7] and [8].

Accepted in principle.

The note is deleted.

D
The reference [9] will soon be withdrawn from the list of international standards.

Accepted.

The reference is deleted.

D
The name of the journal in the reference [52] seems incorrect. It would be "SIAM Journal on Numerical Analysis".

Accepted.

At the resolution of comments meeting, it was suggested that (ISO) standards should be sorted by its ISO number. However, that makes related standards sort far away from each other. The current sorting is therefore still based on programming language name, with standards related to that programming language listed just after the language entry. General programming language related standards are sorted before the programming languages themselves. Subheadings have been inserted.

United Kingdom

UK1) General

The standard is too big with too many functions. The ballot for the previous draft closed 22 months ago. With WG11's limited available effort, a smaller, and thus more rapid, standard is advisable.

Noted.

UK2) General

The set of proj functions for Ada, which are not replicated for other languages, look particularly odd.

It is unclear what is meant by this comment. Ada has no "proj" functions, nor does LIA-3. C has one though, listed in the C binding, with the note "not LIA-3.

UK3) General

Language implementors and standardisers may not find the suggestion that FCD 10967-3 adds lots of new functions and features appealing, particularly when the facilities they provide are already present in the language in other, sometimes only very slightly different, forms.

Firstly, all of the operations are optional, as noted in the conformance clause. Secondly, as also noted in the conformance clause, bindings are allowed to modify the specifications. And thirdly, as should be apparent from the binding examples, features already present in the respective languages are to a large extent matching the specifications given by LIA-3.

UK4) General/Technical

The accuracy of evaluating a computer function f(x) where x is some floating point value should be judged not by its closeness to the value of the corresponding mathematical function and value F(x), but by its closeness to the value of F(x') where x' is some value such that abs(x-x') < 0.5 * ulp and ulp is the value of a unit in the last significant place of x This reflects the fact that a floating point (complex) value really represents one of a range of values. J H Wilkinson used this idea in his scheme of 'backwards error analysis'. To take an extreme case, a computer with floating point with 10 significant digits should not need to find the remainder of exp(100) after dividing by 2pi when asked to calculate sin(exp(100)) but instead give any value between -1 and 1, or better still signal an error (value has no meaning).

Rejected.

The suggestion goes against the approach used in IEC 60559, and more significantly, LIA-1 and LIA-2. However, as for the particular case of a trigonometric operation on a very large magnitude argument, the big_angle parameter signifies a "cutoff" point for these. Still, there are suggestions that there should be no such finite cutoff point, but that one should compute accurate results also for large arguments to these operations.

Note also that the approach suggested would render the (IEC 60559, with which LIA is compatible) "inexact" notification useless, since all results would be inexact, given that all arguments then would be inexact.

The values for the max_error parameters are largely taken from the Ada standard. This is now noted in the rationale.

UK5) General/Technical
It is unnecessary to define the value of a function when one of its arguments is not a number (NaN, or infinity, or overflow, etc). These values can sensibly be left undefined.

Rejected.

IEC 60559 and LIA-2 specify operation results for infinity and NaN arguments. Just about all implementations are based on IEC 60559, and ignoring that is not helpful. The results for these arguments may at times be obvious, but so are the results for "ordinary" arguments in many cases. Indeed, the infinities ARE (extended) numbers. Note also that 'overflow' is an exceptional value, and as defined in LIA, will never occur as an argument to an operation.

UK6) General/Technical
GB cannot imagine a use for a complex integer data type to which so much space is devoted, nor indeed an imaginary (as distinct from a complex) floating point type. Since 1991 Fortran has provided facilities which would allow the user to create their own Gaussian integer type, and yet there has not been a great proliferation of such packages (or applications using such packages) in the public domain - nor has their been any clamour for intrinsic (compiler) support for such a type.

Rejected.

Common Lisp provides for complex integers, and even complex rationals. Ada and C provides for imaginary floating point datatypes (for IEC 60559 based implementations).

In addition, there are requests to extend complex to complex fixed point, and the current specification to complex integers may serve as a boilerplate for such an extension.

UK7) General/Editorial
The 'not LIA-3' notation which appears opposite some operations in each of the language bindings is puzzling. If the operation is not in the standard what is the purpose of the description in the binding?

Accepted in principle.

This is done for some relevant operations that are standardised for the programming language of the example binding, but where that operation is not included in LIA-3. An explanation has been added to the beginning of Annex C.

UK8) General/Editorial
The notation is confusing to the point of opacity. It would have been easier to understand (and use just as much paper) to spell out the argument and result data types rather than forcing those unfamiliar with all the multitudinous functions to hunt through document to find them.

Accepted in principle.

A cross-reference annex of all the LIA operations, helper functions, and parameters, listing part and section where specified, will be added before publication.
[[This is deferred due to time limitations, but the annex will be added.]]

UK9) General/Editorial
In particular, saying that "*" was for syntax that was "already provided" (in contrast to "†") is very strange since, in many cases, the standard in question provides that syntax. What it does not provide is the datatype for the already-provided syntax to act upon! Perhaps the wording for "*" is for where the language provides not only the syntax, but also the operations?

Accepted in principle.

That is indeed the intent, and has been the intent also in LIA-1 and LIA-2, which used the same formulation in the example bindings annex. Please note the preceding sentences of that paragraph. Where the operation in the language is optional, or has syntax that *depends on the datatype, that is noted in the example bindings (editorial mistakes exempted). The introduction to annex C has been clarified on this point.*

UK10) Technical
Fortran provides two COMPLEX datatypes, not one: COMPLEX(KIND(0.0)) aka COMPLEX; [and] COMPLEX(KIND(0d0)).

Accepted.

UK11) Technical
The choice of IEEE_RINT for the "floor", "rounding" and "ceiling" operations is, I believe, incorrect, simply because it is (and should be) only applicable to IEEE machines not in general. In particular, for "rounding", ANINT would seem to be a more obvious choice.

Accepted in principle.

The operations are renamed; FLOOR, CEILING, ROUNDING.

UK12) Technical
Does CMPLX(x)**y not implement what FCD 10967-3 suggests should be POWER(x,y)?

Accepted.

But the editor is still a bit hesitant about the max error requirement then.

UK13) Technical
Does SQRTC(x) not implement what FCD 10967-3 suggests should be SQRT(CMPLX(x)).

Accepted in priciple..

There is no standard "intrinsic function" named SQRTC (that the editor could find) in Fortran. However, LIA-3 now suggests using SQRTC for returning a complex square root value for a real argument.

UK14) Technical
Does LOGC(x) not implement what FCD 10967-3 suggests should be LOG(CMPLX(x)).

There is no standard "intrinsic function" named LOGC (that the editor could find) in Fortran. However, LIA-3 now suggests using LOGC for returning a complex ln value for a real argument.

UK15) Technical
There are two forms provided for "ln[F->c(F)](x)", the first of which is LOGC(x) the second of which is a complicated formula using the variable I. In Fortran, the mathematical "i" is written as "(0,1)". For the maths "i", use italic lower-case to avoid confusion. In any case [see technical comment C.6a] it's already provided as LOG(CMPLX(x)).

Accepted in principle.

The two forms are kept, but more clearly marked as alternatives. (0,1) is used for the complex unit.

UK16) Technical
FCD 10967-3 states that "Arithmetic value conversions for complex in Fortran is done via explicitly converting the real and imaginary part (and recomposing the results)." This does not appear to be true. The CMPLX intrinsic may be used to convert one complex type to another (or indeed any type to complex).

Accepted in principle.

The sentence has been extended with the use of CMPLX as an alternative.

UK17) Editorial
'(not LIA-3)' on page 124 should be 'Not LIA-3' for consistency with other appearances, if they are to remain.

Accepted.

UK18) Editorial
There is a '?' by the <complex> ** <integer> operation, on page 124. If this is meant to denote uncertainty, then this operation is indeed defined in Fortran and has been since the 1966 standard.

Accepted.

UK19) Editorial
Why is there a query at the C++ signb operation on page 114 ?

Accepted in principle.

The “?” is replaced by a dagger, since signb is not mentioned in the C++ standard.*

UK20) Editorial
As the deadline for this vote is October, it seems very likely that by then the revised Fortran standard will, if not published, be on the verge of publication. Therefore '1539-1:1997' could be changed to '1539-1:2004' on pages 118 and 138. This would have no effect on the binding so far as I can see.

Accepted.

The references are updated to the latest versions.

UK21) Editorial
There is a "(" missing in the fifth MODULO line on page 120.

Accepted.

UK22) Editorial
Spelling "cosec" as "CSC" is ugly. "ACSCHC" is even worse, and is equivalent to ACSCH(CMPLX(x)) anyway.

Noted.

csc and arccsc are the spellings used by Handbook of Mathematical functions. “a” is the abbreviation used by Fortran/C/C++ for “arc” (<u>that</u> abbreviation is ugly, but too late to change). The ACSCHC suggested name is kept, and since there is no direct standard alternative, no alternative is given (the bindings examples otherwise do not list several non-standard alternative bindings).