EDITOR'S DRAFT 1.1 (1999-02-03)
DISPOSITION OF COMMENTS ON CD3 OF 10967-2

NOTE: These dispositions have not yet been discussed by WG11, and are at places sketchy. They are to be completed and modified as per decisions to be made at the next WG11 meeting. This document is only a basis for discussion at that meeting.

PROJECT NO: JTC 1.22.33

SUBJECT: CD Ballot for CD 10967-2: Information technology - Language Independent Arithmetic - Part 2: Elementary Numeric Functions (Third CD Ballot); Disposition of Comments

> TITLE: Ballot comments on document ISO/IEC JTC1/SC22N2824 : CD 10967-2:
> Information technology - Language Independent Arithmetic - Part 2:
> Elementary Numeric Functions (Third CD Ballot)
>
> SOURCE: AFNOR
>
>
> France votes NO to document SC22/WG11 CD 10967-2. The vote will be
> reversed if the following comments are satisfactorily resolved.
>
> Comment 1.
> Committee Draft subsection: C.4
> Category: Request for rewording (minor editorial)
> Title: Separation of C binding from C++ binding
>
> Rationale:
>
> This proposed rewording is a consequence of comment 2.
>
> Proposed change:
>
> a) This subsection should read "C" (and not "C and C++")
>
> b) For reasons exposed in comment 2, we proposed that the last
> sentence (refering to C++) of the first paragraph of section C.4 be
> removed.
>

Accepted.

>
> Comment 2.
> Committee Draft Section: C
> Category: subsection that should be included (major editorial)
> Title: Proposal for a C++ binding
>
> Rationale:
>
> C++ is a quite different language from C in its features, namely type
> safety, name space management and overloading. Requiring C++ to have
> the same binding is not necessary and unjustified. C++ has its own
> mecanism to maintain (library functions) compatibility with C.
>
> Proposed change:
>
> We feel that the following subsection should be included as C++
> binding.
>
> ================== cut here ======================

> C.??? C++
>
> The programming language C++ is defined by ISO/IEC 14882:1998,
> Information Technology -- Programming Languages - C++.
>
> An implementation should follow all the requirements of LIA-2 unless
> otherwise specified by this language binding.
>
> The operations or parameters marked "[dag]" 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 identfier is provided.
>
> LIA-2 recommands that all identifers suggested therein be defined in
> the namespace std::math.
>
> The C++ datatype "bool" corresponds to the LIA-1 datatype "Boolean".
>
> Every implementation of C++ has integral datatypes "int", "long int",
> "unsigned int" and "unsigned long int" which (can) conform to LIA-1.
>
> C++ has three floating point datatypes that (can) conform to LIA-1:
> "float", "double", "long double".
>
> The additional integer operations are listed below along with the
> syntax to invoke them:
>

minI(x, y)	min(x, y)	
maxI(x, y)	max(x, y)	
min_seqI(xs)	min_element(xs_start, xs_end)	
max_seqI(xs)	max_element(xs_start, xs_end)	
dimI(x, y)	positive_difference(x, y)	[dag]
sqrtI(x)	sqrt(x)	[dag]
powerI(x, y)	power(x, y)	[dag]
dividesI(x, y) [dag]	divides(x, y)	
evenI(x)	x % 2 == 0	
oddI(x)	x % 2 != 0	
gcdI(x, y)	gcd(x, y)	[dag]
lcmI(x, y)	lcm(x, y)	[dag]
gcd_seqI(xs)	gcd(xs_start, xs_end)	[dag]
lcm_seqI	lcm(xs_start, xs_end)	[dag]
add_wrapI(x, y)	add_wrap(x, y)	[dag]
add_ovI(x, y)	add_overflow(x, y)	[dag]
sub_wrapI(x, y)	sub_wrap(x, y)	[dag]
sub_ovI(x, y)	sub_overflow(x, y)	[dag]
mul_wrapI(x, y)	mul_wrap(x, y)	[dag]
mul_ovI(x, y)	mul_overflow(x, y)	[dag]

>
> where x and y are expressions of type INT and where xs is expression of
> type a sequence (list<INT>, deque<INT>, vector<INT>, valarray<INT>) or
> INT[]. xs_start and xs_end are expressions denoting the beginning and one
> past end iterators of xs.

>
> The additional non-transcendental floating point operations are listed
> below, along with the syntax used to invoke them:
>
> minF(x, y) max(x, y)
> maxF(x, y) max(x, y)
> min_seqF(xs) min(xs_start, xs_end)
> max_seqF(xs) max(xs_start, xs_end)
>
> floorF(x) floor(x)
> roundingF(x) round(x) [dag]
> ceilingF(x) ceil(x)
>
> dimF(x, y) positive_difference(x, y) [dag]
> add3F(x, y, z) add(x, y, z) [dag]
> sumF(sx) accumulate(xs_start, xs_end, 0.)
> dprodF->F'(x, y) ???(x, y) [dag]
> mul_addF(x, y, z) mul_add(x, y, z) [dag]
> iremF(x, y) remainder(x, y) [dag]
> sqrtF(x) sqrt(x)
> rsqrtF(x) reciprocal_sqrt(x) [dag]
>
> add_loF(x, y) add_low(x, y) [dag]
> sub_loF(x, y) sub_low(x, y) [dag]
> div_restF(x, y) div_rest(x, y) [dag]
> sqrt_restF(x) sqrt_rest(x) [dag]
>
> where x and y are expressions of type FLT and where xs is expression of
> type a sequence (list<FLT>, deque<FLT>, vector<FLT>, valarray<FLT>) or
> FLT[]. xs_start and xs_end are expressions denoting the beginning and
> one past end iterators of xs.
>
> The parameters for operations for approximating real valued
> transcendal functions can be accessed by the following syntax:
>
> max_err_hypothF err_hypothenuse<FLT>()
> [dag]
>
> max_err_expF err_exp<FLT>() [dag]
> max_err_powerF(b, x) err_power(b, x) [dag]
>
> max_err_sinhF(x) err_sinh(x) [dag]
> max_err_tanhF(x) err_tanh(x) [dag]
>
> big_radian_angleF(x) err_radian_angle(x) [dag]
> max_err_sinF(x) err_sin(x) [dag]
> max_err_tanF(x) err_tan(x) [dag]
>
> big_angleF big_angle<FLT>() [dag]
> max_err_sinuF(u) err_sin_cycle(u) [dag]
> max_err_tanhF(u) err_tan_cycle(u) [dag]
>
> where b, x and u are expressions of type FLT.
>
>
> << The rest of this subsection is identical to the corresponding part of
> the subsection titled "Java". >>

> ================== cut here ======================

Accepted in principle.

(??? will be changed to dprod; err_radian_angle will be changed to big_radian_angle; C++ documents will be checked to see if the rest is appropriate. See also the US comments on the C and C++ binding(s) below.)

>
> Comment 3.
> Committee Draft subsection: 1.1
> Category: request for rewording (minor editorial)
> Title: reference for bindings
>
> Rationale:
>
> This proposal is a logical consequence of comment 2.
>
> Proposed change:
>
> We propose that the paragraph 1.2p3 be rephrased as follow:
>
> ISO/IEC 10967-2 covers most numerical functions required by the ISO
> standards for Ada, Basic, C, C++, Fortran, Extented Pascal, ISLisp,
> and PL/I. In particular, specifications are provided for...

Accepted.

Summary (France): all of the comments from France are accepted (or accepted in principle).

> _____ end of France comments; beginning of Germany comments _____

> German vote on ISO/IEC CD 10967-2:
> Information technology - Language Independent Arithmetic -
> Part 2: Elementary Numeric Functions
>
> Reference No. ISO/IEC JTC1/SC22 N 2824
>
> Germany votes: NO with comments
>
> The document still contains many editorial and English language errors
> such as incorrect usage of the singular or the plural, of technical
> or mathematical terms (e.g. monotonous instead of monotonic (function)),
> typos and misprints as well as structural errors (e.g. chapter structure
> of sections 5.2.7 - 8 does not match Annex A where 5.2.7 - 10 appear).

Accepted in principle.

The editor will attempt to find and correct English language errors (but since Germany has provided no itemized list, the editor is not sure to find all/the same ones that Germany has found). 'Monotonous' will be changed to 'monotonic'. The editor would appreciate an itemized list of the rest of the incorrectly used terms. The document will be rechecked against typos/misprints, both manually and with the help of automatic spell checkers. But since Germany has provided no itemized list, the editor might not find the same ones. The structure of Annex A (rationale) will be fixed to match the structure of the normative text. (See also the comments from Japan and the US below.)

> Germany still believes that the problem of summation of many terms,
> especially of floating-point numbers or of simple products
> thereof (i.e. the dot product) is not addressed adequately and
> poses a fundamental problem in many numerical applications.
> However, Germany has noticed improvements in this area and is willing
> to provide assistance in this matter.

Rejected, since Germany has not provided (directly or by reference to published papers) a list of operations with semantics to WG 11.

Even so, an exact summation operation has already been provided. For the first revision of the LIA-2, further operations to adequately address e.g. dot product can be added. WG 11 does not wish to further delay LIA-2 by adding operations at this stage.

[dot product can be computed by doing all the multiplications needed, both with mulF and mul_loF, putting all in a 'sequence' (list, vector) and apply sumF. Could add an operation for exact dot product, but not sure if that is what Germany requests.]

> Also, some references are out of date, e.g. to the Fortran standard.
> The new Fortran standard is Fortran 95, published in 1997. It should be
> cited throughout, and adjustments should be made in the text.

Accepted.

Summary (Germany): Comments accepted as best can be, given that itemized lists are lacking, and suggested extra operations with semantics have not been provided.

> _____ end of Germany comments; beginning of Japan comments _____

> CD Ballot (SC22 N2824, due 1999-01-26)
>
> ISO/IEC 10967-2,
> Information technology - Language independent arithmetic - Part 2:
> Elementary numerical functions
>
>
> Japan disapproves the draft for reasons shown below.
>
> ---
>
> 1. Error limit
>
> The definition of conformity to this standard has been improved, but
> there still remain some points to be revised.
>
> We understand that the error limits of each function should be defined
> by each implementation, not by binding standards. This interpretation
> is supported by the descriptions in Clause 8 and Annex B. However, it
> would also be possible to interpret that the error limits may be
> defined by binding standards.
>
> Error limits defined for Ada numeric functions are rather loose than
> those defined in this standard. This leads to an unfortunate
> consequence that Ada implementations conforming to the Ada standard
> does not necessarily conform to this standard. We have heard that, in
> order to escape from this situation, the binding standard for Ada can
> itself define error limits larger than those defined in this standard.
>
> We cannot deduce this explanation from the text of the standard. If
> this explanation is a right one, it should be explicitly mentioned in
> the text. In any case, it should be clearly defined whether a binding
> standard can define error limits or not.
>
> In A.2, it is suggested that a binding standard should define error
> limits, but this can only be taken as informative. The same thing
> should be stated in the normative part of the standard.
>
> In the fourth paragraph of Clause 2, the term "conflict between a
> binding standard and this part of ISO/IEC 10967-2" appears. It is not
> clear what type of conflict is allowed. We consider that some "upper
> bound" of conflict is necessary, since without such upper bound, a
> binding standard can define anything totally disregarding the
> definitions in this standard.

Rejected.

Error limits are fixed by binding standards (not implementations), as already per the normative text. Binding standards can take the error limits as specified in LIA-2, or adjust them (covered by the “conflict between” part).

No concrete suggestion of “upper limit” of “conflicts” has been suggested for consideration. Without such a suggestion it is hard to find one that may fulfill the request. One could add a note saying that ‘conflicts’ should be kept as small as possible; on the other hand, that opinion should be apparent anyway. An alternative is to recommend that certain ‘conflicts’ be handled by specifying a *separate* operation in the binding standard.

> 2. Definition of F
>
> The symbol F is used throughout the document to represent a "floating
> point datatype". There is some confusion on the usage of F:
>
> (a) In each function definition, F looks to represent a subset of R.
>   A condition clause "if x in F" should read "if x is not a special
>   value". This standpoint is stated in the definition of
>   "arithmetic datatype" (4.2).
>
> (b) In other places, F looks to represent the type which may contain
>   special values. See the second paragraph of 5.2.
>
> Usage (a) is the fundamental one, and is the way adopted in LIA-1 and
> the previous versions of LIA-2. The document should clearly state
> this standpoint and avoid any ambiguous usage of F.
>
> We suggest possible improvement below, with <<addition>> and
> <replacement> to the text.
>
> == suggested changes in wording ==================================
>
> 4.1 Symbols
>
> The following symbols are defined in ISO/IEC 10967-1:1994, and used
> in this part <<for an integer type I and for a floating type F and F'>>.
>   ...
> The following symbols represent <<special>> values defined in IEC 559...
>   ...
> These <<special values>> are not part of the <<value>> set F, but if
> iec_599F has the value true, <the floating point datatype corresponding
> to F has these special values available in it>.
>
> <<For an arithmetic datatype F conforming to ISO/IEC 10967-1:1994,
>   the symbol F is also used to designate its value set that consists
>   of only those values in R (for a floating point datatype) or in Z
>   (for an integer datatype), specifically in specification of a function
>   or operation for the datatype.>>
>
> 4.2 Definitions
>   ...
> arithmetic datatype: A datatype whose values are member of Z, R or C.
>   ....
> signature: A summary of information about an operation or function.
>   ... The signature
>   addI: IxI -> I U {integer_overflow}
>   ....
> <<Note: I represents a value set of the target integer datatype which
>   contains only those values in Z, i.e. I does not contain NaN and
>   infinity values.>>
>
> 5.1 Additional basic integer operations
>   ...
> I is an integer datatype conforming to ISO/IEC 10967-1<<:1994>>.
> Integer types conforming to ISO/IEC 10967-1<<:1994>> usually do not
> <have> any NaN or infinity values, even though they may do so.

> Therefore this clause has no specification for such values as arguments.
>   ...
> <<Note: When the datatype I includes such special values, each conforming
>   implementation shall document specification for the cases when such
>   special values appear as arguments. (see 8.)>>
>
> 5.2 Additional basic floating point operations
>   ...
> F is a floating point datatype conforming to ISO/IEC 10967<<:1994>>.
> Floating point datatypes conforming to ISO/IEC 10967-1<<:1994>> usually
> do <have> NaN and infinity values. Therefore, in this clause
> <each function or operation specification includes specifications
> for the cases when such values appear as arguments>.
>
> == end of suggested changes in wording ==========================

Accepted in principle. I and F are subsets of Z and R respectively. Wordings suggesting otherwise will be corrected.

> 3. minor errors
>
> We found many minor errors in the standard. We judge that this CD
> text is still immature, and should be improved before going to DIS
> stage.
>
> Foreword: According to the ISO home page, the full name of ISO is "the
> International Organization for Standardization". They do not use
> British spelling for the name of the organization.

Rejected. British spelling is used throughout the document.

> Foreword: The scope of SC22 should be written correctly.

Accepted.

> 1.1: The current wording "C/C++" implies that C and C++ are compatible
> languages. This is not true, and they should be written as "C, C++"".

Accepted.

> 1.1: An "and" should be added at the end of (e).

Accepted.

> 3.: The second word of the title should be written in lower case.

Accepted.

> 4.1: The operator "|" is not enumerated in the list of notations.

Accepted. Will be added.

> 4.1: "it" should be changed to "is" in the following sentence.
> "... difference of conventions it the arccot function.".

Accepted.

> 4.1: The period after "s2" should be changed to a comma, in "s1 and
> then s2. Etc.". "Etc" should be changed to "etc".

Accepted.

> 4.1: The current list of exceptional values is "integer_overflow,
> floating_overflow, underflow, and undefined". "overflow" should be
> added to this list. "overflow" is used, for example, in 5.4.5.

Accepted in principle. ["integer_overflow" and "floating_overflow" may be merged to "overflow", just as there is only one "invalid".]

> 4.1: "an" should be changed to "a", in "an highly accurate result".

Accepted.

> 4.2, precision: "109671" should be changed to "10967-1".

Accepted.

> 5.1: We cannot understand why the word "additional" is used in the
> title of this section.

They are specified *in addition* to the ones specified in LIA-1.

> 5.1.1, Note 1: The conditions of two alternatives are the same. In
> what case should we use the first alternative, and in what case the
> second alternative?

This is an informative note giving two *equivalent* alternative expressions for n. Since they are equivalent, the conditions are the same. [add 'or equivalently']

> 5.1.2, max_seqI: The definition is not complete. What happens if n =
> 0 and -infinity is available?

This case is intentionally omitted, for reasons explained in the second paragraph of 5.1.

> 5.1.2, min_seqI: The same as above (+infinity).

See above.

> 5.1.4, shift2I: We consider that "invalid" does not occur in this
> function.

Accepted.

> 5.1.4: shift10I: The same as above.

See above.

> 5.1.8, gcdI: The definition is not complete. What happens if n = 0
> and +infinity is available?

See above.

> 5.1.9, add_ovI: We consider that the conditions I /= Z and I = Z are
> not adequate. Even if an implementation supports multiple precision
> arithmetic, an integer type can never cover the whole set of
> mathematical integers. There might be an argument that an integer
> datatype I can be any subset of Z conceptually. But, then, what
> happens if I consists of all the positive values of Z?

Rejected.

It is not uncommon for programming languages to provide an unlimited integer datatype (see e.g. ISLisp). It is unbounded in the sense that there is no preset limit of the size of the values. The storage resources available determine which values can be represented at the moment. In such cases I is said to be equal to Z, even though resource exhaustion may prevent the representation of a particular, even "small", value, at that time and computer process.

> 5.2.3, dimF: "invalid" should be added to the list of possible output
> values.

Rejected. 'invalid' never results from this operation when given values in FxF, only when given 'special' values.

> 5.2.3: The function rndF is used in the definition of dimF, but we
> could not find its definition.

Accepted. rndF will be added to the list (in clause 4.1) of floating point helper functions inherited from LIA-1.

> 5.2.4, Note 3: "handed" should be changed to "handled".

Accepted for Note 2.

> 5.2.4: "invalid" should be added to the list of possible output values
> in the definitions of the following functions. rounding_restF,
> floor_restF, and ceiling_restF.

Rejected. For these operations, 'invalid' never results from an argument in F, only for 'special' arguments not in F.

> 5.2.5, iremF: In the eighth alternative, "yis" should be changed to "y
> is".

Accepted.

> 5.2.7: "invalid" should be added to the list of possible output values
> in the definitions of the following functions. add_loF, sub_loF,
> mul_loF, add3_midF, and mul_add_midF.

Rejected. For these operations, 'invalid' never results from an argument in FxF, only for 'special' arguments not in FxF.

> 5.2.7, add_loF and sub_loF: There are many question marks in the
> definitions, which means that the definitions are not yet fixed. The
> third and the fourth alternatives of add_loF are not consistent with
> those of sub_loF. The symbol "u" appears freely, without definition.

Partially accepted.

These are some of the specifications for which the editor has asked commenters to advise on which specification to select. A consistent, and hopefully useful, selection will be made. [Which one is most useful is still unclear.]

u is the continuation value accompanying the underflow of the corresponding "high" operation, and "appears freely" intentionally.

> 5.2.7, mul_addF: We cannot understand why 0 is handled separately.

It is handled separately because IEEE multiplying 0 and a negative number returns –0, not 0.

> 5.2.8, sum: We cannot understand why the sum is -0 when n = 0. We
> think that 0 is quite reasonable.

Rejected.

It is –0 because sumF([sumF[];sumF(xs)]) should be the same as sumF(xs) even when sumF(xs) is –0, to uphold the append law in the note. –0 is the identity element for IEEE addition, 0 is not, given that addF(-0,-0) is –0, while addF(0,-0) is 0.

> 5.3.3.2, Note 1: There is no predicate verb in the last sentence,
> which starts with "Something which ...".

Accepted. ". Something which" will be changed to ", which".

> 5.3.3.2, expm1F: "underflow" is listed in the list of possible output
> values, but underflow never occurs in this function.

Rejected.

The problem is that underflow may occur for some very abnormal, but LIA-1 conforming, floating point datatypes. Removing "undeflow" here must be accompanied by a small change to LIA-1, or an added restriction in LIA-2.

[If we include the added restriction to LIA-2 (fminN < epsilon/(2*r); or <=), to be retrofitted to LIA-1 when revised, we can accept this comment!]

> 5.3.3.3, powerF: 1^infinity is defined as invalid(1). We think that 1
> is quite reasonable in this case.

This is a controversial point. Some prefer 'invalid', some prefer '1'. In such cases LIA-2 combines these preferences. A binding standard may specify that it shall be '1' instead. (See also the reply to the US comment on powerF.)

> 5.3.3.3, Note 2: Non-overflow conditions is written as "when x = 0 or
> when x = 1". This is not consistent with the style of Note 1, where
> the same condition is written as "x in {0, 1}".

One of these equivalent statements will be selected for consistency of style.

> 5.3.3.4, powerm1F: The result value of the third alternative is -0,
> but this is not adequate. There may be implementations not supporting
> this value. We understand that the result value in such cases should

> be written as negF(0). The same comment applies to 5.2.5, etc.

Accepted.

> 5.3.3.4, powerm1F: 1^infinity-1 is defined as invalid(0). We think
> that 0 is quite reasonable in this case.

See above.

> 5.3.3.8, Note: "requireing" should be changed to "requiring".

Accepted.

> 5.3.3.9, logbaseF: The definition seems to assume that if +infinity is
> available, so is -infinity. Does this assumption always hold?

Accepted.

When not obviously available, the +-infinity occurs as a continuation value to 'pole'. Implementations are free to substitute such continuation values when the specified one is not available. Wording will be added to 4.1 and 6.1 to this effect.

> 5.3.3.9, logbaseF: We consider that the result should be "invalid"
> when x = +infinity and y is in F, and when x = 1 and y = +infinity.

PRELIMINARY: Rejected. The specification follows the principle given in A.5.3.1.5.c. [editor:DOUBLECHECK!!]

> 5.3.3.9, logbaseF: The intersection of the conditions of the 16th and
> the 17th alternatives is not empty.

Accepted (though there is no conflict of results). One of them will have the current intersection removed.

> 5.3.5: Something is wrong in the sentence "... for each the ...
> functions are ...".

Accepted. That sentence will be replaced by: "Three different operations are specified for each of the 'conventional textbook' trigonometric functions."

> 5.3.6, Note: Since this Note gives a definition, it should not be a
> Note, and should be moved to the main part of the text. We could not
> deduce the contents of this Note from other definitions in this
> document.

It is not, and is not intended to be, a consequence of other requirements. The contents of this note (or something equivalent) should not be made normative, but falls under what a binding standard can do.

> 5.3.6.2, sinF: The second axiom is not correct. pi/4 should be
> changed to pi/2.

Accepted.

> 5.3.6.2 -- 5.3.6.8, sinF -- cscF: "invalid" should be added to the
> list of possible output values for these functions.

Rejected. Invalid will not result from an argument in F.

> 5.3.6.9 -- 5.3.6.15, arcsinF -- arccscF, We think that the range of
> result values should be mentioned in the definitions of reverse
> trigonometric functions.

The paragraph in clause 4.1 beginning “Many of the inverses are multi-valued.” explains this. [Could add some text to that paragraph in 4.1.]

> 5.3.6.11, Note 1: The word "section" is not adequate. "clause" should
> be used.

Accepted.

> 5.3.6.11, arcF: There are many question marks in the definition of
> this function, which means that the definition is not yet fixed.

This is one of the operations for which the editor has asked for advice on which selection to make.

> 5.3.6.13, arcctgF: We do not know the mathematical meaning of this
> function. Is this function necessary?

As explained in clause 4.1, arcctg is the arccotangent function that is discontinuous at 0. Some programming languages base its arccotangent operation on the continuous version, some on the discontinuous version. It is felt that it would be inappropriate for WG 11 to fix on one of them, but rather let binding standards choose, or even leave it to programmers to choose.

> 5.3.7, Note 2: It says that the negative angular unit is not
> considered. This is not true. In sinuF, for example, the negative
> angular unit is explicitly considered.

Accepted. The note will be deleted.

> 5.3.7.5, tanuF: "-u/2" in the conditions of the first and the second
> alternatives should be changed to "u/2". When x approaches 180 degrees
> from the second quadrant, tan(x) approaches to 0 from the negative
> side.

Accepted.

> 5.3.9.1: The value "max_error_radF" is used, but we could not find its
> definition.

Accepted. [define or replace by max_error_sin or max_error_tan?]

> 5.4, Note 2: "10967does" should be changed to "10967 does". A period
> should be added after "etc".

Accepted.

> 5.4.1, Note 1: The first "are" in "both are I and I' are" should be
> removed.

Accepted.

> 5.4.1 -- 5.4.6: There are many question marks in the definitions of
> these functions, which means that the definitions are not yet fixed.

These are some of the operations for which the editor has asked advice on how to best specify them. [It is still unclear what is best.]

> 5.4.1 -- 5.4.6: The notion "sNaN without notification" seems strange.
> Is it different from qNaN?

Yes. sNaN and qNaN are special values. qNaN does not result in a notification *when used*, while sNaN does produce a notification *when used*. One can in principle produce/convert to an sNaN *without* giving a notification at production/conversion time.

This text (“sNaN without notification”) occurs in questions on which semantics to select.

> 5.4.4, convert(nearest): "x" in the last alternative should be printed
> in italic font.

Accepted.

> 5.4.5: "overflowif" should be changed to "overflow if".

Accepted.

> 5.4.5, resultD: We feel that the sentence "D is a fixed point type, it
> cannot conform to LIA-1 ..." is somewhat strange as an English
> sentence.

Accepted. This will be changed to “"D is a fixed point type. D cannot conform to LIA-1 ...”

> 5.4.6, convert(nearest): The result value of the sixth and seventh
> alternatives is "overflow". We consider that this should be changed to
> "floating_overflow".

Accepted. [Though ‘overflow’, ‘integer_overflow’, and ‘floating_overflow’ may be merged.]

> 5.5.2: An "and" is necessary in the first sentence. It should be
> inserted between "resultF(x, downF)" and "shall".

Accepted.

> 5.5.2: It is mandatory for an implementation to prepare a numeral for
> positive infinity and NaNs. We think that this rule is an
> over-specification. An implementation supporting these special values
> but not providing numerals is a reasonable one.

Rejected.

Then infinities and NaNs can only be generated by expressions that also normally result in a notification. It is useful to be able to denote these values without catching or disabling notifications. In addition, it is already allowed to omit these ‘operations’ from the set of conforming operations.

> 6.: The usage of "so that" in the second line is not a recommended
> one. It should not be used in a negative statement.

Accepted. “so that” will be replaced by “in such a way that” [better??]

> 6.: The last sentence in the second paragraph contains two "if"s.

Accepted. The sentence will be split in two sentences, one about what shall be returned, the other about what shall be documented.

> 7.: "10967and" should be changed to "10967 and". "of clause 5" should
> be changed to "in clause 5".

Accepted.

> A.1: "Sufficient" should be changed to "sufficient".

Accepted.

> A.4.1: "denormalized" should be changed to "denormalised".

Accepted.

> The clause A.5.1.7 should be added.

There is an A.5.1.7...

> The clauses A.5.2.8 and A.5.2.9 should be deleted.

Partially accepted. Their headings will be deleted, their texts moved to A.5.2.7.

> The clause A.5.3.1.5 should be deleted.

Rejected. This is an additional clause(?) explaining matter that has no heading in the normative text.

> A.5.3.1.5: "thought" should be changed to "though" in "thought they
> are valid if the ...". "the an approach" should be changed to "the
> approach". "if" should be changed to "is" in "approach to zero if
> from ...".

Accepted.

> The clause A.5.5 should be added.

Accepted.

WG11 specially thanks Japan for its very thorough, detailed, and well needed proofing of LIA-2.

> _____ end of Japan comments; beginning of Sweden comments _____

>
> From: Ann Flod (ann.flod@its.sa)
> Subject: Ballot on SC 22 N 2824
>
> Replacement of the Swedish vote on SC 22 N 2824 and FCD 10967-2.
>
> Yes for approval with comments as below:
>
> Minot comment:
>
> 1. A consistent and practical choice must be made for the by the editor
> 7-marked specification lines.

Accepted. The ?-marked specification lines will be given consistent, and hopefully useful, specifications.

> 2. The bindings examples annex should be better completed for more of the
> languages than just Ada.

Accepted. More programming language specifications will be looked through for appropriate binding examples.

> Editorial comment:
>
> Some lines go a bit far into the margin, and should be wrapped or
> otherwise shortened.

Accepted.

> ____ end of Sweden comments; beginning of UK comments _____

> SC22 Letter Ballot N2824 - Approval Ballot for FCD 10967-2 Language
> Independent Arithmetic - elementary Numeric Functions
>
>
> UK votes NO with comments to CD 10967-2 (LIA-2).
>
> STRATEGIC COMMENTS
>
> 1 Clause 5.4.6 includes questions to the reviewers concerning the
> behaviour of convert(... , F to D, x) when x is a signalling NaN. It would
> help reviewers to have a complete list of such questions, and the
> arguments in favour of each possibility (it would also help the editor
> because the questions are more likely to be answered).

Note taken for future such questions.

> 2 A large part of this draft defines the values of the arithmetic
> functions when one or more input arguments is an IEC559 continuation
> value. This is neither desirable nor necessary. Part 1 of the standard
> does not define the basic arithmetic operations under such circumstances,
> and part 2 should not do so either.
>
> Does any programming language standard do so? Or is planning to do so? Or
> has even considered doing so? If there is no candidate for usage of this
> facility then it should be omitted.

The floating point operation specifications for C9x include such specifications.

It is more considered a shortcoming of LIA-1 not to include such specifications for IEC 60559 special values. One of the greater advantages with LIA-2 is that such specifications are included. IEC 60559 special values are available today on all commercially significant computer platforms.

Indeed, all commercially significant computer platforms support denormal numbers. One simplification that should be considered when revising LIA-1 and LIA-2 is to only cover the case that denormF is true, or even that iec_559F is true (which implies that denormF is true and that infinities and NaNs are available).

> TECHNICAL COMMENTS
>
> 1 Clause 4.1 defines three separate meanings for [and] (square
> brackets): (i) designating a closed interval, (ii) to delimit a finite
> sequence of values, (iii) a set of finite sequences. Multiple meanings
> for square brackets, or any other term/usage, may potentially lead to
> errors in understanding by readers.
>
> NOTE 5 says 'it should be clear from context if [X] is a sequence of one
> element ...'. If the note can be rewritten to read truthfully 'It is
> always obvious ...', then perhaps the multiple meanings can be justified,
> but the UK cannot check this since the source was not available in
> electronic form.

The ‘it should be clear’ was intended to be more polite than the more direct statement. In the text of LIA-2 it is always obvious which is intended. The

source has been made available to the UK committee. The note will be reformulated for clarity.

> 2 Annex B - Partial conformity belongs in clause 2, not as an informative
> annex.

Rejected. Compare LIA-1, where a similar partial conformity is defined in an informative annex.

> 3 Clause 5.1.2 - We do not think that -infinity is ever available for an
> integer type, but if it is, then the definitions leave max_seq and
> min_seq undefined for the case when n=0 and -infinity is available.

LIA-1 does not rule it out, so we cannot assume that it is never available. However, it is not at all common to make them available, so adding specifications for them would be over-specifying.

> 4 Clause 5.1.5 - defines the result as 'round(sqrt(x))'. 'round' should
> be replaced by a round function (in math font) which itself needs to be
> defined as a new helper function in 5.1.1.

Font change accepted. The definition of round is already in clause 4.1.

> 5 Clause 5.1 states ''Integer datatypes conforming to ISO/IEC 10967-1
> usually do not contain any NaN or infinity values'. The integer datatypes
> defined in ISO/IEC 10967-1 NEVER contain any NaN or infinity values.

Rejected. LIA-1 does not rule out infinities or NaNs in integer datatypes conforming to LIA-1, though these values never occur in the set I. Compare floating point datatypes in LIA-1, where F never has these values in it, but the corresponding datatype usually does have such values.

> 6 Clause 5.3.7 Natural logarithm. The ln*(F) function is defined with a
> Real -> Real signature. This should be Real -> Complex, i.e. R -> Z.
> Should there be any special consideration when the input argument is a
> denormalized floating point value?

Rejected. LIA-2 does not cover Complex. For ln*F in LIA-2, Real->Real is sufficient. Special consideration is given to denormalised values in ln1p*F, but there is no point in doing so for ln*F.

>
> EDITORIAL COMMENTS
>
> 1 The draft refers throughout to 'sinus hyperbolicus' rather than
> 'hyperbolic sine', and similarly with many other functions. This does not
> appear to the UK to be common usage, nor consistent with LIA-1.

These names are intended to be the Latin names for these functions, names from which the mathematical names derive. Corrections to the Latin language usage are welcome.

> 2 Clause 6, para 3 says 'An implementation shall suppress spurious
> notifications'. What is a 'spurious notification'? The UK is sceptical
> about the value of suppressing any notification.

A “spurious notification” is a notification that arises from a subexpression in the computation of an end result, but when the correct (but maybe approximate) end result does not deserve to be turned into a notification. Such notifications are quite appropriately avoided/caught/suppressed.

> 3 Clause 6 includes 'NOTE'. It should be 'NOTES'.

Accepted.

> 4 Clause 6 includes a sentence 'The results of ... is ...' instead of
> 'The results of ... are ...'.

Accepted.

> 5 Clause 5.4.2 Note 2 Misprint 'ISO/IEC 10967does'

Accepted for 5.4 Note 2.

> 6 Is it necessary for each annex to begin on an odd-numbered page, and
> thus some blank pages? It was not necessary in 10967-1.

It is not necessary, but considered better typographic practice.

> 7 Clause 5.3.1.2 The sentence (Those extentions ... .) should be a note,
> and 'extentions' should be 'extensions'.

Both accepted.

> 8 Clause 5.1.3 What is a 'limited integer type'? Is this what 10967-1
> defines as an integer type with bounded = true?

Accepted. Wording will be changed to use “bounded”.

>
> OTHER COMMENTS
>
> 1. When faced with a new draft standard, it is important that reviewers
> are provided with a list of what has changed in order to simplify their
> task. No such list is provided with this draft. It is strongly
> recommended that such a list is provided with the next versions, although
> this need not be too detailed as long as it indicates the major areas of
> change. It is not necessarily appropriate, for example, to provide change
> bars.

Noted. Given that the changes from CD2 to CD3 have been so major, that nearly everything has changed, such a list would be considerably longer than the document itself in this case.

> 2. Many forms of review are only possible if the source is available, for
> example, how is a particular word used throughout the standard, writing
> programs to check completeness and consistency. The source has not been
> published. It is strongly recommended that an electronic searchable
> version is provided with the next (paper) version

The sources are normally not published. But if asked, sources can be obtained from the editor to national bodies/working groups. In this case the editor has forwarded the (LaTeX) source files to a member of the UK committee.

> _____ end of UK comments; beginning of USA comments _____

> The US National Body votes to Approve with comments ISO/IEC CD 10967-2,
> Information Technology - Language Independent Arithmetic - Part 2:
> Elementary Numeric Functions. Please see comments listed below.
>
> Global:
>
> IEC 559 has been renumbered to IEC 60559. The particular LIA-2
> reference is page 3: 3. Normative References.

Accepted. [ISO/IEC 60559 or just IEC 60559?]

> Change the notations for intervals which do not include the end point:
>]x,z] to (x,z]
> [x,z[to [x,z)
>]x,z[to (x,z)
> based upon CRC Standard Mathematical Tables. The particular LIA-2
> reference is page 4: 4.1 Symbols.

,.,.,.,[could one make an argument about European vs. American practice??]

> Replace "I not equal to Z" (shows up in 5.1.1 for wrapI(x) and elsewhere)
> with "I is bounded". Replace "I equal to Z" with "I is unbounded".

These are equivalent. No change necessary [but could be done if the other
formulation is more pleasing].

> Consider changing conformance to allow some legacy languages such as
> FORTRAN
> to comply to LIA-1 but not LIA-2, but to at least pass through the LIA-1
> conformant data types when dealing with LIA-2 level interchange. This will
> allow interoperation even when languages cannot be extended to conform to
> LIA-2.

We do not understand what specifically should be changed. Implementations are
of course allowed to comply with LIA-1 without necessarily complying with LIA-2.

> Specific pages:
>
> Page 3: 2 Conformity: Note 2: Remove: ", which in turn includes
> (explicitly or by reference) a binding for IEC 559 as well".
> Reason: Our understanding of LIA-1 is that it does not require
> conformance to IEC 559 (IEEE-754).

,.,.,.,[no, but it recommends that bindings be made for IEC 559 anyway]

> Page 8: 4.2 Definitions: Change: The number of digits in the fraction of a
> floating point number to The number of base-r digits in the fraction f of
> a floating point number of the form 0.f * r**e.

Accepted in principle. e will be changed to t (since e is used for other
purposes).

> Page 9: Change subnormal to some other term as it conflicts with the
> usage in IEEE-854.

The intent is that is should coincide with that usage (0, -0, and denormal numbers all being smaller in magnitude than fminN. [editor:DOUBLECHECK!]

> Page 9: Move: "For some operations the exceptional value invalid is
> produced only by input values of -0, +INF, -INF, or sNaN. For these
> operations the signature does not contain invalid." from page 83
> A.5.3.1 Specification format to page 9 5. Specifications for the
> numerical functions. It can be made into a Note.

This, in other words, is also said in 4.2/signature. Moving it appears to be overdoing the emphasis of this.

> Page 9: 5.1 Additional basic integer operations: Rewrite the sentence:
> String formats for integer values usually do contain (signalling) NaNs,
> however, when that string format is regarded as an (non-ISO/IEC 10967-1)
> integer datatype.
>
> It makes no sense and we do not understand the intent, so do not have a
> suggested rewrite. Also, we have yet to seen an integer sNaN, so we
> disagree with the word "usually" here.

Rejected.

The sentence makes perfect sense. A string datatype can be regarded as an integer datatype (and in some programming languages, strings *are* regarded as integers for arithmetic operations). The string “1” then would represent 1, the string “01” also represents 1, the string “NaN” (‘N’ followed by ‘a’ followed by ‘N’) can be used to represent a quiet NaN. Strings such as “Hello” are then signalling NaNs, since they do not represent an (integer) number, and would (expectedly) cause a notification if used as a number.

This view is applied in LIA-2 for conversion of a string to a number datatype which can be a more conventional number datatype.

[Could change “string format” to “string datatypes (regarded as number datatypes)”, or some such.]

> Page 10: 5.1.2 Integer maximum and minimum operations: Either add
> definition for the case of max_seq: n == 0 and -INF is available or remove
> "and -INF is available".

Rejected.

Adding that case would be an over-specification, since INF is usually not available in integer datatypes.

The other alternative of removing that part of the condition would restrict implementations that does provide INF to give a notification in this case, even though an INF value would be more appropriate.

> Either add definition for the case of min_seq: n == 0 and +INF is
> available or remove "and -INF is available".

See above.

> Page 10: 5.1.3 Integer positive difference (monus, diminish) operation:
> Should the note mention the case of x = +INF and y = +INF?

Why? This case is intentionally left open. See above.

> Change "limited integer types" to "bounded integer types".

Accepted.

> Page 10: 5.1.4 Integer power and arithmetic shift operations: Change
> "invalid(1)" to 1. Aside: Based upon other parts of LIA-2, it appears
> that "(1)" is a continuation value (not a footnote). But, where is that
> convention defined?
> Add the case: = 1 if x = 1
> Add the case: = +1 if x = -1 and y is even
> Add the case: = -1 if x = -1 and y is odd

Partially accepted.

The power(0,0) case (both for integers and floating point) is one of much controversy. The invalid with 1 as continuation value is a compromise specification. Binding standards are allowed to override this, and say that 1 is returned without notification in this case.

[We could recommend that binding standards include both the "original LIA" operation and a modified one, and let the programmer choose. The C binding example could exemplify this.]

Text will be added to clause 4 on that (...) after a notification value refers to the continuation value.

For y positive, the suggested three cases are already covered. For y negative the three cases will be added.

> Page 12: 5.1.8 Greatest common divisor and least common multiple
> operations:
> For the invalid case, change invalid to invalid(0) and remove: and +INF
> is not available.
> Change Note 1: "would be incorrect, since the greatest common divisor for
> 0 and 0 is infinity" to "was considered but rejected".
> Reason: INF | 0 means INF * n = 0 for some n. But there is no n that
> satisfies the requirement. Hence, +INF is not the GCD of 0 and 0. Also,
> we believe that 0 <= GCD(x,y) <= min(abs(x),abs(y)).
> In: gcd_seqI ([x1 ; ...; xn]), remove and +INF is not available.
> Add a note to that entry: { 0 } = { x1 ; ...; xn } means all xi are 0.

Rejected (except for the note).

0 is not an appropriate continuation value for gcd(0,0). The cardinal number infinity multiplied with an integer 0 is clearly 0. (This is different from the case of floating point where both the 0 and infinity are likely to be approximate.)

> Page 14: 5.2.1 The rounding and floating point result helper functions:
> Change: "= 0 if x = 0" to "= x if x = 0". Reason: To do the correct
> action for -0 and IEEE-754.

Rejected. The correct action for IEC 60559 is already done. Note that the = is a mathematical equals (not the operation eqF). x=0 is not true if x is negative zero.

> Add a note to that section: If denormF is false, the distinction between
> the
> ordering of rounding and underflow detection is not made. That is, a
> number x might be just under fminNF, so might round to fminNF in one
> implementation and be flushed to zero (before rounding) in another.

Accepted. [editor:DOUBLECHECK any changes needed to the specification of the helper functions resultF and trans_resultF]

> Page 18: 5.2.5 Operation for remainder after division and round to
> integer (IEEE remainder): Add the missing space in "yis" in: "if x is a
> quiet NaN and yis not a signalling NaN".

Accepted.

> Page 19: 5.2.7 Support operations for extended floating point precision:
> add_lo: If x, y, and addF(x,y) are all in F then the specified result is
> fine. But, one or more of x, y, and addF(x,y) are not in F, then there
> are two viewpoints: add_lo(x,y) is always zero (and there are no
> exceptions), or add_lo(x,y) is the same as addF(x,y) with the same
> exceptions. Both viewpoints are acceptable. Whichever you adopt, the
> definitions need to be redone. For example, the "underflow(0)?" entry
> should be either "0" or "underflow(0)". And the entry "0?" should be
> either "0" or "floating_overflow(+/-INF)".
> The same idea applies to sub_lo, mul_lo, div_rest, and sqrt_rest.

Accepted. (But a clear preference one way or the other would have been appreciated.)

> Page 20: Remove add3 as we see no need for it.
>
> Page 21: Remove add3_mid as we see no need for it.
>
> Page 22: Remove mul_add_midF as we see no need for it. It can be
> simulated with *, mul_lo, +, and add_lo.

Accepted. Now that sumF is included, add3 and add3_mid can be removed. Mul_add_mid (as well as add3(_mid)) can be implemented the way you suggest.

> Page 23: 5.2.8 Exact summation operation: Change result in:
> = sNaN if x = +INF and y = -INF to be qNaN.
> Change result in: = sNaN if x = -INF and y = + INF to be qNaN.

Rejected. The sNaN will appropriately result in a notification in the sumF operation.

> Page 25: 5.3.1.2 The trans result helper function:
> = underflow(nearestF(x)) clause has two problems:
> (nearestF (x) < 0 or x 0) seems wrong and we do not know what is should
> be. It also seems that if nearestF(x) == x, e.g., x is exactly
> representable as a denormal, there should not be an underflow.

Rejected (both).

(nearestF (x) < 0 or x > 0) avoids the case when −0 is the appropriate continuation value for the underflow (covered by a later case).

For the operations that use trans_resultF one either gives underflow for all denormal results, or for none of the denormal results. The reason is that the argument to trans_resultF already may contain an approximation, and then an underflow notification is appropriate for all denormal results. On the other hand if the end result will never have any denormalisation loss, underflow is explicitly avoided for those operations. If one wants the behaviour suggested, then resultF should be used instead, and the argument to resultF must be exact, or at least known not to induce any denormalisation loss.

> Page 26: 5.3.2 Hypotenuse operation: Change the specification so that
> hypot(+/-INF, qNaN) and hypot(qNaN, +/-INF) is +INF (not qNaN). Reason:
> if
> one of x or y is an infinity, then no matter what value the other
> argument has, the result is +INF.

This is based on the assumption that the qNaN represents an unknown (real) value. [We could add another operation (hhypotF) that ignores the qNaN, just like we have two f.p. max (maxF and mmaxF).]

> Pages 28-29: We disagree with many of the powerFF and powerFI subcases.
> We
> have complained before and been rejected, so we believe that it will do
> no good to give the detailed rewrites, but we can provide them if you
> would like them.

Rejected for the same reasons as before. It is, however, within the realm of a binding standard to modify these specifications (see the conformity clause). WG11 recommends, but does not require via LIA-2, that one specifies a separate operation, if different behaviour is judged to be needed. [Include an example of this in the example C binding.]

> Page 42: 5.3.6 Operations for radian trigonometrics and inverse radian
> trigonometrics: Change: "It [big_angle_r] shall have the following
> default value" to "It shall have a default value at least as large as".
> Reason: We know of implementations that use enough digits of pi for
> argument reduction to correctly compute the trig functions for all
> arguments.

Rejected.

WG11 is well aware that there are such implementations (indeed the previous LIA-2 editor made one such implementation).

However, the big_angle parameters are introduced for a quite different reason, as explained in the LIA-2 text: for larger and larger angular values the floating point values get sparser and sparser. At some point one or more revolutions go between representable values. The big_angle parameters have been chosen to cut off use of trig. operations long before that happens. Normalisation operations are specified by LIA-2 so that programmers, if those operations are made available to them, can normalise angular values to be within one revolution to avoid this sparsity problem, and maintain accuracy, still allowing angular values within about a thousand revolutions.

> Page 48: arcF: remove the "?inval?" items.

[The question is: which semantics is most appropriate? And there has been a controversy on these cases.]

> Page 48: arcF (we call it arctan2): We disagree with the +/-0, +/-0
> subcases. We have complained before and been rejected, so we believe
> that it will do no good to give the detailed rewrites, but we can provide
> them if you would like them.

The operation(s) corresponding to arcF are sometimes called 'arccot2', sometimes 'acot2', sometimes 'angle', and sometimes (with the arguments swapped!) 'arctan2' or 'atan2'.

The +-0 subcases are a source of controversy. The current specifications is a compromise. It is, however, within the realm of binding standards to override these subcases, or better supplement with an additional operation which overrides these subcases (letting the programmers choose which is most appropriate for the application at hand).

> Page 58: 5.3.7.11 Argument angular-unit arcus operation: Remove "?inval?"
> items.

[The question is: which semantics is most appropriate? And there has been a controversy on these cases.]

> Pages 66-72: For all the 5.4.* functions: Add:
> invalid if x is a quiet NaN and quiet NaN is not available in the
> target type.

Accepted.

> For all the 5.4.* functions: There are several cases to consider for
> signaling NaNs. It the type is changed (int - float, or float - int), or
> if the precision is changed (I - I' or F - F'), that is, a conversion is
> done, then signaling NaNs shall raise invalid and produce a qNaN as a
> continuation value. But, if the type and precison are the same, then it
> is implementation defined if the copy operation leaves sNaN alone or
> raises invalid and produces a qNaN as a continuation value. Reason:
> IEEE-754 allows the implementor to pick if copying sNaNs raises invalid or
> not. IEEE-754 requires sNaNs to raise invalid on conversions.

[Should *conversions* between same types be allowed? If so, should such a conversion be the same as a 'copy' operation?

Editor's current opinion: if conversion between same types are allowed, they are *not* the same as copy. The conversions should have a consistent semantics, same types or not (either sNaNs are converted quietly (if available in target), or notifyingly independent of type 'sameness'). 'Copying' (assigning (which may be a noop, if already at the right place), passing as parameter, or to the programmer invisible movement) an sNaN shall definitely NOT give a notification.]

> [MAJOR] For roundingF, floorF, ceilingF, change integer_overflow to
> integer_overflow or invalid (implementation defined). Reason: There are
> many hardware implementations of IEEE-754 that raise invalid for bad
> conversions from floating-point to integer.

Rejected.

The conversion as such is then not invalid, it is just that the result did not 'fit' in the target. Overflow is then appropriate notification in such cases.

It is, however, within the realm of binding standards to override this, preferably in a separate operation.

> Page 68: 5.4.4 Floating point to floating point conversions: It appears
> that LIA-2 is missing functions to meet IEEE-754 for directed rounding of
> binary <-- decimal conversions, e.g., "input" and "output" of string
> formats. Each function for input or output is supposed to use the current
> rounding direction. This also applies to 5.4.5 and 5.4.6 functions.

[The intent is that those would be covered by the conversion operations. Add text to clarify that...]

> Page 69: 5.4.5 Floating point to fixed point conversions: Need to add a
> definition or reference to LID. Should "limited" be changed to "bounded"?

[Why add a reference to LID? 'limited' will be changed to 'bounded'.]

> Page 73: 5.5.2 Numerals for floating point types: Change statically to
> dynamically in: The rounding circumstance should be statically
> determined, if other than the normal is at all available.
> Check signalling versus signaling. Both are used in LIA-2 (in particular
> on
> page 74).

[Static vs. dynamic. Static direction change should be provided. Dynamic, doubtful...]

> Expand note 2 in clause 7 to include COBOL, which is still the most widely
> used programming language. In particular, please add the following:
> "function asin(x) in COBOL".

Rejected.

Then we should also add SQL, MUMPS, and a whole host of other languages. The point of Note 2 is well served as is, without additions. Many further examples are provided in the bindings examples (though not for Cobol).

> Page 79: A.1.1 Specifications included in ISO/IEC 10967-2: Change
> "intened" to "intended".

Accepted.

> Page 82: A.5.2.7 Support operations for extended floating point
> precision: 3rd paragraph: Add a missing 'by' to: the high parts are
> calculated the corresponding floating point operations.

Accepted.

> Page 93: Annex C: Example bindings for specific languages: Remove: In
> turn, a complete binding for the ISO/IEC 10967-1 will include a binding
> for IEC 559.

,.,.,.,[LIA-1 recommends, but does not require, that one does a binding for IEC559 as well.]

> Page 99: C.2 Ada: Complete: Numerals for infinity...

Accepted.

> Page 99: C.3 BASIC: Complete: The BASIC datatype ???? corresponds to the
> LIA-1 datatype Boolean.

Accepted.

> Page 103: C.4 C and C++: Add the reference number for C++ to the first
> paragraph.

Accepted.

> Page 104: Change: divides(x, y) to y % x == 0.

Rejected. divides(0,y) returns false, while y%0==0 results in a notification. [Could change to (x!=0 && y%x==0).]

> Page 104: Change: evenI(x) x % 2 = 0 to evenI(x) x % 2 == 0.

Accepted.

> Page 104: Change: min(x, y) to fmin(x, y).

Accepted.

> Page 104: Change: max(x, y) to fmax(x, y).

Accepted.

> Page 104: Change: ffloorf(x), ffloor(x), ffloorl(x) to floor(x).

Accepted.

> Page 104: Change: ceiling(x) to ceil(x).

Accepted.

> Page 104: Change: dim(x, y) to fdim(x,y).

Accepted.

> Page 104: Change: ????(x, y) to dprod(x,y).

Accepted.

> Page 104: Change: mul_add(x, y, z) to fma(x,y,z).

Accepted.

> Page 104: Change: sqrtf(x), sqrt(x), sqrtl(x) to sqrt(x).

Accepted.

> Page 105: Change: hypotenuse(x, y) to hypot(x,y).

,.,.,.,[controversial cases? Two operations?]

> Page 105: Change: power(b, y) to pow(b,y).

.,.,.,.[controversial cases? Two operations?]

> Page 105: Change: ln(x) to log(x).

Accepted.

> Page 105: Change: ln1p(x) to log1p(x).

Accepted.

> Page 105: Change: log(b, x) to logbase(b,x).

Accepted.

> Page 105: Change: log1p(b, x) to logbase1p(b,x).

Accepted.

> Page 107: Add to: convertI-I-(x): cast, e.g., (type of I-)x.

Accepted in principle.

> Page 107: Change: rounding(y) to (INT)round(y).

,.,.,. [ties?]

> Page 107: Change: floor(y) to (INT)floor(y).

Accepted.

> Page 107: Change: ceil(y) to (INT)ceil(y).

Accepted.

> Page 107: Add to: cvtnearestF-D(y): sprintf().

,.,.,.,[Accepted in principle. arguments...]

> Page 107: Add to: cvtnearestD-F(z): strtod(z,NULL).

,.,.,.,[Accepted in principle. NULL???]

> Page 107: Either remove or replace the .... in: C provides non-negative
> numerals .... with for all its integer and floating point types.

Accepted.

> Page 112: C.6 Java: Remove long double and change three to two.

Accepted. [? Changes in the works...?]

> Page 121: C.7 ISLisp and Common Lisp: Complete: The details are not
> repeated here, see ....

Accepted.

> Page 125: C.8 Modula 2: Complete: Modula-2 provides non-negative numerals
> ....

Accepted.

> Page 129: C.9 Pascal and Extended Pascal: Complete: Pascal provides
> non-negative numerals ....

Accepted.

> Page 131: Annex D Bibliography: Item [1]: Change 559 to 60559. Add at
> end: (previously designated IEC 559:1989).

Accepted.

> Page 132: Add to item [27]: A corrected tenth edition has been published
> by
> Dover Publications, New York.

Accepted.

WG 11 thanks the US for its detailed and knowledgable review of the LIA-2
document.

> _____ end of USA comments _____
>
>
> _____end of SC22 N2879 _____