ISO/IEC JTC1/SC22/WG11 N469

2000-07-14

Disposition of comments for FCD 4 of 10967-2 (LIA-2)

____ Beginning of National Body Comments _____

FRANCE

France votes NO with comments to the FCD 10967-2: Language Independent Arithmetic - Elementary Numeric Functions. The vote will be reversed if the following comments are satisfactorily resolved.

EDITORIAL COMMENTS

Comment 1.
Final committee draft: whole document
Category: Request for replacing 'C9x' with 'C99' (minor editorial)

Rationale:
The C9x project had successfully been completed, and the revised standard was published in 1999 by ISO.

Proposed resolution:
Change 'C9x' to 'C99'. As a consequence the reference [18] should be updated.

Accepted. Changed in Bibliography and in annex C.4.

Comment 2.
Final committee draft: whole document
Category: Request for consistency with LIA-1 about 'datatype' (minor editorial)

Rationale:
LAI-1 used the term 'data type' (iin two words) whereas the current draft is using 'datatype'. We feel that for the sake of consistency, LIA-2 should adhere to LIA-1 terminology.

Proposed resolution:
Change 'datatype' to 'data type'.

Rejected. The change from "data type" to "datatype" was done at the advice of a delegate to WG11 from England. It reflects a change in the English language, as then noted by the English delagate. We will instead change LIA-1, when revised, to use the term "datatype instead of "data type".

TECHNICAL COMMENTS

Comment 3.
Final committee draft subsection: C.5
Category: Request for improving C++ bindings (major technical)

Comment 3.1
Rationale:

The C++ programming language has its own preferred idioms (e.g. overloading, templates, iterators, ...) to express some constructs. We feel that it is strongly recommanded LIA-2 uses them. For example valarray is not the only possible way to form sequences in C++. One other popular way is 'C-like array' which can't have member function ('min()', 'max(), ...). C++ uses the 'iterator' concept to abstract algorithms over data strutures, which implies among other thing that requiring 'min()', 'max()', 'gcd()', 'lcd()', 'nmax()', 'nmin()' be member functions isn't likely to fit into the general iterator framework.

Proposed resolution:

*change 'xs.max()' to 'max(xs_start, xs_end_past_one)'

* change 'xs.min()' to 'min(xs_start, xs_end_past_one)'

* change 'xs.gcd()' to 'gcd(xs_start, xs_end_past_one)'

* change 'xs.lcm()' to 'lcm(xs_start, xs_end_past_one)'

* change 'xs.nmax()' to 'nmax(xs_start, xs_end_past_one)'

* change 'xs.nmin()' to 'nmin(xs_start, xs_end_past_one)'

These bindings were already suggested in our comments on the last draft.

Rejected. The example bindings are very long already, and they are just informative examples. Adding the above does not add any substantial correction to the binding as it is. In addition 'xs_start' and 'xs_end_past_one' are not well-defined, and there is no suggestion for how to define them for this binding.
A full binding for C++ could of course include also what France suggests.

Comment 3.2
Rationale:

We also feel that distinguishing between functions, having conceptually the same name but operating on values of different types, by prefix or suffix does not fit into the C++ approach to genericity. Instead, overloading should be used.

Proposed resolution:

* Change 'iremainder(x, y)' to 'remainder(x, y)'

* Change 'poweri(b, z)' to 'power(b, z)'

Accepted.

Comment 3.3
Rationale:

C++ allows a template function to be overloaded on the return type. That feature is exactly what should be used to suggest a binding for mult_F'->F. The scheme is extensible.

Proposed resolution:

Change 'dprod(x, y)' to 'product<FLT>(x, y)'

Rejected.

A large majority of mathematical functions and operators are generic/overloaded for all floating point types, so that the result type of dprod cannot be uniquely determined if dprod occurs as an operand/argument within such an expression.
However, the intended and numerically useful cases for dprod are those where the precision of the result is greater than (and preferably at least twice) the precision of the arguments.

Thus, if one has three floating point types, s, d, e, where precision(s) < precision(d) < precision (e), only one of
1. a) dprod(s,s) -> d, or
b) dprod(s,s) -> e, and
2. dprod(d,d) -> e
is useful. If all three types have the same base and prec(d) >= 2 * prec(s), then 1a is sufficient exact results (for arguments in s), and a conversion d->e is always possible without loss of information.

Note again that this is only an example binding. It is up to the C++ community to make a standard binding of LIA-2 for C++.

Comment 3.4
Rationale:

C++ defines a standard template class 'numeric_limits' to provide the user with information about numerical types. We strongly feel that the non-member functions enabling access to the parameters for operations approximating real valued transcendtal functions should be defined members of the template class 'numeric_limits'.

Proposed resolution:

* Change 'err_hypothenuse<FLT>()' to 'numeric_limits<FLT>::err_hypothenuse()'

* Change 'err_exp<FLT>()' to 'numeric_limits<FLT>::err_exp()'

* Change 'err_power<FLT>()' to 'numeric_limits<FLT>::err_power()'

* Change 'err_sinh<FLT>()' to 'numeric_limits<FLT>::err_sinh()'

* Change 'err_tanh<FLT>()' to 'numeric_limits<FLT>::err_tanh()'

* Change 'big_radian_angle<FLT>() to 'numeric_limits<FLT>::big_radian_angle()'

* Change 'err_sin<FLT>()' to 'numeric_limits<FLT>::err_sin()'

* Change 'err_tan<FLT>()' to 'numeric_limits<FLT>::err_tan()'

* Change 'smallest_angle_unit<FLT> to 'numeric_limits<FLT>::smallest_angle_unit()'

* Change 'big_angle<FLT>()' to 'numeric_limits<FLT>::big_angle()'

* Change 'err_sin_cycle<FLT>()' to 'numeric_limits<FLT>::err_sin_cycle()'

* Change 'err_tan_cycle<FLT>()' to 'numeric_limits<FLT>::err_tan_cycle()'

* Change 'err_convert<FLT>()' to 'numeric_limits<FLT>::err_convert()'

* Change 'err_convert_string<FLT>()' to numeric_limits<FLT>::err_convert_string()'

Accepted.

Comment 3.5
Rationale:

As far as casts are concerned, C++ recommands use the 'new style casts' syntax. Especially in the case of numerical conversion, 'static_cast' should be used.

Proposed resolution:

* Change '(INT2)x' to 'static_cast<INT2>(x)'

* Change '(INT)nearbyint(y)' to 'static_cast<INT>(nearbyint(y))'

* Change '(INT)floor(y)' to 'static_cast<INT>(floor(y))'

* Change '(INT)ceil(y)' to 'static_cast<INT>(ceil(y))'

* Change '(FLT)x' to 'static_cast<FLT>(x)'

* Change '(FLT2)y' to 'static_cast<FLT2>(y)'

Accepted.

Comment 3.6
Rationale:

As to how to convert values of arithmetic types to string type (or vice-versa), C++ has a powerful system of streams: these streams are the recommanded means for I/O issues.

Proposed resolution:

Replace use of 'sscanf', 'fscanf', 'fprintf', 'sprintf' with reference to 'locale and facets' (paragraph 22.2.2 of ISO/IEC:14882:1998).

Partailly accepted. The reference is added, but sscanf etc. remain in the example binding.

It is up to the C++ community to elaborate a complete binding of LIA-2 for C++.

Comment 3.7
Rationale:

C++ already defines binding for '+oo', 'qNaN', 'sNaN' as the return values of the functions numeric_limits<FLT>::infinity(), numeric_limits<FLT>::quiet_NaN(), numeric_limits<FLT>::signaling_NaN().

Proposed resolution:

* Replace 'INFINITY' with 'numeric_limits<FLT>::infinity()'

* Replace 'NAN' with 'numeric_limits<FLT>::quiet_NaN()'

* Replace 'SIGNAN' with 'numeric_limits<FLT>::signaling_NaN()'.

Accepted.

----------end French comments----------

JAPAN

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

Japan disapproves the draft for reasons shown below.

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.

----------for the core----------

Forward, 1st paragraph, last line: "organisations" should be "organizations".

**Accepted. (And “standardisation” to “standardization” in Annex B.
Oxford dictionary spelling is (attempted to be) followed.)**

1.1 Inclusions, 1st paragraph, 1st line:
Remove "of" after "Part 1".

Accepted.

4.1.4 Datatypes and exceptional values:
The symbols F and I should be introduced before their usage such as boundedI and eminF.

Accepted.

4.1.4, p.6, after NOTES, 2nd line: "tree" should be "three".

Accepted.

5.1.2 Integer maximum and minimum:
The return values for empty list should be:
max_seqI([]) = invalid(-Infinity), and
min_seqI([]) = invalid(+Infinity),
not as poles. In 4.2, pole is defined and reserved for continuous functions. Those functions max_seqI and min_seqI are not continuous on length of their argument lists in usual sense. (Of course, you may claim they are continuous functions under interpretation of cpo's.)

Rejected. 'pole' is here to be read as 'infinite result from finite arguments', which is a more general interpretation of 'pole'. The reason for not using 'invalid' but 'pole' instead is that the result is really well-defined, and no error has occurred. 'pole' is not reserved for approximations of continuous functions. In order to avoid any confusion, the “pole” exception is renamed to “infinitary”.

5.1.6 Divisibility tests, NOTES 1:
We believe that dividesI(0,0) = true, since dividesI(x,y) = x|y and x|y should be defined as "for some integer m, y= m x", and clearly we have 0= m 0 for any integer m.

Rejected. 0/0 is not well-defined, let alone an integer, so 0|0 must be false.

5.1.8 Greatest common divisor:
We believe that gcdI(0,0)=invalid(+Infinity), since notion of "continuous function" does not apply for this function and, thus, "pole" seems inadequate.

Rejected. 'pole' is here to be read as 'infinite result from finite arguments', which is a more general interpretation of 'pole'. The reason for not using 'invalid' but 'pole' instead is that the result is really well-defined, and no error has occurred. 'pole' is not reserved for

approximations of continuous functions. In order to avoid any confusion, the “pole” exception is renamed to “infinitary”.

5.1.8 Greatest common divisor:

A NOTE on the empty list case should be added in the definition of gcd_seqI, as in lcm_seqI.

Accepted.

5.2.1 The rounding and floating point result helper function, 2nd para.:

We don't see the reason why resultF(x, upF)=overflow(-fmaxF) for the case upF(x) < -fmaxF. "upF(x) < - fmaxF" means that x is not greater than y such that nearestF(y) < -fmaxF and thus resultF(x,upF) cannot be greater than resultF(y,nearestF)=overflow(-Infinity).
Any persuasive reason should be given, or change it to resultF(x, upF)=overflow(-Infinity).

Similar discussion holds for resultF(x, downF)=overflow(fmaxF) for the case downF(x)>fmaxF.

Rejected. result_F is defined this way for compatibility with IEC 60559.
The reason IEC 60559 specifies it so, is because those are the desired results when implementing interval arithmetic. This explanation is added to the rationale.

5.2.2 Floating point maximum and minimum:

The first two lines of the definition in max_seqF([x1, ?, x2]) should be unified into:

max_seqF([x1, ..., xn]) = invalid(-Infinity) if n=0.

max_seqF is not continuous on length of its argument list, and thus the notion of pole doesn't apply for this case. (see the discussion for max_seqI.)
The case when the implementation doesn't have "Infinity" should be left open as "implementation defined" as described in the last paragraph of 4.1.4. (For example, see 5.3.6.6. There is only a description:

powerF(0,y)=pole(+Infinity) if y<0

and it leaves open the case when the implementation doesn't have "Infinity". And it is far better to put a general recommendation or requirement rather than to put in individual case the description saying that the implementation should substitute fmin_N for "Infinity" if it doesn't have "Infintiy".)

Also the first two lines of the definition in min_seqF([x1,...,xn]) should be unified into:

min_seqF([x1,...,xn]) = invalid(Infinity) if n=0.

Partially accepted. The lines are unified, but using the ‘infinitary’ (ex-pole) notification.

5.2.4 Round, floor, and ceiling:

Commas between "Infinity" and "}" should be deleted.

(roundingF, floorF, and ceilingF)

Accepted.

5.2.4 Round, floor, and ceiling:

We are afraid that the value of rounding_restF(x) might be resultF(x-round(x),rnd_F) rather than simply x-round(x), since those values for floor_restF and ceiling_restF are resultF-ed.

No, rounding_rest is an exact operation, while floor_rest is exact for positive arguments, but not always for negative arguments, and ceiling_rest is exact for negative arguments, but not always for positive arguments. No change needed. This explanation is added to the rationale.

5.2.6 Square root:

We don't find any clear reason for sqrtF(-0)= -0.
sqrtF(x) is undefined for all negative value x, and -0 is a negative, or at least the limit of negative values, thus ...

As explained in the rationale, it is defined this way for compatibility with IEC 60559. Note that *lss*$_F$(-0, 0) is false, and *eq*$_F$(-0, 0) is true.

5.2.7 Supporting operations, NOTE1, 2nd line:

"returns" should be "return".

Accepted.

5.3.2 Sign requirements, 1st paragraph, 1st and 3rd lines:

"are shall" should be "shall".

Accepted.

5.3.6.1 powerI:

Add comma between "infinity" and -0.
Remove empty lines above and below this line.

(about powerFI) Partially accpted. Comma added. Empty lines kept, providing a weak grouping of the cases.

5.3.6.6 PowerF:

It is defined now that

powerF(0,0) = result_NaN2_F(0,0), and by definition, = invalid(qNaN),

whereas

powerFI(0,0)=1 and powerI(0,0)=invalid(1).

Are they consistent? Rationale should be given for their treatment.

For powerF the result is completely path dependent. For powerFI the result is 1 along both of the two existing paths. For powerI there is no continuous path, hence the invalid, but 1 is thought to be a useful continuation value, and some programming languages explicitly specify the result to be 1. No change to LIA-2. This motivation (differently worded) is in the rationale.

5.3.6.7 power1pm1:

It should be
power1pm1F*(x,1)=x .

Accepted.

5.3.6.8 Natural logarithm:

All helper functions should be defined as functions on R rather than on F.

In the draft, some helper functions such as lnF* are define as R->R whereas the others are defined as F->R, because, we guess, those specific helper functions have requirements on the real values such as "pi" and e that cannot be members of F. We cannot find out any benefit to distinguish such differences. We prefer to simple style.

Accepted in principle. Each signature for helper functions now specify for which non-F arguments extra requirements exist (or may exist).

5.3.6.12 Argument base logarithm:

It should be

logbaseF(1,y) = invalid(qNaN) for any y,
since for any fixed y0,
lim(x->1) logbaseF(x,y0) is undefined
because the limit depends on the approaching direction.

The only two limits are +infinity and -infinity. For logbase, and several other functions, LIA-2 makes an, in principle arbitrary, choice between +infinity and -infinity as the continuation value for the 'infinitary' notification at the pole at these points.
No change to LIA-2 (but the *notification* 'pole' is renamed to 'infinitary'). Same for holds for other log functions. Note the consistency with ln(y)/ln(x); Such a motivation is added to the rationale.

5.3.6.12 Argument base logarithm:
Remove two empty lines in the definition of logbaseF(x,y).

Empty lines kept, providing a slight grouping of the cases. (There are also NB comments to add empty lines at places...)

5.3.7.8 Inverse hyperbolic cosine:
For each helper function its domain should be specified clearly if it is not required to be total on R (though some are currently defined on F; we propose to change them to R.). For example,
arccoshF*(x) should be defined "if |x|>=1", and
arctanhF*(x) should be defined "if |x|<1".

Partially accepted.
The helper functions are tied to the mathematical functions, and they are intended to be undefined (in the mathematical sense) 'in the same way'. A general paragraph on this, in 5.3, has been added.

5.3.7.12 Inverse hyperbolic cosecant:
We don't find any reason why such a specific relation as
arccschF*(1)=arcsinhF*(1)
is given in the specification. Moreover we don't see whether it is a remark or a requirement since the current description simply says as "Relationship to the arcsinhF* approximation helper funcion". If it is a remark, it should be placed as a "NOTE" or in the annex.

Accepted. That requirement is removed. Other relationship requirements are reformulated with a "shall".

5.3.9.1 Radian angle normalisation, 2nd paragraph, 3rd line:
"x<=big_angle_rF" should be "|x|<=big_angle_rF".

Accepted.

5.3.9.1 Radian angle normalisation:
The signature of axis_radF should be
F -> ((I x I) x F) ...,
since the first part of the returned value is one of (1,0), (0,1), (-1,0) and (0,-1) and this fact could be clearly indicated by "(I x I)" rather than "(F x F)".

Accepted in principle. {(1,0),(0,1),(-1,0),(0,-1)} used. It is up to bindings to specify the type of those integer values. The short form used in the bindings examples, however, don't show the (programming language) types of the operations.

5.3.9.3 Radian cosine:
We don't find reasons why there is no requirement on

cosF*(pi/2) = cosF*(-pi/2) = 0
while there are requirements on the cases when cosF*(x) has the values 1, -1, 1/2 and -1/2.

Because the requrement on the zero result is implied by clause 5.3.2 (Sign requirements). There is no need to repeat them later for each operation. A short explanation about this is added to B.5.3.2.

5.3.9.4 Radian tangent:
"n*2*pi" should be replaced with "n*pi", since the period for tan(x) is pi rather than 2*pi.

5.3.9.5 Radian cotangent:
"n*2*pi" should be replaced with "n*pi", since the period for cot(x) is pi rather than 2*pi.

Accepted in principle. No change needed to the actual specification, but notes are added to explain why 2*pi here works just as well.

5.3.9.8 Radian cosine with sine:
Some functions such as cossinF and cossinuF (5.3.10.8) return a pair of results of other functions. There should be general definition on how to handle the result when one or both of the other functions return notification or special values, especially on how to define the continuation value with those ones returned from the other functions.

This is handled in the same way as in other expressions where there are calls to two or more LIA operations. E.g. recording of notifications are compounded, and for pairs each part is computed, notionally, separately.
A note referring to this, specialised for cosine with sine, has been added. See also the bindings examples, where the pair return is sometimes accomplished by using out parameters.

5.3.9.10 Radian arc cosine:
Is it not necessary to require
arccosF*(1) = 0 ?

Already required via clause 5.3.2 Sign requirements.

5.3.9.10 Radian arc cosine:
The expression "-1<=x<=1" should be rewritten as "|x|<=1".

Accepted (no factual change).

5.3.9.15 Radian angle from Cartesian co-ordinates:
It is defined now that
arcF(0,0)=0
whereas in the previous draft it was
arcF(0,0)=invalid.
Rationale should be given somewhere.

Rationale already given in Annex B Rationale. The reason is consistency with the specification of this operation in C; it is also believed that the current specification is more useful in practice than the one returning invalid.

5.3.10 Operations for trigonometrics with given angular unit:
The angular unit should be limited to positive value.

No one really wants to use a negative unit, and negative units make the current description unduely complicated.

Rejected. There is no mathematical, nor numerical, reason to exclude negative angular units. In addition, the added complication is very small.

5.3.10, NOTE2:
Put space between "is," and "min_angular_unitF".

Accepted.

5.3.10, last paragraph, last line:
A continuation value (in parentheses) should follow the word "invalid".

Accepted, with qNaN.

5.3.10.1 Argument angular-unit angle normalization:
The signature of axis_cycleF should be
F -> ((I x I) x (F...))...,
since the first part of the returned value is one of (1,0), (0,1), (-1,0) and (0,-1) and this fact could be clearly indicated by "(I x I)" rather than "(F x F)".

Accepted in principle. {(1,0),(0,1),(-1,0),(0,-1)} used.

5.3.10.12 arccotu, Range limitation:
Range limitation for arccotuF#(u,x) should be revised.

Though the expression given in the draft well limits the range in the sense that there are no values u and x which produce a result out of it, the expression is too loose (giving just the doubled range) for a specific value u.

Example of revision:
arccotuF#(u,x)
= max{0, min{arccotuF*(u,x), down(u/2)}} if u>0
= min{max{arccotuF*(u,x), upF(u/2)}, 0} if u<0

Rejected. The suggestion implies no factual change (as implied by clause 5.3.2), and the current expression is simpler.

5.3.10.13, definition of arcsecuF(u,x):
"(x<=-1 or x>=1)" should be rewritten as "|x|>=1".

Accepted (no factual change).

5.3.10.14, definition of arccscuF(u,x):
"(x>=1 or x<=-1)" should be rewritten as "|x|>=1".
(By the way, why does it differ from the expression given in arcsecuF?)

Accepted in principle (1<=|x| used where applicable). (No reason for the editorial difference.)

5.3.11.1 rad_to_cycle:
The second argument of rad_to_cycle should be "u" instead of "v", just as in the other arbitrary angular unit functions.

Rejected. v is used for naming consistency with cycle_to_cycle(u, x, v).

5.3.11.1 rad_to_cycle, 1st paragraph, last line:
The maximum error of rad_to_cycleF*(x,u) should be max_error_sinuF rather than max_error_sinF. Note that rad_to_cycle and cycle_to_rad are reciprocal to each other,

and thus one has the maximum error related to sin (cycle_to_rad), the other should have the error related to sinu.

Accepted in principle. Max_error_sin is replaced by max_error_rad for all three operations.

These operations are manifestations of underlying angle normalisation operations that need to have sub-1/2-ulp error bounds (through the use of additional guard digits), composed with a linear conversion.

For full conformity, these operations should have max error 1/2 ulp. The new clause A.4 discusses relaxation of accuracy requirements for angles that are very close to an axis.

----------for annexes ----------

B.4.1.2, 3rd line: The word "programme" looks strange. We prefer "program".

Accepted.

B.4.1.2, 5th line: We think that the word "whether" is inadequate here. It is used as the first word of an adverb phrase.

Accepted. "independently of whether" used instead of "whether".

B.4.1.4, 6th paragraph, 3rd line: There should be a grammatical error in the sentence "... continuation value to cause ...". Perhaps, the word "to" before "cause" should be removed.

Accepted in principle. Reformulated as “For example, \invalid\ without a specified continuation value may cause change of control flow (like an Ada~\cite{ada} exception), while \invalid\ with a specified continuation value may use recording of indicators.”.

B.5.1.2, 5th line: There should be a grammatical error in the sentence "... be possible handle at ...". Perhaps, "to" should be inserted before "handle".

Accepted.

B.5.1.3, 1st line: The direction of the quotation mark before "positive difference" is not correct. The mark before "monus" should also be used here.

Accepted.

B.5.1.7, 1st line: We cannot understand why "47" appears here. Is it the representation of an apostrophe?

Spurious characters deleted.

B.5.1.7, 3rd paragraph, 1st line: We cannot understand what the phrase "... does so for too many arguments, ..." means. The number of arguments of this function is fixed.

Accepted in principle. The phrase does not refer to the number of arguments given to a single call, but to the number of cases. 'arguments' replaced by 'cases'.

B.5.1.8, 1st line: One of two consecutive "a"s should be removed.

Accepted.

B.5.1.8, 4th paragraph, 2nd line: We think that the use of the noun "relative prime" is not common in mathematics. We think that the notion should be phrased as an adjective. Thus we prefer to change "relative primes" to "relatively prime".

Accepted.

B.5.2, 4th paragraph, 1st line: There is no subject corresponding to the verb "summarise".

Accepted in principle. Rephrased to "The following paragraphs is a short summary of the specifications of IEC~60559 regarding the creation and propagation of signed zeros, infinities, and \nans.".

B.5.2, 6th paragraph, 8th line: The word "thought" should be changed to "though".

Accepted.

B.5.2.2, 2nd paragraph, 1st line: The word "one" looks strange in this context.

Accepted in principle. Reformulated as "For floating point datatypes there is also usually a negative zero available,".

B.5.2.5, 2nd paragraph, 2nd line: We think that the word "nominator" should be changed to "numerator". The word "looses" should be changed to "loses".

Accepted.

B.5.3.6, 2nd paragraph of page 80, 4th line: We do not know the word "suppied". Is it a misspelling of "supplied"?

Spelling corrected.

B.5.3.11, 3rd line: The word "looses" should be changed to "loses".

Accepted.

B.5.4, 5th paragraph, 1st line: The phrase "greater that" should be changed to "greater than".

Accepted.

C.2, Ada binding:

The reference to ISO/IEC 11430 is not appropriate. It is provided as a supplement to the old Ada standard (8652:1987). The contents of this standard is included in the new Ada standard (8652:1995), and therefore 11430 is obsolete. It will soon be withdrawn.

Accepted.

The function name "Rem" is not possible. The word "rem" is a reserved word in Ada.

Accepted. Name changed to Ratio.

The type description "array of INT" is not complete. Every Ada array type should have its index type specified. If such generic array types, regardless of their index types are necessary, the "generic" mechanism of Ada should be used. The same comment also applies to Pascal (Pascal does not have generic mechanism).

Accepted in principle. array (Integer range <>) of INT used instead.

Functions returning max_error parameters are defined to have a parameter to differentiate among the floating point types. This is not consistent with the usual Ada style. In the usual Ada coding style, such functions should be defined as "attributes" with the type name as the prefix.

Rejected.
It is WG11's understanding that such 'attributes' can only be defined by the Ada standard itself, not by any separately developed library. That is why example bindings to 'attributes' only have been made to already existing such 'attributes'. If this is not so, then Japan must provide an explanation on how library writers can define new such attributes. If the Ada community so desires, they may augment the Ada standard to have these values as attributes.

We feel strange finding that the parameter name "Cycle" is given to the functions "arctanu" and "arccosu". Yes, the parameter name is necessary to distinguish these functions from the function "arc" with two parameters. However, there is no reason not to give parameter names to other functions. It would be more consistent to give names to all parameters of all functions. This is the usual style of Ada.

Rejected.
The parameter name is not needed for the other cases. When not necessary, it is customary in Ada programming to omit the parameter names in calls. In the binding example we have followed that custom. Note that the format chosen for these examples need not be followed in actual bindings or binding standards, and another format may be chosen that always express the parameter names more naturally (e.g. by using procedure heads rather than calls). This goes for any binding, not just for Ada.

An explanation has been added as to why the parameter name sometimes need to be spelled out in Ada function calls.

After the definition of "mul" function, the variable "z" is defined to have the type FLT, but this variable does not appear in the block. The same comment also applies to Basic, C, C++, Fortran, Haskell, Java, Common Lisp, ISLisp, Modula-2, Pascal, PL/I and SML.

Aceepted.

After the definition of "cycle_to_cycle", the phrase "z is an expressions" is grammatically incorrect. The "s" should be removed. The same comment also applies to C++, Fortran, Haskell, Java, Common Lisp, ISLisp, Modula-2, Pascal, PL/I and SML.

Accepted.

After the definition of "convert" functions, the generic type "INT2" is defined but "FLT2" is not. It is referred to in the definition of the "F->F' conversion" function, and thus should be defined. The same comment also applies to C, C++, Java and Modula-2.

Accepted.

C.3, Basic binding:

Several functions are not given in the binding. For example, we could not find the definitions of "powerFI", "mul", "axis_rad" and "axis_cycle" functions. Is this intentional?

Basic has only one numeric datatype, a floating point one. No integer datatype is provided. Therefore all operations involving integer datatypes are omitted from the example LIA-2/BASIC binding. For axis_rad and axis_cycle it is hard to find a good binding for BASIC, since tuples cannot be returned, nor can “out” parameters be used.

For the function "mul", the same comment also applies to C++, Fortran, Java, Common Lisp, ISLisp, Modula-2, Pascal, PL/I and SML.

Partially accepted. ISLisp has only one floating point type, Modula-2, Pascal, and SML are only required to have one floating point type, for PL/I the situation is not clear to the editor.

Your writing style seems to insert a space after comma for the functions, defined in the programming language, having more than one parameter. This style is often violated. The same comment also applies to many other languages.

Accepted. A space after comma in programming language pieces will be added where missing.

C.4, C binding:

The new version of 9899 has been published. The reference to the standard should be updated. Moreover, references to "C9X" as a preliminary draft are not appropriate. The word "C9X" which appears in many places of this FCD should finally disappear.

Accepted.

There are two definitions of the "divides" function. Is this intentional? How can we choose one of these? The same comment also applies to C++.

Neither of them are definitions. An LIA operation can have several different bindings (ways of expression) in the same programming language.

The definition of the mark "*" says "needs one name per integer datatype". However, there is no suggestion how these names should be constructed. If implementers choose different names, there will be portability problems.

Accepted. Following prior practice in C, a name suffix is now used to indicate the datatype.

We understand that C does not have "type generic macros" mechanism. The implementer of the numeric library cannot write these macros in C.

C99 has 'type generic macros', but library writers cannot define new ones (only use the ones built-in to C). The text about type generic macros is rephrased.

After the definition of "convert" functions, the variable "z" is defined to have the type "FXD". However this variable does not appear in the block. The type "FXD" is defined, but it does not appear in the block. The same comments also apply to C++, Fortran, Haskell, Java, Common Lisp, ISLisp, Pascal, PL/I and SML.

Accepted.

C.5, C++ binding:

The definition of the "divides" function (second one) is incorrect. The subexpression "y != 0" should be changed to "x != 0".

Accepted.

The comment "(unclear)" appears in many functions. We think that this is not acceptable as a description in a standard.

Accepted. The “unclear” remarks are removed. Added sentence explaining that the C++ standard does not make explicit which kind of max/min operations are intended.

After the definition of "cycle_to_cycle", the phrase "... conversions in C++ are can be explicit ..." is grammatically incorrect. The word "are" should be removed. The same comment also applies to Common Lisp, ISLisp, Modula-2, Pascal and PL/I.

Accepted.

C.6, Fortran binding:

The type "FLT" is referred to, but is not defined.

Accepted. Changed to REAL(kind).

The notation (*) appears in many functions (for example, "rounding"), but the meaning of the notation is not explained.

Those operations are not included in the Fortran standard (yet), but are in a supplementary ISO TR. Wording will be added, and notation ("(*)" instead of "*" for "in the standard") explained.

After the definition of "max_error" functions, the variable "b" is defined but does not appear in the block. The same comment also applies to Haskell, Common Lisp and ISLisp.

Accepted. ‘b’ removed.

Several functions are not given in the binding. For example, we could not find the definitions of "axis_rad" and "axis_cycle" functions.

It is unclear (to the editor) how FORTRAN should bind those operations. Therefore no suggestion is given.

The phrase "The Kind of the a numeral ..." is grammatically incorrect. The word "the" should be removed.

Accepted.

C.7, Haskell binding:

The phrase "Haskell provides ... in base is 10" is grammatically incorrect. The word "is" before "10" should be removed. The same comment also applies to ISLisp.

Accepted.

C.8, Java binding:

The types "INT" and "FLT" are referred to, but are not defined. The same comment also applies to Common Lisp, ISLisp, Modula-2, Pascal and PL/I.

Accepted, bindings texts are modified.

The names of the parameters of the function "max_err_powerF" is "b" and "x". We cannot understand why the name "b" is chosen here.

Accepted, ‘b’ removed.

C.9, Common Lisp binding:

After the definition of the "sqrt_rest" function, a grammatically incorrect phrase "a data objects" appears. The "s" should be removed.

Accepted.

After the definition of the "convert" functions, a misspelled word "Convertion" appears. It should be changed to "Conversion".

Accepted.

In the last paragraph of the binding, a grammatically incorrect phrase "... operations returns a complex ..." appears. The "s" at the end of "returns" should be removed.

Accepted.

C.10, ISLisp binding:

In the first paragraph of page 128, "a ISLisp object" should be changed to "an ISLisp object".

Accepted.

After the definition of the "sqrt_rest" function, "an data objects" should be changed to "a data object".

Accepted.

C.11, Modula-2 binding:

After the definition of the "cycle_to_cycle" function, "... conversions in C" is referred to. "C" should be changed to "Modula-2".

Accepted.

After the definition of "sNaN", "an Modula-2 exception" should be changed to "a Modula-2 exception". The line containing "WHOLE-ZERO-..." is too long. The question mark in "(for pole, not overflow?)" is not adequate.

Accepted.

C.12, Pascal binding:

After the definition of "convert" functions, "FDIS 9899" which is the standard of C appears. The clause numbers "xxxxxx" and "yyyyy" should be changed to concrete numbers.

Accepted.

C.14, SML binding:

In the definition of the "powerFI" and "pow" functions, the parameter names "x" and "y" are referred to. They are different from the parameter given in the specification.

Accepted. Names changed.

After the definition of "convert" functions, the type "INT2" is defined which does not appear in the block.

Accepted. "INT2" sentence removed.

----------end of comments from Japan----------

SWEDEN

Clause 4.1.4, add before "The following symbols are defined in Part~1":
Let I be the non-special value set for an integer datatype conforming to Part~1. Let F be the non-special value set for a floating point datatype conforming to Part~1.

Accepted.

Clause 4.1.4, note 3, replace "justly allow avoiding" with "be able to avoid" (better English).

Accepted.

Clause 4.2, def. of cont. val.:
replace "value in F" with "(in the datatype representable) value in > $\RR$" (slight generalisation, suitable for the definitions section).

Accepted.

Clause 4.2, def. of monotonic approximation:
>Replace "operation $op_F: ... \times F \times ... \rightarrow F$" with "floating point operation $op_F: ... \times F \times ... \rightarrow F$, for a floating point datatype with non-special value set F" (more explicit).

Accepted.

Clause 4.2, def. of monotonic approximation: add "$a < b$,".

Accepted.

On power_FI: Generalise:
$power_{\!FI}^*(1,y) = 1 $ \>\>if $y \in I$\\

to
$power_{\!FI}^*(x,y) = x^y $ \>\>if $y \in I$
and $x \in \ZZ\cap F$ and ($y>0$ or $|x|=1$)\\

Accepted.

On power_FI: Add:
Relationship to other $power_{\!FI'}^*$ helper functions for $power_{\!FI'}$ operations in the same library:
\begin{example}\atab
$power_{\!FI}^*(x,y) = power_{\!FI'}^*(x,y)$ \>\>if $x\in F$ and $y \in I\cap I'$
\end{example}

Accepted.

On power_I: change "since the exact result then is not in $\ZZ$" to "(unless $|x|=1$) since the exact result then is not in $\ZZ$"

Accepted.

On power1pm1: change "greater than -1" to "greater than or equal to -1"

Accepted.

On power1pm1: Add:

$\mathit{power1pm1}_{\!F}^*(x, y) = (1+x)^y-1 $

\>\>if $x,y \in F\cap\ZZ$ and $x\geq-1$ and $y > 0$\\

Accepted.

On power_F: Change

Relationship to the $power_{\!FI}^*$ approximation helper function:

to

Relationship to the $power_{\!FI}^*$ approximation helper function for a $power_{\!FI}$ operation in the same library:

Accepted (also other places). This particular relationship is needed for the C (and C++, and some others) amalgamation of powerF and powerFI into powF.

On arccot_F: change the note

\begin{notes}

$arccot_F(x) \approx arc_F(x,1)$.

\next

There is no ``jump'' at zero for $arccot_F$.

\end{notes}

to

\begin{note}

$arccot_F(x) \approx arc_F(x,1)$ (the arc_F operation is specified below).

\end{note}

Accepted in principle (see USA comments below).

On arcctgu, add in the note:

Due to the range limitation, $arcctgu_F(0)$ need not equal $arccotu_F(0)$.

Accepted.

About big angle parameter for angular unit parameter operations: change

value greater than or equal to 1 and such that $\ulp_F(big\_angle_F) \leq 1/2000$.

to

value greater than or equal to 1 and such that $\ulp_F(big\_angle\_u_F) \leq 1/2000$.

Accepted.

and change

In order to allow $\ulp_F(big\_angle_F) \leq 1/2000$, $p_F \geq 2+\log_{r_F}(1000)$ should hold.

to

In order to allow $\ulp_F(big\_angle\_u_F) \leq 1/2000$, $p_F \geq 2+\log_{r_F}(1000)$ should hold.

Accepted.

On arcctg_F: add notes:

\begin{notes}

$arcctgu_F(neg_F(u,x)) = neg_F(arcctgu_F(u,x))$.

\next

Due to the range limitation, $arcctg_F(u,0)$ need not equal $arccot_F(u,0)$.

\end{notes}

Accepted.

Change the title:
\subsection{Conversion operations}
to
\subsection{Operations for conversion between numeric datatypes}

Accepted.

On notification handling: change
\item When ``recording of indicators'' is the method of notification, the datatype used to represent $Ind$, the method for denoting the values of $Ind$ (the association of these values with the subsets of $E$ must be clear), and the notation for invoking each of the ``indicator'' operations. (See 6.1.2 of Part~1.) In interpreting 6.1.2 of Part~1, the set of indicators E
to
\item When ``recording of indicators'' is the method of notification, the datatype used to represent $Ind$ (see clause 6.1.2 of Part~1), the method for denoting the values of $Ind$, and the notation for invoking each of the ``indicator'' operations. E is the set of notification indicators. The association of values in Ind with subsets of E must be clear. In interpreting clause 6.1.2 of Part~1, the set of indicators E

Accepted.

Partial conformity annex: change
according to the normative text
to
according to the main normative text

Accepted.

On clause B.4.2: Add:
Note also the LIA distinction between denormal and subnormal.
Subnormal include zero values, while denormal does not.

Rejected. Use the term subnormal for IEEE 754 subnormal and LIA-1 denormal. (Also change LIA-1 on this point when revised. This is noted in a new annex on planned changes to LIA-1)

Rationale: change
A report (\cite{soren}) issued by the ANSI X3J11 committee discusses possible
to
The report {\em Floating-Point C Extensions}\cite{fpce}, issued by the ANSI X3J11 committee, discusses possible

Accepted (with the modification that the reference to the ANSI committee is removed).

----------end comments from Sweden----------

UNITED STATES

USA comments:
Key:

USA comment number: --- 001 through 056
Clause/subclause #, WG11 N462 page # affected: --- C.sub &c, 000 through 154
Comment Category: --- M(major technical), m(minor technical),
E(major editorial), e(minor editorial), or Q(question - clarification desired)

Title: --- one-line summary
Rationale: --- original description of the problem
Proposed change: --- original

Comments:

Comment number: --- USA-001
Clause/subclause, page #: --- 1.2(g), 001
Category: --- e
Title: --- excess "of"
Rationale: --- meaningless use of "of" on first line
Proposed change: --- remove "of"

Accepted.

Comment number: --- USA-002
Clause/subclause, page #: --- 1.2(g), 002
Category: --- e
Title: --- "spcification"
Rationale: --- misspelled; reads better if plural
Proposed change: --- replace "spcification" by "specifications"

Accepted.

Comment number: --- USA-003
Clause/subclause, page #: --- 2 paragraph 4 , 003 paragraph 3
Category: --- e
Title: --- singular specification needed
Rationale: --- In 2nd sentence "In the case of ...", a singular "specification" is needed, both to convey the meeting and to satisfy rules of grammar.
Proposed change: --- replace "specifications .. takes" by "specification .. takes"

Accepted.

Comment number: --- USA-004
Clause/subclause, page #: --- 2 paragraph 5 , 003 paragraph 4
Category: --- e
Title: --- comma needed
Rationale: --- a comma between "operations" and "independently" is needed to improve readability.
Proposed change: --- replace "... operations independently ..." by "… operations, independently ..."

Accepted.

Comment number: --- USA-005
Clause/subclause, page #: --- 4.1.1, 004
Category: --- e
Title: --- define before use
Rationale: --- "F" is used in Note 2 in 4.1.3 before it is defined
Proposed change: --- Move the definition of I (in 5.1) and F (in 5.2) here, or provide forward references in subclause 4.1.3.

Accepted in principle.

Comment number: --- USA-006
Clause/subclause, page #: --- 4.1.4 note 3, 006
Category: --- e
Title: --- Unnecessary value judgment
Rationale: --- 'justly allow avoiding' needs to be changed. One suggestion is: 2) in order to avoid an underflow notification ...
Proposed change: --- delete "justly"

Accepted in principle (rephrased to 'in order to be able to avoid').

Comment number: --- USA-007
Clause/subclause, page #: --- 4.1.4 paragraph after note 4, 006
Category: --- e
Title: --- Misspelled "three"
Rationale: --- spelled as "tree"
Proposed change: --- change "tree" to "three"

Accepted.

Comment number: --- USA-008
Clause/subclause, page #: --- 4.2 "monotonic approximation", 007
Category: --- m
Title: --- relationship of a to b
Rationale: --- In the definition of a and b, it is assumed that a<b (because of the English meaning of non-decreasing and non-increasing)
Proposed change: --- after "and b (element of) F" add ", a<b"

Accepted.

Comment number: --- USA-009
Clause/subclause, page #: --- 4.2 "monotonic non-decreasing" etc., 007-008
Category: --- e
Title: --- duplicate term definitions
Rationale: --- unnecessary duplication
Proposed change: --- Remove one of the duplicate definitions of monotonic non-decreasing and monotonic non-increasing

Rejected. They are not duplicates, but two needed complements to the definition of monotonic.

Comment number: --- USA-010
Clause/subclause, page #: --- 4.2 "subnormal", 009
Category: --- e
Title: --- inter-standard inconsistency
Rationale: --- LIA-2 vs (IEEE-854 and C9x) use inconsistent definitions for the term "subnormal number". IEEE 854 and C9x do not include 0 and -0 as possible values of "subnormal" quantities.
Proposed change: --- Choose a different term to represent the concept of the union of denormalized numbers with 0 and -0 (it may be possible to finesse this issue by rewording the (few) places where this term is used in the body of LIA-2).

Accepted.

Comment number: --- USA-011
Clause/subclause, page #: --- 5.2.1 paragraph 1, 014
Category: --- e
Title: --- provide friendly reference
Rationale: --- readers might not remember where to look for definition of F*
Proposed change: --- Consider adding after 'F*' as a note: "F* is defined in LIA-1. It is the 'unbounded extension of F'".

Accepted.

Comment number: --- USA-012
Clause/subclause, page #: --- 5.2.1 result_NaN3F helper function, 015
Category: --- e
Title: --- poor grammar
Rationale: --- "not .. neither .." is not idiomatic English
Proposed change: --- Change "not" to "neither" (three places)

Accepted.

Comment number: --- USA-013
Clause/subclause, page #: --- 5.3.2 paragraph 1, 021
Category: --- e
Title: --- "are shall"
Rationale: --- poor grammar
Proposed change: --- change "are shall" to "shall" in two places

Accepted.

Comment number: --- USA-014
Clause/subclause, page #: --- 5.3.6.6-7, 026-027
Category: --- e
Title: --- function ordering
Rationale: --- grouping "power*" functions makes them easier to find Proposed change: --- move them after 5.3.6.1, also make corresponding change in Annex B (purely personal preference, based on naming conventions of helper functions. This change separates the "Exponentiation of" titles, so there is no perfect ordering).

Rejected. The different power operations have different properties, e.g., powerFI(0,0) = 1, while powerF(0,0)=invalid(NaN). So the three power opeartions (I, FI, F) are best kept separate, despite the naming.

Comment number: --- USA-015
Clause/subclause, page #: --- 5.3.6.7 Further requirements..., 027
Category: --- m
Title: --- incorrect requirement
Rationale: --- The requirement that power1pm1*F(x,1) is equal to "1 + x" is incorrect
Proposed change: --- It should be equal to "x"

Accepted.

Comment number: --- USA-016
Clause/subclause, page #: --- 5.3.9.10 paragraph 2, 041
Category: --- m
Title: --- additional "Further" requirement
Rationale: --- improves interfunctional consistency
Proposed change: --- Add arccos*F(-x)=pi-arccos*F(x)

Rejected. May prove too hard for implementations to fulfill (assuming that pi refers to the exact value), and the added value of such a requirement is small.

Comment number: --- USA-017
Clause/subclause, page #: --- 5.3.9.9-5.3.9.15 Range limitation sections, 041-044
Category: --- e
Title: --- noun needed
Rationale: --- title insufficiently descriptive
Proposed change: --- "Range limitation" is an adjectival phrase - it needs a noun to state the category that the "arc...#()" functions belong to. Change 'Range limitation' to 'Range limitation helper function' everywhere.

Accepted.

Comment number: --- USA-018
Clause/subclause, page #: --- 5.3.9.13 paragraph 2, 043
Category: --- m
Title: --- additional "Further" requirement
Rationale: --- improves interfunctional consistency
Proposed change: --- Add arcsec*F(-x)=pi-arcsec*F(x)

Rejected. May prove too hard for implementations to fulfill,
and the added value is small.

Comment number: --- USA-019
Clause/subclause, page #: --- 5.3.9.15, 044-045
Category: --- e
Title: --- define before use
Rationale: --- 5.3.9.11 Note 1 refers to arcF, not yet defined
Proposed change: --- Move 5.3.9.15 to before 5.3.9.9 (alternate: give a forward reference in the Note)

Accepted by giving a forward reference.

Comment number: --- USA-020
Clause/subclause, page #: --- 5.3.10 last paragraph, 045
Category: --- m
Title: --- error ranges too large
Rationale: --- Error ranges specified for max_error_sinuF(u) and max_error_tanuF(u) have upper bounds of 2 and 4 respectively. These values are so large as to be useless for practical implementations (this is most likely a typographical error, albeit one with technical consequences).
Proposed change: --- replace "2]" with "2*rnd_errorF]" and "4]" with 4*rnd_errorF]" respectively.

Accepted in principle, with 2*max_error_sin and 2*max_error_tan. In addition the intervals are changed (in general) to upper bounds only, with a general statement about the smallest value.

Comment number: --- USA-021
Clause/subclause, page #: --- 5.4.4 &5.4.5, 059 & 060
Category: --- Q
Title: --- Choice of error bounds
Rationale: --- Need clear rationale for maximum error bounds for round-to-nearest conversions.
Proposed change: --- The rationale for the use of 0.97 ulp as the upper bound for these errors in IEEE 754 is based on a technical paper proving that such accuracy is sufficient for converting back and forth from a given floating type without incurring an error in the binary representation of the original number. Accordingly, we ask either that the upper bound for such errors be changed to 0.97ulp, or that an explanation be provided in B.5.4 for the difference.

Accepted in principle. Main text to have 0.5 ulp max_error for conversions, since that is what C requires. 0.97 ulp limit required for partial conformity (annex A).

Comment number: --- USA-022
Clause/subclause, page #: --- 5.5 Note, 062
Category: --- e
Title: --- give specific reference
Rationale: --- above where?
Proposed change: --- after "above", specify "(5.4)"

Accepted.

Comment number: --- USA-023
Clause/subclause, page #: --- 5.5.2 paragraph 7, 062-063
Category: --- m
Title: --- unnecessary requirement
Rationale: --- The requirements labeled a) and b) seem to require such numerals, even in implementations that do not support the IEEE 754 standard.
Proposed change: --- Clarify that these numerals need only be provided for implementations and bindings that support IEEE 754.

Accepted. New text:

```
There should be a numeral for positive infinity.
There shall be a numeral for positive infinity
if there is a positive infinity in the floating point datatype.
```

Similar wording is used for integer datatypes too.

Comment number: --- USA-024
Clause/subclause, page #: --- B.3, 070
Category: --- e
Title: --- remove unfortunate implication
Rationale: --- current wording implies IEEE754 is obsolete
Proposed change: --- reword as: 'The referenced IEC 60559 standard is identical to IEEE 754 and the former IEC 559 standards.'

Accepted.

Comment number: --- USA-025
Clause/subclause, page #: --- B.5.1.2 paragraph 1 & B.5.2.2 paragraph 1, 072 & 076
Category: --- e
Title: --- Improve explanation
Rationale: --- Improves rationale for Pole exception for empty list
Proposed change: --- Add words along the lines of: "Taking the minimum/maximum of an empty sequence is an undefined operation. But since it has a meaningful result (+INF/-INF), it is a pole exception, not an invalid exception."

Rejected. Max and min of an empty sequence are not undefined. Ideally, the result is an infinity (negative for max, positive for min), since that result is the identity element for max and min respectively. The special handling in LIA for these operations are made because infinities are not always present in arithmetic datatypes. For clarity the "pole" exception is renamed to "infinitary".

Comment number: --- USA-026
Clause/subclause, page #: --- B.5.1.7 paragraph 1, 073
Category: --- e
Title: --- extraneous digits
Rationale: --- meaningless digits after "integers"
Proposed change: --- remove the "47"

Accepted.

Comment number: --- USA-027
Clause/subclause, page #: --- B.5.1.7 paragraph 1, 073
Category: --- e
Title: --- Improve rationale
Rationale: --- Improved coverage of cases
Proposed change: --- add something like: "Overflow only happens on MIN/ -1"

Accepted in principle. The note

```
\begin{note}
$\mathit{divf\!}_I(minint_I, -1)$, for a signed integer datatype,
is the only case where this operation may overflow.
\end{note}
```

is added to the main text.

Comment number: --- USA-028
Clause/subclause, page #: --- B.5.1.8 paragraph 1, 074
Category: --- e
Title: --- extraneous word
Rationale: --- "a" is repeated
Proposed change: --- remove one "a"

Accepted.

Comment number: --- USA-029
Clause/subclause, page #: --- B.5.2.3, 076
Category: --- e
Title: --- Motivate controversial choice

Rationale: --- A controversial choice deserves an explanation. J11 prefers that dim(+INF,+INF) and dim(-INF,-INF) return 0 rather than an invalid(qNan).
Proposed change: --- Explain that LIA-2 interprets multiple different instances of +/-INF as potentially representing different quantities, rather than the same quantity. This allows the programmer to more easily detect sources of errors.

Accepted in principle. The sentence "Note that $dim_F(\posinf, \posinf) = invalid(\qNaN)$ is consistent with that $sub_F(\posinf, \posinf) = \invalid(\qNaN)$ according to IEC~60559." is added.

Comment number: --- USA-030
Clause/subclause, page #: --- B.5.2.5 paragraph 2, 076
Category: --- e
Title: --- misunderstanding of an assertion
Rationale: --- The original commenter suggests "cannot be exact" be changed to "need not be exact". This reflects a misunderstanding - the sentence is making an assertion about reality, not a requirement of an implementation (reinterpreted from original suggestion).
Proposed change: --- Reword the sentence so its assertive nature is clear, and indicates that its explanation is forthcoming.

Accepted. Clarified to "Remainder after floating point division and floor to integer cannot be exact for all pairs of arguments from F." (note the addition at end of sentence).

Comment number: --- USA-031
Clause/subclause, page #: --- B.5.2.7, 077
Category: --- e
Title: --- Add IEEE plug
Rationale: --- Clarify a difference between IEEE conforming and non IEEE conforming implementations
Proposed change: --- Include a sentence explaining that add_loF() and sub_loF() underflow only when denormals are not supported. Also, consider adding a statement to the effect that, if the high part underflows, then the low part is zero.

Accepted. The sentences "add_lo_F and sub_lo_F can underflow only when denormals are not supported. In addition, if the high part underflows, then the low part is zero." are added.

Comment number: --- USA-032
Clause/subclause, page #: --- B.5.3.5, 079
Category: --- e
Title: --- cover all cases
Rationale: --- explanation is missing a useful fact
Proposed change: --- Add words like: "hypot() only underflows if both arguments are denormal numbers.

Accepted.

Comment number: --- USA-033
Clause/subclause, page #: --- B.5.3.6, 079
Category: --- e
Title: --- Which specification is being referred to?
Rationale: --- Pronoun use is ambiguous in its context
Proposed change: --- In the paragraph that mentions X/OPEN, change 'This specification' to 'That specification'. Throughout the document, the term "This specification" refers to itself, LIA-2.

However, here, it is being used to refer to the X/OPEN Portability Guide, thus it is confusing. (alternate: "Those specifications")

Accepted.

Comment number: --- USA-034
Clause/subclause, page #: --- B.6 paragraph 1, 086
Category: --- e
Title: --- imprecise use of English
Rationale: --- The use of "must not" here is understandable, but lacks the explanatory power for which Annex B exists.
Proposed change: --- Reword sentence to explain in a neutral tone that intermediate overflow is harmful to user programs and unnecessary in implementations (wording changed from original suggestion).

Accepted. New text:

An intermediate overflow on computing approximations to x^2 or y^2 during the calculation of $hypot_F(x,y) \approx \sqrt{x^2 + y^2}$) does not result in an overflow notification, unless the end result overflows. This is clear from the specification of the $hypot_F$ operation in this Part.
It is not helpful for the user of an operation to let intermediary overflows or underflows that are not reflected in the end result be propagated. Implementations of LIA-2 operations are required to shield the user from such intermediary overflows.

Comment number: --- USA-035
Clause/subclause, page #: --- C, 094 &c
Category: --- e
Title: --- inter-standard consistency
Rationale: --- Desire for LIA-2 to use consistent terminology with the IEEE 854 standard
Proposed change: --- Change 'SIGNAN' to 'NANS'. IEEE-854 suggests that all representations of NAN (both quiet and signaling) start with the substring 'NAN', for ease of parsing the string representation of this symbol.

Accepted in principle. Exception is made for programming languages where they already have other names, or whose naming conventions differ from the suggestion.

Comment number: --- USA-036
Clause/subclause, page #: --- C.3 next to last paragraph, 098
Category: --- Q
Title: --- incompleteness
Rationale: --- Did someone intend complete a description of exception numbers for BASIC?
Proposed change: --- Eliminate the "(?)" at the end of the 2nd to last sentence.

Accepted. The description is made more complete.

Comment number: --- USA-037
Clause/subclause, page #: --- C.4 paragraph 1, 098
Category: --- e
Title: --- update

Rationale: --- reference to C9x is not current. The FDIS was approved on September 15, 1999, so it is now an IS. It is not clear when it will be published.
Proposed change: --- When transmitting the Final CD to the editors, please alert them to the need to amend this reference. Alternatively, it is up to the editors at ISO/IEC to insure that inter-standard references are up-to-date, as part of the publication process.

Accepted.

Comment number: --- USA-038
Clause/subclause, page #: --- C.4 max/min definitions, 099
Category: --- e
Title: --- improve suggested binding
Rationale: --- C has a ternary operator to allow selection
Proposed change: --- Change 'imax(x,y)' to '(x<y ? y : x)'; also, change 'imin(x,y)' to '(x<y ? x : y)'

Accepted as alternate bindings.

Comment number: --- USA-039
Clause/subclause, page #: --- C.4, 100
Category: --- e
Title: --- improve readability
Rationale: --- separate unrelated groups of items in a list
Proposed change: --- Add a blank line between ceiling_rest and remr

Accepted.

Comment number: --- USA-040
Clause/subclause, page #: --- C.4 hypot section, 101
Category: --- e
Title: --- imprecise reference to C
Rationale: --- The original C standard (1989) does not have a hypot()
Proposed change: --- Change "C has a hypot..." to "C9x has a hypot..."

Accepted in principle (but in the binding line).

Comment number: --- USA-041
Clause/subclause, page #: --- C.4 elementary floating operations list, 101
Category: --- e
Title: --- imprecise references to C
Rationale: --- The original C standard does not have these functions
Proposed change: --- After the *, add "(C9x)" to the line for each of the functions: exp2, log2, asinh, acosh, atanh, nearbyint

Accepted (with "C99").

Comment number: --- USA-042
Clause/subclause, page #: --- C.4 "pow" discussion, 101
Category: --- e
Title: --- improve completeness
Rationale: --- Note that pow() function in C is different in an additional way from LIA-2
Proposed change: --- Need to consider pow(+/-1,+/-INF) in not qNaN in C9x

Accepted.

Comment number: --- USA-043
Clause/subclause, page #: --- C.4 convert section, 102-103
Category: --- e
Title: --- choose symbol implying the proper type
Rationale: --- current representation implies misleading data type
Proposed change: --- Change '&r' to '&i'. Change 'x' to 'i'. The choice of "r" and "x" symbols may imply a floating type to some people.

Accepted in principle (variables explained in text).

Comment number: --- USA-044
Clause/subclause, page #: --- C.4 following lists & C.5 following lists, 103 & 108
Category: --- e
Title: --- unnecessary term definitions
Rationale: --- "?" is not used in the tables for C or C++
Proposed change: --- remove 'A ? above indicates that the parameter is optional.' (for both C and C++ sections)

Accepted.

Comment number: --- USA-045
Clause/subclause, page #: --- C.4 3rd from last paragraph & C.5 3rd from last paragraph, 103 & 108
Category: --- e
Title: --- improve completeness
Rationale: --- In C and C++, the string form of floating point values may be indicated by the presence of an exponent, as well as by the presence of a ".".

Proposed change: --- Change "must have a '.' in them" to "must have a '.' or an exponent in them".

Accepted.

Comment number: --- USA-046
Clause/subclause, page #: --- C.5 paragraph 4, 104
Category: --- e
Title: --- unfortunate hyphenation
Rationale: --- technical term can be read incorrectly
Proposed change: --- Change the hyphenation of "names-pace" to "name-space"

Accepted.

Comment number: --- USA-047
Clause/subclause, page #: --- C.5 lists, 104 & 105
Category: --- e
Title: --- organization of tables
Rationale: --- easier for readers to find
Proposed change: --- move sqrt to after shift10, move mul to after sqrt_rest

Accepted.

Comment number: --- USA-048
Clause/subclause, page #: --- C.5 lists, 105
Category: --- e
Title: --- misspelled function name
Rationale: --- easier for readers to find
Proposed change: --- Rename reciprocal_sqrt to rsqrt

Accepted.

Comment number: --- USA-049
Clause/subclause, page #: --- C.5 lists, 106 & 107
Category: --- e
Title: --- excessive credit
Rationale: --- misleading status of C++ standard
Proposed change: --- Change the * to a dagger for the functions: asinh, acosh, atanh, nearbyint

Accepted.

Comment number: --- USA-050
Clause/subclause, page #: --- C.5 convert-related entries, 107 & 108
Category: --- e
Title: --- choice of conversion functions
Rationale: --- The idiomatic C++ syntax for triggering conversions to and from digit strings is to use the overloaded << and >> operators. (use of C operators is acceptable and does have the advantage of more compactly specifying non-default format codes.)
Proposed change: --- use << and >> operators as the examples for convert bindings for C++

Rejected. The examples are very long already. The << and >> are only overloaded convenience names for default format. A full binding for C++ would include these too, but for the example in LIA-2 adding them does not seem necessary.

Comment number: --- USA-051
Clause/subclause, page #: --- C.7, 117
Category: --- e
Title: --- spurious characters
Rationale: --- meaningless characters need deleting
Proposed change: --- In the last section of tables for Haskell, remove the (apparently) spurious sequences of "." characters preceding the "*"s.

Accepted.

Comment number: --- USA-052
Clause/subclause, page #: --- C.8, 120
Category: --- Q
Title: --- overloading?
Rationale: --- both lnF(x) and logbaseF(b,x) are mapped to the Java function "log()". Does Java support function name overloading in this way?
Proposed change: --- Verify that Java supports overloading in this manner, to allow the use of "log()" for the logbaseF(b,x) case, or change the spelling for the suggested representation for logbaseF(b,x).

Java allows this kind of overloading. No change to LIA-2.

Comment number: --- USA-053
Clause/subclause, page #: --- C.10 arcsinuF binding, 131
Category: --- e
Title: --- missing ")"
Rationale: --- simple omission
Proposed change: --- Supply ")" (4 places)

Accepted.

Comment number: --- USA-054
Originator comment #: J11FT-54
Clause/subclause, page #: --- C.11 & C.12 discussions prior to the convert tables, 135 & 139
Category: --- e
Title: --- cut/paste errors
Rationale: --- fix spurious references to C in Modula & Pascal subclauses
Proposed change: --- Change the references to C in the Modula and Pascal sections respectively and fix to reflect the appropriate languages

Accepted.

Comment number: --- USA-055
Clause/subclause, page #: --- C.12 4th paragraph from last, 140
Category: --- e
Title: --- missing clause specifications
Rationale: --- placeholders need filling-in
Proposed change: --- Replace references to clauses xxxxxx and yyyyy with the correct clause numbers

Accepted.

Comment number: --- USA-056
Clause/subclause, page #: --- D [20], 152
Category: --- e
Title: --- spurious date
Rationale: --- "Currently under revision (1998)" I don't know what that "1998" is, but I can't believe that it is anything that is current or correct. (There is no real clue at the beginning of Annex D to explain what it is supposed to mean.)
Proposed change: --- remove the "(1998)"

Accepted.

----------end of comments from USA----------

----------end of comments on LIA-2 FCD2----------