

IT/99-0125

\*\*\*\*\*

National Committee for Information Technology Standards

NCITS Secretariat, Information Technology Industry Council (ITI)  
1250 Eye St. NW, Suite 200, Washington, DC 20005  
Telephone 202-737-8888; Fax 202-638-4922;  
Email: NCITS@itic.org

Date: March 18, 1999  
Ref. Doc: ISO/IEC FCD 9899  
Project: 381-RI  
Reply to: Deborah J. Donovan  
Phone: 202-626-5746  
email: ddonovan@itic.org

\*\*\*\*\*

To: NCITS Members - For Information  
From: Deborah J. Donovan, Coordinator, Standards Processing, NCITS  
Subject: J11 Responses to Comments on the Public Review of ISO/IEC  
FCD 9899, Information Technology - Programming Languages C  
(Revision of ISO/IEC 9899:1990)

During the public review of ISO/IEC FCD 9899, seventeen comments were received and registered, IT/98-0398. Attached are the J11 responses to the comments.

---

NCITS/J11 Responses to Public Review Comments Received for ISO/IEC FCD 9899, Information Technology - Programming Languages - Programming Language C (Revision of ISO/IEC 9899:1990)

Committee NCITS/J11 Document # J11/99-011

This document contains the responses to all public comments submitted for the public review (from 09/11/98 to 11/10/98) of ISO/IEC FCD 9899, Information Technology - Programming Languages - Programming Language C (Revision of ISO/IEC 9899:1990).

From the receipt of these responses, the submitters of public comments have 20 working days in which to reply and indicate if the comment(s) has not been satisfactorily resolved. Replies must be sent (preferably by email) to:

Ms. Deborah Donovan  
ITI  
1250 Eye Street, suite 200  
Washington D.C.  
ddonovan@itic.org

CCing me, the NCITS/J11 Committee chair

Rex Jaeschke  
2051 Swans Neck Way  
Reston, VA 20191-4023  
rex@aussie.com

At the February, 1999, co-located meeting of SC22/WG14 and NCITS/J11 all comments from all national bodies were resolved and all countries that voted No on the FCD had their No vote turned into a Yes. As such, the Final Draft International Standard (FDIS) is expected to be submitted for a 2-month ballot early in April.

The responses to your comments are shown below under the entry US-x.

Rex Jaeschke  
NCITS/J11 Committee chair

---

US-1 (Douglas Walls)

Comment #1:

The atexit function is free to return non-zero to indicate failure if called after exit, but atexit is not permitted to crash the program. Any functions successfully registered by atexit after exit is called are called in reverse order of registration after the function that registered them returns.

---

US-2 (John Hauser)

Comment #1:

The special cases for the atan2 function match those desired for carg to support the general approach to complex special cases, which entails avoiding NaNs, even in indeterminate cases where a non NaN value is deemed more useful. See the response to US-16, Comment 9.

Comment #2:

The second bullet of F.9.5.4 will be replaced by

- tgamma(+/-0) returns +/-inf and raises the divide-by-zero exception.
- tgamma(x) returns a NaN and raises the invalid exception if x is a negative integer.

The cases

$\text{pow}(x, +\text{infinity})$   $x < -1$   
 $\text{pow}(x, -\text{infinity})$   $-1 < x < 0$

are defined to be +infinity, instead of NaN or -infinity, because large -magnitude finite floating-point numbers are all even integers, hence  $\text{pow}(x, n)$  is +infinity if  $x < -1$  and  $n$  is any floating-point value of sufficiently large magnitude, and similarly for the second case.

Regarding  $\text{pow}(-0, y)$   $y < 0$  and  $y$  not an integer for points near and to the left of  $(-0, y)$ , pow returns NaN, and for points near and to the right, pow returns large positive values, with  $\text{pow}(+0, y)$  returning +infinity.

Particularly as the sign of zero can be a computational accident, +infinity was thought to be the most useful result.

The case  $\text{pow}(-\text{infinity}, y)$   $y > 0$  and  $y$  not an integer is defined to be consistent with the previous case.

The case of  $\text{tgamma}$  of  $x$ ,  $x$  a negative integer, deserves a NaN result, as the mathematical function approaches +infinity on one side of the argument and -infinity on the other (and hunkers on the  $x$  axis between poles).

Comment #3:

The current  $\text{fdim}$  spec is believed to be a better match for existing practice, in both C and Fortran. Support for Fortran codes was a primary reason for including  $\text{fdim}$  in the International Standard.

Comment #4:

The Committee believes a significant body of applications will be better served by the current specification, though acknowledging that the suggested one would be better for others. The IEEE treatment of NaNs is not compelling, as these functions, like the bit manipulation functions mentioned in the comment, need not be regarded as arithmetic.

-----  
US-3 (David Tribble)

Comment #1:

This has been discussed many times within the Committee, and the current status represents the solution that prompted the greatest consensus. There are many varying opinions on this issue. In fact, the opposite change was proposed in another public comment. Considering this, and in order to maintain consensus within the Committee, no change will be made.

-----  
US-4 (Eric Rudd)

Comment #1:

In the draft standard, the first sentence of 7.12.4.4 [ #2] will be changed to:

The atan2 functions compute the value of the arc tangent of  $y/x$ , in the range  $[-\pi, +\pi]$  radians, using the signs of both arguments to determine the quadrant of the return value.

Comment #2:

A domain error is not required for these cases. As 7.12.4.4 allows but does not require a domain error, whether one occurs may vary among implementations. For IEC 60559 implementations, atan2(0,0) and atan2(inf,inf) each has a defined numerical result (F.9.1.4), hence no domain error. Other implementations are allowed to adopt a definition for atan2 that excludes the inputs (0,0) and (inf,inf) and regard these cases as domain errors.

Comment #3:

A statement in the draft standard that the range is a closed interval does not imply that each value in the interval (including endpoints) is achieved by the function.

Comment #4:

As indicated in the response above to Comment 2, there is some latitude for definitions of special cases. For certain special cases, Annex F specifies a numerical result and no floating-point exception corresponding to a domain or range error. Hence for implementations supporting the annex, these cases need not be regarded as domain or range errors. Also see the response to US-16, Comment 9.

Comment #5:

As indicated in the response above to Comment 2, there is some latitude for definitions of special cases. Whether a domain error occurs for pow(0,0) depends on the implementation.

Comment #6:

The following statement will be appended to the F.3 bullet that begins "The fegetround and fesetround functions ‡": "The values 0, 1, 2, and 3 of FLT\_ROUNDS are the IEC 60559 directed rounding modes."

As in other cases, Annex F adopts IEC 60559 specification by reference, in order to keep down the size of the draft and to reduce the chance of documentation errors.

The current specification in 5.2.4.2.2, like in 9899:1990, does not define the named rounding modes, and in particular leaves unspecified the treatment of halfway cases in rounding to nearest. The Committee does not see sufficient benefit from introducing a requirement that non-IEEE implementations document details of their rounding modes.

Comment #7:

The term exception was used in two ways. The draft will be changed to distinguish (a) what will now be called /exceptional conditions/ as in 6.5 [ #5] from (b) floating-point exceptions as in 7.6.

Comment #8:

The draft standard will be changed to require, like 9899:1990, that calls to math functions shall not terminate a program.

Comment #9:

The draft standard will be changed to require that each implementation support either errno, as required in 9899:1990, or the floating-point exception flags corresponding to domain and range errors for math functions.

-----  
US-5 (Larry Jones)

Comment #1:

Accepted, with some editorial rewording of point 3.

Comment #2:

Accepted in principle; new words:

Add a subclause to clause 3:

Unspecified value

A value of the relevant type where this International Standard imposes no requirements on which value is chosen in any instance.

Note: an unspecified value cannot be a trap representation.

Add a subclause to clause 3 defining "implementation-defined value" using wording based on that defining "implementation-defined behaviour".

Comment #3:

Duplicate of UK #0246, which was responded to as follows:

Accepted in principle; new wording:

Change 6.7.2.2p4 first sentence to read:

Each enumerated type shall be compatible with /char/ or a signed or unsigned integer type.

Comment #4:

Accepted as editorial. The draft has been corrected.

Comment #5:

Accepted as editorial. The draft has been corrected.

-----

US-6 (David Thornley)

Comment #1:

No change will be make, this has been discussed several times within the Committee.

Comment #2:

This has been discussed many times within the Committee, and the current status represents the solution that prompted the greatest consensus. There are many varying opinions on this issue. In fact, the opposite change was proposed in another public comment. Considering this and in order to maintain consensus within the Committee, no change will be made.

Comment #3:

The problem described cannot arise, because a conforming implementation cannot claim to provide a signed integer type when it does not provide the

corresponding unsigned integer type (See subclause 6.2.5 Types, and 7.18.1 Integer types).

Comment 4:

Removing long long would reduce consensus within the Committee. The inclusion of long long has been discussed and reaffirmed on several occasions, so no change will be made.

Comment #5:

The Committee is not considering requests for new features at this point.

-----

US-7 (Robert Corbett)

Comment #1:

This was accepted as an editorial change.

Comment #2:

In F.9.4.4, the bullet

-- `pow(+/-1,+/-inf)` returns ...

will be changed to

-- `pow(+/-1,+/-inf)` returns +1

The case `(-1,+/-inf)` is included in the change because `pow(-1,y)` is +1 for all sufficiently large magnitude floating-point numbers `y`.

Comment #3:

`cpow` was defined by the usual formula for ease of implementation, and because the special cases were thought to be somewhat less important for `pow`. In order not to preclude a more careful treatment of special cases, the draft standard will be changed to allow but not require `cpow` to be implemented with the formula

$$\text{cpow}(z,c) = \text{cexp}(c*\text{clog}(z))$$

A statement will be added that `cpow` raises exceptions if appropriate for

the calculation of the parts of the result, and may raise spurious exceptions (as may complex multiply and divide).

Comment #4:

The present semantics of /sizeof/ are those intended by the Committee. The proposer may wish to read the section of the Rationale document dealing with VLAs.

Comment #5:

A careful reading of the Standard will show that linkage can only apply to a specific identifier, and not to two identifiers with different spellings. The editor has been asked to consider adding a footnote attached to 6.2.2p1:

Linkage applies to a single identifier. Two different identifiers (such as /x/ and /y/) can never refer to the same object through linkage.

Comment #6:

The draft standard will be changed to append to the Semantics section of 6.4.4.2 the following paragraph:

"Floating-point constants are converted to internal format (as if) at translation time. The conversion of floating-point constants shall not raise an exceptional condition nor a floating-point exception at execution time."

Strengthening requirements for consistent evaluation of decimal constants was feared too great a burden for some existing implementations (which may, for example, deliver different values depending on trailing zeros). Note that the draft standard requires correctly rounded, therefore consistent, conversion of decimal constants (except for extreme cases) for IEC 60559 implementation. Strengthening consistency requirements for hexadecimal constants did not seem worthwhile, as exact conversion is already required where possible.

-----  
US-8 (John Hauser)

Comment #1:

The following statement will be appended to the Description of frexp: "The

results are unspecified if the value of /value/ is not a floating-point number", with a forward reference to 5.2.4.2.2.

Comment #2:

The suggested fix will be accepted.

Comment #3:

The second and third sentences in 7.12.7.4 [ #2] will be changed to say "A domain error occurs if x is finite and negative and y is finite and not an integer value.

The fact that  $\text{pow}(+0,+0)$  is defined in F.9.4.4 may suggest but does not imply that implementations not supporting Annex F must adopt this definition. Non IEC 60559 implementations have latitude in how they define special cases.

Comment #4:

The suggested fix will be accepted.

-----

US-9 (Thomas MacDonald)

Comment #1:

The Committee has voted for this idea.

Comment #2:

The Committee has voted for this idea.

Comment #3:

The following wording change was agreed:

In 6.7.5.2p3, delete the words:

It is unspecified whether side effects are produced when the size expression is evaluated.

Add a new paragraph between p3 and p4:

Where a size expression is part of an operand to sizeof, and changing the value of the size expression would not affect the value of the sizeof expression, it is unspecified whether or not the size expression is evaluated.

Comment #4:

Addressed by N866, which was adopted.

Comment #5:

Accepted.

Comment #6:

Addressed by N867, which was adopted with the following changes:

On line 90, add the third syntax option:

direct-declarator [static type-qualifier-list/opt assignment-expression]

On line 98, change:

The optional type qualifiers preceding ...

to:

The optional type qualifiers and the keyword static preceding ...

On line 128, change:

If a size expression follows such a type qualifier ...

to:

If the keyword static appears ...

-----

US-10 (Peter Seebach)

Comment #1:

The Committee has reaffirmed its previous decision not to include this

function.

Comment #2:

The has been addressed by an editorial change, "indeterminately..." removed from definition.

Comment #3:

Yes, changes along the lines you suggested will be made.

-----

US-11 (Andrew Josey)

Comment #1:

Accepted in principle; new words:

Append to 7.19.8.2p3:

If /size/ or /nmemb/ is zero, /fwrite/ returns zero and the state of the stream remains unchanged.

-----

US-12 (Larry Jones)

Comment #1:

Making the suggested change would potentially destabilize the document at this time.

Comment #2:

Accepted as editorial. Jones will provide words.

-----

US-13 (Randy Meyers)

Comment #1:

Accept Future Directions and footnote. Normative text is not needed.

Comment 2:

Accepted.

---

US-14 (Antoine Leca)

Comment #1:

Duplicated by a French comment, which was responded to as follows:

Addressed by removal of the struct tmx-based functions.

Comment #2:

Duplicated by a French comment, which was responded to as follows:

Agreed, the struct tmx material was removed.

---

US-15 (Douglas Gwyn)

Comment #1:

A careful reading of the Standard will show that a strictly conforming program cannot exceed any minimum translation limit.

Comment #2:

A technical report that will address conformance issues will be debated at some future date.

Comment #3:

The present rules for /main/ were preferable.

Comment #4

The Committee has voted for this idea.

Comment #5:

Withdrawn

Comment #6:

The Committee has voted for this idea.

Comment #7:

This is viewed as an editorial change which has been accepted.

Comment #8:

The Committee has voted for this idea.

Comment #9:

Accepted.

Comment #10:

Accepted.

Comment #11:

The Committee considered that this would be too difficult to provide on many implementations. There were also technical flaws in the proposal.

Comment #12:

Duplicate of Norway #12, which was responded to as follows:

The Committee believes that the suggested change does not improve the document. The primary virtue of the example is its historical significance as a common, if poor, implementation.

Comment #13:

Duplicate of US-1 Walls #1.

Comment #14:

The Committee has voted for this idea.

-----

US-16 (John Hauser)

Comment #1:

Accepted. See the response to US-2 Comment #2.

Comment #2:

Moving the formulas into normative text might lead users to believe the functions are required to be implemented by the formulas. The association of "the usual mathematical formulas" in normative text to the formulas in the footnote is clear enough.

Comment #3:

Accepted as editorial.

Comment #4:

Accepted as editorial.

Comment #5:

A bullet will be added to G.5.2.3 to cover the missing case:

--  $\operatorname{catanh}(1+i0)$  returns  $+\infty+i0$  and raises the divide-by-zero exception

Although the imaginary part is indeterminate, 0 was chosen so the function will equal the real  $\operatorname{atanh}$  function on the x axis, in keeping with the general approach for complex special cases, e.g. in the specification for  $\operatorname{clog}(0+i0)$ .

Comment #6:

The draft will be changed to incorporate the suggested fix.

Comment #7:

In G.5.2.6, the bullet

--  $\operatorname{ctanh}(+\infty+iy)$  returns  $1+i0$ , for all positive-signed numbers y

will be replaced by the first two suggested bullets in the comment.

The remaining suggestions in this comment were not accepted because the

refinements

$\text{ctanh}(+0+i*\text{inf}) = +0+i*\text{NaN}$

$\text{ctanh}(+0+i*\text{NaN}) = +0+i*\text{NaN}$

would be problematic due to the possible 0/0 indeterminacy in the real part of  $\text{ctanh}(x+i*y)$ :

$\sinh(2x) / (\cosh(2x) + \cos(2y))$

when  $\cos(2y) = -1$ .

Comment #8:

Editorial. Typo. Already fixed.

Comment #9:

The comment reflects a good understanding of the thinking underlying the specification of complex special cases. We should add that even if the partial information represented by a special-case non-NaN value is lost in subsequent calculations, the value still may be more useful than a NaN. In a significant number of calculations, the actual numerical value of one part or the other of an intermediate complex value doesn't matter, but a NaN value in that part would cause an undesirable NaN in the final result. The computation of a cube root as  $w = \text{cexp}(\text{clog}(z)/3)$  illustrates the point, as  $z = 0+0i$  would yield  $w = \text{NaN}+i\text{NaN}$  if  $\text{carg}(z)$ , and hence the imaginary part of  $\text{clog}(z)$ , were required to be a NaN. In such cases the programmer is spared the burden for coding to defend against certain inputs, and the program is more efficient. The programmer guarding against bad inputs will be no worse off with the current specification.

Admittedly the current specification entails some risk for non defensive programmers, who might gain some benefit from a NaN result giving an after-the-fact indication of a problem. This benefit is thought less than might first appear because the programs that could benefit are likely to first encounter erroneous calculations that don't produce NaNs, e.g.  $\text{carg}(z)$  may be questionable if  $z$  is very small but not 0.

Even though the suggested specification would be more helpful in some instances, the approach of allowing two treatments of the contentious cases was rejected for portability reasons.

-----

US-17 (Paul Eggert)

Comment #1:

The Committee considered that there were good reasons to leave this choice for implementations.

Comment #2:

The Committee has voted for this idea.

---