ISO/IEC 1539-1:2018 Defect Report

January 12, 2021

This document lists all the reported defects in ISO/IEC 1539-1:2018 (Fortran 2018) that are to be addressed by a planned Corrigendum 1. JTC1/SC22/WG5 voted on these and comments received follow each item.

Defect F18/015 failed to win approval and is not reflected in this corrigendum

In this document, "J3" refers to INCITS PL22.3, the US TC, which is responsible for developing the technical content of the Fortran standard.

See also the following WG5 documents:

- N2176 - WG5 letter ballot 1 on Fortran 2018 interpretations
- N2178 - Results of WG5 letter ballot 1 on Fortran 2018 interpretations

NUMBER: F18/001
TITLE: ACOSH principal value specification is wrong
KEYWORDS: ACOSH
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Clause 16.9.5 ACOSH(X) p5 Result Value has:
"If the result is complex the imaginary part is expressed in radians and lies in the range 0 <= AIMAG (ACOSH (X)) <= {pi}".

There is general agreement that the principal value of ACOSH should have the real part non-negative which means the imaginary part must range from -pi to +pi.

Is this a mistake in the definition of ACOSH?

ANSWER:

Yes, this is a mistake. Edits are included to correct this error.
This will be an incompatibility with Fortran 2008.

EDITS to 18-007r1:

[26:17+] 4.3.3 Fortran 2008 compatibility, p10+, insert new para
"Fortran 2008 required ACOSH of a complex value to have the imaginary part nonnegative and had no requirement on the real part. This document requires ACOSH of a complex value to have a nonnegative real part and has no such requirement on the imaginary part."

[340:34-35] 16.9.5 ACOSH, p5 Result Value,
after "is complex"
insert "the real part is nonnegative and",
change "range 0" to "range $-\pi$".
Making the whole p5 read
"Result Value. The result has a value equal to a processor-dependent approximation to the inverse hyperbolic cosine function of X. If the result is complex the real part is non-negative, and the imaginary part is expressed in radians and lies in the range $-\pi$ <= AIMAG (ACOSH (X)) <= π."

SUBMITTED BY: Anton Shterenlikht

HISTORY: 18-236 m217 Submitted
18-236r1 m217 Put into interp format, revised edits.
18-236r2 m217 Revised draft - Passed by J3 meeting
19-228 m220 Passed as amended by J3 letter ballot #35

No WG5 Comments

NUMBER: F18/002
TITLE: Internal procedures in generic interface blocks
KEYWORDS: Internal procedure, generic interface block
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

An internal procedure is allowed in a GENERIC statement, but apparently it cannot appear in a generic interface block.

Section 15.4.3.4.1 (12.4.3.4.1 in F2008) Generic identifiers p1 states

"A generic interface block specifies a generic interface for each of the procedures in the interface block. The PROCEDURE statement lists procedure pointers, external procedures, dummy procedures, or module procedures that have this generic interface."

Is the apparent prohibition of internal procedures in the PROCEDURE statement intended?

ANSWER:

No, this was not intended. An internal procedure should be allowed in a generic interface body. An edit is provided to correct this mistake.

DISCUSSION:

In F2003, Section 12.3.2.1 Interface Block has rule R1206

<procedure-stmt> is [MODULE] PROCEDURE <procedure-name-list>

with constraint C1270

"A <procedure-name> shall have an explicit interface and shall refer to an accessible procedure pointer, external procedure, dummy-procedure, or module procedure."

Edits in paper 05-202r1 allowed internal procedures to be passed as actual arguments and to be targets of procedure pointers. This meant an internal procedure could be the target of a procedure pointer which was specified in a generic interface body.

Paper 08-178 asked the question "What is the point of excluding
internal procedure in C1207?" and replaced C1207 with

"A <procedure-name> shall be a nonintrinsic procedure that has an explicit interface."

The normative text cited in the question was not modified.
This was an oversight.

EDIT to 18-007r1:

[294:11-12] 15.4.3.4.1 Generic identifiers, p1, second sentence,
Change "procedure pointers, external procedures, dummy procedures, or module procedures"
to "nonintrinsic procedures with explicit interfaces".

This makes the sentence read:
"The PROCEDURE statement lists nonintrinsic procedures with explicit interfaces that have this generic interface."

SUBMITTED BY: Jon Steidel

HISTORY: 18-251 m217 Submitted
18-251r1 m217 Revised draft - Passed by J3 meeting
n/a n/a Typos noted at meeting corrected by /Interp.
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/003
TITLE: Pointer association of component of non-definable selector
KEYWORDS: Pointer association, Associate name
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the following

```
 type :: T
   real, pointer :: X
 end type T

 type(t), external :: F
 real, target :: P

 associate ( A => F(42) )
   nullify ( A%X )          !***
   A%X => P                 !***
 end associate
```

The topic is whether the statements marked "!***" (the NULLIFY statement and the pointer assignment statement) conform to the standard.

11.1.3.3p5 ([175:18-21] of 18-007r1) has two requirements that are relevant to this topic:

> "The associating entity itself is a variable, but if the selector is not a definable variable, the associating entity is not definable and shall not be defined or become undefined. If a selector is not permitted to appear in a variable definition context (19.6.7), neither the associate name nor any subobject thereof shall appear in a variable definition context."

With regards to the second sentence, neither NULLIFY nor a pointer assignment statement is a variable definition context (19.6.7, [516:12-30] lists fifteen variable definition context: neither statement appears in the list. Therefore it seems that this requirement is satisfied.

With regards to the first sentence, the associate-name A "shall not

be defined or become undefined". It appears that neither of these statements cause A to be defined or become undefined, because
(a) An object of derived type is defined if and only all of its nonpointer components are defined (see 19.6.1p4, [511:11]). The pointer association status of the component A%X is thus irrelevant to the question of whether A is defined.
(b) Neither statement appears anywhere in the giant lists of "Events that cause variables to become defined" (or "undefined") in 19.6.5 and 19.6.6.

Therefore we must reluctantly conclude that the requirements of the first sentence also appear to be satisfied.

Against this, it is certainly true that the *value* of A is affected by the statements in question: see 7.5.8 Derived-type values, which states that the "component value" of a pointer component is its association status, and
"The set of values of a particular derived type consists of all possible sequences of the component values..."

However, there is no rule in 11.1.3.3 about changing the values, even though it might seem contradictory that something that changes the value of an undefinable object would be permitted.

So the question is, are the two statements standard-conforming?

ANSWER:

No, these statements were not intended to be standard-conforming.
The lack of an explicit rule is an error in the standard.
An edit is provided to correct this error.

EDIT to 18-007r1:

[175:21] 11.1.3.3 Other attributes of associate names, p5,
After "variable definition context"
insert " or a pointer association context",
making the whole paragraph read:
"The associating entity itself is a variable, but if the selector is not a definable variable, the associating entity is not definable and shall not be defined or become undefined. If a selector is not permitted to appear in a variable definition context (19.6.7), neither the associate name nor any subobject thereof shall appear in a variable definition context or pointer

association context."
{Prohibit the marked statements.
The extra context is only relevant to pointer subobjects, but this
need not be stated explicitly.}

{This might appear to be an incompatibility since the explicit
prohibition is missing from Fortran 2003 and 2008, and thus should
require an edit to the "compatibility" subclauses in clause 4.
However as it seems somewhat contradictory, it is argued that those
standards do not establish an unambiguous interpretation of the code
in question, so no compatibility issue arises.}

SUBMITTED BY: Van Snyder

HISTORY: 18-262r1 m217 Submitted
18-262r2 m217 Revised draft - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 Comments

NUMBER: F18/004
TITLE: Program execution sequence with failed images
KEYWORDS: Program execution, Termination of execution, Failed image,
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider execution of a program with more than one image, and some image has failed (perhaps by execution of FAIL IMAGE, or perhaps by some actual failure). Let us further suppose that all other images have initiated normal termination.

According to 5.3.7 "Termination of execution", paragraph 1:
"Termination of execution of the program occurs
when all images have terminated execution."
This text is unchanged from Fortran 2008, which did not have failed images.

As an image that has failed "has ceased participating in program execution but has not terminated execution", it would seem that in this situation, execution of the program has not terminated.

Should failed images cause execution of the program not to terminate?

ANSWER:

No, failed images have stopped participating in program execution and therefore their existence should not prevent program termination. That this definition is unchanged from Fortran 2008 is an oversight.

An edit is supplied to correct this.

EDIT to 18-007r1:

[38:8] 5.3.7 Termination of execution, p1,
After "when all images have terminated execution"
insert "or failed",
making the last sentence of the paragraph read
"Termination of execution of the program occurs when all images have terminated execution or failed."

SUBMITTED BY: Malcolm Cohen

HISTORY: 19-129 m218 Submitted
19-129r1 m218 Revised draft - Passed by J3 meeting
19-228 m220 Passed as amended by J3 letter ballot #25

No WG5 comments

NUMBER: F18/005
TITLE: Does INPUT_UNIT really identify the same unit as *?
KEYWORDS: Connection, INPUT_UNIT
DEFECT TYPE: Interpretation
STATUS: Passed by J3 letter ballot

QUESTION:

According to 16.10.2.13 INPUT_UNIT [429:8-9],
"The value of the default integer scalar constant INPUT_UNIT identifies the same processor-dependent external unit preconnected for sequential formatted input as the one identified by an asterisk in a READ statement;"

Consider the program:

```
PROGRAM input_unit_test
   USE iso_fortran_env
   CHARACTER(80) line
   OPEN(input_unit,FILE='test.dat',ACTION='read')
   READ(input_unit,'(a)') line
   PRINT *,TRIM(line)
   READ(*,'(a)') line
   PRINT *,TRIM(line)
END PROGRAM
```

Furthermore, let us suppose the file test.dat contains the following two lines:
dat line 1
dat line 2
and let us further suppose the file that was preconnected before the OPEN statement contains the following single line:
inp line 1

Is the output of the program (1):
dat line 1
dat line 2
or is it (2):
dat line 1
inp line 1
?

The plain meaning of the words
"identifies the same processor-dependent external unit...
...as the one identified by an asterisk in a READ"
would seem to imply that (1) is expected, not (2); however, only some Fortran processors produce output (1), while others have been observed to produce output (2).

ANSWER:

Output (1) is correct. According to the quoted words, using INPUT_UNIT in a READ statement must have the same effect as UNIT=* (which is the same effect as not having a input/output control list).

It is common for unit 5 to be effectively preconnected to stdin, but on some processors changing the connection of unit 5 (e.g. with an OPEN statement) does not affect unit=*. On such a processor, the value of INPUT_UNIT should not be equal to 5, but to whatever value, possibly negative, that will continue to connect to unit=*.

EDIT to 18-007r1:

None.

SUBMITTED BY: Malcolm Cohen

HISTORY: 19-130 m218 Submitted
19-130r1 m218 Revised draft - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/006
TITLE: Connection of INPUT_UNIT on different images
KEYWORDS: Connection, INPUT_UNIT
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

According to 16.10.2.13 INPUT_UNIT [429:8-9],
"The value of the default integer scalar constant INPUT_UNIT identifies the same processor-dependent external unit preconnected for sequential formatted input as the one identified by an asterisk in a READ statement;"
and according to 12.5.1 Referring to a file [217:10-11],
"In a READ statement, an io-unit that is an asterisk identifies an external unit that is preconnected for sequential formatted input on image 1 in the initial team only (12.6.4.3)."

This leaves unanswered the status of the i/o unit identified by INPUT_UNIT on images other that 1. Plausible interpretations are:
(a) it is not preconnected on other images;
(b) it is preconnected, but to a processor-dependent file;
(c) it is preconnected, but it is not standard-conforming for a program to use it;
(d) it is preconnected, but it raises an i/o error condition if a program attempts to use it;
(e) attempting to use INPUT_UNIT in any way, including connecting it to another file, is not conforming;
(f) INPUT_UNIT may be connected to another file, but it is still processor-dependent whether it may be used;
(g) on an image other than image 1,
- it is processor-dependent whether it is preconnected,
- if it is preconnected,
  * it is processor-dependent to what file,
  * it is processor-dependent whether use is permitted,
- if it is subsequently connected, it is processor-dependent whether it may be used.

Q1. What is the preconnection status of INPUT_UNIT on images other than image one?

Q2. On an image other than image one, if INPUT_UNIT happens to be preconnected, may it be used?

Q3. On an image other than image one, if INPUT_UNIT is connected to
a different file by an OPEN statement, may it be used?

Also, the definition text for INPUT_UNIT in 16.10.2.13 makes no mention of any caveat, but implies it is preconnected on every image. Perhaps this text could be clarified.

ANSWER:

A1. INPUT_UNIT (and thus the unit identified by an asterisk in a READ statement) was not intended to be preconnected on images other than image one in the initial team.

An edit is supplied to correct this omission.

A2. Moot.

A3. Yes.

EDITS to 18-007r1:

[217:11] 12.5.1 Referring to a file, p4,
After "on image 1 in the initial team only (12.6.4.3)"
insert "; it is not preconnected on any other image".
{Clarify preconnection state on images other than one.}
{Editor notes that "image 1" should be "image one".}

[429:9,10] 16.10.2.13 INPUT_UNIT, p1,
Delete "preconnected for sequential formatted input".
{This is a mere parenthetical remark which could confuse.}
After "input/output control list (12.6.4.3)." insert new sentence
"This unit is preconnected for sequential formatted input on image one in the initial team only, and is not preconnected on any other image."
{Make this match clause 12.}

SUBMITTED BY: Malcolm Cohen

HISTORY: 19-131 m218 Submitted
19-131r1 m218 Revised draft - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/008
TITLE: Contradictory assumed-rank requirements
KEYWORDS: Assumed rank
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Paragraph 1 of subclause 8.5.8.7 (Assumed-rank entity) states that an assumed-rank entity is a dummy data object ... or the associate name of a RANK DEFAULT block in a SELECT RANK construct.

C837 requires an assumed-rank entity to be a dummy data object. It does not permit an associate name of a RANK DEFAULT block in a SELECT RANK construct to have assumed rank.

Is an associate name of a RANK DEFAULT block in a SELECT RANK construct permitted to have assumed rank?

ANSWER:

Yes, the associate name of a RANK DEFAULT block may have assumed rank.

An edit is supplied to remove the contradiction.

EDIT to 18-007r1:

[101:13-14 C837 in 8.5.8.7 Assumed-rank entity]

At the end of C837, after "or VALUE attribute", insert
", or the associate name of a RANK DEFAULT block in a SELECT RANK construct whose selector has assumed rank",
making the whole constraint read:

C837 An assumed-rank entity shall be a dummy data object that does not have the CODIMENSION or VALUE attribute, or the associate name of a RANK DEFAULT block in a SELECT RANK construct whose selector has assumed rank.

SUBMITTED BY: Van Snyder

HISTORY: 19-119 m218 Submitted
19-119r1 m218 Revised draft - Passed by J3 meeting

19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/009
TITLE: Bad examples in IEEE_ARITHMETIC functions
KEYWORDS: IEEE_VALUE, IEEE_ARITHMETIC module, examples
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

In 17.11 the examples for IEEE_MAX_NUM, IEEE_MIN_NUM, IEEE_QUIET_compare functions, and IEEE_SIGNALING_compare functions all contain a call to IEEE_VALUE(IEEE_QUIET_NAN). IEEE_VALUE has two non-optional arguments, not one. Should calls to IEEE_VALUE in each of these examples have an additional X= argument?

ANSWER:

Yes, IEEE_VALUE has two non-optional arguments, X and CLASS.
The X argument was mistakenly omitted.

Edits are provided to fix these issues.

EDITS to 18-007r1:

[448:12] 17.11.17 IEEE_MAX_NUM p8
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[449:9] 17.11.19 IEEE_MIN_NUM p8
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[451:4] 17.11.24 IEEE_QUIET_EQ p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[451:18] 17.11.25 IEEE_QUIET_GE p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[451:32] 17.11.26 IEEE_QUIET_GT p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[452:10] 17.11.27 IEEE_QUIET_LE p7

Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[452:24] 17.11.28 IEEE_QUIET_LT p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[452:38] 17.11.29 IEEE_QUIET_NE p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[457:37] 17.11.41 IEEE_SIGNALING_EQ p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[458:13] 17.11.42 IEEE_SIGNALING_GE p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[458:27] 17.11.43 IEEE_SIGNALING_GT p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[459:4] 17.11.44 IEEE_SIGNALING_LE p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[459:18] 17.11.45 IEEE_SIGNALING_LT p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

[459:32] 17.11.46 IEEE_SIGNALING_NE p7
Change "IEEE_VALUE (IEEE_QUIET_NAN)" to
"IEEE_VALUE (1.0, IEEE_QUIET_NAN)"

SUBMITTED BY: Jon Steidel

HISTORY: 19-124 m218 Submitted - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/010
TITLE: Categories of pure procedures
KEYWORDS: pure
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Subclause 15.7 Pure procedures begins with a list of categories of pure procedures. The list does not include procedure pointers or type-bound procedures. Dummy procedures are included in the list, and so dummy procedure pointers are allowed, but other procedure pointers are not.

Q1. Can procedure pointers that are not dummy procedures be pure?

Q2: Can type-bound procedures be pure?

ANSWER:

A1. Procedure pointers that are not dummy procedures are allowed to be pure. The absence of procedure pointers in the list in Subclause 15.7 is an oversight. An edit to correct the oversight is provided.

A2. Type-bound procedures that are bound to pure procedures are pure. The absence of type-bound procedures in Subclause 15.7 is an oversight. An edit to correct the oversight is provided.

EDITS to 18-007r1:

[324:1, 15.7 "Pure procedures" p1]
Replace
"specified to be PURE, or"
with
"specified to be PURE,
* a procedure pointer that has been specified to be PURE,
* a type-bound procedure that is bound to a pure procedure, or"
{ Add missing entries to the list in Subclause 15.7. }

SUBMITTED BY: Robert Corbett

HISTORY: 19-151 m218 Submitted - Passed by J3 meeting

19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/011
TITLE: Categories of elemental procedures
KEYWORDS: elemental
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

The list of categories of elemental procedures at the start of Subclause 15.8 "Elemental procedures" [325:11-12] is incomplete. Procedures that are defined to be elemental in intrinsic modules are not included. Type-bound procedures are not included. Dummy procedures and procedure pointers are not included, but there is evidence that those omissions were intentional.

The standard intrinsic modules specify several procedures to be elemental. The omission of module procedures declared in intrinsic modules from the list in Subclause 15.8 creates a contradiction.

Subclause 7.5.6.2 [80:5-8] and Subclause 7.5.7.3 [82:16] both assume that type-bound procedures may be elemental.

Nonintrinsic elemental procedures cannot be used as actual arguments [301: 29, 15.5.1 "Syntax of a procedure reference" p1]. Elemental procedures specified in the standard intrinsic modules are generic, and so they cannot be passed as actual arguments. Elemental intrinsic procedures can be passed as actual arguments [309: 2-5, 15.5.2.9 "Actual arguments associated with dummy procedure entities" p1], but the text cited indicates that dummy procedures cannot be elemental.

Procedure pointers pose a bit of a problem. A nonintrinsic elemental procedure cannot be the target of a pointer assignment [165:34, 10.2.2.2 "Syntax of the pointer assignment statement"]. However, Subclause 10.2.2.4 "Pointer procedure assignment" paragraph 3 states

> If the pointer object has an explicit interface, its characteristics shall be the same as the pointer target except that the pointer target may be pure even if the pointer object is not pure and the pointer target may be an elemental intrinsic procedure even if the pointer object is not elemental.

The final phrase suggests that a pointer object might be elemental, but it does not say that it can be elemental.

Q1. Is a procedure in an intrinsic module that is specified
to be elemental an elemental procedure?

Q2. Can a type-bound procedure be elemental?

Q3. Can a dummy procedure be specified to be elemental?

Q4. Can a procedure pointer be specified to be elemental?

ANSWER:

A1. Procedures in intrinsic modules that are specified to be elemental
are elemental. An edit is provided to correct the omission in
Subclause 15.8.

A2. A type-bound procedure that is bound to an elemental
procedure is elemental. An edit is provided to correct
the omission in Subclause 15.8.

A3. A dummy procedure cannot be specified to be elemental.
An edit is provided to make this restriction explicit.

A4. A procedure pointer cannot be specified to be elemental.
An edit is provided to make this restriction explicit.

The edits provided are intended to make the impact on the existing
standard small. More extensive changes are probably desirable, and
could be made in a revision of the standard. In particular, the
restrictions on dummy arguments and procedure pointers could be made
constraints.

EDITS to 18-007r1:

[167:8] 10.2.4.4 Procedure pointer assignment, p3, Replace
"elemental intrinsic procedure even if the pointer object
is not elemental."
with
"elemental intrinsic procedure, even though the pointer
object is not elemental."
{ Remove the suggestion that a procedure pointer might
be elemental.}

[325:11-12] 15.8.1 Elemental procedure declaration and interface, p1,
Replace the sentence

"An elemental procedure is ... an elemental subprogram."
with
"An elemental procedure is
* an elemental intrinsic procedure (16.1),
* a module procedure in an intrinsic module, if it is specified to be elemental,
* a procedure that is defined by an elemental subprogram, or
* a type-bound procedure that is bound to an elemental procedure."
and insert a new paragraph following it:
"A dummy procedure or procedure pointer shall not be specified to be ELEMENTAL."
{ Add missing items to the list in Subclause 15.8. Add a missing restriction to Subclause 15.8. }

SUBMITTED BY: Robert Corbett

HISTORY: 19-153 m218 Submitted
19-153r1 m218 Amended - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/012
TITLE: Internal procedure in a generic interface
KEYWORDS: Internal procedure, Generic interface
DEFECT TYPE: Clarification
STATUS: Passed by J3 letter ballot

QUESTION:

In Fortran 2003, an internal subprogram was not allowed to define a specific procedure for a generic identifier:

Fortran 2003 [285:14-15]
C1207 (R1206) A <procedure-name> shall have an explicit interface and shall refer to an accessible procedure pointer, external procedure, dummy procedure, or module procedure.

Paper 08-178 proposed the new feature of permitting an internal procedure in this context, and this was included in Fortran 2008:

[Fortran 2008 281:8]
C1207 (R1206) A <procedure-name> shall be a nonintrinsic procedure that has an explicit interface.

However, this new feature was not mentioned in the Introduction of Fortran 2008, nor is it mentioned in Annex C.1 of Fortran 2018 (which lists Fortran 2008 features not originally mentioned in its Introduction).

Should this be mentioned in Annex C.1?

ANSWER:

Yes, this new feature should be mentioned in Annex C.1.

An edit is provided.

EDIT:

[528:22+] C.1 Fortran 2008 features not mentioned in its Introduction, p1,
Insert an item at the end of the bullet list:
"- An internal procedure name can appear in a <procedure-stmt> in a generic interface block."

SUBMITTED BY: Van Snyder

HISTORY: 19-179 m219 Submitted - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/013
TITLE: TEAM_NUMBER arguments and intrinsic function are ambiguous
KEYWORDS: NUM_IMAGES, IMAGE_INDEX, TEAM_NUMBER, teams
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider

```
PROGRAM p
  n = NUM_IMAGES()
  n1 = MERGE(1,2,THIS_IMAGE()<2)
  n2 = MERGE(1,3,THIS_IMAGE()<3)
  FORM TEAM(n1,x)
  FORM TEAM(n2,y)
  IF (n1==1) THEN
    CHANGE TEAM (x)
      PRINT *,NUM_IMAGES(TEAM_NUMBER=1) !(A)
      PRINT *,NUM_IMAGES(TEAM_NUMBER=3) !(B)
    END TEAM
  END IF
END PROGRAM
```

According to 16.9.145 NUM_IMAGES, the TEAM_NUMBER argument
"shall identify the initial team or a team whose parent is the
same as that of the current team".

However, at (A), there are two teams with team number 1 and the same parent, one created by the FORM TEAM with "n1", the other created by the FORM TEAM with "n2".

At (B), there is only one team with team number 3 and the same parent as the current team, however, it is not a "sibling" team in that it was not created by the same FORM TEAM statement.

Referring to the definition of the term "team number" at 3.145.4:
"-1 which identifies the initial team, or positive integer that
identifies a team within its parent team"
which has the same flaw in that there could be multiple teams within the parent team that have that number.

Q1. Was TEAM_NUMBER in NUM_IMAGES, and the "team number" definition, intended to limit the identification to teams created by

corresponding FORM TEAM statement executions?

Q2. The TEAM_NUMBER argument of IMAGE_INDEX suffers from similar wording. Was this also intended to be limited to "sibling" teams?

ANSWER:

Yes, this is the intention. Edits are provided.

The edits use the terminology "corresponding" executions, as execution on more than one image can hardly be the "same" execution.

The edits also use the term "sibling teams". As sibling teams are always created as a group, and really only have meaning when there are more than one, the plural form is probably most appropriate.

EDITS:

[20:27+] Terms and definitions:
Insert after the definition of "parent team"
"3.145.3+
sibling teams
teams created by a single set of corresponding executions of the FORM TEAM statement (11.6.9)"

[20:30] 3.145.4 team number, definition, change
"within its parent team" -> "among its sibling teams"
making the definition read
"-1 which identifies the initial team, or a positive integer that identifies a team among its sibling teams (5.3.4)

[36:23] 5.3.4 Program execution, para 2, final sentence
Replace "Within its parent team,"
with "Among its sibling teams,"
making the whole sentence read
"Among its sibling teams, each team is identified by its team_number; this is the integer value that was specified in the FORM TEAM statement."
{This is just waffle, no need to index "sibling teams".}

[131:18-19] In 9.6 Image selectors, p3 second sentence
Replace "one of the teams that were formed by execution of the FORM TEAM statement for"
with "a sibling team of"

making the whole sentence read
"If a TEAM_NUMBER= specifier appears in an image-selector and the current team is not the initial team, the value of scalar-int-expr shall be equal to the value of a team number for a sibling team of the current team, and the team of the image selector is that team; the object shall be an established coarray in an ancestor team of the current team, or an associating entity of the CHANGE TEAM construct."

[204:35] 11.6.9 FORM TEAM statement, p1,
change "creates new teams"
to "creates a set of sibling teams"
and index "sibling teams" here.
This makes that whole paragraph read:
"The FORM TEAM statement creates a set of sibling teams whose parent team is the current team."

[380:12-13] In 16.9.97 IMAGE_INDEX, p3 TEAM_NUMBER
Change
"team whose parent is the same as that of the current team"
to
"sibling team of the current team"
and index "sibling teams" here.
making the whole argument read:
"TEAM_NUMBER shall be an integer scalar. It shall identify the initial team or a sibling team of the current team."

[401:24-25] 16.9.145 NUM_IMAGES, p3 Arguments, TEAM_NUMBER argument,
Change
"team whose parent is the same as that of the current team"
to
"sibling team of the current team"
and index "sibling teams" here.
This makes that whole argument read:
"TEAM_NUMBER shall be an integer scalar. It shall identify the initial team or a sibling team of the current team."

[421:9]16.9.189 TEAM_NUMBER([TEAM]), p5 Result Value,
Change "within its parent team"
to "among its sibling teams",
and index "sibling teams" here.
This makes that whole paragraph read:
"The result has the value -1 if the specified team is the initial team; otherwise, the result value is equal to the positive integer

that identifies the specified team among its sibling teams."

SUBMITTED BY: Malcolm Cohen, Jon Steidel, and John Reid

HISTORY: 19-190 m219 Submitted
19-190r1 m219 Merged with 19-178 - Passed by J3 meeting
19-228 m220 Passed as amended by J3 letter ballot #35

WG5 Comments:

Edit for [20:30] 3.145.4 team number, definition.
The edit and its expanded version are not consistent. I suggest removing "(5.3.4)" from the expanded version. I think that this reference is unnecessary with the term "sibling teams" defined (see edit for [20:27+] Terms and definitions).

NUMBER: F18/014
TITLE: Type of OPERATION arguments to the REDUCE intrinsic
KEYWORDS: REDUCE, OPERATION, polymorphic, type
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Section 16.9.161 REDUCE p3 describes the arguments to the intrinsic subroutine CO_REDUCE.

The description of the ARRAY argument states:
"ARRAY shall be an array of any type."

It describes the OPERATION argument as:

"OPERATION shall be a pure function with exactly two arguments; each argument shall be a scalar, nonallocatable, nonpointer, nonoptional dummy data object with the same type and type parameters as ARRAY. If one argument has the ASYNCHRONUS, TARGET, or VALUE attribute, the other shall have that attribute. Its result shall be a non-polymorphic scalar and have the same type and type parameters as ARRAY. OPERATION should implement a mathematically associative operation. It need not be commutative."

The function result of OPERATION cannot be polymorphic. The arguments to OPERATION must have the same type as its result, and thus cannot be polymorphic. However, ARRAY can be polymorphic. This means the dynamic type of ARRAY must be the same type as the arguments and result of OPERATION. This seems like an unfortunate requirement.

Was it intended that the dynamic type of ARRAY be required to match the type and type parameters of the arguments of OPERATION?

ANSWER:

No, this requirement was not intended. Edits are provided to correct the problem.

EDITS:
[408:36] change "type and type parameters"
to "declared type and type parameters"

[408:39] change "type and type parameters"
to "declared type and type parameters"

so that the description of OPERATION reads:
"shall be a pure function with exactly two arguments; each argument shall be a scalar, nonallocatable, nonpointer, nonoptional dummy data object with the same declared type and type parameters as ARRAY. If one argument has the ASYNCHRONOUS, TARGET, or VALUE attribute, the other shall have the attribute. Its result shall be a non-polymorphic scalar and have the same declared type and type parameters as ARRAY. OPERATION should implement a mathematically associative operation. It need not be commutative."

[409:5] change "type and type parameters"
to "declared type and type parameters"

so that the first sentence of Result Characteristics reads:
"The result is of the same declared type and type parameters as ARRAY."

SUBMITTED BY: Jon Steidel

HISTORY: 19-192 m219 Submitted
19-192r1 m219 Revised draft - Passed by J3 meeting
19-228 m220 Passed by J3 letter ballot #35

No WG5 comments

NUMBER: F18/016
TITLE: Host association changes in Fortran 2018
KEYWORDS: Host association
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

The default semantics for accessing entities by host association from an interface body appear to be different in Fortran 2018 than in Fortran 2008.

Problem 1:
In Fortran 2008, an interface body that is not a module procedure interface body cannot access entities in its host by host association unless an IMPORT statement is present in the interface body. The same rule applies by default in Fortran 2018 if the interface body is for an external or dummy procedure, but not if the interface body is for an abstract interface or a procedure pointer that is not a dummy procedure pointer (see 8.8 "IMPORT statement" [117:17-19]).

For example, in

```
  DOUBLE PRECISION X
  ABSTRACT INTERFACE
    SUBROUTINE SUB(A)
      REAL(KIND(X)) A
    END SUBROUTINE
  END INTERFACE
```

Fortran 2008 specifies that X is default REAL, and that therefore so is argument A, but Fortran 2018 specifies that X is accessed by host association and so argument A is double precision.

Problem 2:
The Fortran 2008 standard specified that a submodule has access to host entities, but the Fortran 2018 standard does not specify any default host association semantics for a submodule (it specifies IMPORT semantics only for nested scoping units (see 8.8 "IMPORT statement" [117:23-26]). That makes submodules using host association not conforming.

For example, in

```
  MODULE mod
    INTERFACE
```

```
      MODULE SUBROUTINE S
      END SUBROUTINE
    END INTERFACE
    INTEGER,PARAMETER :: WP = KIND(0.0)
  END MODULE
  SUBMODULE (mod) submod
    REAL(WP) X
  END SUBMODULE
```

the submodule references WP by host association in Fortran 2008, but Fortran 2018 does not specify any semantics and so the whole thing is not conforming.

Problem 3:
The Fortran 2008 standard specified that generic identifiers were accessible by host association, but the Fortran 2018 standard specifies that host association is for named entities.

For example, in

```
  INTERFACE OPERATOR(.plus.)
    PROCEDURE plusfun
  END INTERFACE
  ...
  CONTAINS
    SUBROUTINE SUB(a,b,c)
    ...
    c = a.plus.b
```

Fortran 2018 would not permit access to the user-defined operator.

Problem 4:
The Fortran 2018 standard specifies that BLOCK constructs access named entities in their hosts by host association. This makes no sense because BLOCK constructs already have access to entities in their hosts through inclusive scoping.

Were these changes intended?

ANSWER:

No, none of these changes were intended.

Edits are provided to restore the semantics specified in the Fortran 2008 standard.

EDITS to 18-007r1:

[117:18-19] 8.8 IMPORT statement, p2, second sentence,
Replace "interface body for an ... procedure."
with
"interface body that is not a module procedure
interface body."

making the sentence read

"This is the default for an interface body that is not
a module procedure interface body."

[117:25-26] 8.8 IMPORT statement, p4, second sentence,
Change "for a nested scoping unit ... procedure"
to "for a derived-type definition, internal subprogram,
module procedure interface body, module subprogram, or
submodule"
making the sentence read
"This is the default for a derived-type definition,
internal subprogram, module procedure interface body,
module subprogram, or submodule."

[502:7] 19.5.1.4 "Host association", p1, first sentence
Change "nested scoping unit"
to "derived-type definition, interface body, internal
subprogram, module subprogram, or submodule",
Delete "named",
Making the sentence read
"A derived-type definition, interface body, internal
subprogram, module subprogram, or submodule has access to
entities from its host as specified in 8.8."

SUBMITTED BY: Robert Corbett

HISTORY: 19-257 m220 F18/016 Submitted
19-257r1 m220 Revised draft - Passed by J3 meeting
20-132 m222 Passed by J3 letter ballot #36

No WG5 comments

NUMBER: F18/017
TITLE: Final subroutine invocation order
KEYWORDS: FINAL ALLOCATABLE
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider

```
 Module m
  Type base
  Contains
   Final basef
  End Type
  Type other
  Contains
   Final otherf
  End Type
  Type,Extends(base) :: t
   Type(other),Allocatable :: comp
  Contains
   Final tf
  End Type
 Contains
  Subroutine basef(a)
   Type(base),Intent(InOut) :: a
   Print *,'basef'
  End Subroutine
  Subroutine otherf(b)
   Type(other),Intent(InOut) :: b
   Print *,'otherf'
  End Subroutine
  Subroutine tf(c)
   Type(t),Intent(InOut) :: c
   Print *,'tf'
  End Subroutine
 End Module
 Program test
  Use m
  Call sub
 Contains
  Subroutine sub
```

```
    Type(t) x
    Allocate(x%comp)
  End Subroutine
End Program
```

When the subroutine is executed, it will finalize X on exit,
so what is the expected output?

Finalization of X occurs before auto-deallocation of X%COMP;
this follows from 9.7.3.2 paragraph 9.

According to 7.5.6.2, in sequence
 (1) the object's final procedure is invoked, i.e. TF is called,
 (2) finalizable components are finalized, i.e. OTHERF is called,
 (3) the parent is finalized, i.e. BASEF is called.
And according to 7.5.6.3, deallocating X%COMP finalizes it,
and so
 (4) OTHERF is called.
I.e. the output is
 TF
 OTHERF
 BASEF
 OTHERF

However, this violates the principle that you only finalize something
once.

Q1. Is X%COMP actually finalized twice?

It could be argued that "finalizing X before deallocating X%COMP"
only puts an order on invocation of TF, and in particular, finalizing
the parent pseudo-component need not precede the deallocation. But
this would still invoke OTHERF twice.

Q3. Is the auto-deallocation of an allocatable component required to
    follow the finalization of other components and the parent
    pseudo-component?

Now consider the case where X%COMP is not allocated (i.e. delete the
ALLOCATE statement). According to 7.5.6.2, it should invoke
 (1) TF on X
 (2) OTHERF on X%COMP
 (3) BASEF on X%BASE
however, as X%COMP is unallocated, the invocation in step (2) does

not conform to the procedure reference requirements, i.e. the program is not conforming.

Q2. Is X%COMP required to be allocated when X is finalized?

DISCUSSION:

An object is only finalized in situations listed in 7.5.6.3.
Every such situation would also unconditionally deallocate any allocatable component, and if that component were finalizable, such deallocation would also unconditionally finalize the component (* except for intrinsic assignment, where a previous interpretation added an exclusion).

Therefore it seems to be broken to finalize any allocatable component during finalization of the object it is contained in, as either it will be non-conforming, or will be finalized twice (* except for the previously-added exception).

The design where allocatable entities are finalized at the time of deallocation would seem to be simpler, easier to understand, and less buggy.

Perhaps the finalization of allocatable components in 7.5.6.2 step (2) should be removed, and the exclusion for intrinsic assignment for deallocation-finalization should also be removed?

ANSWER:

A1. No object should be finalized twice.
A2. No, a finalizable allocatable component should not be required to be allocated when its containing object is finalized.

The inclusion of allocatable components in 7.5.6.2 step (2) is an error in the standard, and the intrinsic assignment exception for finalization on deallocation is likewise an error.

Edits are provided to correct these errors.

A3. An allocatable component should be able to be finalized as soon as the object's final subroutine returns, i.e. there should be no requirement on the processor to produce a particular invocation order here.

The ambiguity in whether component deallocation and component finalization should be ordered is inadvertent. An edit is provided to remove any implication that these need to have a specific order.

FURTHER ELUCIDATION:

Finalization ordering is a partial ordering. When final subroutines will be executed for a type (TFIN), an allocated allocatable component thereof (AFIN), an ordinary (viz nonallocatable nonpointer) component thereof (OFIN), and the type's parent type (PFIN), the orderings are:

TFIN<AFIN
TFIN<OFIN<PFINAL

Note there is no ordering between AFIN and OFIN, and no ordering between AFIN and PFIN. Thus, after applying the edits below to the standard, the following execution orders are valid:

TFIN, AFIN, OFIN, PFIN
TFIN, OFIN, AFIN, PFIN
TFIN, OFIN, PFIN, AFIN

EDITS to 18-007r1:

[80:9] 7.5.6.2 The finalization process, p1, item (2),
"All finalizable" -> "All nonallocatable finalizable".
{Remove redundant finalization.}

[80:22] 7.5.6.3 When finalization occurs, p2,
After
"unless it is the variable in an intrinsic assignment statement"
Delete "or a subobject thereof".
{Remove allocatable component exclusion in intrinsic assignment.}

[137:28] 9.7.3.2 Deallocation of allocatable variables, p9,
Change "that object is finalized"
To "any final subroutine for that object is executed",
Making the whole paragraph read
"If an allocatable component is a subobject of a finalizable object, any final subroutine for that object is executed before the component is automatically deallocated."

SUBMITTED BY: Malcolm Cohen

HISTORY: 20-117 m221 F18/017 Submitted - Passed by J3 meeting
20-132 m222 Passed as amended by J3 letter ballot

No WG5 comments

NUMBER: F18/018
TITLE: Public namelist and private variable
KEYWORDS: NAMELIST PUBLIC PRIVATE
DEFECT TYPE: Clarification
STATUS: Passed by J3 letter ballot

QUESTION:

Consider

```
 Module m1
   Real,Public :: x
 End Module
 Module m2
   Use m1
   Private x
   Namelist/nml/x
 End Module
```

On the face of it, module M2 appears to violate
 C8105 (R868) A namelist-group-object shall not have the PRIVATE
 attribute if the namelist-group-name has the PUBLIC attribute.
as the local X indeed has the PRIVATE attribute. On the other hand,
it is just a local name for the remote X which is PUBLIC, which
raises doubts.

Comment: This seems to be a very old constraint dating back to when
 the standard was much more restrictive about such things.
 It is not clear why this should be disallowed.
 Even if X were a local variable of M2, it is not clear what
 purpose this constraint serves.

A quick compiler survey revealed that most but not all compilers
think that it is where the variable is defined that matters, i.e.
many accept the example code.

Q1. Should PRIVATE local variables really be prohibited from a PUBLIC
 namelist?

If the answer to Q1 is yes,
Q2. Is it PRIVATE on the local name that matters, or PRIVATE on the
 variable where it is defined?

COMMENT:

A NAMELIST statement can contain several namelist-group-names, so it is also somewhat ambiguous as to which namelist-group-objects this constraint applies to.

ANSWER:

A1. Yes. Although it would be reasonable to lift this restriction in a future standard, it is a deliberate restriction in this standard.

A2. It is whether the local name has the PRIVATE attribute that matters, not where the variable is declared.

An edit is provided.

EDIT to 18-007r1:

[119:8-9] 8.9 NAMELIST statement, C8105,
After "PRIVATE attribute" insert "in the local scope",
and change "the namelist-group-name" to "its namelist-group-name",
making the whole constraint read
"C8105 (R868) A namelist-group-object shall not have the PRIVATE attribute in the local scope if its namelist-group-name has the PUBLIC attribute."

SUBMITTED BY: Malcolm Cohen

HISTORY: 20-127 m221 F18/018 Submitted
20-127r1 m221 Passed by J3 meeting
20-132 m222 Passed as amended by J3 letter ballot #36

No WG5 comments