N2205

ISO/IEC 1539-1:2018 Defect Report for Corrigendum 2

August 11, 2022

This document lists all the reported defects in ISO/IEC 1539-1:2018 (Fortran 2018) that are to be addressed by a planned Corrigendum 2. JTC1/SC22/WG5 voted on these and comments received follow each item. 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:

- N2195 WG5 letter ballot 2 on Fortran 2018 interpretations
- N2197 Results of WG5 letter ballot 2 on Fortran 2018 interpretations

N2205

NUMBER: F18/007
TITLE: Problems with C_FUNLOC and C_F_PROCPOINTER being PURE
KEYWORDS: C_FUNLOC, C_F_PROCPOINTER, ISO_C_BINDING
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

1) Regarding C_FUNLOC (X),

In 15.7 Pure procedures, constraint C1590 is:

"C1590 The specification-part of a pure subprogram shall specify
that all its dummy procedures are pure."

In 15.5.2.9 Actual arguments associated with dummy procedure entities,
the first paragraph says

"If the interface of a dummy procedure is explicit, its
characteristics as a procedure (15.3.1) shall be the same as those
of its effective argument, except that a pure effective argument
may be associated with a dummy argument that is not pure and an
elemental intrinsic actual procedure may be associated with a dummy
procedure (which cannot be elemental)."

If C_FUNLOC is PURE, then these together imply that the actual
argument, X, has to be a PURE procedure. This is not stated in the
specification of this function in 18.2.3.5. This has the effect of
limiting the uses of C_FUNLOC compared to the specification in Fortran
2008. This is not indicated in 4.3.3 Fortran 2008 compatibility. Was
this an intentional change, or was making C_FUNLOC PURE a mistake?

2) Regarding C_F_PROCPOINTER (CPTR, FPTR), a similar argument to that
above implies that the INTENT(OUT) procedure pointer, FPTR, is a PURE
procedure. However, there is no stated requirement that the input
argument CPTR be PURE, and indeed there is no specification of what
that even means if CPTR is a pointer to an interoperable C function.
This suggests that C_F_PROCPOINTER provides a backdoor allowing an
impure procedure to appear to be pure, invalidating the assumptions
that are associated with a pure procedure. Was making C_F_PROCPOINTER
PURE a mistake?

ANSWER:

1) It was not an error to make C_FUNLOC pure in principle. But if a
reference to C_FUNLOC appears in a pure procedure, its argument should
have been required to be pure.

It is noted that constraint C1590 does not apply to C_FUNLOC as it is
a procedure from an intrinsic module, and as such is not defined by a
subprogram. The only question is whether its argument is required to
be pure, and in what circumstances.

2) Making C_F_PROCPOINTER pure was a mistake.

Edits are included to correct these errors.

EDITS to 18-007r1:

[325:8+] In 15.7 Pure Procedures, following constraint C1599, add a new constraint:

"C1599a A reference to the function C_FUNLOC from the intrinsic module ISO_C_BINDING shall not appear in a pure subprogram if its argument is impure."

[469:26-27] In 18.2.3 Procedures in the module, 18.2.3.1 General, second sentence, change "C_F_POINTER subroutine is" to "C_F_POINTER and C_F_PROCPOINTER subroutines are".

Making the whole sentence read

"The C_F_POINTER and C_F_PROCPOINTER subroutines are impure; all other procedures in the module are pure."

[472:16] In 18.2.3.4 C_F_PROCPOINTER (CPTR, FPTR), Class paragraph, Change "Pure subroutine" to "Subroutine".

SUBMITTED BY: Bill Long

```
HISTORY: 19-114   m218  Submitted
         19-114r1 m218  Revised draft
         19-114r2 m218  Passed as amended by J3 meeting
         19-228   m220  Failed J3 letter ballot #35
         21-124   m223  Repair edits to satisfy objections raised in
                        letter ballot, passed by J3 meeting 223, 11-2.
         21-184   m225  Passed by J3 letter ballot #37
```

Comment:
F18/007
Muxworthy:
Also [xiv:3] after "C_F_POINTER" add "and C_F_PROCPOINTER".

```
----------------------------------------------------------------------

----------------------------------------------------------------------
```

NUMBER: F18/015
TITLE: Example in C.6.8 is wrong
KEYWORDS: failed images
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

The example code for failed images in C.6.8 raises several issues about its correctness.

QUESTION:

```
Q1.A: In the example in C.6.8, the assignments
        me[k] = failures(i)
        id[k] = 1
      are made by image 1 and the assignments
```

```
        me = THIS_IMAGE ()
        id = MERGE (1, 2, me<=images_used)
      are made by image k in unordered segments. Was this intended?
```

B: In the example in C.6.8, the assignment

```
          me[k] = failures(i)
```

is made by image 1 and me[k] is referenced on other images in the FORM TEAM statement in unordered segments. Was this intended?

Q2. Suppose the program in C.6.8 is executed by 11 images, so 1 is intended to be a spare. If image 9 in the initial team fails immediately before it executes the first FORM TEAM statement, then image 10 in the initial team, which executes FORM TEAM with a team-number == 1 and NEW_INDEX == 10 (== me), will have specified a NEW_INDEX= value greater than the number of images in the new team. Should there be a test for this in the code?

Q3.A: If a replacement image has failed, its image index will be the value of an element of the array failures, a replacement for it will be found, and the replacement will be placed in team 1. Was this intended?

B: The value of images_used increases each time the setup loop is executed. However, the array failures will contain the image indices of all the failed images and allocate all of them fresh replacements. Was this intended?

Q4. The variable images_used is incremented only on image 1 but is referenced by other images near the beginning and end of DO setup. Was this intended?

Q5. The intention is that on each cycle of the DO iter loop, a calculation is performed on the worker images and if any of them fail during this, the calculation is resumed from a checkpoint with the failed images replaced by spares. On resumption, the variable read_checkpoint has the value true on all the old worker images to indicate that they should access the checkpoint data. On a replacement image, this variable will still have its initial value of false. Was this intended?

Q6. The code for choosing the number of spares does not correspond to the comment for it. Was this intended?

ANSWER:

1-A: No. An image control statement that provides segment ordering is needed.

1-B: No.

2: This is quite a low-probability event, so exiting with the error condition seems appropriate.

3-A: No.

3_B: No. It was intended to allocate replacements only for the newly failed images.

Furthermore, the example contains more errors than in the list above. Therefore an edit is provided that replaces the entire example with a complete rewrite, involving correction of additional errors, a better choice of names, and more comments.

4: No. The problem near the end of DO setup may be avoided by replacing the statement

```
    IF (THIS_IMAGE () > images_used) done = done[1]
```

by

```
    IF (team_number == 2) done = done[1]
```

The problem near the beginning of DO setup may be avoided by making the variable images_used a coarray and referencing images_used[1]. This will need the addition of a SYNC ALL statement just before the statement

```
    outer : DO
```

to ensure that the correct value is used on all images on the first iteration of the loop.

5. No. The variable read_checkpoint is not needed and should be removed. The initial entry is just the special case of the checkpoint data being null so that the calculation needs to be started.

6. No. In the line

```
  images_spare = MAX(NUM_IMAGES()/100,0,MIN(NUM_IMAGES()-10,1))
```

"10" should be "9".

Some of the noteworthy additional changes are:
 - declarations separated out and many comments added or changed;
 - logical variable START added to distinguish the first execution of
   the outer do loop when READ_CHECKPOINT should be false;
 - rename ME to LOCAL_INDEX and ID to TEAM_NUMBER;
 - code added to calculate the local indices of team 2;
 - THEN keyword added to ELSE IF (done) statement.

EDITS to 18-007r1:

```
[543:42-545:17] C.6.8 Example involving failed images,
                Replace the entire example with the code below.
                Note that many lines and comments are broken to keep
                them within 70 columns, these should be joined up or
                reformatted in the actual standard.
"
```

```
PROGRAM possibly_recoverable_simulation
  USE, INTRINSIC :: ISO_FORTRAN_ENV, ONLY:TEAM_TYPE, STAT_FAILED_IMAGE
  IMPLICIT NONE
  INTEGER, ALLOCATABLE :: failures (:) ! Indices of the failed images.
  INTEGER, ALLOCATABLE :: old_failures(:) ! Previous failures.
  INTEGER, ALLOCATABLE :: map(:) ! For each spare image k in use,
             ! map(k) holds the index of the failed image it replaces.
  INTEGER :: images_spare ! No. spare images.
                          ! Not altered in main loop.
  INTEGER :: images_used [*] ! On image 1, max index of image in use.
  INTEGER :: failed ! Index of a failed image.
  INTEGER :: i, j, k ! Temporaries
  INTEGER :: status ! stat= value
  INTEGER :: team_number [*] ! 1 if in working team; 2 otherwise.
  INTEGER :: local_index [*] ! Index of the image in the team.
  TYPE (TEAM_TYPE) :: simulation_team
  LOGICAL :: done [*] ! True if computation finished on the image.

  ! Keep 1% spare images if we have a lot, just 1 if 10-199 images,
  !                                              0 if <10.
  images_spare = MAX(NUM_IMAGES()/100,0,MIN(NUM_IMAGES()-9,1))
  images_used = NUM_IMAGES () - images_spare
  ALLOCATE ( old_failures(0), map(images_used+1:NUM_IMAGES()) )
  SYNC ALL (STAT=status)

  outer : DO
    local_index = THIS_IMAGE ()
    team_number = MERGE (1, 2, local_index<=images_used[1])
    SYNC ALL (STAT = status)
    IF (status/=0 .AND. status/=STAT_FAILED_IMAGE) EXIT outer
    IF (IMAGE_STATUS (1) == STAT_FAILED_IMAGE) &
        ERROR STOP "cannot recover"
    IF (THIS_IMAGE () == 1) THEN
    ! For each newly failed image in team 1, move into team 1 a
    ! non-failed image of team 2.
       failures = FAILED_IMAGES () ! Note that the values
                   ! returned by FAILED_IMAGES increase monotonically.
       k = images_used
       j = 1
       DO i = 1, SIZE (failures)
          IF (failures(i) > images_used) EXIT ! This failed image and
          ! all further failed images are in team 2 and do not matter.
          failed = failures(i)
          ! Check whether this is an old failed image.
          IF (j <= SIZE (old_failures)) THEN
             IF (failed == old_failures(j)) THEN
                j = j+1
                CYCLE ! No action needed for old failed image.
             END IF
          END IF
          ! Allow for the failed image being a replacement image.
          IF (failed > NUM_IMAGES()-images_spare) failed = map(failed)
          ! Seek a non-failed image
          DO k = k+1, NUM_IMAGES ()
            IF (IMAGE_STATUS (k) == 0) EXIT
          END DO
          IF (k > NUM_IMAGES ()) ERROR STOP "cannot recover"
          local_index [k] = failed
          team_number [k] = 1
          map(k) = failed
       END DO
       old_failures = failures
       images_used = k
       ! Find the local indices of team 2
       j = 0
       DO k = k+1, NUM_IMAGES ()
            IF (IMAGE_STATUS (k) == 0) THEN
            j = j+1
            local_index[k] = j
          END IF
       END DO
    END IF
    SYNC ALL (STAT = status)
    IF (status/=0 .AND. status/=STAT_FAILED_IMAGE) EXIT outer
    !
    ! Set up a simulation team of constant size.
    ! Team 2 is the set of spares, so does not participate.
    FORM TEAM (team_number, simulation_team, NEW_INDEX=local_index, &
               STAT=status)
    IF (status/=0 .AND. status/=STAT_FAILED_IMAGE) EXIT outer

    simulation : CHANGE TEAM (simulation_team, STAT=status)
      IF (status == STAT_FAILED_IMAGE) EXIT simulation
      IF (team_number == 1) THEN
         iter : DO
           CALL simulation_procedure (status, done)
           ! The simulation_procedure:
           !  - sets up and performs some part of the simulation;
           !  - starts from checkpoint data if these are available;
           !  - stores checkpoint data for all images from time to
           !  - time and always before return;
           !  - sets status from its internal synchronizations;
           !  - sets done to .TRUE. when the simulation has completed.
           IF (status == STAT_FAILED_IMAGE) THEN
              EXIT simulation
           ELSE IF (done) THEN
              EXIT iter
           END IF
         END DO iter
      END IF
    END TEAM (STAT=status) simulation

    SYNC ALL (STAT=status)
    IF (team_number == 2) done = done[1]
    IF (done) EXIT outer
  END DO outer
  IF (status/=0 .AND. status/=STAT_FAILED_IMAGE) &
    PRINT *,'Unexpected failure',status
END PROGRAM possibly_recoverable_simulation
"
```

SUBMITTED BY: John Reid

```
HISTORY: 19-182   m219  Submitted
         19-182r3 m219  Revised draft - Passed by J3 meeting
         19-228   m220  Failed J3 letter ballot #35
         20-105   m221  Revised answer - Passed by J3 meeting
         20-132   m222  Passed as amended by J3 letter ballot #36
         N2178    n/a   Failed WG5 letter ballot N2176.
         21-105   m223  Revised answer
         21-105r1 m223  Further revised, Passed by J3 meeting 223.
         21-184   m225  Passed by J3 letter ballot #37
```

No WG5 comments

NUMBER: F18/019
TITLE: PURE and default initialization
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

An essential property of pure procedures is that they do not modify the values of nonlocal variables except through dummy arguments. The addition of default initialization of pointer components makes it possible to violate this property. Because default initialization does not imply the SAVE attribute, a local variable of derived type in a pure procedure can include a pointer component whose target is a nonlocal variable. The definition of pure procedures in subclause 15.7 allows a program to modify the value of a nonlocal variable through such a component.

For example,

```
    PROGRAM EXAMPLE1
      REAL, TARGET :: X = 1.0
      TYPE T
        REAL, POINTER :: P => X
      END TYPE T
      CALL SUBR
      PRINT *, X ! X has been changed to 2.0
    CONTAINS
      PURE SUBROUTINE SUBR
        TYPE(T) Y
        Y%P = 2.0
      END SUBROUTINE SUBR
    END
```

A local variable is not needed, for example,

```
    PROGRAM example2
      REAL,TARGET :: x = 123
      TYPE t
        REAL,POINTER :: p => x
      END TYPE
      CALL sub
      PRINT *,x ! No longer == 123?
    CONTAINS
      PURE SUBROUTINE sub()
        ASSOCIATE(y=>t())
          y%p = -999 ! Affects x.
        END ASSOCIATE
      END SUBROUTINE
    END PROGRAM
```

A polymorphic variable can be used, for example,

```
    PROGRAM example2
      REAL,TARGET :: x = 123
      TYPE t
      END TYPE
      TYPE,EXTENDS(t0) :: t
        REAL,POINTER :: p => x
      END TYPE
      CALL sub
      PRINT *,x ! No longer == 123?
    CONTAINS
      PURE SUBROUTINE sub()
        CLASS(t0) y ! Declared type has no initialized ptr comp.
        ALLOCATE(t::y) ! Without SOURCE=, gets a pointer to x.
        SELECT TYPE(y)
        TYPE IS (t)
          y%p = -999 ! Affects x.
        END SELECT
      END SUBROUTINE
    END PROGRAM
```

ALLOCATE with MOLD= instead of a type-spec can do the same.

If component initialization is not an attribute of the component, more convoluted examples are possible using SEQUENCE types so that the local variable does not have default initialization but can be initialized using a compatible type that does.

Q. Was it intended to allow nonlocal variables to be modified by a pure procedure in this way?

ANSWER:

A: No. An edit is supplied to correct the standard.

Note: The question of whether component initialization is an attribute should be addressed by a separate interpretation request.

EDIT to 18-007r1:

```
[324:20-] 15.7 Pure procedures, between NOTE 1 and C1590,
          insert new constraint
   "C1589a A named local entity or construct entity of a pure
           subprogram shall not be of a type that has default
           initialization of a data pointer component to a target at
           any level of component selection."
{With no local or construct entity designator of a "bad" type being
 allowed, one cannot write a pointer component reference to it.
 This is stricter than strictly necessary, as it effectively forbids
 such types from any usage within pure, even unproblematic usage.}
```

SUBMITTED BY: Robert Corbett

```
HISTORY: 20-151   m222  Submitted
         21-123   m223  Revised with answer
         21-123r1 m223  Passed by J3 meeting 223.
         21-184   m225  Passed as amended by J3 letter ballot #37
```

Comment:
F18/019

Corbett:
I do not object to the answer. I object to the proposed edit.

The text explaining the answer in essence says that if no local or construct variable or named constant is "of a type that has default initialization of a data pointer component to a target any level of component selection" the problems described in the QUESTION portion of the interpretation request cannot arise. I believe this assertion is correct.

The proposed edit might ban a bit more than is intended. It clearly suffices to ban the cases that need to be banned. However, it might be read to ban cases that do not need to be banned. Specifically, it might be read as banning type names that name derived types that include default initializations of pointer components to targets. I do not think it does, but I am not sure it does not.

A type name can name a local entity that might seem to be subject to the new constraint. However, I do not think that a derived type is "of a type" (it is a type), and therefore, is not subject to the constraint. I find that to be a slender reed. An alternate edit might be

```
    C1589a A named local or construct data entity of a pure
           subprogram shall not be of a {declared} type that
           has default initialization of a data pointer
           component to a target at any level of component
           selection.
```

The word "declared" need not be part of the edit, but I find the sentence easier to read with it. If the the edit is changed as indicated, I will change my answer to "-Y-".

If the proposed ban is intended to include type names, I still do not object to the interpretation, but different edits will be needed.

(Further discussion in email with Malcolm Cohen disagreeing with Robert Corbett's objections.)

NUMBER: F18/023
TITLE: CLASS(*) ambiguous operator overloading
KEYWORDS: CLASS(*) generic OPERATOR
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the program

```
  Module m
```

```
    Interface Operator(==)
      Module Procedure mp
    End Interface
  Contains
    Logical Function mp(a,b)
      Class(*),Intent(In) :: a,b
      mp = .False.
      Select Type(a)
      Type Is (Integer)
        Select Type(b)
        Type Is (Integer)
          mp = .True.
        End Select
      End Select
    End Function
  End Module
  Program ambiguous
    Use m
    If (13==999) Then
      Print *,'Invoked the user function'
    Else
      Print *,'Did not invoke the user function'
    End If
  End Program
```

```
Is this program valid?
If so, does it invoke the user function or the intrinsic operation?

It is clear from the standard that the user is not supposed to be able
to override intrinsic operators, merely extend them. The last sentence
of 15.4.3.4.2p1 "Defined operations" says:
  "If the operator is an intrinsic-operator (R608), the number of dummy
   arguments shall be consistent with the intrinsic uses of that
   operator, and the types, kind type parameters, or ranks of the dummy
   arguments shall differ from those required for the intrinsic
   operation (10.1.5)."

However, CLASS(*) encompasses these while "differing".

ANSWER:

The program is not conforming because it is ambiguous.

However, the interface was intended to be forbidden; as noted, the
standard clearly intended to prohibit overriding intrinsic operations.
An edit is provided to correct this oversight.

EDITS to 18-007r1:

[295:11] 15.4.3.4.2 Defined operations, p1, last sentence,
    After "(10.1.5)"
    insert ", treating a CLASS(*) dummy argument as not differing in
            type or kind".
{A bit ugly, but we can't use "distinguishable" because operations have
 operands not arguments."

SUBMITTED BY: Paul Richard Thomas
```

N2205

HISTORY: 21-113   m223 Submitted, passed by J3 meeting 223.
         21-184   m225 Passed by J3 letter ballot #37

No WG5 comments

NUMBER: F18/024
TITLE: CFI_setpointer with result a deferred length character
KEYWORDS: C interoperability
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Is the requirement in CFI_setpointer that the elem_lem member have the same value in the source and result arguments intended to apply if the result is of type CHARACTER with the length parameter deferred?

ANSWER:

No. It is an oversight in the standard that this rule apply to the case of result being of type CHARACTER and having the length parameter deferred.

Currently in section 18.5.5.9 "The CFI_setpointer function" states in para 2 that

"If source is not a null pointer, the corresponding values of the elem_len, rank, and type members shall be the same in the C descriptors with the addresses source and result."

The function can therefore not work correctly if the "result" parameter is a descriptor address for a deferred-length character entity coming in from Fortran, because the elem_len value is not available. Manual updates to result->elem_len are prohibited by 18.6 para 1.

The CFI_setpointer function in the intrinsic module ISO_C_BINDING is intended to provide a means for C programmers to replicate the pointer association capability in Fortran, for dummy arguments in interfaces with with interoperable interfaces. Thus, for the case of a character result having deferred length, which is allowed at [478:21], the correct action should be for the elem_len value from the source to be copied to the same member of result as part of execution of the function.

Edits are supplied to correct this defect.

EDITS to 18-007r1:

In 18.5.5.9 The CFI_setpointer function

- [491:27] In the description of the source formal parameter, second sentence, delete "elem_len, ".

- [491:28+] In the description of the source formal parameter, after the second sentence add new sentence:

  "If source is not a null pointer and the C descriptor with the address result does not describe a deferred length character pointer, the corresponding values of the elem_len member shall be the same in the C descriptors with the addresses source and result."

- [491:31] In the first sentence of paragraph 3, "Description" replace "base_addr and dim" by "base_addr, dim and possibly elem_len".

- [491:38] At the end of the second bullet point of paragraph 3, "Description", add new sentence;

  "If the C descriptor with the address result describes a character pointer of deferred length, the value of its elem_len member is set to source->elem_len".

SUBMITTED BY: Reinhold Bader

```
HISTORY: 21-102r1 m223  Submitted
         21-102r3 m223  Passed by J3 meeting 223.
         21-184   m225  Passed by J3 letter ballot #37
```

Comment:
F18/024
Muxworthy:
At [491:27] also delete "," after "rank".

NUMBER: F18/026
TITLE: C_SIZEOF argument
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the example

```
    SUBROUTINE test(b)
      USE iso_c_binding
      REAL(c_double) b(:),a(SIZE(b))
      PRINT *,c_sizeof(a)               ! A
      PRINT *,c_sizeof(b)               ! B
      PRINT *,c_sizeof(a(::2))          ! C
      PRINT *,c_sizeof(a+1)             ! D
      PRINT *,c_sizeof(1.0_c_double)    ! E
    END SUBROUTINE
```

```
18.2.3.7 C_SIZEOF (X) states
    "X shall be an interoperable data entity..."

According to that, the reference to C_SIZEOF marked A is valid, as A
is interoperable (an explicit-shape array of interoperable type and
type parameters). And the reference marked B is invalid, as only
explicit-shape arrays and assumed-size arrays are interoperable, thus
assumed-shape arrays are definitely not.

For the references at C and D, the standard seems to be silent on the
matter of whether they are interoperable. It is clear for named
variables, but although subobject designators can denote variables,
they are not names, and expressions are not variables at all. Being
silent implies non-conformance as no interpretation is established.

The reference at E also appears to be non-conforming, as the standard
specifies no criteria for interoperability of expressions.

However, the description of the result of C_SIZEOF only makes use of
the interoperability of the type and type parameters.

Are these references intended to be conforming?
If not, should the standard be clarified to say that X shall be an
interoperable named variable?

ANSWER:

Yes, these references were all intended to be conforming.
An edit is supplied to correct this mistake, with an addition to
require that any pointer or allocatable argument be associated or
allocated.

EDIT to 18-007r1:

[473:27] 18.2.3.7 C_SIZEOF (X), p3 Argument,
  Replace the paragraph with:

   "Argument. X shall be of interoperable type and type parameters,
    and shall not be an assumed-size array, an assumed-rank array
    that is associated with an assumed-size array, an unallocated
    allocatable variable, or a pointer that is not associated."
{Loosen the requirements.}

SUBMITTED BY: Malcolm Cohen

HISTORY: 21-134   m224  Submitted
         21-134r1 m224  Revised
         21-134r2 m224  Revised again
         22-101r1 m226  Disallow unassociated/unallocated argument.
                        Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.

No WG5 comments

----------------------------------------------------------------------

----------------------------------------------------------------------
```

```
NUMBER: F18/027
TITLE: CO_BROADCAST with allocatable component
KEYWORDS: CO_BROADCAST allocatable component
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the program

program example
  type :: my_string_t
    character(len=:), allocatable :: contents
  end type
  type(my_string_t) :: string
  if (this_image() == 1) string%contents = "Hello, World!"
  call co_broadcast(string, source_image=1)
  print *, string%contents
end program

Is this program valid?
If so, what is the status of string%contents on images other than 1?

The description for argument A in the standard (Section 16.9.46)
states that A:
  "shall have the same shape, dynamic type, and type parameter value,
  in corresponding references."

and given the example that follows, would indicate that allocatable
array arguments must be allocated to the same shape on all images
prior to the call to CO_BROADCAST. However, it also states that A:
  "becomes defined, as if by intrinsic assignment, on all images"

which would indicate that objects with allocatable components would
have those components (re)allocated.

Is it intended for CO_BROADCAST to be usable by objects with
allocatable components, and have those components be (re)allocated on
the receiving images? Furthermore, can those components be polymorphic,
(i.e. be declared with CLASS instead of TYPE)?

ANSWER:

The program is conforming because argument A satisfies the
requirements:
  "shall have the same shape, dynamic type, and type parameter value,
  in corresponding references."

and string%contents should be allocated with value "Hello, World!"
because string shall:
  "becomes defined, as if by intrinsic assignment, on all images"

EDIT to 18-007r1:

[355:22] 16.9.46 CO_BROADCAST, p3 Arguments, argument A, last sentence
    At the end of the last sentence
    add ", including (re)allocation of any allocatable ultimate
         component, and setting the dynamic type of any polymorphic
         allocatable ultimate component".
```

Somewhat redundant because "as if by intrinsic assignment" should already indicate it, but clearly the clarification is needed, as some compilers have not interpreted it as such.

SUBMITTED BY: Brad Richardson

```
HISTORY: 21-139   m224  Submitted
         21-139r1 m224  Removed alternate answer, repaired edit,
                        passed by J3 meeting 224, 8-5.
         21-184   m225  Passed by J3 letter ballot #37
```

No WG5 comments

NUMBER: F18/028
TITLE: Specification inquiry
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the following program:

```
      implicit type (t) (a-z)
      integer,parameter :: n = storage_size(x)
      type t
        character(len=n) :: s
      end type t
      print *, n
      end
```

Two compilers tested issue a compile time error message while a third compiler compiled the program and printed "0" when executed.

Is the use of storage_size(x) a valid specification expression in this program?

DISCUSSION:

In the above program, x is implicitly typed to be type(t). It is used in a specification expression requiring knowledge of type(t), prior to the specification of type(t). There is no text in the standard to prohibit this usage.

The rules for specification expressions require the type, type parameters, array bounds, and cobounds of a variable to be known via prior specification, or use or host association when the variable is used in a specification expression. If an element of an array whose value is used in a specification expression, the array must be fully specified in prior declarations.

The intent is that specification expressions which are constant expressions can be evaluated when seen.

ANSWER:

No, the program is not conforming. The use of an implicitly typed variable of derived type in a specification inquiry prior to the specification of the derived type was overlooked. An edit is provided to correct this oversight.

EDIT to 18-007r1:

```
[158:4] 10.1.11 Specification expression p6
  Insert at the end of the paragraph:
    "If a specification inquiry depends on the type of an object of
     derived type, that type shall be previously defined."
{It is only possible for the type of a variable not to be previously
 defined if it is typed by the implicit typing rules, but a pointer
 component can refer a type before its definition.}
```

SUBMITTED BY: Jon Steidel

```
HISTORY: 21-146   m224  Submitted
         21-146r1 m224  Revised edit, passed by J3 meeting 224.
         21-184   m225  Passed by J3 letter ballot #37
```

No WG5 comments

NUMBER: F18/029
TITLE: Type of main argument of CO_REDUCE
KEYWORDS: CO_REDUCE
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

The arguments of OPERATION are required to be scalar data objects that are nonallocatable, nonpointer, and nonpolymorphic. Was it intended to allow the type to have components that are allocatable, pointer, or polymorphic?

ANSWER:

No. The restrictions were imposed to allow an implementation to involve an image copying a scalar value to another image as for intrinsic assignment to a scalar coarray. This would not be possible if the type were to have an ultimate component that is allocatable, a pointer, or polymorphic. It was intended for this to be forbidden. An edit is provided to correct this oversight. Forbidding allocatable and pointer ultimate components is sufficient to forbid polymorphic ultimate components too.

EDIT to 18-007r1:

[356:42] In 16.9.49 CO_REDUCE, para 3, after first sentence, add
    "It shall not be of a type with an ultimate component that is
    allocatable or a pointer".

SUBMITTED BY: John Reid

```
HISTORY: 21-137   m224  Submitted
         21-137   m224  Passed by J3 meeting 224, 8-6.
         21-184   m225  Passed by J3 letter ballot #37
```

No WG5 comments

NUMBER: F18/030
TITLE: CO_REDUCE/REDUCE OPERATION with coarray argument
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the function

```
  Pure Real Function f(a,b) Result(r)
    Real,Intent(In) :: a[*],b[*]
    r = a[1]*b[1]
  End Function
```

This function appears to satisfy all the requirements on the OPERATION
argument to CO_REDUCE.

Q1. Was this intended to be a valid operation for CO_REDUCE?

Q2. Was this intended to be a valid operation for REDUCE?

ANSWER:

A1. No, this was not intended to be valid; the arguments of OPERATION
    should not have been permitted to be coarrays.

A2. Likewise, this was not intended to be valid.

Edits are supplied to correct this oversight.

EDIT to 18-007r1:

```
[357:9] 16.9.49 CO_REDUCE, p3 Arguments, OPERATION argument,
        After "nonallocatable," insert "noncoarray,".
        That makes the first sentence of the argument read:
"OPERATION shall be a pure function with exactly two arguments; the
           result and each argument shall be a scalar, nonallocatable,
           noncoarray, nonpointer, nonpolymorphic data object with the
           same type and type parameters as A."
{The "noncoarray" requirement is superfluous for the result,
 but that is not harmful.}

[408:36] 16.9.161 REDUCE, p3 Arguments, OPERATION argument,
         Before "nonpointer," insert "noncoarray,".
         That makes the first sentence of the argument read:
  "OPERATION shall be a pure function with exactly two arguments; each
             argument shall be a scalar, nonallocatable, noncoarray,
             nonpointer, nonpolymorphic, nonoptional dummy data object
             with the same type and type parameters as ARRAY."
{It is "interesting" that these two very similar requirements are
 being expressed differently. Perhaps there is a good reason.}

SUBMITTED BY: Malcolm Cohen

HISTORY: 21-150   m224  Submitted, passed by J3 meeting 224.
         21-184   m225  Passed by J3 letter ballot #37

No WG5 comments
----------------------------------------------------------------------

----------------------------------------------------------------------

NUMBER: F18/031
TITLE: CO_BROADCAST with polymorphic argument
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the code fragment

    Subroutine asgn_for_image(a,b,srcimg)
      Class(*) a,b
      Integer,Intent(In) :: srcimg
      ! The assignment A = B is not allowed, so...
      If (This_Image()==srcimg) Then
        Call asgn(b)
      Else
        Call asgn(a)
      End If
    Contains
      Subroutine asgn(x)
        Class(*) x
        Call co_broadcast(x,Source_Image=srcimg)
      End Subroutine
    End Subroutine

There is no requirement forbidding the A argument of CO_BROADCAST
from being polymorphic, so on the face of it, this appears to get
around the prohibition against nonallocatable polymorphic
assignment.

However, CO_BROADCAST states
    "A becomes defined, as if by intrinsic assignment"
but intrinsic assignment is not defined when the variable is a
nonallocatable polymorphic. It can be convincingly argued that
the standard therefore does not establish an interpretation, and
```

thus the call to CO_BROADCAST is not valid.

Philosophically, it would seem to be strange to allow polymorphic
broadcast across images, but not to allow polymorphic assignment
of a single variable within an image let alone across images.

Is the call to CO_BROADCAST in the example standard-conforming?

(And if so, what are the actual semantics?)

ANSWER:

No, this was not intended to be conforming. Edits are supplied to
correct this.

EDITS to 18-007r1:

[355:19] 16.9.46 CO_BROADCAST, p3 Arguments, argument A, sentence 1,
delete "dynamic" to make the sentence read
    "A shall have the same shape, type, and type parameter values, in
    corresponding references."
{The word "dynamic" would be confusing here.}

[355:20] 16.9.46 CO_BROADCAST, p3 Arguments, argument A, sentence 2,
after "It shall not be" insert "polymorphic or" making the sentence read
    "It shall not be polymorphic or a coindexed object."

SUBMITTED BY: Malcolm Cohen

```
HISTORY: 21-151   m224  Submitted
         21-151r1 m224  Eliminated alternative answer,
                        passed by J3 meeting 224.
         21-184r1 m225  Failed J3 letter ballot.
         22-100   m226  Revised
         22-100r1 m226  Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.
```

No WG5 comments

NUMBER: F18/032
TITLE: CO_BROADCAST and pointer components
DEFECT TYPE: Clarification
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the following program fragment:

```
    TYPE t
        CHARACTER(100) name
        REAL location(3)
        TYPE(t),POINTER :: antecedent
    END TYPE
    TYPE(t) x
    INTEGER src_image
    ...
    CALL co_broadcast(x,src_image)
```

On every image other than src_image itself, this will leave x with its antecedent component having an undefined pointer association status (except when it is disassociated on src_image).
This would seem to be not useful and likely to lead to further errors.

Was it intended to permit the A argument of CO_BROADCAST to have an ultimate pointer component?

ANSWER:

Yes, this was intended. Otherwise, individual CO_BROADCAST executions would be required for every other component; this would be very inconvenient.

EDIT to 18-007r1:

None.

SUBMITTED BY: Malcolm Cohen

```
HISTORY: 21-167   m224  Submitted
         21-167r1 m224  Selected alternative answer,
                        passed by J3 meeting 224.
         21-184   m225  Passed by J3 letter ballot #37
```

No WG5 comments

NUMBER: F18/033
TITLE: E/EN/ES/D output exponent when w=0
KEYWORDS: I/O, E_format
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the following program:

```
integer, parameter :: DP = selected_real_kind (10,300)
integer, parameter :: QP = selected_real_kind (10,3000)
real(DP) :: A
real(QP) :: B

A = 0.12345E123_DP
write (*,'(E0.7)') A ! (1)

B = 0.12345E1234_QP
write (*,'(E0.7)') B ! (2)
```

end

13.7.2.1 p1(6) says:

"On output, with I, B, O, Z, D, E, EN, ES, EX, F, and G editing, the specified value of the field width w may be zero. In such cases, the processor selects the smallest positive actual field width that does not result in a field filled with asterisks."

and (5) above it says:

"On output, if an exponent exceeds its specified or implied width using the E, EN, ES, EX, D, or G edit descriptor, or the number of characters produced exceeds the field width, the processor shall fill the entire field of width w with asterisks. However, the processor shall not produce asterisks if the field width is not exceeded when optional characters are omitted."

If we then look at 13.7.2.3.3 (E and D editing), table 13.1 (Exponent forms) says that for the Ew.d form and where the absolute value of the exponent is greater than 99 but less than or equal to 999, the form of the exponent omits the exponent letter.

Q1: According to the text of the standard, the exponent form for the output at (1) must be "+123", omitting the exponent letter. Was this intended? Isn't the whole point of w=0 to produce minimal width but complete values? Note that the exponent letter is not an "optional character".

Q2: What should the output at (2) be? It would seem that the standard does not provide an interpretation that results in anything but the whole field being filled with asterisks, since the exponent overflows three digits, yet w=0 disallows that.

ANSWER:

A1: No, this was not intended. E0.d should behave as if it were E0.dE0 (similarly for EN and ES), where the exponent letter is present and there are the minimum number of digits needed to represent the exponent. There is no D0.dE0 form, however, so it needs to be a special case in the standard.

A2: The current text provides no interpretation. The proposed change to behave as E0.dE0 provides a reasonable interpretation.

Note that G0.d does not have this problem, as the exponent form is specified as a "reasonable processor-dependent value ... of e". The EX descriptor also does not have this problem.

EDITS to 18-007r1:

13.7.2.3.3 (E and D editing)

264:Table 13.1 (E and D Exponent forms)

Row 1: Add after "Ew.d": " with w > 0"

```
Row 3: Add after "Ew.dE0": " or E0.d"
Row 4: Add after "Dw.d": " with w > 0"
Add new Row 5:
  Column 1: "D0.d"
  Column 2: "any"
  Column 3: "D\pmz1z2 . . . zs or E\pmz1z2 . . . zs"

13.7.2.3.4 (EN editing)

265:Table 13.2 (EN Exponent forms)

Row 1: Add after "ENw.d": " with w > 0"
Row 3: Add after "ENw.dE0": " or EN0.d"

13.7.2.3.5 (ES editing)

266:Table 13.3 (ES Exponent forms)

Row 1: Add after "ESw.d": " with w > 0"
Row 3: Add after "ESw.dE0": " or ES0.d"

SUBMITTED BY: Steve Lionel

HISTORY: 21-172    m225  Submitted
         21-172r1  m225  Revised, passed by J3 meeting 225.
         21-130    m226  Passed by J3 letter ballot #38

No WG5 comments

----------------------------------------------------------------------

----------------------------------------------------------------------

NUMBER: F18/034
TITLE: Purity of IEEE_GET_FLAG and IEEE_GET_HALTING_MODE
KEYWORDS: IEEE_GET_FLAG, IEEE_GET_HALTING_MODE, PURE
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

17.11.1 General (in 17.11 Specifications of the procedures) states
    "all the subroutines are impure unless otherwise stated".

Table 17.3 in 17.10 Summary of the procedures classes these as "ES",
where "ES indicates that the procedure is an elemental subroutine".

Since Fortran 2008, being elemental has no bearing on whether a
procedure is pure or impure. If it is declared with the ELEMENTAL
keyword it is pure by default, that is, if the IMPURE keyword does not
appear; this however only applies to elemental procedures defined by
subprograms, not ones defined by standard intrinsic modules.

Looking at 17.11.5 and 17.11.6, the standard merely has
    "Class: Elemental subroutine"
which again, does not indicate purity or impurity.
```

The lack of a statement to the contrary means that the specification
in 17.11.1 is operative, and therefore IEEE_GET_FLAG and
IEEE_GET_HALTING_MODE must be impure.

However, these were considered to be pure in Fortran 2003, as Fortran
2003 had no concept of an impure elemental procedure - it only had
pure elemental procedures.

Further evidence that the current situation might be a mistake is that
the non-elemental subroutines IEEE_SET_FLAG and IEEE_SET_HALTING_MODE
are explicitly specified to be pure.

Are IEEE_GET_FLAG and IEEE_GET_HALTING_MODE intended to be pure?

ANSWER:

Yes, these subroutines were intended to be pure.
Edits are provided to correct this mistake.

EDITS to 18-007r1:

```
[440:8] 17.10 Summary of the procedures, p3, line "ES indicates...",
        "an elemental subroutine" -> "a pure elemental subroutine".

[443:23] 17.11.5 IEEE_GET_FLAG (FLAG, FLAG_VALUE), para 2 Class,
         "Elemental" -> "Pure elemental".

[443:34] 17.11.6 IEEE_GET_HALTING_MODE (FLAG, HALTING), para 2 Class,
         "Elemental" -> "Pure elemental".
```

SUBMITTED BY: Malcolm Cohen

```
HISTORY: 21-202    m225  Submitted, passed by J3 meeting 225.
         21-130    m226  Passed by J3 letter ballot #38
```

No WG5 comments

NUMBER: F18/035
TITLE: Defining or referencing a coarray component of a dummy argument
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION 1:

On page 311 of 18-007r1, in 15.5.2.13, we have
"While an entity is associated with a dummy argument, the following
restrictions hold.

```
      ...
(3) Action that affects the value of the entity or any subobject of it
    shall be taken only through the dummy argument unless
     ...
   (d) the dummy argument is a coarray and the action is a coindexed
       definition of the corresponding ultimate argument coarray by a
       different image."

Should there be a similar exception for a coarray that is an ultimate
component of a dummy argument?

QUESTION 2:

On page 312 of 18-007r1, in 15.5.2.13, we have
"While an entity is associated with a dummy argument, the following
restrictions hold.
      ...
(4) If the value of the entity or any subobject of it is affected
    through the dummy argument, then at any time during the invocation
    and execution of the procedure, either before or after the
    definition, it shall be referenced only through that dummy argument
    unless
     ...
   (d) the dummy argument is a coarray and the reference is a coindexed
       reference of its corresponding ultimate argument coarray by a
       different image.

Should there be a similar exception for a coarray that is an ultimate
component of a dummy argument?

ANSWERS:

For both questions, the answer is "yes". It was intended that a
subobject of a coarray that is an ultimate component of a dummy
argument may be referenced or defined on another image by coindexing
the corresponding coarray subobject of the actual argument.

Edits are provided.

EDITS to 18-007r1:

[xiv] Introduction, Program units and procedures, last sentence,
      Insert ", or a coarray ultimate component of a dummy argument,"
      After "argument", making that sentence read
  "A coarray dummy argument, or a coarray ultimate component of a
   dummy argument, can be referenced or defined by another image."
{Add to new feature list since Fortran 2008.}

[311:44-46] In 15.5.2.13 Restrictions on entities associated with dummy
arguments, para 1,
at the end of (3)(c) delete "or", and
at the end of (3)(d) replace "image." by
"image, or
(e) the dummy argument has a coarray ultimate component and the action
    is a coindexed definition of the corresponding coarray by a
    different image."

[312:9-11] In 15.5.2.13 Restrictions on entities associated with dummy
arguments, para 1,
at the end of (4)(c) delete "or", and
at the end of (4)(d) replace "image." by
"image, or
(e) the dummy argument has a coarray ultimate component and the
```

```
    reference is a coindexed reference of the corresponding coarray
    by a different image."

[314:1-]. At the end of 15.5.2.13 Restrictions on entities associated
with dummy arguments, NOTE 5, sentence 1, replace "exception" by
"exceptions" and "coarrays enables" by "arguments that are coarrays or
have coarray ultimate components enable" so that the sentence reads
"The exceptions to the aliasing restrictions for dummy arguments that
are coarrays or have coarray ultimate components enable cross-image
access while the procedure is executing."


SUBMITTED BY: John Reid

HISTORY: 22-107   m226 Submitted
         22-107r1 m226 Revised. Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.

No WG5 comments

----------------------------------------------------------------------

----------------------------------------------------------------------

NUMBER: F18/036
TITLE:  Array element argument for sequence association
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Paragraph 14 of 15.5.2.4 Ordinary dummy variables states

If the actual argument is a noncoindexed scalar, the corresponding
dummy argument shall be scalar unless
 - the actual argument is default character, of type character with
   the C character kind (18.2.2), or is an element or substring of an
   element of an array that is not an assumed-shape, pointer, or
   polymorphic array,
 - the dummy argument has assumed-rank, or
 - the dummy argument is an assumed-type assumed-size array.

15.5.2.11 Sequence association, paragraphs 2 and 3, say

   "An actual argument represents an element sequence if it is ... an
    array element designator...
    If the dummy argument is not of type character with default or C
    character kind, and the actual argument is an array element
    designator, the element sequence consists of that array element
    and each element that follows it in array element order.

    If the dummy argument is of type character with default or C
    character kind, and has nonzero character length, the storage unit
    sequence is as follows:
    ...
    - if the actual argument is an array element or array element
      substring designator, the storage units starting from the first
```

      storage unit of the designator and continuing to the end of the
      array;"

Consider
```
    SUBROUTINE sub2
      CHARACTER(10),TARGET :: a2(20,30)
      CHARACTER(:),POINTER :: p2(:,:)
      p2 => a2(::2,::3)(3:4)
      CALL bad2(p2(1,1))
    END SUBROUTINE
    ! There is no explicit interface here.
    SUBROUTINE bad2(b2)
      CHARACTER b2(*)
      ...
    END SUBROUTINE
```

15.5.2.11 says that the storage units of b2 are the storage units of p2 starting from the array element - in this case p2(1,1) - going on to the end of the array (p2); these are
```
    a2(1,1)(3:3), a2(1,1)(4:4), a2(3,1)(3:3), a2(3,1)(4:4), ...
```
which are discontiguous in storage. But as an assumed-size array, b2 is supposed to have contiguous storage units.

If there were an explicit interface, the compiler could gather all the elements of p2 into a temporary, and pass that, similarly to how it handles passing p2 itself as an actual argument to an "old style" dummy array. But there is not, and it would be completely unreasonable to copy the whole of the rest of p2 starting from some arbitrary array element into a temporary, when the dummy argument might be scalar.

Q1. Was sequence association for default/C character to the storage units of the whole of the rest of the array intended to apply to non-contiguous arrays, or was the sequence association intended to apply only to the storage units of the array element itself?

Further consider

```
    SUBROUTINE sub1
      REAL,TARGET :: a(10,20,30)
      REAL,POINTER :: p(:,:,:)
      p => a(::2,::4,::3)
      CALL bad1(p(1,1,1))
    END SUBROUTINE
    SUBROUTINE bad1(b)
      TYPE(*) b(*)
      ...
    END SUBROUTINE
```

15.5.2.11 says that the element sequence of the actual argument is the element sequence of the whole array p, starting from the specified array element - in this case p(1,1,1), i.e.
```
    p(1,1,1), p(2,1,1),...
```
But these correspond to the discontiguous array elements
```
     A(1,1,1), A(3,1,1),...
```

One cannot do anything much with an assumed-type variable other than pass it to C, but a C routine is allowed to access all the elements of

```
an array so passed. That would be difficult in this case unless the
Fortran processor makes a copy of the entire rest of P and pass that
as an argument instead (and copy-back on return if not INTENT(IN)).

Q2. Was sequence association for assumed type to the element sequence
    of the whole of the rest of the array intended to apply to
    non-contiguous arrays, or was the sequence association intended to
    apply only to the array element itself?

ANSWER:

A1. When the array element or array element substring designator is of
    a potentially discontiguous array is passed, only the storage
    units of that array element or substring are intended to be
    passed.

An edit is provided to correct this error.

A2. When an array element of a potentially discontiguous array is
    passed to an assumed-type assumed-size dummy argument, only that
    element is intended to be passed.

An edit is provided to correct this error.

EDITS to 18-007r1:

[310:14] 15.5.2.11 Sequence association, p2,
    After the first sentence, ending "C character kind (18.2.2).",
    insert a paragraph break.
    After that (in was-end-of-p2 now-new-p3), factor out
        "If the dummy argument is not of type character with default
         or C character kind,"
    changing the comma to a colon, and turn those two sentences into
    a bullet list with semicolons.
    After "and the actual argument is an array element designator"
    insert "of a simply contiguous array",
{Avoid the hostages to fortune of listing the allowed ones or the
 disallowed ones, in favour of what property we want the array to
 have.}
    At the end of (was-p2 now-new-p3) append bullet
        "otherwise, if the actual argument is scalar, the element
         sequence consists of that scalar".

This makes the new p3 read
   "If the dummy argument is not of type character with default or C
    character kind:
    - if the actual argument is an array expression, the element
      sequence consists of the elements in array element order;
    - if the actual argument is an array element designator of a
      simply contiguous array, the element sequence consists of that
      array element and each element that follows it in array element
      order;
    - otherwise, if the actual argument is scalar, the element
      sequence consists of that scalar."

[310:19-21] Same subclause, next paragraph (was p3),
    After "substring designator"
    insert "of a simply contiguous array".
    Change the last "if the actual" to "otherwise, if the actual",
    and delete "and not an array ... designator".

This makes the old p3 read
   "If the dummy argument is of type character with default or C
    character kind, and has nonzero character length, the storage unit
    sequence is as follows:
    - if the actual argument is an array expression, the storage units
      of the array;
    - if the actual argument is an array element or array element
      substring designator of a simply contiguous array, the storage
      units starting from the first storage unit of the designator and
      continuing to the end of the array;
    - otherwise, if the actual argument is scalar, the storage units
      of the scalar object."

SUBMITTED BY: John Reid

HISTORY: 22-105   m226  Submitted
         22-136   m226  Revised. Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.

No WG5 comments

----------------------------------------------------------------------

----------------------------------------------------------------------

NUMBER: F18/037
TITLE: Locality spec limitations
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

BACKGROUND

In 11.1.7.2 Form of the DO construct, C1128 states
  "A <variable-name> that appears in a LOCAL or LOCAL_INIT
   <locality-spec> shall not have the ALLOCATABLE, INTENT (IN), or
   OPTIONAL attribute, shall not be of finalizable type, shall not be
   a nonpointer polymorphic dummy argument, and shall not be a coarray
   or an assumed-size array."

QUESTION:

Q1. Was it intended to allow a variable with an ultimate component of
    finalizable type here?

Q2. Was it intended to allow a variable with a coarray ultimate
    component here?

Q3. Was it intended to allow a variable with an allocatable ultimate
    component here?

ANSWER:

A variable with an ultimate allocatable component was not intended to
```

be allowed here. All the questions above involve such a component.

An edit is provided to correct this oversight.

EDIT to 18-007r1:

```
[181:22-24] 11.1.7.2 Form of the DO construct, C1128, first sentence,
    After "of finalizable type,"
    insert "shall not have an ultimate allocatable component,"
    making the whole constraint read

  "C1128 A variable-name that appears in a LOCAL or LOCAL_INIT
         locality-spec shall not have the ALLOCATABLE, INTENT (IN), or
         OPTIONAL attribute, shall not be of finalizable type, shall
         not have an allocatable ultimate component, shall not be a
         nonpointer polymorphic dummy argument, and shall not be a
         coarray or an assumed-size array. A variable-name that is not
         permitted to appear in a variable definition context shall
         not appear in a LOCAL or LOCAL_INIT locality-spec."
```

SUBMITTED BY: John Reid

```
HISTORY: 22-109   m226  Submitted
         22-109r1 m226  Revised
         22-109r2 m226  Further revised. Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.
```

No WG5 comments

NUMBER: F18/038
TITLE: SIZE= with no reason
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Fortran 2018 permits SIZE= in an input statement without ADVANCE=.
SIZE= assigns to its variable the number of character transferred by
edit descriptors.

This means that SIZE= is also allowed for list-directed and namelist
input/output, but no edit descriptor is involved, so this will always
assign the value zero to the variable.

For example,

```
    READ(*,*,SIZE=N) X
    PRINT *,N ! Always prints zero.
```

Was this superfluous additional way of assigning zero to an integer
variable deliberately added?

ANSWER:

```
No, this was not a deliberate addition.
An edit is provided to remove this inadvertent feature.

EDIT to 18-007r1:

[225:29+] 12.6.2.1 Syntax, in 12.6.2 Control information list,
    After constraint C1213 that begins
        "(R1213) A BLANK=, PAD=, END=, EOR=, or SIZE= specifier..."
    insert new constraint
        "C1213a A SIZE= specifier shall not appear in a list-directed
                or namelist input statement."

SUBMITTED BY: Malcolm Cohen

HISTORY: 22-137   m226  Submitted
         22-137r1 m226  Passed by J3 meeting
         22-150   m227  Passed by J3 letter ballot #39.

No WG5 comments

----------------------------------------------------------------------

----------------------------------------------------------------------

NUMBER: F18/039
TITLE:  Corresponding coarrays in recursive procedures
KEYWORDS: coarray, recursive, allocatable
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the following program:

program recu_caf_1
    call csub( 1 )
contains
    recursive subroutine csub ( depth )
        integer :: depth
        real, allocatable :: x[:]
        integer :: k

        if (this_image() == depth) then
            allocate(x[*], source=real(depth))
        else if (depth < num_images()) then
           call csub ( depth + 1 )
        end if
        if (allocated(x)) then
           write(*, *) "image: ", this_image(), "depth: ", depth, &
                    "x: ", (x[k],k = 1, num_images())
        else
           write(*, *) "image: ", this_image(), "depth: ", depth, &
                    "x not allocated"
        end if
    end subroutine
end program
```

```
Q1: Is program recu_caf_1 standard-conforming?
    It establishes a single coarray on each image, but at different
    recursion depths on each image, so the question is really whether
    these correspond (each recursion level has its own set of unsaved
    local variables).

Q2: If the SAVE attribute is added to the declaration of coarray "x"
    in the example, does that make the program standard-conforming?
    (There is only one saved variable, shared with each recursion
    level.)

ANSWER:

A1: No, unsaved local variables are different at each level, and it
    was intended that coarrays at the same level correspond.
    An edit is provided to clarify this.

A2: Yes, the modified program is standard-conforming.

EDITS to 18-007r1:

[41:25] 5.4.7 Coarray, paragraph 2, append new sentence
   "If a coarray is an unsaved local variable of a recursive
    procedure, its corresponding coarrays are the ones at the same
    depth of recursion on each image."

[134:17] 9.7.1.2 Execution of an ALLOCATE statement, paragraph 3,
         append new sentence
"If the coarray is an unsaved local variable of a recursive procedure,
 the execution of the ALLOCATE statement shall be at the same depth of
 recursion on every active image in the current team."

SUBMITTED BY: John Reid and Reinhold Bader

HISTORY: 22-113   m226  Submitted
         22-113r1 m226  Revised. Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.

Comment:
F18/039 and F18/040
Iwashita:
I don't think "depth of recursion" is clear. Is it the number of
invocations of the recursive procedure in question, or the total
number of invocations of all recursive procedures?
For example, is the following program conforming?


program rec_wrap
>    if (this_image() =3D=3D 1) then
>       call wrapper( 3 )
>    else
>       call csub( 3 )
>    end if
>
> contains
>    recursive subroutine wrapper( n )
>        integer :: n
```

```
>
>        call csub( n )
>    end subroutine wrapper
>
>    recursive subroutine csub ( n )
>        integer :: n
>        real, allocatable :: x(:)[:]
>
>        if ( n =3D=3D 0 ) then
>           return
>        end if
>
>        allocate( x(n*100)[*] )
>
>        if (this_image() =3D=3D 1) then
>           call wrapper( n - 1 )
>        else
>           call csub( n - 1 )
>        end if
>
>    end subroutine
> end program
>
```

NUMBER: F18/040
TITLE: Allocating a dummy argument with a coarray ultimate component
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

On page 134 of 18-007r1, 9.7.1.2, para 3, we have
"If an allocation specifies a coarray, its dynamic type and the values of corresponding type parameters shall be the same on every active image in the current team. The values of corresponding bounds and corresponding cobounds shall be the same on those images. If the coarray is a dummy argument, its ultimate argument (15.5.2.3) shall be the same coarray on those images."
Should there be a similar restriction for a coarray that is an ultimate component of a dummy argument? For example, does the following program conform to the standard?

```
program test
   type new
      real, allocatable :: a[:]
   end type
   integer :: i
   type(new) x,y
   if(this_image()<=2) then
      call work(x)
   else
```

```
      call work(y)
   end if
   sync all
   if (this_image()==2)then
      do i = 1, num_images()
         x%a[i] = i
      end do
   end if
   sync all
   if (this_image()==4)then
      do i = 1, num_images()
         write(*,*) i, y%a[i]
      end do
   end if
contains
   subroutine work(z)
      type(new) :: z
      allocate (z%a[*])
   end subroutine
end program
```

Here, the calls of subroutine work create a coarray that is accessible on images 1 and 2 as x%a[i] and on other images as y%a[i]. Was this intended?

ANSWER:

A similar restriction was intended. It was not intended to allow the creation of a coarray that is accessible in a scope as an ultimate component of one object on some images and as an ultimate component of another object on other images.

An edit is provided. We have taken the opportunity to change "same" in the quoted text to "corresponding". Coarrays on different images cannot be the same, but they can correspond, see 5.4.7. We need the concept of "same" for objects of a type with coarray ultimate components. It seems appropriate to require that they be declared with the same name in the same set of statements.

EDIT to 18-007r1:

[134:16-17] In 9.7.1.2 Execution of an ALLOCATE statement, para 3, replace the final sentence by the two sentences
"If the coarray is a dummy argument, the ultimate arguments (15.5.2.3) on those images shall be corresponding coarrays. If the coarray is an ultimate component of a dummy argument, the ultimate arguments on those images shall be declared with the same name in the same scoping unit and if in a recursive procedure at the same depth of recursion."

SUBMITTED BY: John Reid

```
HISTORY: 22-110   m226 Submitted. Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.
```

See comment at F18/038

```
----------------------------------------------------------------------

NUMBER: F18/041
TITLE: NULL() passed to assumed-rank dummy
KEYWORDS: NULL, assumed-rank
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider:

  call foo(null())
contains
  subroutine foo(x)
    integer, pointer, intent(in) :: x(..)
    print *, rank(x)
  end subroutine
end

What should be printed? According to Table 16.5 (Characteristics of
the result of NULL()), the actual argument has the rank of "the
corresponding dummy argument". In this case, however, the corresponding
dummy has no defined rank, instead taking its rank from the actual
argument. Was this intended to be undefined?

ANSWER:

No, this combination was intended to be non-conforming.
An edit is provided to correct this mistake.

EDITS to 18-007r1:

[400:33] 16.9.144 NULL, p6 "If the context...",
         Add a new sentence to the end of the paragraph:
   "If the context of the reference to NULL is an <actual argument>
    corresponding to an <assumed-rank> dummy argument, MOLD shall be
    present."
{Add restriction. It could also be appended to p5, or be a new para.}

SUBMITTED BY: Steve Lionel

HISTORY: 22-146   m226  F18/041 Submitted. Passed by J3 meeting 226.
         22-150   m227  Passed by J3 letter ballot #39.

No WG5 comments
```