ISO/IEC JTC 1/SC 22/WG 5 "Fortran"
Convenorship: **ANSI**
Convenor: **Lionel Steve Mr**

WG5 letter ballot 1 on Fortran 2023 interpretations

Document type	Related content	Document date	Expected action
General / Other		2025-06-14	

To: J3 J3/25-134
From: Malcolm Cohen
Subject: WG5 letter ballot 1 on Fortran 2023 interpretations
Date: 2025-May-16

This J3 paper will be promoted to be a WG5 paper in due course.

This is the first set of draft interpretations for Fortran 2023.
They have all been approved in a J3 letter ballot.

The rules we operate on say:

4. The chair of J3/interp gathers all interp answers that are marked
   "passed by J3 letter ballot" and forwards them to the WG5 convenor.
   The WG5 convenor holds a ballot of individual members; a no vote
   must be accompanied by an explanation of the changes necessary to
   change the member's vote to yes. The answers that pass this ballot
   become "WG5 approved".

   J3/interp reserves the right to recall an interp answer for more
   study even if the answer passes.

5. "WG5 approved" answers are processed into a corrigendum document by
   taking the edits from the interp answers and putting them in the
   format required by ISO. A WG5 vote is made on forwarding the
   corrigendum to SC22.

The following Fortran 2023 interpretations are being balloted:

Yes	No	Number	Title
---	---	F23/003	Conflicting rules for COMMON block names
---	---	F23/004	OUT_OF_RANGE and ROUND argument
---	---	F23/005	Defined assignment/operators and dynamic type
---	---	F23/006	Underflow in IEEE_SCALB
---	---	F23/008	Real argument I in IEEE_SCALB
---	---	F23/009	Coarray subobject of component
---	---	F23/010	MOVE_ALLOC with coarray arguments
---	---	F23/011	NULL and procedure pointers
---	---	F23/012	Coarray correspondence in DEALLOCATE
---	---	F23/013	BOZ literals in interoperable enumerators
---	---	F23/015	Coindexed objects in structure constructors
---	---	F23/016	Segments associated with allocation
---	---	F23/017	CFI_establish nonalloc nonpointer null base address
---	---	F23/018	Correspondence of unallocated coarrays

The text of these interpretations appears below.
Each interpretation starts with a row of "-"s.

Please mark the above -Y- in the Yes column for "yes", -C- in the Yes column for "yes with comment", or -N- in the No column for a "no" answer {be sure to include your reasons with "no"} and send ONLY the vote section to

sc22wg5@open-std.org

by 23:59 UTC on June 15, 2025, in order to be counted.

Thanks,
Malcolm on behalf of Steve Lionel

NUMBER: F23/003
TITLE: Conflicting rules for COMMON block names
KEYWORDS: COMMON Block, Named Constant, USE association renaming
DEFECT TYPE: Clarification/Erratum
STATUS: Passed by J3 letter ballot
REFERENCES: N2209

QUESTION:

A survey of five compilers finds that three disallow a named constant from having the same name as a COMMON block in a give scope, while two compilers permit named constants sharing the same name as a COMMON block accessible in the same scope.

19.3.1 Classes of local identifiers, paragraph 1 establishes identifiers of names constants to be class (1) identifiers. Paragraph 2 states:

"Within a scope, a local identifier of an entity of class (1) or class(4) shall not be the same as a global identifier used in that scope unless the global identifier
   o ...
   o is a common block name
   o ..."

However, 19.3.2 Local identifiers that are the same as common block names states:

"A name that identifies a common block in a scoping unit shall not be used to identify a constant or an intrinsic procedure in that scoping unit. ..."

which disallows a named constant from having the same name as an accessible common block.

14.2.2 The USE statement and use association contains Note 4 which states

"The constraints in 8.10.1, 8.10.2, and 8.9 prohibit the local-name from appearing in a COMMON statement, and equivalence-object in an EQUIVALENCE statement, or a namelist-group-name in a NAMELIST statement, respectively. There is no prohibition against the local-name appearing as a common-block-name or a namelist-group-object."

The last sentence of this note contradicts the restrictions in 19.3.2.

Q. Is the intent to disallow a local identifier that identifies a named constant from being the same as an accessible common block?

ANSWER:

Yes, the local identifier of a named constant cannot be the same as that of an accessible common block. This issue goes back to FORTRAN 77 when the PARAMETER statement was introduced. When Fortran 90 introduced MODULEs and USE association, the restriction on common block names was overlooked when the note was written.

Clarifying edits are provided.

EDITS to N2209:

[299] 14.2.2 The USE statement and use association, NOTE 4, sentence 2
  delete
    "a common-block-name or"

```
  add at the end of the note
    "Restrictions on local-name being the same as a common-block-name
     are detailed in 19.3.2."

[528:15] 19.3.1 Classes of local identifiers, p2,
         after "is a common block name (19.3.2)"
         insert "and the local identifier is not that of a named
                 constant or intrinsic procedure,"

SUBMITTED BY: Jon Steidel

HISTORY: 23-119   m229  Submitted
         23-119r1 m229  Revised edit location/description, approved uc
         25-133   m236  Passed by J3 letter ballot #40

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

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

NUMBER: F23/004
TITLE: OUT_OF_RANGE and ROUND argument
KEYWORDS: OUT_OF_RANGE
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

In the description of the intrinsic function

    OUT_OF_RANGE (X, MOLD [, ROUND])

it states "ROUND shall be present only if X is of type real and
           MOLD is of type integer."

Consider

    Program test
        Call s(1.5,2.5)
    Contains
        Subroutine s(x,y,r)
            Logical,Optional :: r
            Print *,out_of_range(x,y,r) ! Valid?
        End Subroutine
    End Program

As R is an optional dummy argument, it is present only if the actual
argument is present. In this example, R is not present, and so the
requirement in 16.9.157 is satisfied.

Was it intended that this program be valid?
(Of the two compilers that implement OUT_OF_RANGE that I tested,
 both of them rejected the program.)

ANSWER:

No, it was not intended that the example be valid; the requirement
should have stated that the argument shall not appear.

An edit is provided.

EDIT to N2209.

[422] 16.9.157 OUT_OF_RANGE, Arguments paragraph, ROUND argument,
      change "shall be present only" to "shall appear only".
```

SUBMITTED BY: Malcolm Cohen

HISTORY: 23-114   m229 Submitted, approved uc
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/005
TITLE: Defined assignment/operators and dynamic type
KEYWORDS: Defined assignment, defined operations, dynamic type
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

10.2.1.4 Defined assignment statement states that a subroutine
"defines the defined assignment x1 = x2 if
    (1) the subroutine is specified with a SUBROUTINE (15.6.2.3) ...
        statement that specifies two dummy arguments, d1 and d2,
    [and]
    (3) the types of d1 and d2 are compatible with the dynamic types
        of x1 and x2, respectively,"

Furthermore, 10.2.1.2 Intrinsic assignment statement states
        "An intrinsic assignment statement is an assignment statement
         that is not a defined assignment statement (10.2.1.4)."

This means that
(i) in the situation where there is no intrinsic assignment, whether
    there is a defined assignment (and thus the program is valid)
    depends on the dynamic types, not the declared types;
(ii) in the situation where there may be an intrinsic assignment for
     a derived type, whether this is overridden by defined assignment
     again depends on the dynamic types.

Similar wording appears in in 10.1.6 Defined operations, specifically
in 10.1.6.1 Definitions, paragraph 2, item (3), and paragraph 5 item
(3). Similar conclusions follow.

These consequences are contrary to the principle that generic
resolution is determined statically, that is, at compile time.
Generic resolution is indeed done statically for generic names and
generic type-bound procedures. It would be very surprising for it
not to be done statically for generic assignment and operators.

For example, consider

```
    Module c23_005
        Type t
            Integer c
        End Type
        Type,Extends(t) :: t2
            Integer d
        End Type
        Generic :: Assignment(=) => assign_t2_int
    Contains
        Subroutine assign_t2_int(a,b)
            Class(t2),Intent(InOut) :: a
            Integer,Intent(In) :: b
            a%c = b
            a%d = -b
```

```
        End Subroutine
    End Module
    Program test
        Use c23_005
        Class(t),Allocatable :: x
        Allocate(x,Source=t2(1,2))
        x = 12345                   ! (1)
        Print *,x%c
    End Program
```

There is no intrinsic assignment at (1), so the program would be invalid unless generic resolution is done on the dynamic type.

Consider

```
    Module c23_005a
        Type t
            Integer c
        End Type
        Type,Extends(t) :: t2
            Integer d
        End Type
        Generic :: Assignment(=) => assign_t2_t
    Contains
        Subroutine assign_t2_t(a,b)
            Class(t2),Intent(InOut) :: a
            Type(t),Intent(In) :: b
            a%c = b%c**2
            a%d = -b%c**2
        End Subroutine
        Subroutine sho(w)
            Class(t),Intent(In) :: w
            Select Type (w)
            Type Is (t)
                Print *,'t:',w
            Type Is (t2)
                Print *,'t2:',w
            End Select
        End Subroutine
    End Module
    Program test
        Use c23_005a
        Class(t),Allocatable :: x
        Allocate(x,Source=t2(1,2))
        x = t(9)                    ! (2)
        Call sho(x)
    End Program
```

Intrinsic assignment would be available for the assignment at (2), but if generic resolution were done on the dynamic type, the defined assignment would be executed, not the intrinsic assignment. That affects the result - does the program print "t: 9" or "t2: 81 -81".

It does not have to be the left-hand-side that is polymorphic: consider

```
    Module c23_005b
        Type t
            Integer c
        End Type
        Type,Extends(t) :: t2
            Integer d
        End Type
```

```
        Generic :: Assignment(=) => assign_t_t2
    Contains
        Subroutine assign_t_t2(a,b)
            Type(t),Intent(Out) :: a
            Class(t2),Intent(In) :: b
            a%c = b%c**2 - b%d**2
        End Subroutine
    End Module
    Program test
        Use c23_005b
        Class(t),Allocatable :: x
        Type(t) y
        Allocate(x,Source=t2(1,2))
        y = x                       ! (3)
        Print *,y
    End Program
```

Here the question is whether the program prints "1" for generic resolution of the statement at (3) done at compile time (and thus the intrinsic assignment), or "-3" for generic resolution done at execution time (and thus the defined assignment).

For defined operations, consider

```
    Module c23_005c
        Type t
            Integer c
        End Type
        Type,Extends(t) :: t2
            Integer d
        End Type
        Generic :: Operator(+) => t2_plus_int
    Contains
        Type(t2) Function t2_plus_int(a,b) Result(r)
            Class(t2),Intent(In) :: a
            Integer,Intent(In) :: b
            r%c = a%c + b
            r%d = a%d - b
        End Function
    End Module
    Program test
        Use c23_005c
        Class(t),Allocatable :: x
        Allocate(x,Source=t2(1,2))
        Print *,x+10                    ! (4)
    End Program
```

This is valid only if the dynamic type is used to resolve the generic operation at (4) in favour of the defined operation, even though the compiler has no idea what the declared type might be at runtime.

One final consideration is that type compatibility is between entities, not between types. Therefore, the quoted words

> "the types of d1 and d2 are compatible with the dynamic types of x1 and x2"

have no meaning, and thus by subclause 4.2 Conformance, paragraph 1,

> "A program (5.2.2) is a standard-conforming program ... if the program has an interpretation according to this document"

all programs which attempt to use defined assignment or operators are non-conforming.

So the question is, should generic resolution of defined assignment and defined operators follow the dynamic type, as suggested by the current wording?

DISCUSSION:

These words come from paper 02-129r2 at meeting 160, which claimed to be making no technical change. The editor wrote in his report
    "These seem more than editorial. Looks like a couple of dynamic and declared types got switched around for a start. And same type got changed to compatable type. I'll just assume it is all correct."

Unfortunately, no-one followed up on the report.

No compilers have been reported as following the implications of the current wording - they all use the declared types for generic resolution, just like normal type compatibility.

ANSWER:

No, generic resolution of defined assignment and defined operations should follow the rules for type compatibility, which uses the declared type not the dynamic type.

It is noted that the rules for argument association already include the correct wording; 15.5.2.5 Ordinary dummy variables, p2, states
    "The dummy argument shall be type compatible with the actual argument."

Edits are provided to correct this misstatement.

EDITS to N2209.

[161:6] 10.1.6 Defined operations, 10.1.6.1 Definitions, p2, (3),
    Change
        "the type of d2 is compatible with the dynamic type of x2,"
    to
        "d2 is type-compatible with x2,"

[161:23] Same subclause, p5, (3),
    Change "the types of d1 and d2 are compatible with the dynamic
            types of x1 and x2, respectively,"
    to
        "d1 and d2 are type-compatible with x1 and x2, respectively,"

[172:13] 10.2.1.4 Defined assignment statement, p2, (3),
    Change "the types of d1 and d2 are compatible with the dynamic
            types of x1 and x2, respectively,"
    to
        "d1 and d2 are type-compatible with x1 and x2, respectively,"
{Note the lines in NOTE 8 at the top of the page do not count towards the line number in the interpretation document.}

SUBMITTED BY: Malcolm Cohen & Jon Steidel

HISTORY: 23-118   m229 Submitted, approved uc
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/006
TITLE: Underflow in IEEE_SCALB
KEYWORDS: Underflow, IEEE_SCALB
DEFECT TYPE: Erratum

STATUS: Passed by J3 letter ballot

QUESTION:
If X * 2**I is too small to be represented with full accuracy, was it intended that IEEE_SCALB(X,I) should return the representable number having a magnitude nearest to ABS(2**I) and the same sign as X? For example, if X is IEEE binary32 with the value 2E-38, was it intended that IEEE_SCALB(X,-1) should return the value 0.5?

ANSWER:
No, it was intended that it should return the representable number having a magnitude nearest to ABS(X*2**I) and the same sign as X. An edit is supplied.

This error dates back to Fortran 2003. Therefore this is an incompatibility with Fortran 2003, 2008, and 2018. Edits to the compatibility subclause are provided.

EDITS to N2218:

[xiii] Introduction, Intrinsic modules bullet, append sentence
"The result of the function IEEE_SCALB from the intrinsic module IEEE_ARITHMETIC has been corrected to conform to \theIEEEstd."

[33:13+] 4.3.3 Fortran 2018 compatibility, last paragraph, new bullet
"- The result of a reference to the function IEEE_SCALB from the intrinsic module IEEE_ARITHMETIC has been corrected to return the representable number having a magnitude nearest to ABS(X*2**I) with the same sign as X, if X*2**I is too small to be represented with full accuracy."

[34:17+] 4.3.4 Fortran 2008 compatibility, last paragraph, new bullet with text identical to preceding edit.

[35:13+] 4.3.5 Fortran 2003 compatibility, last paragraph, new bullet with text identical to the edit for [33:13+].

[487:15] 17.11.37 IEEE_SCALB, Result Value paragraph, Case (iii), change "|2^I|" to "|X \times 2^I|".

SUBMITTED BY: John Reid

HISTORY: 23-156 m230 Submitted
23-156r1 m230 Revised edits, approved uc
25-133 m236 Passed by J3 letter ballot #40

NUMBER: F23/008
TITLE: Real argument I in IEEE_SCALB
KEYWORDS: Real, IEEE_SCALB
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

The first sentence of 17.1 Overview of IEEE arithmetic support, states
"The intrinsic modules IEEE_EXCEPTIONS, IEEE_ARITHMETIC, and IEEE_FEATURES provide support for the facilities defined by ISO/IEC 60559:2020."
However, nothing claims to support the IEEE function scaleB. This is

very like IEEE_SCALB (X,I) except that the IEEE standard requires
that the second argument to scaleB be the same type as logB, and that
is Real type whereas IEEE_SCALB only accepts Integer type.

Was it intended to support the IEEE scaleB operation?
If so, is that intended to be provided by IEEE_SCALB?

ANSWER:

Yes, the IEEE scaleB operation should have been supported.

Yes, IEEE_SCALB should provide the scaleB operation.

EDITS to N2218:

[xiv] Introduction, bullet "Changes to the intrinsic module
        IEEE_ARITHMETIC for conformance with ISO/IEC 60559:2020",
      append sentence
    "The function IEEE_SCALB from the intrinsic module
     IEEE_ARITHMETIC now performs the scaleB operation."

[470:6-7] 17.9 IEEE arithmetic, p1, bullet list, last item,
    After "logB," insert "scaleB,",
    After "IEEE_LOGB," insert "IEEE_SCALB".

[487:6] 17.11.37 IEEE_SCALB, Arguments, I, change "integer" to
    "integer or of type real with the same kind type parameter as X"
  so that the line reads
    "I shall be of type integer or of type real with the same kind
     type parameter as X.".

[487:9+] Same subclause, Result Value, before "Case (i)", insert
    "The value of the result shall conform to the scaleB operation
     of \theIEEEstd.".

SUBMITTED BY: John Reid

HISTORY: 23-157   m230 Submitted
         23-157r1 m230 Revised
         23-157r2 m230 Revised, passed by J3 meeting
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/009
TITLE: Coarray subobject of component
KEYWORDS: coarray, allocatable, array, component
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

The Introduction of Fortran 2023 says
  "A data object with a coarray component can be an array or
   allocatable."

This appears to be true for named variables, but there is a constraint
that makes it impossible for an array component or an allocatable
component:
  "C753 A data component whose type has a coarray potential subobject
        component shall be a nonpointer nonallocatable scalar and
        shall not be a coarray."

That means that given the type

```
    Type real_coarray
      Real,Allocatable :: c[:]
    End Type
```

the statements

```
    Type(real_coarray) x(100)
    Type(real_coarray),Allocatable :: y
```

are acceptable as type declaration statements, but unacceptable as component definition statements.

Is this irregularity deliberate?

ANSWER:

No, this constraint was accidentally overlooked when extending types with coarray components to be subobjects of arrays and allocatables.

An edit is provided to correct this mistake.

EDIT to N2218 (Fortran 2023 FDIS):

[79:constraint C753] In subclause
  Delete this constraint, which begins
    "C753 A data component whose type has a coarray"

[80:After NOTE 1] Insert new NOTE
    "NOTE 1.5
     A data component whose type has a coarray potential subobject component cannot be a coarray or a pointer, see constraint C825."
{C825 says "An entity whose type has a coarray potential subobject..."
 and components are certainly entities. We specifically wrote "entity"
 instead of "named variable" to cover the component case.
 I prefer to say it once and refer to it.}

SUBMITTED BY: John Reid & Reinhold Bader

HISTORY: 23-210   m231 Submitted
         23-210r1 m231 Revised
         23-210r2 m231 Passed as amended by J3 meeting 231.
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/010
TITLE: MOVE_ALLOC with coarray arguments
KEYWORDS: MOVE_ALLOC, coarray
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

If the FROM and TO arguments to MOVE_ALLOC are coarrays, are corresponding invocations of MOVE_ALLOC required to have their FROM (and TO) arguments be corresponding coarrays?

For example, this program ends up with A and B not having the same allocation status on all images, which surely cannot be right.

```
    program trouble
      real, allocatable :: a[:], b[:]
      allocate(a[*],b[*])
      if (this_image()>1) then
        call sub(a,b)
        ! Now, A is deallocated and B is allocated
      else
        call sub(b,a)
        ! Now, B is deallocated and A is allocated
      end if
    contains
      subroutine sub(s,t)
        real, allocatable :: s[:], t[:]
        call move_alloc(s,t)
      end subroutine
    end program
```

ANSWER:

Yes, those arguments are required to be corresponding coarrays.
An edit is supplied to correct this error.

EDIT to N2218 (Fortran 2023 FDIS):

[423] In 16.9.147 MOVE_ALLOC, Arguments paragraph,
      At the end of the FROM argument description append
        "If it is a coarray, it shall correspond to the FROM arguments
         in all corresponding invocations of MOVE_ALLOC."
      At the end of the TO argument description append
        "If it is a coarray, it shall correspond to the TO arguments
         in all corresponding invocations of MOVE_ALLOC."

SUBMITTED BY: John Reid & Reinhold Bader

HISTORY: 23-170   m230 Submitted
         23-220   m231 Revised
         23-220r1 m231 Revised, passed by J3 meeting 231.
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/011
TITLE: NULL and procedure pointers
KEYWORDS: NULL, procedure pointer
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider the procedure declaration statement

```
      PROCEDURE(), POINTER :: PROCPTR => NULL()
```

The statement appears to violate the requirements of the standard.
The function reference does not match any of the conditions listed in
Table 16.5.

The closest case is "initialization of an object in a declaration",
but a procedure pointer is not an object (see 3.42).

Therefore, a MOLD argument is required.

However, constraint C813 prohibits a MOLD argument from appearing.

The same does not apply to default initialization of procedure pointer components in a derived type definition, as component initialization appears in Table 16.5.

Is this irregularity deliberate?

ANSWER:

No, this is not deliberate. Table 16.5 should be extended to include initialization in a procedure declaration statement.

An edit is provided.

EDIT to N2218 (Fortran 2023 FDIS):

[428] 16.9.155 NULL, Result Characteristics, Table 16.5
In the entry "initialization for an object..."
change "object" to "entity", twice, making the whole entry read
"initialization for an entity in a declaration | the entity".
{Subtle but effective.}

SUBMITTED BY: Robert Corbett & Malcolm Cohen

HISTORY: 23-233 m231 Submitted
23-233r1 m231 Revised edit, passed by J3 meeting 231.
25-133 m236 Passed by J3 letter ballot #40

NUMBER: F23/012
TITLE: Coarray correspondence in DEALLOCATE
KEYWORDS: DEALLOCATE, coarray
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

In 9.7.3.2 Deallocation of allocatable variables, paragraph 11 requires that coarray dummy arguments on the active images have ultimate arguments that are corresponding coarrays.

However, there appears to be no such requirement for coarray components of dummy arguments.

Thus this program appears to be valid (apart from having no interpretation due to coarray allocation status inconsistency):

```
    Program trouble
        Type t
            Real,Allocatable :: c(:)[:]
        End Type
        Type(t) x,y
        Allocate(x(1)[*],y(1)[*])
        If (This_Image()==1) Then
            Call oops(x)
        Else
            Call oops(y)
        End If
        Print *,Allocated(x%c),Allocated(y%c)
```

```
        Sync All
    Contains
        Subroutine oops(z)
            Type(t) :: z
            Deallocate(z%c)
        End Subroutine
    End Program
```

Should there be a requirement on coarray components of dummy arguments in DEALLOCATE?

There is also an editorial glitch in paragraph 10 where it says that a coarray does not "become deallocated on an image unless it is successfully deallocated on all active images", since "it" exists on only one image (the others being the corresponding coarrays), and "all active images" includes the image in question, so is circular. Should this not be corrected?

ANSWER:

Yes, this requirement should be explicit.

Yes, the wording in the last sentence of paragraph 10 is poor, and should be improved.

Edits are supplied to address these defects.

As the question noted, the example is not conforming as it violates the semantics that corresponding coarrays have the same allocation status, bounds, etc. on all images on which they are established.

EDIT to N2218:

[152] 9.7.3.2 Deallocation of allocatable variables, paragraph 10, last sentence,
change "it is" to "the corresponding coarrays are",
and insert "other" between "all" and "active",
making that whole sentence read:
"A coarray shall not become deallocated on an image unless the corresponding coarrays are successfully deallocated on all other active images in this team."
{Clarify "it" and "all".}

[152] Same subclause, paragraph 11 beginning "If an allocate-object is a coarray dummy argument", append new sentence
"If an allocate-object is a coarray subcomponent of a dummy argument, those components of the ultimate arguments on those images shall be corresponding coarrays."
{Requirement needed to maintain coarray correspondence semantics.
A "subcomponent" is a component that is an immediate component, of a component of a subobject.}

SUBMITTED BY: John Reid & Reinhold Bader

HISTORY: 23-218 m231 Submitted
23-243 m231 Revised
23-243r1 m231 Passed by J3 meeting 231.
25-133 m236 Passed by J3 letter ballot #40

NUMBER: F23/013

TITLE: BOZ literals in interoperable enumerators
KEYWORDS: BOZ, enumerators
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

For Fortran 2023, work item US-23 expanded the contexts in which BOZ constants were allowed, to include "as the <initialization> for a named constant of type INTEGER or REAL,..." (C7119). 19-256r1 had the primary edits, with 21-101 containing additional edits not relevant to this paper.

19-256r1 contained examples showing use of BOZ constants in PARAMETER statements, but did not mention interoperable enumerators, which are "named integer constant[s]" (7.6.1p1).

Consider the following:

ENUMERATOR :: FOO = Z'123'

is this conforming in Fortran 2023? Clearly, the intent of US-23 was that it should be, but the syntax rule for <enumerator> is:

R762 enumerator is named-constant [= scalar-int-constant-expr]

Since a BOZ constant has no type (7.7p1), it can't be an integer, and thus isn't a scalar-int-constant-expr. Was this exclusion intended? (Note: PARAMETER does not have this issue.)

ANSWER:

No - it was intended to allow BOZ constants as the value for interoperable enumerators just as they are in the PARAMETER statement. Edits to correct this are provided.

EDITS to 24-007:

[7.6.1, 95:18+ Interoperable enumerations and enum types]

insert after:
R762 enumerator is named-constant [= scalar-int-constant-expr]

"   or <named-constant> [= <boz-literal-constant>]"

(The Editor is welcome to substitute an alternate expression of this definition, such as creating a new term for the initializer.)

[7.6.1p6, 96:5-9 Interoperable enumerations and enum types]

In the numbered list following "The enumerator is a scalar named constant, with the value determined as follows.", make the following changes.

Insert after (1):
"(1a) if boz-literal-constant appears, the enumerator has the value specified by INT(boz-literal-constant, C_INT), where C_INT is from the intrinsic module ISO_C_BINDING."

In the current (2) and (3), replace "If scalar-int-constant-expr does not appear" with "If neither scalar-int-constant-expr nor boz-literal-constant appears" such that the new list items read:

(2a) If neither scalar-int-constant-expr nor boz-literal-constant

appears and the enumerator is the first enumerator in enum-def, the
enumerator has the value zero.
(3a) If neither scalar-int-constant-expr nor boz-literal-constant
appears and the enumerator is not the first enumerator in enum-def,
it has the value obtained by adding one to the value of the
enumerator that immediately precedes it in the enum-def.

[7.7,100:30 Binary, octal, and hexadecimal literal constants]

In C7119, after "variable of type integer or real"

insert:

", as the value in an <enumerator>"

SUBMITTED BY: Steve Lionel

HISTORY: 24-101   m232 Submitted
         24-101r1 m233 Passed by J3 meeting 233.
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/015
TITLE: Coindexed objects in structure constructors
KEYWORDS: Pointer component, coindexed object, structure constructor
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider

```
  TYPE realptrwrap
    REAL,POINTER :: p
  END TYPE
  TYPE t
    TYPE(realptrwrap) c
  END TYPE
  TYPE(realptrwrap) x[*]
  TYPE(t) y
  REAL,TARGET :: z[*]

  x = realptrwrap(z[2])     ! (A) harmful
  y%c = realptrwrap(z[2])   ! (B) harmful
  x = realptrwrap(z)        ! This assignment is valid and harmless.
  y = t(x[2])               ! (C) harmful
```

(Aside: "harmful" means "copies a pointer from one image to another".)

The structure constructors in the statements (A) and (B) are invalid,
by combination of

```
    R758 component-data-source  is  expr
                                or  data-target
                                or  proc-target
    C7109 (R758) A data-target shall correspond to a data pointer
          component; ...
    R1038 data-target  is  expr
    C1029 (R1038) A data-target shall not be a coindexed object.
```

However, the statement (C) has the same undesirable effect as (B), but
does not violate those constraints, and thus appears on the face of it

to be a valid statement.

Statement (C) would, however, be prohibited in a pure procedure, even
though it cannot cause side effects, merely undefined pointers.

This appears to be inconsistent. Is this deliberate?

ANSWER:

This apparent inconsistency was inadvertent.
Edits are provided to correct the issue.

EDITS to 24-007:

[93:21+] 7.5.10 Construction of derived-type values,
         after the penultimate constraint C7109, insert new
         constraint
    "C7109a (R758) If <expr> is a coindexed object, it shall not have
            a pointer component at any level of component selection."

[93:23-] Same subclause, after NOTE 1, insert new NOTE
    "NOTE 1a
     Although a coindexed object with a pointer subcomponent is not
     the only way for the structure constructor to produce a value
     with an undefined pointer subcomponent, copying a pointer from
     another image is particularly likely to cause undiagnosed
     incorrect results, and thus precluded in this context."

[34:1+] 4.3.3 Fortran 2018 compatibility, after paragraph 4,
        insert new paragraph
    "Fortran 2018 permitted a <component-data-source> in a structure
     constructor to be a coindexed object with a pointer subcomponent.
     This document does not permit such usage."

[35:8+] 4.3.4 Fortran 2008 compatibility, after paragraph 14,
        insert new paragraph
    "Fortran 2008 permitted a <component-data-source> in a structure
     constructor to be a coindexed object with a pointer subcomponent.
     This document does not permit such usage."

SUBMITTED BY: Malcolm Cohen

HISTORY: 24-149   m233 Submitted
         24-149r1 m233 Passed by J3 meeting 233.
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/016
TITLE: Segments associated with allocation
KEYWORDS: segment, allocate, coarray
DEFECT TYPE: Clarification
STATUS: Passed by J3 letter ballot

QUESTION:

Is it possible for a coarray to be allocated on an image in the
current team without there being a corresponding allocated coarray on
another active image in the current team?

For example, consider the program

```
PROGRAM ALLOCATION
   IMPLICIT NONE
   INTEGER :: I
   REAL, ALLOCATABLE :: A[:]
   ALLOCATE (A[*])
   A = THIS_IMAGE()
   SYNC ALL
   DO I = 2,THIS_IMAGE()
      IF (A[I]/=I) WRITE(*,*) "Value incorrect on image", I
   END DO
   DEALLOCATE (A)
END
```

Is it possible that an execution of the IF statement fails because the coarray A has already been deallocated on image I?

ANSWER:

No, it is not possible.

For the ALLOCATE statement, the standard says
> "execution on the active images of the segment (11.7.2) following the statement is delayed until all other active images in the current team have executed the same statement the same number of times in this team."

That means that after execution of the ALLOCATE statement on an image, the coarray is allocated on all the images (in the team).

For the DEALLOCATE statement, the standard says
> "When a statement that deallocates a coarray or an object with a coarray potential subobject component is executed, there is an implicit synchronization of all active images in the current team."

It then goes on to say
> "A coarray shall not become deallocated on an image unless it is successfully deallocated on all active images in this team."

That means that the DEALLOCATE statement cannot actually deallocate a coarray on the executing image until all other active images have reached that DEALLOCATE and confirmed that they can deallocate their corresponding coarray.

Furthermore,
> "execution on the active images of the segment (11.7.2) following the statement is delayed until all other active images in the current team have executed the same statement the same number of times in this team"

means that after the DEALLOCATE statement execution is complete on one image, the coarray is unallocated on all active images.

EDIT to 24-007.

None.

SUBMITTED BY: Reinhold Bader

HISTORY: 24-145 m233 Submitted
24-145r1 m233 Revised, passed by J3 meeting 233.
25-133 m236 Passed by J3 letter ballot #40

NUMBER: F23/017
TITLE: CFI_establish nonallocatable nonpointer null base address
KEYWORDS: CFI_establish
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

[524:29-31] 18.5.5.5 The CFI_establish function, p3 Description, states that if CFI_establish is called with the base_addr a null pointer, and attribute CFI_attribute_other (i.e. not a pointer or allocatable), it establishes

"a C descriptor that has the attribute CFI_attribute_other but does not describe a data object".

However, looking at the definition of base_addr in 18.5.3 [518:15-19], it does not allow this:

"If the object is an unallocated allocatable variable or a pointer that is disassociated, the value is a null pointer; otherwise... {cases that are not null pointers}."

Note that CFI_establish is required to follow these rules, as p1 of 18.5.3 states

"The values of these members of a structure of type CFI_cdesc_t that is produced by the functions and macros specified in this document... shall have the properties described in this subclause."

That is a contradiction, which means that when CFI_establish is called with CFI_attribute_other, and base_addr is a null pointer, the program is not standard-conforming and so any behaviour (including memory corruption or program termination) may result.

Either this needs to be permitted in 18.5.3, or forbidden in 18.5.5.5. Permitting it does not seem useful, as there seems to be little or nothing one could possibly do with such a C descriptor.

How should this contradiction be resolved?

ANSWER:

This should not be permitted, as it is useless.

Edits are provided to explicitly forbid this.

EDITS to 24-007:

[524:15] 18.5.5.5 The CFI_establish function, p2 Formal Parameters,
attribute parameter,
Append new sentence

"If it is CFI_attribute_other, \cf{base_addr} shall not be a null pointer."

[524:29-31] Same subclause, p3 Description,
After "for an unallocated allocatable"
change the comma to "or",
After "disassociated pointer"
delete ", or is... data object",
making that sentence read

"If \cf{base_addr} is a null pointer, the established C descriptor is for an unallocated allocatable or a disassociated pointer."

{J3: The paragraph is a wall of text, so I won't regurgitate it here.}

{J3: The standard is contradictory here, so no compatibility issue.}

SUBMITTED BY: Malcolm Cohen

HISTORY: 24-150   m233 Submitted
         24-150r1 m233 Passed by J3 meeting 233.
         25-133   m236 Passed by J3 letter ballot #40

NUMBER: F23/018
TITLE: Correspondence of unallocated coarrays
KEYWORDS: corresponding, unallocated, coarray
DEFECT TYPE: Erratum
STATUS: Passed by J3 letter ballot

QUESTION:

Consider

```
    Program example
        Real,Allocatable :: a[:]
        Call sub(a)
    Contains
        Subroutine sub(x)
            Real,Allocatable :: x[:]
            Allocate(x[*])
            ... do something with A.
        End Subroutine
    End Program
```

According to 5.4.7 paragraph 3 corresponding coarrays have to be "established (5.4.8)" in a team.

According to 5.4.8 paragraph 2,
    "An unallocated allocatable coarray is not established."

Therefore the coarray A on image one does not correspond to the coarray A on any other image.

However, 9.7.1.2 Execution of an ALLOCATE statement, paragraph 4, requires
    "If the coarray is a dummy argument, the ultimate arguments (15.5.2.4) on those images shall be corresponding coarrays."

The program cannot satisfy that requirement, and thus does not conform to the standard. That makes it impossible to allocate any allocatable coarray that is a dummy argument.

Is this intended?

ANSWER:

No, this was not intended.
The definition of correspondence in subclause 5.4.7 is incomplete.

Edits are supplied to make the definition of correspondence complete, extending it to cover unallocated allocatable coarrays. This let us simplify and correct the requirements for allocation.

EDITS to 24-007:

[49:26] 5.4.7 Coarray, p3,
        Change "For each coarray"
        to "For each established coarray".
[49:27] After "in which it is established (5.4.8)."
        insert new sentence
            "For each unallocated coarray, there exists a
             corresponding unallocated coarray with the same declared
             type, rank, corank, and non-deferred type parameters on
             each active image of the current team."
        and then end the paragraph (the rest of paragraph becoming a
        new paragraph).
[49:27] Insert a new paragraph in between the above insertion and the
        rest of what was paragraph 3:
   "For a named coarray that is not a dummy argument, its
    corresponding coarrays are the ones with the same name in that
    scoping unit. For a coarray that is a component at any level of
    component selection, its corresponding coarrays are the same
    components of the base object that has the same name in that
    scoping unit. If a coarray component is a potential subobject
    component of an array element, the array element for its
    corresponding coarrays has the same position in array element
    order on each image."
{Take the correspondence specification from 9.7.1.2 and put it here
 where it belongs. Correspondence is not just for ALLOCATE!}

***ASIDE:

This makes paragraph 3 of 5.4.7 into these three paragraphs:
   "For each established coarray on an image, there is a corresponding
    coarray with the same type, type parameters, and bounds on every
    other image of a team in which it is established (5.4.8). For each
    unallocated coarray, there exists a corresponding unallocated
    coarray with the same declared type, rank, corank, and non-
    deferred type parameters on each active image of the current
    team.

    For a named coarray that is not a dummy argument, its
    corresponding coarrays are the ones with the same name in that
    scoping unit. For a coarray that is a component at any level of
    component selection, its corresponding coarrays are the same
    components of the base object that has the same name in that
    scoping unit. If a coarray component is an ultimate component of
    an array element, the array element for its corresponding coarrays
    has the same position in array element order on each image.

    If a coarray is an unsaved local variable of a recursive
    procedure, its corresponding coarrays are the ones at the same
    depth of recursion of that procedure on each image."

***END ASIDE.

[49:30] Same subclause, paragraph 4,
        After "The set of corresponding"
        insert "established",
        making the whole sentence read
            "The set of corresponding established coarrays on all
             images in a team is arranged in a rectangular pattern."
{Unallocated coarrays are not arranged in any pattern.}

[148:32-40] In "9.7.1.2 Execution of an ALLOCATE statement", replace
            the third sentence "If the coarray is a..." to the end of
            the paragraph with
   "The coarray shall be corresponding (5.4.7) on those images."

{If we got the definition of correspondence right, that is all we need
 to say. It would be inappropriate to define what correspondence means
 in the middle of the ALLOCATE statement execution.}

[148:40+] Insert new paragraph
   "If an allocation specifies a coarray, the same ALLOCATE statement
    shall be executed on every active image of the current team. 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 of that procedure on those images."
{The first requirement actually follows from the segment rules, but
 stating it explicitly means the reader does not have to go off and
 prove a theorem. The second requirement is the last sentence of the
 existing p4, with slightly simplified wording.}

SUBMITTED BY: John Reid and Reinhold Bader.

HISTORY: 23-219   m231 Submitted
         23-219r1 m231 Rejected
         24-146   m233 Revised but not processed
         24-178   m234 Revised again
         24-178r1 m234 Passed by J3 meeting 234.
         25-133   m236 Passed by J3 letter ballot #40
