To: WG5
From: Janice Shepherd
Subject: Proposed content of corrigendum 3

Corrigendum 3 will contain edits from the following defect items that were approved as described in N1080:
28, 30, 41, 86, 121, 129, 135, 137, 139, 143, 147, 177, 181, and 182

100 is not on the list, as its edits overlapped with the new edits for 27. There is a new version of 100 that refers to the edits in 27.

Corrigendum 3 will also contain edits from the following defect items that were approved at the WG5 San Diego meeting (N1141):
0c, 54, 58, 83, 90, 91, 101, 127, 141, 146, 149, 158, 161, 167, 168, 171, 173, 175, 184, 186, 188, 189, 192, 193, 195, 198, and 199.

where a change was made to defect item 83 edit 3, change in last sentence "applies to" to "is consistent with".

Corrigendum 3 will also contain edits from the following defect items that were approved at the WG5 San Diego meeting subject to approval in an X3J3 letter ballot:

NUMBER: 000027
TITLE: Requirements for pointer and target association KEYWORDS: POINTER attribute, TARGET attribute, pointer association DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: If PTR has the POINTER attribute and TGT has the TARGET or POINTER attribute, under which of the following other conditions are PTR and TGT considered to be pointer associated, i.e., under which of the following conditions does ASSOCIATED(PTR, TGT) return a value of .TRUE.:

a) PTR and TGT have different types?
b) PTR and TGT have different type parameters? c) PTR and TGT have different ranks?
d) PTR and TGT have different sizes?
e) PTR and TGT have different shapes?
f) PTR and TGT have different bounds?
g) PTR and TGT refer to the same set of array elements/storage units, but not in the same array element order?
h) PTR and TGT have array elements/storage units whose range of memory

addresses
overlap, but they have no identical array elements/storage units? i) PTR and TGT have at least one but not all identical array elements/storage units
and all the identical elements have the same subscript order value in both PTR and
TGT?
j) PTR and TGT have at least one but not all identical array elements/storage units
but not all the identical elements have the same subscript order value in both PTR
and TGT?

ANSWER: Any of the above conditions except for f) are sufficient for ASSOCIATED (PTR, TGT) to return a value of .FALSE.. In determining whether a pointer and a target are associated, the bounds are relevant only for determining which elements are referenced. The extents of each dimension of PTR and TGT must be the same, thus their shapes must match which is covered by condition (e). If TGT is zero sized, ASSOCIATED (PTR, TGT) returns .FALSE..

Discussion: It is the intent of the standard that the two argument form of the ASSOCIATED intrinsic function returns true only if the association of POINTER and TARGET is as if the pointer assignment statement

POINTER => TARGET

was the last statement executed prior to the execution of the statement invoking the ASSOCIATED function. This is not clear in the definition of the ASSOCIATED intrinsic or elsewhere in the standard. Clarifying edits are provided.

In the example program fragment below, for the first invocation of the ASSOCIATED function the result is true, the second invocation of the function is invalid as the type, type parameters and rank of IR and CX are not the same.

```
SUBROUTINE SUB ()
INTEGER, TARGET        :: I
REAL,TARGET,DIMENSION(2):: R
INTEGER, POINTER       :: IP
REAL, POINTER, DIMENSION(:) :: IR
COMMON /BLOCK/ I, R
IP => I
IR => R(:)
CALL INNER()

CONTAINS
SUBROUTINE INNER()
```

```
INTEGER,TARGET :: J
COMPLEX,TARGET :: CX
COMMON /BLOCK/ J, CX

PRINT *, ASSOCIATED (IP, J) ! true
PRINT *, ASSOCIATED (IR, CX) ! error, though the target of IR and
! CX occupy exactly the same storage
! units in the same order

END SUBROUTINE INNER
END SUBROUTINE SUB
```

EDITS:
1. In section 13.13.13 [198:31] in the description of the TARGET dummy argument add
", and have the same type, type parameters, and rank as POINTER" following
"must be a pointer or target"

2. In section 13.13.13 replace Case (ii) with [198:35-36]
"Case (ii): If TARGET is present and is a scalar target, the
result is true if TARGET is not a zero sized storage
sequence and the target associated with POINTER occupies
the same storage units (there may be only one) as TARGET.
Otherwise the result is false. If POINTER is
disassociated, the result is false."

3. In section 13.13.13 replace Case (iii) [199:1-3] with
"Case (iii): If TARGET is present and is an array target, the
result is true if the target associated with POINTER and
TARGET have the same shape, are not of size zero or arrays
whose elements are zero sized storage sequences, and
occupy the same storage units in array element order.
Otherwise the result is false. If POINTER is
disassociated, the result is false.

Case (iv): If TARGET is present and is a scalar pointer, the
result is true if the target associated with POINTER and
the target associated with TARGET are not zero sized
storage sequences and they occupy the same storage units
(there may be more than one). Otherwise the result is
false. If either POINTER or TARGET is disassociated, the
result is false.

Case (v): If TARGET is present and is an array pointer, the result
is true if the target associated with POINTER and the
target associated with TARGET have the same shape, are

not of size zero or arrays whose elements are zero sized
storage sequences, and occupy the same storage units in
array element order. Otherwise the result is false. If
either POINTER or TARGET is disassociated, the result is
false. "

SUBMITTED BY: Jon Steidel, 120-JLS-4 (120.022) HISTORY: 120-LJM-3A
(120.081A)
m121 Original response proposed
92-061 (121-ADT-9) p9 & X3J3 / 92-061 Questioned response
(121-ADT-13) item 27
92-093A m121 Approved
92-329 (jw note)
m123 Approval rescinded at m123 (uc)
93-099r1 m124 Revised response adopted (15-2) 93-111 m125 ballot failed, returned
to subgroup 93-135r m125 Based on comments returned with the X3J3 letter
ballot following m124, the revised response was
prepared and adopted 15-3.
93-255r1 m127 ballot failed 17-7
94-289 m130 revised answer, approved u.c. 94-306 m131 X3J3 ballot failed 16-3
95-281 m135 revised response, added edits, WG5 approved

NUMBER: 000081
TITLE: Pointer actual argument overlap
KEYWORDS: pointer, target, argument - actual, argument - dummy,
argument association
DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: Section 12.5.2.9, Restrictions on entities associated with dummy
arguments, clearly states that if there is partial or complete overlap between
actual arguments associated with two different dummy arguments of the same
procedure, the overlapping portions may not be defined, redefined, or undefined
during the procedure execution. It continues:

"This restriction applies equally to pointer targets. For example, in

```
REAL, DIMENSION (10), TARGET :: A
REAL, DIMENSION (:), POINTER :: B, C
B => A (1:5)
C => A (3:9)
CALL SUB (B, C)
```

B (3:5) cannot be defined because it is part of the actual argument associated with the second dummy argument. C (1:3) cannot be defined because it is part of the argument associated with the first dummy argument. A (1:2) [which is B (1:2)] remains definable through the first dummy argument and A (6:9) [which is C (4:7)] remains definable through the second dummy argument."

Unfortunately, this example does not contain an explicit interface for the called subroutine, nor are there sufficient words to clearly state what is meant by the words and example provided.

Question 1: Do the above restrictions hold when both dummy arguments are nonpointers? In this case the following interface describes SUB.

```
INTERFACE
SUBROUTINE SUB (X, Y)
REAL, DIMENSION (:) :: X, Y
END SUBROUTINE
END INTERFACE
```

Question 2: Same as question 1, only add the TARGET attribute to one or both of the dummy arguments. The following interfaces may describe SUB.

```
INTERFACE
SUBROUTINE SUB (X, Y)
REAL, DIMENSION (:), TARGET :: X, Y
END SUBROUTINE
END INTERFACE
```
or

```
INTERFACE
SUBROUTINE SUB (X, Y)
REAL, DIMENSION (:) :: X, Y
TARGET X
END SUBROUTINE
END INTERFACE
```

Question 3: Do the above restrictions hold *upon entry* when both dummy arguments are pointers? That is, *upon entry* to SUB, is it safe to assume that pointer dummy arguments do not have overlapping elements which may get defined during execution of SUB? The following interface describes SUB.

```
INTERFACE
SUBROUTINE SUB (X, Y)
REAL, DIMENSION (:), POINTER :: X, Y
END SUBROUTINE
END INTERFACE
```

Question 4: Same as question 3, but one dummy argument is a pointer, one has the target attribute? *Upon entry* to SUB, is it safe to assume a pointer dummy argument does not point to any elements of a target dummy argument which may get defined during execution of SUB, but during the execution of SUB such an association may come to exist? The following interface describes SUB.

```
INTERFACE
SUBROUTINE SUB (X, Y)
REAL, DIMENSION (:) :: X, Y
POINTER X
TARGET Y
END SUBROUTINE
END INTERFACE
```

Question 5: Two derived type dummy arguments each have a subobject (or a subobject of a subobject etc.) which are pointers with the same type, kind type parameter, and rank. *Upon entry* to the subroutine, is it safe to assume such pointer subobjects do not have overlapping targets which may get defined? That is, in the following fragment, *upon entry* to SUB, is it safe to assume X%PTR_1 and Y%PTR_2 cannot have overlapping target elements which get defined during execution of SUB?

```
SUBROUTINE SUB (X, Y)
TYPE A
SEQUENCE
REAL, DIMENSION(:), POINTER :: PTR_1
END TYPE

TYPE B
SEQUENCE
REAL, DIMENSION(:), POINTER :: PTR_2
END TYPE

TYPE (A) :: X
TYPE (B) :: Y
```

ANSWER: There is a deficiency in the standard. The restrictions always apply to the allocation status of the entities, but do not apply to:

(1) the values when the dummy argument or a subobject of the dummy argument
has the POINTER attribute,

(2) the values when the dummy argument has the TARGET attribute, is a scalar or an assumed-shape array and does not have the INTENT(IN) attribute

and the actual argument is a target other than an array section with a vector subscript, and

(3) the pointer association status.

Edits are supplied to correct this. The answers to the specific questions are:

Answer 1: Yes for the interface specified.

Answer 2: Yes for the allocation status of the entities. Yes for the values except when (2) holds.

Answer 3: Yes for the allocation status of the entities. No for the pointer association status. No for the values. Overlapping elements of dummy arguments may be defined during execution of SUB.

Answer 4 (first part): Yes for the allocation status of the entities. No for the pointer association status of the entities. No for the value of the pointer argument. No for the value of the target argument when it is not INTENT(IN), it is scalar or an assumed-shape array and the actual argument is a target other than an array section with a vector subscript. Yes for the value of the target argument in other cases.

Answer 4 (second part): Upon entry to SUB, a pointer dummy argument may point to an element of a target assumed-shape dummy argument that is defined during execution of SUB.

Answer 5: Yes for the allocation status of the entities. No for the pointer association status of the entities. No for the values. Overlapping elements of pointer components of dummy arguments may be defined during execution of SUB.

Discussion: The restrictions of Section 12.5.2.9 on entities associated with dummy arguments are intended to facilitate a variety of optimizations in the translation of the procedure, including implementations of argument association in which the value of an actual argument that is neither a pointer nor a target is maintained in a register or in local storage. The latter mechanism is usually known as "copy-in/copy-out". The text assumed that the rules applied to an actual argument with the TARGET attribute, too, but defect item 125 has made it clear that this is not the case for all actual arguments with the TARGET attribute and edits are needed in 12.5.2.9.

The present text is incorrect for the pointer association status of a pointer dummy argument. The implementation must allow for the pointer association status to be altered through another pointer.

The present text is incorrect for the value of a pointer dummy argument. Here, the implementation must allow for the value to be altered through another pointer as in the example

```
INTEGER, POINTER :: IP
CALL INNER(IP)
CONTAINS
SUBROUTINE INNER(IARGP)
INTEGER, POINTER :: IARGP
INTEGER, TARGET :: J
IP = 0 ! OK. This alters the value of IARGP, too.
```

It is also incorrect where the dummy argument has the TARGET attribute, the dummy argument does not have INTENT(IN), the dummy argument is a scalar object or an assumed-shape array and the actual argument is a target other than an array section with a vector subscript. Here the implementation must allow for the value to be altered through a pointer as in the example

```
INTEGER, POINTER :: IP
CALL INNER(IP)
CONTAINS
SUBROUTINE INNER(IARGT)
INTEGER, TARGET :: IARGT
IP = 0 ! OK. This alters the value of IARGT, too.
```

EDITS:
1. In section 12.5.2.9, [180:3-4] Replace the first two lines of item (1) by
"No action that affects the allocation status of the entity may be taken. Action that affects the value of the entity or any part of it must be taken through the dummy argument unless
(a) the dummy argument has the POINTER attribute, (b) the part is all or part of a pointer subobject, or (c) the dummy argument has the TARGET attribute, the dummy
argument does not have INTENT(IN), the dummy argument is a scalar object or an assumed-shape array and the actual argument is a target other than an array section with a vector subscript.
For example, in"

2. In section 12.5.2.9, [180:32] in the line following the line "DEALLOCATE (A)" change "availability of A" to "allocation of B".

3. In section 12.5.2.9, [180:34-35] in the lines following the line "DEALLOCATE(B)"
change ", but would ... attribute."
to ". If B were declared with the POINTER attribute the statements
DEALLOCATE(A)

and
DEALLOCATE(B)
would both be permitted."

4. In section 12.5.2.9, [180:37] in the second to last paragraph on page
180, after "the same procedure" add

"and the dummy arguments have neither the POINTER nor the TARGET
attribute".

5. In section 12.5.2.9, [181:8] in line 8 of page 181, after "CALL SUB(B,C)" add

"! The dummy arguments of SUB are neither pointers nor targets".

6. In section 12.5.2.9, [181:17-19] Replace the first three lines of item (2) by

"If the value of any part of the entity is affected through the dummy
argument, then at any time during the execution of the procedure, either before
or after the definition, it may be referenced only through that dummy argument
unless

(a) the dummy argument has the POINTER attribute, (b) the part is all or part of a
pointer subobject, or (c) the dummy argument has the TARGET attribute, the
dummy
argument does not have INTENT(IN), the dummy argument is a scalar object or
an assumed-shape array and the actual argument is a target other than an array
section with a vector subscript.

For example, in"

7. In section C.12.7, [291:40-42] Replace lines 3 to 5 by

"The restrictions on entities associated with dummy arguments are intended to
facilitate a variety of optimizations in the translation of the procedure, including
implementations of argument association in which the value of an actual
argument that is neither a pointer nor a target is maintained in a register or in
local storage."

SUBMITTED BY: Jon Steidel
HISTORY: 92-208 m123 Submitted
93-85 m124 Proposed response, withdrawn
93-174 m125 Revised response, withdrawn
93-213 m126 Revised response, adopted by unanimous consent 93-255r1 m127
ballot failed 19-5
94-248 m130 Revised response and edits.

94-282r1 m130 Clarified terminology, approved u.c. 94-306 m131 ballot failed 11-7 94-309r1 m131 modified edits and answer, approved 12-1 94-366r1 m131 First edit replaced with revised text from 94-366r1
approved u.c.
95-034r1 m132 X3J3 ballot failed, 16-4
95-043 m132 revised words of answer/edit (no change in intent) and
change status to X3J3 approved; approved 10-4
95-244 m134 Updated answer and edits based on new answer to
defect item 125, approved u.c.
95-256 m135 X3J3 ballot failed, 10-6
95-281 m135 revised response and edits, WG5 approved

NUMBER: 000125
TITLE: Copy in/copy out of target dummy arguments KEYWORDS: argument - dummy, target, interface - explicit,
argument association
DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION:Previous Fortran standards have permitted copy in/copy out as a valid implementation for argument passing to procedures, as does Fortran 90. Fortran 90 introduces POINTER and TARGET attributes. Sections 12.4.1.1 and C.12.8 indicate that it was intended that copy in/copy out also be a valid implementation for passing an actual argument that has the TARGET attribute to a dummy argument that has the TARGET attribute. The following example demonstrates a case where a copy in/copy out implementation may get different results from an implementation which does not use a copy in/copy out method for such a combination of arguments.

```
POINTER IPTR
TARGET I
IPTR => I
CALL SUB (I, IPTR)
...
CONTAINS
SUBROUTINE SUB (J, JPTR)
POINTER JPTR
TARGET J
PRINT *, ASSOCIATED (JPTR, J)
END SUBROUTINE
END
```

Is this a flaw in the standard?

ANSWER:Yes, there is a flaw in the standard. The edits supplied disallow copy in/copy out as a valid implementation for passing an actual argument that has the TARGET attribute to a corresponding argument that has the TARGET attribute, and is either scalar or is an assumed-shape array.

Discussion: The changes apply only to target dummy arguments. Without the changes the behaviour of the example in the question would surprise many programmers. Other examples not involving the ASSOCIATED function are also affected by these changes in such a way that they too will have a more expected behaviour. One such example is included in the edit to section C.12.8.

An earlier answer to this defect did not contain the following words at the end of the first paragraph of edit 2

"If such a dummy argument is associated with a dummy argument with the TARGET attribute, whether any pointers associated with the original actual argument become associated with the dummy argument with the TARGET attribute is processor dependent."

The earlier answer also included different wording for the start of the second paragraph of edit 2:

"If the dummy argument has the TARGET attribute and the corresponding actual argument has the TARGET attribute but is not an array section with a vector subscript:"

and did not include the paragraph:

"If the dummy argument has the TARGET attribute and is an explicit-shape array or is an assumed-size array and the corresponding actual argument has the TARGET attribute but is not an array section with a vector subscript:

(1) On invocation of the procedure, whether any pointers associated with the actual argument become associated with the corresponding dummy argument is processor dependent.

(2) When execution of the procedure completes, the pointer association status of any pointer that is pointer associated with the dummy argument is processor dependent."

An earlier answer to this defect included different wording for the first paragraph of edit 3.

"When execution of a procedure completes, any pointer that remains defined and that is associated with a dummy argument that has the TARGET attribute, remains associated with the corresponding actual argument if the actual argument has the TARGET attribute and is not an array section with a vector subscript."

An earlier answer to this defect included the edit:
Section 12.4.1.1, add at the end of the fourth paragraph [173:6],

"If the dummy argument has the TARGET attribute and the actual
argument has the TARGET attribute but is not an array section with a vector subscript, the dummy and actual arguments must have the same shape."

The earlier versions of edits 2 and 3, along with edit 1 and the just mentioned deleted edit were included in corrigendum 2.

EDITS:
1. Section 12.4.1.1, fifth paragraph, last sentence [173:10-13]
delete, "with a dummy argument of the procedure that has the TARGET attribute or"

2. Section 12.4.1.1, delete the sixth paragraph [173:14-17] and
replace with,

"If the dummy argument does not have the TARGET or POINTER
attribute, any pointers associated with the actual argument do not become associated with the corresponding dummy argument on invocation of the procedure. If such a dummy argument is associated with a dummy argument with the TARGET attribute, whether any pointers associated with the original actual argument become associated with the dummy argument with the TARGET attribute is processor dependent."

If the dummy argument has the TARGET attribute and is either scalar or is an assumed-shape array, and the corresponding actual argument has the TARGET attribute but is not an array section with a vector subscript:

(1) Any pointers associated with the actual argument
become associated with the corresponding dummy argument on invocation of the procedure.

(2) When execution of the procedure completes, any
pointers associated with the dummy argument remain associated with the actual argument.

If the dummy argument has the TARGET attribute and is an explicit-shape array or is an assumed-size array, and the corresponding actual argument has the TARGET attribute but is not an array section with a vector subscript:

(1) On invocation of the procedure, whether any pointers
associated with the actual argument become associated with the corresponding
dummy argument is processor dependent.

(2) When execution of the procedure completes, the pointer
association status of any pointer that is pointer associated with the dummy
argument is processor dependent.

If the dummy argument has the TARGET attribute and the corresponding actual
argument does not have the TARGET attribute or is an array section with a
vector subscript, any pointers associated with the dummy argument become
undefined when execution of the procedure completes."

3. Section C.12.8, delete the second paragraph through the end of
the section [292:5-37] and replace with

"When execution of a procedure completes, any pointer that remains
defined and that is associated with a dummy argument that has the TARGET
attribute and is either scalar or is an assumed-shape array, remains associated
with the corresponding actual argument if the actual argument has the TARGET
attribute and is not an array section with a vector subscript.

```
REAL, POINTER    :: PBEST
REAL, TARGET     :: B (10000)
CALL BEST (PBEST, B)     ! Upon return PBEST is associated
...       ! with the "best" element of B
CONTAINS
SUBROUTINE BEST (P, A)
REAL, POINTER    :: P
REAL, TARGET     :: A (:)
...       ! Find the "best" element A(I)
P => A (I)
RETURN
END SUBROUTINE
END
```

When the procedure BEST completes, the pointer PBEST is associated with an
element of B.

An actual argument without the TARGET attribute can become associated with a
dummy argument with the TARGET attribute. This permits pointers to become
associated with the dummy argument during execution of the procedure that
contains the dummy argument. For example:

```
INTEGER LARGE(100,100)
CALL SUB(LARGE)
```

```
...
CALL SUB()
CONTAINS
SUBROUTINE  SUB(ARG)
INTEGER, TARGET, OPTIONAL :: ARG(100,100) INTEGER, POINTER,
DIMENSION(:,:) :: PARG IF (PRESENT(ARG)) THEN
PARG => ARG
ELSE
ALLOCATE (PARG(100,100))
PARG = 0
ENDIF
... ! Code with lots of references to PARG IF (.NOT. PRESENT(ARG))
DEALLOCATE(PARG) END SUBROUTINE SUB
END
```

Within subroutine SUB the pointer PARG is either associated with the dummy argument ARG or it is associated with an allocated target. The bulk of the code can reference PARG without further calls to the PRESENT intrinsic."

SUBMITTED BY: Jon Steidel - X3J3/93-095
HISTORY: 93-095 m124 submitted with draft response and adopted (15-1)
93-111 m125 ballot, returned to subgroup based on Leonard, Maine
comments. Problems with placement of edit 1,
content of edit 4
93-139r m125 revised response adopted 17-1. 93-255r1 m127 ballot failed 13-10
94-092r1 m128 revised response, approved 11-5 94-116r1 m129 X3J3 ballot failed 10-13
94-177r1 m129 revised response closer to 93-255r1; approved 19-2 94-221 m130 X3J3 ballot, approved 21-2
94-327 m131 WG5 approved, edit changed to reflect change in
corrigendum 2.
95-177 m134 revised response and edits, approved 14-2 95-256 m135 X3J3 ballot, approved 15-1
95-281 m135 revised edits after errors were discovered, WG5 approved
(N1161)

NUMBER: 000145
TITLE: Expressions in <type-spec> of a FUNCTION statement KEYWORDS: expression - specification, expression - initialization,
FUNCTION statement, host association, use association DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: The syntax rule R1217 shows that the type and type parameters of a function can be specified in the FUNCTION statement (12.5.2.2).

(a) If a <type-spec> appears in a FUNCTION statement, can the initialization and specification expressions of that <type-spec> involve names of entities that are declared within the function or are accessible there by host or use association?

(b) Section 5.1 states:

"The <specification-expr> (7.1.6.2) of a <type-param-value> (5.1.1.5)
or an <array-spec> (5.1.2.4) may be a nonconstant expression provided the specification expression is in an interface body (12.3.2.1) or in the specification part of a subprogram."

As a FUNCTION statement is not part of the specification part of a subprogram, this text in the standard appears to distinguish between FUNCTION statements that are in interface blocks and ones that are not. This text seems to prohibit such examples as:

```
INTEGER I
...
CONTAINS
CHARACTER*(I+1) FUNCTION F()
...
COMMON / / I
...
```

where it can be confusing as to which I is being referenced in the FUNCTION statement. While host association does not apply to interface bodies, for consistency should the text quoted from Section 5.1 have been "... is in the specification part of an interface body (12.3.2.1) or in the specification part of a subprogram."?

(c) Section 7.1.6.1 states:

"If an initialization expression includes a reference to an inquiry function for a type parameter or an array bound of an object specified in the same <specification-part>, the type parameter or array bound must be specified in a prior specification of the <specification-part>."

Was this text intended to apply to FUNCTION statements even though they are not part of any <specification-part>, thus disallowing fragments such as:

```
INTEGER (KIND=KIND(X)) FUNCTION F()
INTEGER(KIND=KIND(0)) X
...
```

Similar text appears in Section 7.1.6.2.

ANSWER:
(a) A specification expression in the <type-spec> of a FUNCTION statement may involve names of entities that are declared within the function or are accessible there by host or use association, but an initialization expression in such a <type-spec> may only involve names that are accessible by host or use association.

(b) No. It was not the intent of the standard to distinguish between the two types of FUNCTION statements cited. As elaborated in the discussion of part (a), the standard intended to allow the <type-spec> expression of a FUNCTION statement to be a nonconstant expression. The sentence cited is corrected with a supplied edit.

(c) Yes, the text cited from 7.1.6.1 was intended to apply to FUNCTION statements. The sentence quoted and the corresponding sentence in 7.1.6.2 are corrected with supplied edits. The code fragment is not standard conforming.

Discussion:

(a) An initialization expression is a constant expression with an additional rule relating to exponentiation (7.1.6.1). Since it is a constant expression, the only names it can contain are the names of named constants, structure constructors, intrinsic procedures, and variables whose type parameters or bounds are inquired about.

* Named constant

Section 5.1.2.1 states:

"A named constant must not be referenced in any ... context unless
it has been defined in a prior PARAMETER statement or type declaration statement using the PARAMETER attribute, or made accessible by use association or host association."

Since the FUNCTION statement is the first statement of the scoping unit, there can be no prior PARAMETER statement or type declaration statement using the PARAMETER attribute, so the first clause does not apply. A named constant can appear in a <type-spec> of a function statement if it is accessible within the function by host or use association.

* Structure constructor

Rule R502 shows that the only opportunities for expressions to appear in <type-spec>s are in a <kind-selector> or in a <char-selector>. However, a structure constructor can not appear in a <kind-selector> because rule R505 shows that a <kind-selector> must be an integer expression. Similarly, R506 shows that any initialization expression in a <char-selector> must be type integer. Therefore, a structure constructor can not appear in an initialization expression in the <type-spec> of a FUNCTION statement.

* Intrinsic procedure

The intrinsic procedure names or classes of intrinsic procedures that may appear in an initialization expression are given in 7.1.6.1.

* Variables whose type parameters or bounds are inquired about

The text from section 7.1.6.1 as cited in question (c) was intended to apply to initialization expressions in the <type-spec> of a FUNCTION statement. With the correction supplied, this means that if a variable appears as the argument to an inquiry intrinsic in the <type-spec> of a FUNCTION statement, the function must be a module procedure or an internal procedure, and the variable must exist in (be accessible from) the host scoping unit.

Rule R502 defines <type-spec>. The only opportunity for a <type-spec> to contain a <specification-expr> is when the data type is character (<type-param-value> may be a <specification-expr>). Section 7.1.6.2 states that a specification expression is a restricted expression that is scalar, of type integer, and each operation must be intrinsic. In addition, rule (2) of 7.1.6.2 states that a primary of a specification expression can be a dummy argument that has neither the OPTIONAL nor INTENT(OUT) attribute. The following code fragment demonstrates a use of such a dummy argument:

```
CHARACTER*(N+1) FUNCTION S(N)
INTEGER, INTENT(IN) :: N
```

Rule (2) also states that the primary can be a subobject of such a dummy argument. Section 6.1.2 indicates that a structure component must not be referenced or defined before the declaration of the parent object. Similar rules are needed to prevent a substring from being referenced ahead of the declaration of its parent, and an array element or array section from being referenced ahead of the declaration of the array. Edits are provided to supply these rules. Since a subobject can not be referenced before its parent object is declared and the FUNCTION statement is the first statement of the subprogram, the parent's declaration could not have occurred. Thus a subobject must not be referenced in the <type-spec> on a FUNCTION statement for objects declared within the function.

Rule (3) states that a primary can be a variable that is in a common block. The following code fragment demonstrates a use of such a common block member:

```
CHARACTER*(N+1) FUNCTION S()
...
COMMON N
```

As in rule (2), rule (3) allows a subobject of such a variable but for the same reasons as above, such a subobject designator can not appear in the <type-spec> expression of a FUNCTION statement.

Rule (4) states that a primary may be a variable that is accessible by use association or host association. The following code fragments demonstrate uses of such variables:

```
PROGRAM MAIN
INTEGER :: N = 21
...
CONTAINS
CHARACTER(LEN = 2*N) FUNCTION SS(K) ! N is host associated. ...
END FUNCTION
END PROGRAM
```

and

```
MODULE MOD
INTEGER K
DATA K /20/
END MODULE

CHARACTER*(K*2) FUNCTION CHECK(STR)      ! K is use associated.
USE MOD
...
END FUNCTION
```

Rule (4) also states that the primary can be a subobject of such a use or host associated variable.

A structure constructor can not appear in a FUNCTION <type-spec> specification expression because the expression must be of type integer and any operations (which might yield an integer value from one or more structure constructors) must be intrinsic.

Other rules of 7.1.6.2 state which intrinsic procedure names or classes of intrinsic procedures may appear in a specification expression.

Section 7.1.6.2 also states:

A variable in a specification expression must have its type and type parameters, if any, specified by a previous declaration in the same scoping unit, or by the implicit type rules currently in effect for the scoping unit, or by host or use association.

The discussion above regarding specification expressions has already ruled out "previous declarations" so the first clause of the cited sentence does not apply. The other clauses apply equally to a FUNCTION statement <type-spec> and to type declaration statements inside the function.

(b) When the discussion for part (a) is applied to the code fragment provided, it means that the 'I' referenced in the <type-spec> of the FUNCTION statement is the common block member.

EDITS:
1. Section 5.1, in the first sentence of the paragraph that starts
"The <specification-expr> (7.1.6.2)" [40:39-41],

change "in an interface body (12.3.2.1) or in the specification part of a subprogram"

to "contained in an interface body (12.3.2.1), is contained in the specification part of a subprogram, or is in the <type-spec> of a FUNCTION statement (12.5.2.2)"

2. Section 6.1.1, add to the end of the paragraph before the examples [62:29]

"A substring must not be referenced or defined before the declaration of the type and type parameters of the parent string, unless the type and type parameters are determined by the implicit typing rules of the scope."

3. Section 6.2.2, add after the sentence "An array section is an array."
[64:16]
"An array element or array section must not be referenced or defined before the declaration of the array bounds."

4. Section 7.1.6.1, in the paragraph after the constraints [78:21-22]

change "object specified in the same <specification-part>, the type parameter or array bound must be specified in a prior specification of the <specification-part>."

to "object declared in the same scoping unit, the type parameter or array bound must be specified in a specification prior to the initialization expression."

5. Section 7.1.6.2, in the 2nd paragraph after the constraint [79:28-29]

change "entity specified in the same <specification-part>, the
type parameter or array bound must be specified in a prior specification of the
<specification-part>."

to "entity declared in the same scoping unit, the type
parameter or array bound must be specified in a specification prior to the
specification expression."

SUBMITTED BY: Janice C. Shepherd
HISTORY: 93-193 m126 submitted
94-023r1 m128 response, approved uc
94-116r1 m129 X3J3 ballot failed 22-1
94-336 m131 revised response, approved u.c 95-034r1 m132 X3J3 ballot failed 15-5
95-281 m135 revised response, reworded edit 3, WG5 approved (N1161)

NUMBER: 000148
TITLE: RANDOM_SEED, RANDOM_NUMBER
KEYWORDS: RANDOM_SEED intrinsic, RANDOM_NUMBER intrinsic
DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION:
(1) After executing the following sequence :
CALL RANDOM_SEED
CALL RANDOM_NUMBER(X)
CALL RANDOM_SEED
CALL RANDOM_NUMBER(Y)
is it the intent of the standard that X=Y?

The description of RANDOM_SEED, section 13.13.84 (page 228), specifies that if
no argument to RANDOM_SEED is present the processor sets the seed to a
processor-dependent value. Was it the intent that the same processor-dependent
value be set into the seed on all such argumentless calls to RANDOM_SEED?

(2) After executing the following sequence:
INTEGER SEED(2)
CALL RANDOM_NUMBER(X1)
CALL RANDOM_SEED(GET=SEED)
CALL RANDOM_NUMBER(X2)
CALL RANDOM_SEED(PUT=SEED)
CALL RANDOM_NUMBER(X3)

is it the intent of the standard that X2=X3? i.e. that the seed is updated on each call to RANDOM_NUMBER and that by restoring the seed value to that before the last call of RANDOM_NUMBER the last number will be generated again.

There is nothing in the standard that specifies this behavior.

An alternative implementation that conforms to the current standard does not update the seed on each call to RANDOM_NUMBER. Rather the put argument to RANDOM_SEED effectively initializes a sequence of numbers and remains unchanged until the next put. Whenever a put is done with a given seed the same sequence of numbers will always be generated. If a different seed is put a different seed will be generated. With this approach the value X3 has the same value as X1, not X2.

ANSWER:
(1) No, it is not the intent of the standard that X must equal Y after the the example calls to RANDOM_SEED and RANDOM_NUMBER. The standard states:

If no argument is present, the processor sets the seed to a processor dependent value.

in 13.13.84. This leaves the value of the seed and the method of generating that value up to the processor. Therefore, the answer to the second question is no, it is not the intent of the standard that the same processor-dependent value be set into the seed on all such argumentless calls to RANDOM_SEED.

(2) Yes. It is the intent of the standard that X2=X3. An edit is supplied to clarify that this is the intent.

Note that the program fragment in question 2 is standard conforming for a given processor only if that processor returns the value '2' or '1' in the SIZE argument when RANDOM_SEED is called.

EDIT: In section 13.13.84 add the following text [228:13+]

"The pseudorandom number generator accessed by RANDOM_SEED and RANDOM_NUMBER maintains a seed that is updated during the execution of RANDOM_NUMBER and that may be specified or returned by RANDOM_SEED. Computation of the seed from argument PUT is performed in a processor dependent manner. The value specified by PUT need not be the same as the value returned by GET in an immediately subsequent reference to RANDOM_SEED. For example, following execution of the statements

```
CALL RANDOM_SEED(PUT=SEED1)
CALL RANDOM_SEED(GET=SEED2)
```

SEED1 need not equal SEED2. When the values differ, the use of either value as the PUT argument in a subsequent call to RANDOM_SEED will result in the same sequence of pseudorandom numbers being generated. For example, after execution of the statements

```
CALL RANDOM_SEED(PUT=SEED1)
CALL RANDOM_SEED(GET=SEED2)
CALL  RANDOM_NUMBER(X1)
CALL RANDOM_SEED(PUT=SEED2)
CALL  RANDOM_NUMBER(X2)
```

X1 equals X2."

SUBMITTED BY: Graham Barber
HISTORY: 93-203 m126 submitted
94-051r2 m128 response, approved uc
94-116 m129 X3J3 ballot failed 12-11
94-201 m129 revised response and added edit, approved u.c. 94-221 m130 X3J3 ballot failed 21-2
94-325 m131 revised response, approved 16-1 95-034r1 m132 X3J3 ballot failed 13-7
95-142r1 m133 revised response, approved 10-2 95-183 m134 X3J3 ballot failed 12-5
95-281 m135 revised edit, approved by WG5 (N1161)

NUMBER: 000154
TITLE: EQUIVALENCE and zero-sized sequences KEYWORDS: EQUIVALENCE statement, zero-sized sequences DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: Section 5.5.1 and its subsections discuss equivalence association and zero-sized sequences, but not in detail sufficient to answer the following questions:

Given the following

```
CHARACTER (LEN=4) :: A, B, C
```

Question 1: are the following equivalence statements valid?

```
COMMON /X/ A
COMMON /Y/ B
EQUIVALENCE (A, C(10:9)), (B, C(10:9)) ! the same zero-sized object is
! equivalenced to two different
! common blocks.
```

```
EQUIVALENCE (A, C(10:9)), (B, C(100:90)) ! two different zero-sized objects
! are equivalenced to two different
! common blocks
```

Question 2: is the following equivalence valid

```
EQUIVALENCE (C, C(10:9))
```

Question 3: Is the following valid?

```
EQUIVALENCE (C(10:9), C(100:90))
```

Question 4: It is generally the case that

```
given
CHARACTER (LEN=4) :: D, E, F

then
EQUIVALENCE (D, E)
EQUIVALENCE (F, E)
and
EQUIVALENCE (D, E, F)
```

both specify that D is equivalenced to F. Do the following specify the same storage associations?

```
EQUIVALENCE (D, E(2:1))        ! E(2:1) is zero sized
EQUIVALENCE (F, E(2:1))
and
EQUIVALENCE (D, E(2:1), F)


EQUIVALENCE (D, E(2:1))        ! E(2:1) is zero sized
EQUIVALENCE (F, E(3:1))        ! E(3:1) is also zero-sized
and
EQUIVALENCE (D, E(2:1), F)
```

Question 5: is E(2:1) a subobject of E?

ANSWER 1: Both equivalence statements are invalid. Introducing complex semantics to explain the meaning of equivalencing zero-length substrings with each other or with other variables adds little useful functionality to the language. Instead such equivalences should have been prohibited. An edit is included to make the prohibition.

ANSWER 2: No.
ANSWER 3: No.
ANSWER 4: No.
ANSWER 5: Yes.

EDIT: In section 5.5.1, add a new constraint after the existing constraints [57:17+] "Constraint: A <substring> must not have length zero."

SUBMITTED BY: Dick Weaver
HISTORY: 93-264 m127 submitted
93-323 m127 response approved 14-5
94-034 m128 X3J3 ballot failed 10-18
95-281 m135 revise response, WG5 approved (N1161)

NUMBER: 000155
TITLE: Multiple USE statements, rename and only lists. KEYWORDS: USE statement, use renaming, ONLY DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: Section 11.3.2 states: (with some formatting to better show logic, tags added to assist references):

R1109 <only> is <access-id>
or [<local-name> =>] <use-name>

Constraint: Each <access-id> must be a public entity in the module. Constraint: Each <use-name> must be the name of a public entity in the
module.

......

More than one USE statement for a given module may appear in a scoping unit.

If one of the USE statements is without an ONLY qualifier, (a) all public entities in the module are accessible and (b) the <rename-list>s and <only-list>s are interpreted as a single
concatenated <rename-list>.

Questions:

1. Is the syntax ambiguous? Note:

R1107 <use-stmt> is USE <module-name> [,<rename-list>]
or USE <module-name>, ONLY: [<only-list>]

R1109 <only>   is <access-id>
or [<local-name> =>] <use-name>

R522 <access-id> is <use-name>
or <generic-spec>

Thus <use-name> in an <only-list> can parse either as

R1109 <use-name>
or R1109 <access-id> -> R522 <use-name>.

2. Can <rename-list>s and <only-list>s be concatenated as a <rename-list>
as specified in (b)? <Only-list>s permit "<access-id>", while rename lists require
"<local-name> => <use-name>".

ANSWER:

1. Yes. The syntax is ambiguous ("<access-id>" should be changed to "<generic-spec>"). However removal of the ambiguity will be deferred to the next revision of the standard (Fortran 95).

2. No. The syntax is such that it might appear only the renames from <only-list>s are being concatenated into the single <rename-list>. An edit is supplied to make it clear that all items from any <only-list> contribute to the list of accessible local names and accessible <access-id>s.

Discussion: There are many other places in the standard besides R1109 where the same interpretation can be reached by more than one path through the syntax (e.g. <boz-literal-constant> could be removed from R533 since it is a special case of a <scalar-constant>). Rather than fix this one ambiguity now, it will be deferred to the more general clean up in Fortran 95.

Note that in Fortran 95 the appropriate edits to remove the ambiguity in R1109 are :

1. In section 11.3.2, R1109 [158:11]
change "<access-id>"
to "<generic-spec>"

2. In section 11.3.2, the first constraint following R1109 [158:13]
change "<access-id>"
to "<generic-spec>"

3. In section 11.3.2, paragraph beginning "A USE statement" [158:19]

change "<access-id>s"
to "<generic-spec>s"

EDITS:
1. In section 11.3.2, in the first sentence after the constraints [158:16]
change "the local name is the <use-name>"
to "the local name of a named entity is the <use-name>"

2. In section 11.3.2, in the 2nd sentence of the 4th paragraph after the
constraints [158:21-23]
change " and the <rename-list>s and <only-list>s are interpreted
as a single concatenated <rename-list>"
to ", the <rename-list>s and renames in <only-list>s are
interpreted as a single concatenated <rename-list>, and entities in the remaining
<only-list> items are accessible by those <use-name>s or <access-id>s (they may
also be accessible by one or more renames)"

SUBMITTED BY: Dick Weaver
HISTORY: 93-265 m127 submitted
93-305 m127 response approved uc
94-034 m128 X3J3 ballot passed 26-1
94-160 m129 failed WG5 ballot
94-352 m131 revised response, approved u.c. 95-034r1 m132 X3J3 ballot failed 19-1
95-281 m135 revised response and edits, WG5 approved.

NUMBER: 000176
TITLE: Definition of RANDOM_SEED
KEYWORDS: RANDOM_SEED intrinsic
DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: The definition in 13.13.84, RANDOM_SEED, for the optional
argument GET is:

"must be a default integer array of rank one and size >= N. It is
an INTENT(OUT) argument. It is set by the processor to the current value of the
seed."

There is similar text for the PUT argument. For both cases

-- "set" is used in a manner that appears to specify assignment.
For similar uses of "set" see DATE_AND_TIME, IBSET, and SET_EXPONENT.

-- "current value" is not the same thing as "physical representation". Setting, or assignment, of a value and necessary conversions, are described in section 7.5. "physical representation", however, was more likely intended for RANDOM_SEED.

-- The shape of the seed is processor dependent and the
specification for both GET and PUT results in different semantics depending on whether the processor's seed is an array of default integers, an array of some other numeric type, or a scalar.

Further, the descriptions of PUT "set the seed value" and GET "set . . . to the current value of the seed" would appear to specify that:

-- the processor must accept whatever is specified for a seed
value when some values, 0 for example, are often not suitable

-- PUT and GET can be used for global communication; assigning a
value with PUT and later retrieving that same value with GET.

1. Is the following what was intended for GET?

must be a default integer array of rank one and size >= N. It is an INTENT(OUT) argument. Denoting this array by "a" and the current value of the seed by "s", TRANSFER (SOURCE=s, MOLD=a) is assigned to "a".

TRANSFER hides both type and shape, assigning not the current value of the seed but the physical representation of that value independent of shape. Thus the seed can be of any type and any shape.

Alternately

must be a default integer array of rank one and size >= N. It is an INTENT(OUT) argument. It is assigned the physical representation of the seed value in a processor dependent manner.

2. PUT semantics are more complicated in that the argument specified for PUT may not always be suitable as a seed. Thus while the assignment semantics currently specified may not have been intended, neither is it appropriate to specify TRANSFER semantics.

Are PUT semantics processor dependent? This would allow processor seeds of any type and shape.

3. Given a value specified for PUT, can processors alter that value for use as a seed?

ANSWER: 1. Yes, for GET "TRANSFER" semantics were intended. However the existing wording in Fortran 90 implies that the integer array value itself is to be called a seed, regardless of the internal interpretation. EDIT 2 clarifies this.

2. Yes, specifying PUT semantics as processor-dependent is correct. This does not mean that seeds of any type and shape are allowed.

3. Yes, the value supplied by PUT may be altered in constructing a seed.

Discussion:Edits provided in defect item 000148 describe the operation of RANDOM_SEED and RANDOM_NUMBER. Edits to the description of RANDOM_SEED arguments PUT and GET are provided here.

EDIT:
1. In 13.13.84, RANDOM_SEED, PUT argument, replace the last sentence (beginning "It is used by the processor . . .") with [228:9]:

"It is used in a processor-dependent manner to compute the seed value accessed by the pseudorandom number generator."
2. In 13.13.84, RANDOM_SEED, GET argument, replace the last sentence (beginning "It is set by the processor . . .") with [228:12]:

"It is assigned the current value of the seed."

SUBMITTED BY: Dick Weaver
HISTORY: 94-142r1 m129 submitted, approved u.c.
94-221 m130 X3J3 ballot, failed 21-2
94-324r1 m131 revised response, approved 16-1 95-034r1 m132 X3J3 ballot, approved 19-1 HOLD for defect item 000148
95-155 m133 revised response, approved 12-1 95-183 m134 X3J3 ballot, failed 16-2
95-281 m135 revised response and 2nd edit, WG5 approved (N1161)

NUMBER: 000183
TITLE: Unambiguous procedure overloading KEYWORDS: generic interface, interface - generic, argument - dummy DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: The standard considers (14.1.2.3) the following example to be ambiguous:

```
INTERFACE BAD
SUBROUTINE S1(A)
END SUBROUTINE
SUBROUTINE S2(B,A)
END SUBROUTINE
END INTERFACE ! BAD
```

because it requires a single argument which disambiguates both by position and by keyword (A of S2 disambiguates by position but not by keyword; B of S2 disambiguates by keyword but not by position).

Note that the above is the simplest example of unambiguous overloading which the standard disallows; other cases exist where the number of nonoptional arguments is the same, e.g.

```
INTERFACE DOUBLE_PLUS_UNGOOD
SUBROUTINE S1(I,P,A)
END SUBROUTINE
SUBROUTINE S2(J,A,I)
END SUBROUTINE
END INTERFACE
```

where S2 takes two nonoptional integer arguments and S1 takes one nonoptional integer argument, but there is still no single argument which disambiguates between them.

A third example shows an (seemingly forbidden) unambiguous overloading where the number of arguments of each data type are the same:

```
INTERFACE BAD3
SUBROUTINE S1(I,J,A,B)
REAL I
INTEGER B
END SUBROUTINE
SUBROUTINE S2(J,I,B,A)
REAL I
INTEGER A
END SUBROUTINE
END INTERFACE
```

Was the overly strict nature of the disambiguation rules an unintentional oversight?

ANSWER: Yes. A change is needed to the rules in 14.1.2.3 to avoid rejecting examples such as the first two examples shown. While the first two examples can be considered to have been incorrectly rejected by the text in the standard, the text necessary to avoid rejecting the last example is sufficiently complicated to indicate that it was specifically not included in the standard. Future versions of the standard may consider extending the language to include the third example as standard conforming.

The change is to allow a difference in the number of arguments of each data type to disambiguate overloads.

Discussion: To allow the third example it would be necessary to have text that would place an ordering relationship between the positional disambiguators and keyword disambiguators.

Consider the following code fragment, which includes an ambiguous call. This code fragment must be nonstandard conforming.

```
INTERFACE AMBIGUOUS
SUBROUTINE S1(I,B,J,A)
END SUBROUTINE
SUBROUTINE S2(K,I,A,J)
REAL:: I
END SUBROUTINE
END INTERFACE
CALL AMBIGUOUS(3, 0.5, A=0.5, J=0) ! Cannot tell which of S1 or S2 to
! call
```

EDITS:
1. In section 14.1.2.3, in the third paragraph [242:38]

change "and at least one of them must have a nonoptional dummy argument that" to "and
(1) one of them has more nonoptional dummy arguments of a
particular data type, kind type parameter, and rank than the other has dummy arguments (including optional dummy arguments) of that data type, kind type parameter, and rank; or
(2) at least one of them must have a nonoptional dummy argument
that"

and indent numbers (1) and (2) to be a sublist of list item (2); changing them to be (a) and (b) respectively.

2. In section 14.1.2.3, in the text after the example [243:10-11]
change "(1)" to "(2)(a)" twice
change "(2)" to "(2)(b)" twice

SUBMITTED BY: Malcolm J. Cohen
HISTORY: 94-281r1 m130 submitted with proposed response, approved u.c.
94-306 m131 X3J3 ballot failed 17-2
94-359r1 m131 revised answer to only change conformance of first
example; approved u.c.
95-034r1 m132 X3J3 ballot approved 19-1, with edit 95-281 m135 revised text in

question (including 3rd example),
revised response to indicate 2nd example will be valid.
WG5 approved (N1161).

NUMBER: 000187
TITLE: TARGET attribute, storage association, and pointer association
KEYWORDS: TARGET attribute, association - storage, COMMON block,
association - pointer
DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: Defect item 92, the response to which has been approved by WG5
and is now in the "B" section of interpretation requests, contains an answer that
is desirable but there is no text in the standard that supports the response. X3J3
was no doubt considering the following text from section C.5.3 as the base text for
the response but (1) this text is in an appendix, not in the body of the standard,
and (2) the text is flawed:

"The TARGET attribute ... is defined solely for optimization purposes.
It allows the processor to assume that any nonpointer object not explicitly
declared as a target may be referred to only by way of its original declared name.
The rule in 5.1.2.8 ensures that this is true even if the object is in a common block
and the corresponding object in the same common block in another program
unit has the TARGET attribute."

The only part of 5.1.2.8 that could reasonably be considered the "rule" to which
C.5.3 refers is:

"An object without the TARGET or POINTER attribute must not
have an accessible pointer associated with it."

This "rule" does not seem to provide the insurance mentioned in C.5.3. Rather, it
seems that this sentence exists to clarify the "may" in the first sentence of 5.1.2.8.
That is, it seems to be just reiterating that the following is not standard
conforming:

```
INTEGER I
INTEGER, POINTER :: P
P => I
```

In actuality, there is no way that a pointer can become *pointer associated* with
an object that does *not* have the TARGET (or POINTER) attribute. The
confusion seems to arise when an object with the TARGET attribute is storage
associated with an object that does not have the TARGET attribute.

What, then, is the meaning of the sentence in 5.1.2.8 cited above?

ANSWER: The sentence from 5.1.2.8 was intended to prohibit a nontarget object from being referenced both via a pointer and via the object's name within a single scoping unit but, it fails to do so. Edits are provided to add the prohibition alluded to in C.5.3.

Discussion: The following example is provided to illustrate the problem and clarify the edits:

```
PROGRAM  MAIN_PROG

!        Variable M (a member of blank common) does not have the TARGET
!        attribute.

INTEGER M
COMMON M
INTEGER, POINTER :: P

INTERFACE
SUBROUTINE  SUB(P)
INTEGER, POINTER :: P
END SUBROUTINE
END INTERFACE

CALL SUB(P)
M = -1
PRINT *, "M = ", M
PRINT *, "P's target = ", P
END


SUBROUTINE  SUB(P)
INTEGER, POINTER :: P

!        Variable M (a member of blank common) has the TARGET attribute.

INTEGER, TARGET :: M
COMMON M
M = 100
P => M
END
```

In the main program, the storage unit represented by M and P is accessible by two different methods: the variable name M and the pointer P. The text in C.5.3 is intended to prevent this multiple accessibility but the sentence it is referencing in 5.1.2.8 is not relevant with respect to this example. Pointer P is not pointer associated with variable M in the main program. This could be demonstrated by adding the statement

```
PRINT *, ASSOCIATED(P, M)
```

to the main program but this statement would be invalid because M has neither the POINTER nor TARGET attribute in that scoping unit.

Since there is no text in 5.5.2.3 that states that if an item in a common block has the TARGET attribute, it may correspond only with another item (in another declaration of the common block) that also has the TARGET attribute, the edits add this rule.

Note that defect item 160 also quotes this same passage from C.5.3 in its question. That defect item resulted in the addition of a constraint that prohibits an object in an EQUIVALENCE list from having the TARGET attribute. Without this prohibition, there could again possibly be more than one avenue of reference to a data object in a single scoping unit. In the common block case, however, it is desirable to allow variables with the TARGET attribute so the edit adds the rule stating if a variable in a common block has the TARGET attribute, any corresponding variable in another instance of the common block with the same name must also have the TARGET attribute.

REFERENCES: ISO/IEC 1539:1991 5.5.1 (as modified by defect item 160) EDITS:

1. Delete the second sentence of 5.1.2.8 [48:16-17].

2. Insert as the (new) last paragraph of 5.5.2.3 [59:42+]:

"An object with the TARGET attribute must become storage associated
only with another object that has the TARGET attribute."

3. Delete the fourth sentence of C.5.3 [269:23-25].

SUBMITTED BY: Larry Rolison
HISTORY: 94-299r1 m131 submitted, with proposed response, passed 12-1
95-034r1 m132 X3J3 ballot, failed 14-6
95-281 m135 revised text in question and answer. WG5 approved (N1161)

NUMBER: 000196
TITLE: Inaccessibility of intrinsic procedures KEYWORDS: intrinsic procedure, INTRINSIC attribute, generic identifier,

names class
DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION: Section 14.1.2 states:

"Note that an intrinsic procedure is inaccessible in a scoping
unit containing another local entity of the same class and having the same name.
For example, in the program fragment

```
SUBROUTINE SUB
...
A = SIN (K)
...
CONTAINS
FUNCTION SIN(X)
...
END FUNCTION SIN
END SUBROUTINE SUB "
```

Are the following two comments about this text correct?

(1) The example is not strictly correct because the resolution of the procedure reference "SIN" depends on the contents of the first "...":

(1a) If "..." does not contain an "INTRINSIC SIN" statement, the behavior is as specified: In SUB, the name SIN is established specific due to condition 14.1.2.4 part (2b), it is not established generic, and the internal function SIN is referenced due to 14.1.2.4.2 part (3).

(1b) If "..." does contain an "INTRINSIC SIN" statement, SIN is established specific as above, but also established generic due to condition 14.1.2.4 (1b). So the reference is resolved according to 14.1.2.4.1 part (2): the intrinsic function SIN is called. (At least if there is a suitable specific function for data) (object K. If not, the reference is resolved according to) (14.1.2.4.1 (4) which also requires a consistent reference.)

(2) The first sentence of the cited text is wrong (incomplete), because it does not consider the case of generic identifiers:

* Intrinsic procedures are local entities of class (1). * Generic identifiers are local entities of class (1). * Various instances in the standard indicate that it is possible to extend the generic interface of intrinsic procedures.

Consequently, in the example

```
MODULE my_sin
CONTAINS
LOGICAL FUNCTION lsin (x)
LOGICAL, INTENT(IN) :: x
...
END FUNCTION lsin
END MODULE my_sin

SUBROUTINE sub
USE my_sin
INTERFACE SIN
MODULE PROCEDURE lsin
END INTERFACE SIN
...
END SUBROUTINE sub
```

the intrinsic procedure SIN remains accessible in SUB although that scoping unit contains another local entity of class (1) named SIN.

ANSWER: Comment 1 is incorrect. See the answer to (1b).

Comment 1a is correct.

Comment 1b is incorrect.

SIN is a local name for the internal procedure, which is a specific procedure, and adding an "INTRINSIC SIN" statement is prohibited by 14.1.2, 3rd paragraph.

Comment 2 is correct. Edits to remove the contradiction are included below.

EDITS:
1.In section 14.1.2, 4th paragraph [241:36], change
"having the same name" in the first sentence to
"having the same name, except when the other local entity
and the intrinsic are both generic procedures"

2.In Section 14.1.2, 3rd paragraph [241:33], change
"in the case of"
to
"when both are"

SUBMITTED BY: Michael Hennecke (hennecke@rz.uni-karlsruhe.de) HISTORY:
95-252 m135 submitted
95-281 m135 response WG5 approved (N1161)

NUMBER: 000201
TITLE: SELECTED_REAL_KIND result
KEYWORDS: SELECTED_REAL_KIND, intrinsic, result DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION:The Result Value portion of the description of SELECTED_REAL_KIND states in part:

"The result has a value equal to a value of the kind type parameter of a real data type with decimal precision, as returned by the function PRECISION, of at least P digits ..., or if no such kind type parameter is available on the processor, the result is -1 if the precision is not available..."

When SELECTED_REAL_KIND is invoked with a P value of -1, some vendors return a positive value as the result (almost certainly because they've focused on the phrase "of at least P digits" in the above quote from the standard) and some vendors return -1 (almost certainly because they've focused on the phrase "the result is -1 if the precision is not available" in the above quote from the standard).

Question 1:Is either one of these results "more correct" than the other ?

Question 2:Are both answers acceptable?

ANSWER 1:Yes. For this example, the processor should return a positive value.

ANSWER 2:No.

Discussion:The phrase "if the precision is not available" in section 13.13.93 of the standard is confusing. Edits are supplied to clarify the intent.

EDIT(S):In section 13.13.93, in the paragraph prefaced with "Result Value:" [232:6-7]
change "if the precision is not available"
to "if the processor does not support a real data type with a precision greater than or equal to P"

and change "if the exponent range is not available"
to "if the processor does not support a real data type with an exponent range greater than or equal to R"

SUBMITTED BY: Larry Rolison
HISTORY: 95-175 m134 submitted
95-281 m135 response, approved by WG5 (N1161)

NUMBER: 000203
TITLE: Scope of operator / assignment symbols KEYWORDS: intrinsic / defined operators / assignment, local / global entities DEFECT TYPE: Erratum
STATUS: WG5 approved; ready for X3J3

QUESTION 1: Section 14.4 (Scope of operators) states that
"The intrinsic operators are global entities." and that
"A defined operator is a local entity.".

But a defined-operator (R311) may be an extended-intrinsic-operator (R312) which in turn is an intrinsic-operator (R310). This means that if intrinsic operators are global entities, so are at least some of the defined operators, so the second quotation given above cannot be true. Is this correct ?

PROPOSED EDIT: Add " that is not an extended intrinsic operator"
after "A defined operator" in [F90,245:30].

QUESTION 2: In Section 14.4, the sentence "Within ..." applies to the first sentence only. Defined-operators (other than extended-intrinsic-ops) have no intrinsic meaning and so "additional operations" makes no sense. Is this correct ?

PROPOSED EDIT: Move sentence 2 to the end of 14.4.

QUESTION 3: Section 14.5 (Scope of the assignment symbol) does not address the replacement of the intrinsic derived type assignment operation, as defined in 7.5.1.2 and 12.3.2.1.2. Is this correct ?

PROPOSED EDITS: Add ", or replace the intrinsic derived type assignment operations" after "operations" in [F90,245:34].

QUESTION 4: The entities under consideration in 14.4 and 14.5 are the operator and assignment SYMBOLS. These are to be distinguished from the associated OPERATIONS in the same way that a generic name is to be distinguished from the specific names it comprises. Is this correct ?

PROPOSED EDITS: in [F90,245], replace:
* line 29: "operators" => "operator symbols" * line 30: "operators" => "operator symbols" * line 30: "operator" => "operator symbol" * line 31: "operator" => "operator symbol" * After the "Within ..." sentences (lines 30,33), add
"The procedures corresponding to these operations
are local entities."
(The intrinsic operations have no "specific" entities associated
with them and so need not be mentioned. If they had, these would be global entities.)

ANSWERS:
ANSWER 1: Yes. An edit is supplied below to clarify that some defined operators are indeed global entities.

ANSWER 2: Yes. Although the word "additional" only applies to the first sentence, the proposed edit does not substantially improve the clarity, and the existing text is not in error.

ANSWER 3: Yes. An edit is supplied below which adds the case of redefining the assignment operation when the two arguments have the same derived type.

ANSWER 4: Yes. However, the current use of the words "operator" and "operations" already makes this distinction in a sufficiently precise manner.

EDITS:

1.In Section 14.4, after "A defined operator" [245:30], add
"that is not an extended intrinsic operator"

2.In Section 14.5, after "operations" [245:34], add
", or replace the intrinsic derived type assignment operation"

SUBMITTED BY: Michael Hennecke (hennecke@rz.uni-karlsruhe.de) HISTORY:
95-254 m135 submitted
95-281 m135 response, WG5 approved (N1161)