Subject: Concerning deferred type parameters, including issues 78, 79, 134, 135, 136, 137, 138, 140, 141
From: Van Snyder

# 1 Edits

Edits refer to 99-007r1. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by immediately following text, while a page and line number followed by + indicates that immediately following text is to be inserted after the indicated line. Remarks for the editor are noted in the margin, or appear between [ and ] in the text.

## 1.1 Problems not addressed by editor's issues

The deferred parameters of the result of NULL() are undefined. 138:15+

| Is this needed if item 4 of the proposed new section 14.8 advocated to address item 134 below is accepted? | *Question to J3* |
|---|---|

Replace "with the same shape." by ". If it is an array, it is allocated with the same bounds. If it has deferred type parameters, the values of type parameters of the component of *expr* are used for the values of corresponding parameters of the component of *variable*." 157:42

At the invocation of the procedure, a dummy procedure pointer becomes disassociated if it has INTENT(OUT). If it does not have INTENT(OUT) then it receives the pointer association status of the actual argument and, if the actual argument is currently associated, the dummy procedure pointer becomes associated with the same target. 279:32+

If a dummy procedure pointer is a function procedure pointer for which the result type has deferred type parameters, the corresponding actual argument procedure pointer shall have a result type that has deferred corresponding parameters.

| This would not be needed if dummy procedure pointers were considered to be dummy data objects, not dummy procedures. | *Note to J3* |
|---|---|

(7) If it has deferred parameters, they shall not be the subject of parameter inquiry. 281:13+

## 1.2 Issues 78 and 141

Constraint: If *pointer-object* is a data object, all deferred or assumed parameters of *target* shall correspond to deferred parameters of *pointer-object*. 159:33+

| Otherwise, a run-time check is required to make sure the nondeferred parameter value of *pointer-object* is the same as the deferred or assumed parameter value of *target*. | *Note to J3* |
|---|---|

[Editor, replace starting with "If *pointer-object*".] 160:12-15

The definition status of each deferred parameter of *pointer-object* is assumed from the definition status of the corresponding parameter of *target*. If parameters of *target* that correspond to deferred parameters of *pointer-object* are defined, their values are assumed by corresponding parameters of *pointer-object*.

| | |
|---|---|
| I couldn't find whether "definition status" includes value. Does anybody know if "definition status" includes value? If so, the second sentence above isn't needed. Or, is it OK to say "definition status and value are assumed..." with the understanding that the value isn't assumed if the definition status is "undefined." | *Question to J3* |

[Editor: delete through "related" if this paper adequately addresses this part of issue 78.] 160:18-24

[Editor: delete if this paper adequately addresses issue 141.] 160:32:48

### 1.3 Issue 79

[Editor: Change the period that ends the sentence to a colon and replace "That is..." by the following:] 78:34-36

- If the actual argument is a pointer, a reference to the associated dummy data object or dummy function procedure pointer may occur if the actual argument pointer is associated with a target.
- If the actual argument is allocatable, a reference to the associated dummy data object may occur if the actual argument is allocated.
- A reference to the dummy data object may occur if the associated actual argument is defined.
- Inquiry about a type parameter of the dummy data object may occur if the corresponding type parameter of the associated actual argument is defined.
- The dummy data object, its pointer association status and deferred type parameters, or allocation status and deferred type parameters may be changed if the associated actual argument is definable.

[Editor: delete.] 276:39-41

[Editor: delete if this paper adequately addresses issue 79.] 277:1-16

[Editor: Add the following after "target.":] 277:37

Deferred or assumed type parameters of the dummy argument become associated with corresponding type parameters of the actual argument.

### 1.4 Issue 134

A **deferred type parameter** is a nonkind type parameter for which an expression to calculate a value is not specified in the declaration of an object. A type parameter is indicated to be deferred by using a colon in a type declaration statement or a component definition statement. 32:34+

Values of deferred type parameters of an object become defined or undefined as specified in 14.8.

[Editor: delete. Was in the wrong place, anyway.] 32:41-33:19

[Editor: delete if this paper adequately addresses issue 134.] 33:8-19

[Editor: Add a new section 14.8:] 379:11+

**14.8 Definition and undefinition of deferred type parameters**

A deferred type parameter of an object may be defined or may be undefined and its definition status may change during execution of a program. An action that causes a deferred type parameter to become undefined does not imply that it was previously defined. An action that causes a deferred type parameter to become defined does not imply that it was previously undefined.

The definition status of deferred type parameters is changed by the following events:

1. Successful execution of an ALLOCATE statement or allocation of an allocatable component during intrinsic assignment (7.5.1.5) causes deferred type parameters of the allocated object to become defined.

2. Execution of a pointer assignment statement causes the values of deferred parameters of *pointer-object* and its ultimate components to assume the definition status of corresponding parameters of *target* and its ultimate components. This includes pointer assignments that result from intrinsic assignment of objects of derived type that have components with the POINTER attribute.

3. Execution of a NULLIFY statement causes deferred parameters of *pointer-object* and its ultimate components to become undefined.

4. Events that causes a pointer to become disassociated (14.6.2.1.2) or undefined (14.6.2.1.3) cause the deferred type parameters of the pointer and its ultimate components to become undefined.

5. Deallocating (6.4.3) an allocatable object causes the deferred type parameters of the object and its ultimate components to become undefined. This includes deallocation of an allocatable component of a derived type object during intrinsic assignment (7.5.1.5).

6. Reference to a procedure causes the deferred type parameters of dummy arguments that do not have INTENT(OUT), and their ultimate components, to assume the same definition status as corresponding type parameters of corresponding actual arguments and their ultimate components.

7. Any change in the definition status of deferred type parameters of an object, or the object's ultimate components, causes the same change in the corresponding deferred type parameters of an associated object of the same type and its ultimate components.

| | |
|---|---|
| If the change suggested for VALUE in 99-133 is accepted, add "other than a dummy arugment that has the VALUE attribute" after the first "object" in the last item above. | *Note to J3* |

## 1.5 Issue 135

[Editor: The conjecture in issue 135 was correct: The change was an attempt to get rid of "only," to which the editor has expressed some revulsion. The old wording doesn't work, either, for the same reasons as those mentioned in issue 137. Maybe this is correct:] 58:8-11

Constraint: An asterisk may be used as a *type-param-value* only in the *type-spec* that declares the type of a dummy argument.

Constraint: A colon may be used as a *type-param-value* only in the *type-spec* that declares the type of an entity or component that has the POINTER or ALLOCATABLE attribute.

[Editor: delete through "Also" on 58:24.] 58:14-24

### 1.6 Issue 136

[Editor: Replace "other than within an ALLOCATE statement" by "within a *type-declaration-stmt* (5.1)."] 58:34-35

[Editor: delete if this paper adequately addresses issue 136.] 58:36-43

### 1.7 Issue 137

[Editor: new stuff underlined] 75:31-32
It may be used as a parameter of a *type-spec* that explicitly specifies the type of *allocate-objects* in an ALLOCATE statement....

[Editor: delete if this paper adequately addresses issue 137.] 75:34-38

### 1.8 Issue 138

Deferred type parameters of the result type of a function procedure pointer are parameters of the type of the result value of functions that are targets of the procedure pointer, not type parameters of the procedure pointer. It is therefore not possible to inquire the values of deferred parameters of function procedure pointers. 88:1-2

Note $5.21\frac{1}{2}$

It is possible to inquire the values of deferred parameters of values that result from function execution, provided they are defined.

[Editor: delete if this paper adequately addresses issue 138.] 88:3-25

[Editor: delete] 160:32-36

Deferred type parameters of the result type of a function are parameters of the type of the result value of the function, not type parameters of the function. It is therefore not possible to inquire the values of deferred parameters of functions. 286:17+

Note $12.31\frac{1}{2}$

It is possible to inquire the values of deferred parameters of values that result from function execution, provided they are defined.

### 1.9 Issue 140

A deferred type parameter of a disassociated pointer, of a function procedure pointer, of a pointer with undefined association status, or of an unallocated variable shall not be the subject of a type inquiry. 108:22-23

[Editor: delete if this paper adequately addresses issue 140.] 108:24-33