Subject: Comments on section 4
From: Van Snyder

1 Edits

Edits refer to 01-007r1. 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 (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

1.1 Minor syntax change in the procedure binding statement

It would be useful and harmless to allow several procedure bindings to be specified in a single statement. I don't use a *foobar-list* kind of syntax rule construction, because it would be difficult to tie in the constraint at [43:13].

■ [, *binding-name* [=> *binding*]] ... 43:12+

[Editor: "the binding is" ⇒ "all bindings are" (or "each binding is"?).] 43:14

1.2 Final subroutine specifications are stricter than necessary

In writing data types and procedures for a simple data structure, i.e. a doubly-linked list, it appeared that it might be useful to allow final subroutines to be the same as subroutines used for other purposes, and to that end, it might be useful to allow them to have optional dummy arguments. Also, it doesn't seem to hurt if the first argument of the type is optional, because it will, after all, be present when the subroutine is used for finalization. In fact, this would allow a final subroutine to be used for several types. Of course, all of this could be done with an additional layer of subroutines, but this adds to code bulk, and therefore adds to lifetime cost.

1.2.1 Specs

Allow a final subroutine to have any number of arguments, so long as at least one is of the type to which the subroutine is bound as a final subroutine, and all of the rest are optional.

1.2.2 Syntax

No changes are required in the syntax.

1.2.3 Edits

Constraint: Each *final-subroutine-name* shall be the name of a module subroutine. It shall have at least one dummy argument of the derived type being defined. The first of these shall be be a nonpointer, nonallocatable, nonpolymorphic argument; all of its nonkind parameters shall be assumed. It shall not have INTENT(OUT). The term "finalized argument" is defined in 4.5.10 to specify this argument. If the subroutine has additional arguments, they shall be optional. 43:31-35

Constraint: The rank and kind type parameters of the finalized argument (4.5.10) of a final subroutine shall not be the same as those for the finalized argument of another final subroutine specified for the same derived type.	43:38-40 *See §1.4*
The first dummy argument of a subroutine that is of the same type as the type to which the subroutine is bound as a final subroutine is the **finalized argument**. If a subroutine is bound to several types as a final subroutine, different dummy arguments will be finalized arguments in different contexts.	58:26+
[Editor: "dummy" ⇒ "finalized".]	58:28
[Editor: After "argument" insert "associated with the finalized argument, and any other arguments are not present" twice.]	58:30,32
[Subroutines don't have kind type parameters. Editor: "with" ⇒ "whose finalized argument has".]	58:31 *See §1.4*
finalized argument (4.5.10) : The first dummy argument of a final subroutine that is of the same type as the type to which the subroutine is bound as a final subroutine. If a subroutine is bound to several types as a final subroutine, different dummy arguments will be finalized arguments in different contexts.	402:4+

1.3 It is a mistake that enumerations are aliases

Some of the advantages of enumerations being new types, not aliases for integers, are described in section 4 of 98-171r2. Other advantages that are not therein described include the possibility to use enumerators for input and output.

We should leave room for the possibility that the authors of a future revision of the standard may conclude that defining enumerations to be aliases was a mistake, and wish to define enumerations that *are* new types, with differently spelled syntax. A natural dichotomy would be that ENUM introduces a new type, and ENUMALIAS introduces an alias. It would be inconsistent with the syntaxes for types and type aliases to keep ENUM meaning "introduce an alias" and expect a future committee to come up with something different, say NEWENUM, to introduce a new one. On the other hand, it would preserve what in the 1980's was called "The beloved FORTRAN tacked-on look." Instead, the syntax should be changed now so as to make it explicit that enumerations, as presently defined, are aliases.

Although it is possible and reasonable that the authors of a future revision of the standard may prefer a syntax such as

```
TYPE, ENUM :: type-name
  enumerator-def-stmt
END TYPE [ type-name ]
```

it would be unkind to that committee to paint them into a corner now.

[Editor: "enum" ⇒ "ENUMALIAS".]	14:39
[Editor: "ENUM" ⇒ "ENUMALIAS" twice.]	60:22,23
[Editor: "ENUM" ⇒ "ENUMALIAS" twice.]	60:26,29
[Editor: "ENUM" ⇒ "ENUMALIAS" four times.]	62:3,5,6,9
[Editor: "ENUM" ⇒ "ENUMALIAS".]	462

1.4 Miscellaneous edits

[Editor: Insert a space between " ■ " and "[".] 40:41

[Editor: Insert "(4.6)" after "alias" because it's a forward reference.] 41:7

Constraint: The same *type-attr-spec* shall not appear more than once within a single *derived-type-stmt*. 41:4+

[Editor: Add these instances of "DIMENSION", "ALLOCATABLE" and "*access-spec*" to the index. The instance of "POINTER" on the previous line is already in the index.] 41:40-42

[Editor: There is only one *declaration-type-spec*: "a *declaration-type-spec*" ⇒ "the *declaration-type-spec*" twice.] 42:6,9

[Editor: Delete "statement" (compare to style at [43:8]).] 43:3

[In two constraints on final subroutine names, the type to which the terms "that type" and "that derived type" refers is unclear.]

[Editor: Delete "derived" (for consistency with the previous constraint).] 43:40

[We seem to have lost the constraint on PASS_OBJ for type-bound procedures that is parallel to the one at [42:40-43] for procedure pointer components.] 43:44+

Constraint: If PASS_OBJ appears, the interface specified by *abstract-interface-name* or the procedure specified by *binding* shall have a scalar, nonpointer, nonallocatable dummy variable of type *type-name*. The first such dummy variable shall be polymorphic if and only if *type-name* is extensible.

[Pointers don't "point to". Editor: "to" ⇒ "of".] 45:24

[Editor: Insert a comma after "TODAY".] 46:31

[Editor: The first "status" ⇒ "association status (5.1.2.11)".] 47:37

[Editor: "in" ⇒ "by".] 48:11

Note 4.31$\frac{1}{2}$ 49:24+

If a module procedure is bound to several types as a type-bound procedure, different dummy arguments might be the passed-object dummy argument in different contexts.

[Editor: "parant" ⇒ "parent".] 57:11

[Editor: Delete "PV".] 57:17

[This sentence appears to have no connection to anything previous or subsequent in this or nearby subclauses, nor does it contribute anything on its own. It's substance is adequately covered in section 12. Editor: Delete.] 58:25

[Subroutines don't have kind type parameters. Editor: "with" ⇒ "whose dummy argument has".] 58:31

[Prevent circular dependence among type aliases:] 59:42+

Constraint: A *declaration-type-spec* shall specify an intrinsic type or a previously defined derived type.

R458 *array-constructor* **is** (/ *ac-spec* /) 62:29-31
or *left-square-bracket ac-spec right-square-bracket*

R458$\frac{1}{2}$ *ac-spec*	**is** *type-spec* :: **or** [*type-spec* ::] *ac-value-list*	
[Now handled by revised syntax rules. Editor: Delete.]		62:34
[The constraint seems to be requiring only that all of the *ac-value* expressions have the same type and type parameters as each other. The intent is almost certainly that they have the same type and type parameters as specified by the *type-spec*. Editor: "of the same" ⇒ "of that"; after "values" insert "as specified by *type-spec*".]		63:5-6
[Duplicates [32:38-39]. Editor: Delete.]		69:39-40

2 Potential problems with crappy edits or no edits offered

1. Assignment isn't an operation. 2. Intrinsic assignment isn't defined for polymorphic objects.	32:6
It requires substantial sophistication to understand that "execution of a derived-type intrinsic assignment statement" determines the values of deferred type parameters only indirectly. It would be better to delete it.	32:43-44
Intrinsic assignment is not defined for polymorphic objects.	33:15
This paragraph isn't connected to anything previous or subsequent in this subclause, and it contributes nothing on its own. Its substance belongs in 13.11.100.	37:43-45
The sentence "Pointers ... undefined" belongs in 5.1.2.11, where its substance isn't even mentioned!	47:35-36
Move to 4.5.1.5$\frac{1}{2}$ (to be with other type-bound procedures), or move 4.5.1[56] to be just before this subclause.	51:28-41
Subclause 4.5.11 should be 4.5.10.1.	58:41
Subclause 4.5.12 should be 4.5.10.2.	59:23