Subject: Editorial comments
From: Van Snyder

1 Introduction

As one might expect, there are still problems in the draft.

2 Edits that are unavoidable or that ought not to be substantive

Edits refer to 02-007r3. 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 associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

[Editor: "CMPLX function" ⇒ "function CMPLX" (for consistency with the rest of the paragraph).] 35:16

[Editor: Insert " The " after "(R534)".] 86:7

[C576 says that "a *proc-entity-name* shall also be declared in a *procedure-declaration-stmt*." But at [260:25] the declared thing-o is called a *procedure-entity-name*. Editor: "*proc-entity-name*" ⇒ "*procedure-entity-name*" twice.] 89:4-5

[The phrase "the derived-type definition of the declared type" makes inheritance association worthless. The components inherited from the parent type *are* components of the declared type, but they're gotten by inheritance association, rather than by being declared in its derived-type definition. Editor: Delete "the derived-type definition of".] 103:5

[Mostly doesn't belong here because it mostly has nothing to do with type, type parameters and shape. The part about type, type parameters and shape is at [123:31-32], which implicitly refers to 5.1.0 and 5.1.1, [32:22-26] and [77:21-22]. The part about reference doesn't belong here, but it is at [101:2-4]. A primary's value isn't tied in to expressions, however, so we'll do that in 7.1.8.1. The part about a pointer not being associated with a target is wrong because a primary isn't what appears in those contexts, which are correctly described at [81:24-25], [143:29-30] and [265:24], and if this stuff did belong here and we wanted to repeat it, it would be misleadingly incomplete because it doesn't account for inquiry functions. [124:11-12] can't possibly be related to primaries because a disassociated pointer or an unallocated allocatable can't be used as a primary. The parts about having rank but no shape are covered in 5.1.2.5.3. The part about the NULL intrinsic is covered in 13.7.84.] 124:3-12

NOTE 7.8$\frac{1}{2}$

If a primary is a pointer it shall be associated with a target, and the target is referenced.

[Unlike Note 7.13$\frac{1}{2}$, this note is to remind the reader that the type, type parameters and shape come from the pointer's target.]

[The type, type parameters and shape of a primary are tied into an expression by 7.1.4.1, but nothing ties the *value* of a primary into an expression. This seems like a good place to do it. This will also hook Section 6 to an expression, because Section 6 begins with "The appearance of a data object designator in a context that requires its value. . . ." Nothing specifies what *type-param-inquiry* and *type-param-name* mean where they appear as primaries.] 129:11-New ¶

If an operand is a primary, the value of the operand is the value of the primary. If the primary is a data object designator its value is as specified in 2.4.2 and 2.4.3. If it is a function reference, its value is as specified in 2.4.3. If it is a *type-param-inquiry* or a *type-param-name* its value is the value of the specified type parameter (4.2, 6.1.3). If it is an expression enclosed in parentheses its value is the value of the expression.

[I don't think we do an adequate job of specifying that the syntax of a primary denotes its value, but Malcolm contends the first sentence is a mere sophistry, and that 6.1.3 does an adequate job for a

type-param-inquiry or a *type-param-name*.]

NOTE 7.13$\frac{1}{2}$

If a primary is a pointer it shall be associated with a target, and the target is referenced.

[Unlike Note 7.8$\frac{1}{2}$, this note is to remind the reader that the value comes from the pointer's target. I'm not comfortable with the description of a function result in 2.4.3 in the case of a function result that's a pointer. Does it hang together?]

[The whole of subclause 7.4.1.3 is about intrinsic assignment. Editor: Delete "in an intrinsic assignment" twice. We didn't find it to be necessary at [139:28].] 140:1,3

[We don't want to force these events if they're unnecessary – just the appearance that they happen. Editor: Insert "result is as if the" before "following"; "is' ⇒ "were". Malcolm says this is unnecessary: If a standard-conforming program cannot tell the difference, there is no difference.] 141:9

[The "or a procedure pointer" part of C727 makes it clear we intended to allow `pointer1 => pointer2` where both pointers are procedure pointers, but it doesn't work because the syntax of *procedure-name* is too limited: There's no explicit rule for *procedure-name*, so it's covered by R102 and R304.]

or *proc-pointer-name* 143:23+
or *data-ref % procedure-component-name*

C726$\frac{1}{2}$ (R741) The *procedure-component-name* shall be the name of a procedure pointer component of the declared type of *data-ref*. 143:24+

[Editor: Insert "or" before "a" at [143:25] and delete ", or a procedure pointer" at [143:26].] 143:25-26

[The strategy of defining "interface" by first defining "abstract interface" and then using that concept to define the interface of arbitrary entities that have explicit interface was changed at meeting 162. This instance of "abstract interface" is an oversight. Editor: "abstract interface" ⇒ "characteristics".] 253:14

[In order for keyword-identified arguments to work, the interface needs to include the argument keywords. Editor: After "name" insert ", the argument keywords".] 253:14

[An interface body specifies an explicit specific interface, not just an explicit interface. Editor: Insert "specific" after "explicit".] 255:42

[Editor: Insert a blank after the first comma.] 258:5

[Assignment isn't an operation. Editor: delete "the" and "operation".] 259:14

[Could be construed to make inheritance worthless. Editor: "in" ⇒ "declared within or inherited into" twice.] 263:11-12

[The result types of AMAX0, AMIN0, MAX1 and MIN1 aren't specifically given in 13.6. Are the kludges for these four functions good enough? They should be in a separate description that doesn't claim "AMAX0(...)" and "REAL(MAX(...))" are names.] 294:9-12+1

13.6 Names and characteristics of specific standard intrinsic functions

References to the specific standard intrinsic functions AMAX0, AMIN0, MAX1 and MIN1 are equivalent to references to generic functions shown in Table 13.1.

[The table number really will be 13.1 when it appears in 007.]

Table 1: **Characteristics of AMAX0, AMIN0, MAX1 and MIN1**

Specific Reference	Generic Reference	Argument Type	Result type
AMAX0(...)	REAL(MAX(...))	Default integer	Default real
AMIN0(...)	REAL(MIN(...))	Default integer	Default real
MAX0(...)	INT(MAX(...))	Default real	Default integer
MIN0(...)	INT(MIN(...))	Default real	Default integer

The result type of a specific function listed in Table 13.2 is the same as the result type of the corresponding generic function would be if it were invoked with the same arguments as the specific function.

[The table number really will be 13.2 when it appears in 007.]

Table 2: **Characteristics of specific standard intrinsic functions**

[Remove AMAX0, AMIN0, MAX1 and MIN1 from Table 13.2.

At [80:5], [143:26], [260:29], [263:38] and [268:6], replace "13.6" by "Table 13.2".]

[What's the point of identical names in columns one and two of the table in 13.6 where the row is marked with a bullet? The only cases where a specific intrinsic is important are where they are required to be one *not* marked with a bullet. Editor: Delete the rows for ICHAR, INT, LGE, LGT, LLE, LLT, and REAL from the table.] 295

[Why be coy about the type? And REAL only has one type parameter. Editor: "variable of the same type and kind type parameters" ⇒ "real variable with the same kind type parameter".] 340:10

["... would have if its value was... " is ungrammatical. It could be repaired with "was" ⇒ "were" but it's simpler to put it directly. Editor: "would have" ⇒ "has"; "were" ⇒ "is".] 340:10-11

[C doesn't use an asterisk to denote an unspecified size. One might be tempted to delete the asterisk, but then the remaining [ ] syntax doesn't specify a size, which is what the first phrase of the second clause specifies. Editor: Delete "or specifies a size of [*]".] 388:13

[Editor: "FORTRAN" ⇒ "FORTRAN".] 451:32

3 Edits that might be substantive

[Did we really intend not to allow a list on a type-bound procedure binding? If we want a list:] 44:21-22

■ [[, *binding-attr-list*] ::] *proc-binding-list*

R441$\frac{1}{2}$ *proc-binding* **is** *binding-name* [=> *binding*]

C444 If => *binding* appears, the double-colon separator shall appear before the *proc-binding-list*.

[Why do we allow several *allocate-object*s if a *type-spec* appears, but only one if SOURCE= appears? If we want to allow several:]

[Editor: "*allocation-list* ... which" ⇒ "each *allocate-object*".] 109:20-21

C302 (R623) If *source-variable* is not a scalar, every *allocate-object* shall have the same rank as *source-variable*. 109:22

[Editor: "*allocation*" ⇒ "each *allocate-object*".] 110:6

[Editor: Insert "each" after "to".] 110:9

[We don't provide for a procedure pointer to get its target from a binding. If this was an oversight:]

or *data-ref % binding-name* 143:23+

C726$\frac{1}{2}$ (R741) The *binding-name* shall be the name of an accessible specific binding declared within or inherited into the declared type of *data-ref*. 143:24+

If *proc-target* is *data-ref % binding-name* the *proc-pointer-object* becomes pointer associated with the dynamic binding (12.4). 144:26+