

Subject: Another whack at embedding conditionals in expressions etc.
 From: Van Snyder
 Reference: 03-258r1, section 2.8.1; 04-192

1 Number

2 TBD

3 Title

4 Another whack at embedding conditionals in expressions etc.

5 Submitted By

6 J3

7 4 Status

8 For consideration.

9 5 Basic Functionality

10 Allow to embed decisions within expressions, as actual arguments, or as pointer-assignment targets.
 11 Allow to compute whether an actual argument is present or absent.

12 6 Rationale

13 The syntaxes proposed below in the Detailed Specification section are a functional syntax of the form
 14 `IF(condition, true-result, false-result)`, and an operator-like syntax. It is intended that the semantics do
 15 not depend on the chosen syntax. The functional form is used in the examples here.

16 One sometimes needs to select one thing or another to use within an expression or as a pointer-assignment
 17 target. At present, for the former usage, one creates a temporary variable, sets that variable with an
 18 if-then-else construct, and then evaluates the expression using the temporary variable. It would be more
 19 convenient if one could embed the “use A or B depending on C” decision within an expression. These
 20 may be such things as “`IF (A > 0, log(A), -huge(0.0))`” or “`IF (present(A), A(:,i), B)`” where B is of
 21 rank one. In these cases, the value of one or the other of the outcomes is desired, but it’s important
 22 not to calculate the “wrong” one because it will raise an error condition. In cases such as “`p => IF (`
 23 `associated(A), A(:,i), NULL())`,” it’s important not to try to take the section A(:,i) if A is not associated.
 24 Even if A is associated, the value of A(:,i) is not needed.

25 One would like to be able to compute whether an actual argument that is to be associated with an
 26 optional dummy argument is to be considered to be present or absent. This isn’t a compelling desire in
 27 the case of one of these beasts, but for n of them, one needs a 2^n -way if-elseif...-else-endif construct with
 28 a different one of the 2^n possible combinations of present actual arguments in each branch. It would be
 29 more convenient if one could use an actual argument of the form “`IF(A, B)`” meaning “if A then B is
 30 the actual argument else the actual argument is absent.” In these cases, it’s important that B, not the
 31 value of B, is the actual argument — at least in the `INTENT([IN]OUT)` case. An example of this might
 32 be “`IF(present(A), A(:,i))`” meaning the actual argument is A(:,i) if A is present and it is absent if A is
 33 not present, or something similar with “present” replaced by “associated” or “allocated.”

34 One would like to be able to select one actual argument or another depending on some condition. The
 35 ways to do this are with a 2^n -way if-elseif...-else-endif construct for n arguments, or to have n sequences
 36 that associate a pointer with one or another of the desired arguments (or nullify it, if that was one of the
 37 outcomes). It would be more convenient to be able to write “`IF(A, B, C)`.” In these cases, it’s important
 38 that either B or C, not the value of one or the other, becomes the actual argument — at least in the
 39 `INTENT([IN]OUT)` case.

1 No matter whether a functional or operational syntax is chosen, the entity behaves somewhat differently
 2 from existing functions or operators, in that the “result” *is* one of the arguments/operands, not the value
 3 of one of them. That is, these entities behave more like run-time macro substitutions than functions or
 4 operators.

5 **7 Estimated Impact**

6 Small, both for standard and implementors.

7 **8 Detailed Specification**

8 **8.1 Functional syntax**

9 Provide two new intrinsic functions, named IF here but the particular names are not important. In both
 10 cases, the first argument is of type logical, and is evaluated before the function is “invoked.” In the
 11 three-argument case, the result is the second argument if the first is true, and the third argument if the
 12 first is false.

13 In the two-argument case, a reference to which is permitted only as an actual argument associated with
 14 an optional dummy argument, the result is the second argument iff the first is true, else it is an absent
 15 actual argument.

16 Notice that the specification carefully specifies “the result is ...,” not “the result is the value of ...”
 17 For all other functions, the result is an entity distinct from its arguments. For these functions, the result
 18 *is* one of the arguments. The “functions” behave more like run-time macro substitutions than functions.

19 **8.1.1 Illustrative edits w.r.t. 04-007, to indicate the scope of the proposed change**

20 C1220 $\frac{1}{2}$ (R1217) A reference to the two-argument form of the IF intrinsic function shall not appear 266:16+
 21 except as an actual argument corresponding to an optional dummy argument.

22 [Replace “it” by “any function other than the IF intrinsic function (13.7.51 $\frac{1}{2}$)”.] 276:3

23 **13.5.17 $\frac{1}{2}$ Conditional functions** 298:2+

24 IF (MASK, TSOURCE, FSOURCE)	Result is TSOURCE or FSOURCE, depending
	on MASK.
25	
26 IF (MASK, TSOURCE)	Result is TSOURCE if MASK is true, else result
27	is an absent actual argument.

28 **13.7.51 $\frac{1}{2}$ IF (MASK, TSOURCE, FSOURCE) or IF (MASK, TSOURCE)** 322:23+

29 **Description.** Embed a decision within an expression, or calculate whether an actual argument
 30 is present.

31 **Class.** Transformational.

32 **Arguments.**

33 MASK	shall be of type logical and shall be scalar.
34 TSOURCE	may be of any type, and may have any type parameter values. Shall be TKR compatible (5.1.1.2) with FSOURCE. It is not evaluated before the function is invoked. It may be undefined. If it is a pointer it need not be associated. If it is allocatable it need not be allocated.
35 FSOURCE	shall be TKR compatible with TSOURCE. It shall be polymorphic if and only if TSOURCE is polymorphic. It is not evaluated before the function is invoked. It may be undefined. If it is a pointer it need not be associated. If it is allocatable it need not be allocated.

36 **Result Characteristics.**

37 *Case (i):* Three arguments: The result characteristics are the same as TSOURCE if MASK
 38 is true, else the same as FSOURCE.

1 *Case (ii):* Two arguments: The result characteristics are the same as TSOURCE if MASK
2 is true, else the result is an absent actual argument.

3 **Result.**

4 *Case (i):* Three arguments: The result is the TSOURCE argument if the MASK argument
5 is true, else it is the FSOURCE argument. The result, and therefore the function
6 reference, may appear in a variable-definition context (16.5.7) if TSOURCE and
7 FSOURCE are permitted to appear in a variable-definition context.

8 *Case (ii):* Two arguments: The result is the TSOURCE argument if and only if the MASK
9 argument is true. If MASK is false the result is undefined, and the actual argu-
10 ment consisting of the function reference is absent. The result, and therefore
11 the function reference, may appear in a variable-definition context (16.5.7) if
12 TSOURCE is permitted to appear in a variable-definition context.

13 **Examples.**

14 *Case (i):* The result of IF (PRESENT(X), X, 0.0) is X if X is present, else it is 0.0.

15 *Case (ii):* The result of IF (ASSOCIATED(P), P(:,2), NULL()) is the array section P(:,2),
16 which is not a pointer, if P is associated, and NULL(), which is a pointer, if P is
17 not associated. Both are valid targets in a pointer assignment.

18 *Case (iii):* The result of IF (ASSOCIATED(P), P(:,2)) is a present actual argument that
19 is the array section P(:,2) if P is an associated pointer, else it is an absent actual
20 argument.

21 *Case (iv):* The result of IF (PRESENT(D), D(:,J)) is a present actual argument consisting
22 of the array section D(:,J) if D is a present dummy argument, else it is an absent
23 actual argument.

24 8.2 Operational syntax

25 Provide a distfix-operator-like syntax, wherein one operand is of type logical, and is initially evaluated,
26 and the result is one or the other of two remaining operands. The two remaining operands are not
27 evaluated as a consequence of their appearance within the distfix-operator syntax, but the selected one
28 might be evaluated if the context of the appearance of the distfix-operator syntax demands a value. The
29 syntax suggested in 04-192 was “A ? B : C”, but operators spelled like defined operators, e.g., “B .else. C
30 .if. A” or “A .picks. B .else. C” would work also.

31 Provide an infix-operator-like syntax, wherein one operand is of type logical, and is initially evaluated,
32 with the result being the other operand if the initially-evaluated one is true, and an absent actual
33 argument otherwise. The syntax suggested in 04-192 was “A ? B” but an operator spelled like a defined
34 operator, e.g., “B .if. A” would work also. In either case, B would not necessarily be evaluated even
35 if A is true. Instead, if A is true, B would be associated with the corresponding dummy argument.
36 Considering B to be an actual argument may or may not cause it to be evaluated. For example, if the
37 actual and dummy arguments are pointers, B is not evaluated, while if B is an expression — a more
38 complicated one than a *variable* — it is evaluated. But that is entirely dependent on the context of the
39 appearance of the “operator,” not the fact of the appearance of B as an “operand.”

40 Illustrative edits are not provided in this case, but would not be much different in magnitude from the
41 functional-syntax case.

42 9 History

43 Concept originally submitted in 04-192 at J3 meeting 167.