Subject: Embedding conditionals in expressions etc.
From: Van Snyder
Reference: 03-258r1, section 2.8.1; 04-192, 04-357r1

1 Number

TBD

2 Title

Embedding conditionals in expressions etc.

3 Submitted By

J3

4 Status

For consideration.

5 Basic Functionality

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

6 Rationale

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

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

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

When a procedure is invoked, one would like to be able to select one actual argument or another depending on some condition. For one argument, this isn't too bad: Put two different invocations in two branches of an IF construct. For n arguments, the ways to do this are with a 2^n-way if-elseif...-else-endif construct, or to have n IF statements or constructs that associate a pointer with one or another of the desired arguments (or nullify it, if that was one of the outcomes), followed by the invocation. It would be more convenient to be able to write something of the form "IF(A, B, C)" for each argument. In these

cases, it's important that either B or C, not the value of one or the other, becomes the actual argument — at least in the INTENT([IN]OUT) case.

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

7 Estimated Impact

Small, both for standard and implementors. Judged at meeting 169 to be at 4 on the JKR scale.

8 Detailed Specification

8.1 Functional syntax

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

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

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

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

---

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

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

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

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

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

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

Class. Transformational.

Arguments.

MASK	shall be of type logical and shall be scalar.
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.
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.

Result Characteristics.

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

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

Result.

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

Case (ii): Two arguments: The result is the TSOURCE argument if and only if the MASK argument is true. If MASK is false the result is undefined, and the actual argument consisting of the function reference is absent. The result, and therefore the function reference, may be associated with an argument that does not have INTENT(IN) if TSOURCE is permitted to appear in a variable-definition context (16.5.7).

Examples.

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

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

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

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

8.2 Operational syntax

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

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

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

9 History

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