Subject: Allow more general forward type reference for components
From: Van Snyder

1 Number

TBD

2 Title

Allow more general forward type reference for components.

3 Submitted By

J3

4 Status

For consideration.

5 Basic Functionality

This is a two-level proposal.

5.1 Level 1

Allow a forward reference for the type of an allocatable component.

5.2 Level 2

Allow a forward reference for the type of any component, so long as any circular dependence that arises involves a component with the POINTER or ALLOCATABLE attribute.

6 Rationale

6.1 Level 1

It's a little bit weird that the appearance of the POINTER attribute allows a forward reference for the type of a component, but ALLOCATABLE doesn't. In the discussion that led to MOVE_ALLOC, it was argued that one could use it to create linked lists without using pointers. Without allowing allocatable components to have declared types that are not yet defined, that dream is in fact not realizable. That the constraints were not updated when allocatable components were done was probably an oversight rather than a conscious decision.

6.2 Level 2

The only purposes served by C438 are to prevent infinite-size types — those that have themselves as a direct component (as defined below), and to avoid expensive circularity checks. Prohibiting a forward reference was a simple way to do this — but it also prevents orderings of type definitions that some would find clearer than the reverse of a depth- or breadth-first traversal of the dependency DAG.

Processors have to cope with forward type references for FUNCTION statements, so that's clearly not an unsolved problem. Processors have to prevent circular module dependencies, so circular-reference detection also is clearly not an unsolved problem.

7 Estimated Impact

Trivial for the standard — at 3 on the N1594 scale; probably in the small range for processors.

The obvious way to discover circularity requires $O(n^2)$ operations where n is the number of direct components (as defined below) in a type. It can be done with less work:

For each type, mark whether it has any nonpointer nonallocatable components for which the type definition is not yet accessible, or the type of some component is marked.

For each marked type, construct a list of the types of the nonpointer nonallocatable components that are either not yet accessible or have types that are marked.

To check for circularity, it is only necessary to check the types in the list.

Thereby, types for which all nonpointer nonallocatable components are all accessible — i.e., types defined in a presently-allowed order — would have zero-length lists, would not be marked, and would not appear in any other type's "check these types for circularity" lists.

A processor could update the "marked components" and "marked types" lists when a type is defined if it discovers that that definition causes another type to become completely defined.

8 Detailed Specification

The **Basic Functionality** section covers it. Here are illustrative edits, to give an idea of the effect on the standard.

8.1 Level 1

[Replace "POINTER ... not" by "neither the POINTER nor ALLOCATABLE attribute is".] 50:18

[Replace "POINTER" by "POINTER or ALLOCATABLE". Ummm, shouldn't C439 be a note anyway?] 50:21

8.2 Level 2

The **direct components** of a derived type are its nonpointer nonallocatable components of derived type, and the direct components of the types of those components. 44:29+ New ¶

This definition of **direct components** is different from (and simpler than) the definition in 01-007.	*Note to J3*

[Replace C438 and C439:] 50:18-23

C438 (R440) A derived type shall not have a direct component of the same declared type.

9 History