Subject: Sections or elements of pointer arrays are pointers
From: Van Snyder

# Number

TBD

# Title

Sections or elements of pointer arrays are pointers.

# Submitted By

J3

# Status

For consideration.

# Basic Functionality

Define sections and elements of pointer arrays to be pointers. Allow sections of disassociated pointer arrays, and define them to be disassociated pointers.

# Rationale

If a dummy argument is a pointer and one needs to use a section of an array pointer for an actual argument, one needs to pointer-assign the section to an auxiliary pointer. If the pointer is disassociated one needs to be careful how it is assigned:

```
auxPointer => NULL() ! or nullify ( auxPointer )
if ( associated(arrayPointer) ) auxPointer => arrayPointer(:,i)
```

This is especially useful where an intent(in) pointer is used as a proxy for optional – a proxy that can be computed during execution, instead of being cast in concrete by the syntax.

The embedded-decision proposals in 04-192 would render the need for this proposal less urgent, *viz.*

```
auxPointer => associated(arrayPointer) ? arrayPointer(:,i) : NULL()
call SUB ( A, B, associated(arrayPointer) ? arrayPointer(:,i) : NULL() )
```

# Estimated Impact

Minor, both for standard and implementors.

# Detailed Specification

Define sections and elements of pointer arrays to be pointers. Allow sections of disassociated pointers, and define them to be disassociated pointers. This requires very little change in the standard or processors. Care will be needed to prohibit changing the association status of an actual-argument pointer that doesn't have a name by way of an associated dummy argument, similarly to how the standard prohibits an actual-argument expression from being assigned a value by way of an associated dummy argument. We may need to make an exception to the "subscripts shall be in bounds" rule, but it may be possible to finesse it other ways. For example, we could distinguish between subscripts used to calculate pointerness, and ones to access targets.

This isn't any more of a problem for expressions than the present possibility to use a disassociated pointer in an expression — which is prohibited. A disassociated pointer that arises from a section of a disassociated pointer array would be nothing different.

It may be possible for implementations to arrange descriptors for null pointers so that sections of them end up being null pointers after the usual create-a-new-descriptor instructions are executed. For example if disassociated pointers are represented by a null pointer, and the descriptor has zero lower bound and zero stride in every dimension, computing a descriptor from a section as if it were associated should result in a descriptor for a disassociated pointer, i.e., one with a null pointer and zero lower bound and stride in every dimension. It wouldn't matter what special values are used for bounds and strides for a disassociated pointer's descriptor, even one that results from a section of a disassociated pointer, because it would still be illegal to invoke all of the array inquiry functions using a disassociated pointer, even one that results from a section of a disassociated pointer.

# History