This subject was previously proposed as a standalone MTE, but it was not given a high enough priority by WG5 to be included in the Fortran 2000 requirements. It is no longer being considered on that basis. However, it is has shown up as a possible part of many of the requirements that were approved, including integration of the Data Type Enhancements TR, parameterized data types, constructor/destructors, and object-oriented programming. Rather than deferring consideration of the issues underlying this proposal to the integration phase of Fortran 2000, I am beginning this consideration now.

Preliminary Specification

Data objects with the ALLOCATABLE and POINTER are extended to allow deferring of the specification of non-KIND type parameters as well as array bounds. The ALLOCATE statement is extended to provide a means of providing values for those deferred type parameter values at the time the data objects are allocated. The description of the pointer assignment statement is extended to allow the value of deferred type parameters to be determined by the value those parameters have in the target.

Details: the ALLOCATE Statement

A parenthesized list of the deferred data type parameters is specified before the name of data object being allocated. For example,

```
ALLOCATE((ROWS=R,COLS=C)MATRIX_VAR)
ALLOCATE((R,C)ANOTHER_MATRIX_VAR)
ALLOCATE((LEN=L)CHAR_VAR(ARRAY_SIZE))
```

Questions (and possible answers):

- Should the type parameter specification be allowed to include type parameters whose value was not deferred in the declaration of the data object? Is the answer the same for KIND type parameters as for non-KIND type parameters? (My inclination is to allow such respecification as long as the value provided confirms the value provided in the declaration of the object. There is at least one case that is significantly simplified if this respecification is allowed for non-KIND type parameters (see later discussion of the interaction of assumed and deferred type parameters), and I would allow it for KIND parameters as a matter of consistency.)

- If respecification of the "other" type parameters is not allowed, how does this affect which combinations of positional and keyword specification of type parameters allowed? (To allow for possible extension (either by current vendors or in a future revision of the standard) that would allow the respecification of these "other" type parameters, I would require that type parameters following them in the original type definitions always be specified by keyword. An even more conservative choice might be to require all type parameters in an ALLOCATE statement to be specified in the keyword form.)

Details: Declaration

Given that the syntax for deferred shape is nearly identical to that for assumed shape, I suggest that deferred type parameters, like assumed type parameters, be denoted by an asterisk(*).

Questions (and possible answers):

- Noting that all dimensions of the shape must be deferred for any dimension to be deferred, we must determine what rules govern the interaction of deferral of type parameters. If the shape is deferred,

must the non-KIND type parameters also be deferred? (Obviously not, as this case is already legal Fortran 90/95.)

- If the non-KIND type parameters are deferred, must the shape be deferred? (I would suggest not, as the comparable restriction does not hold for assumed shape and assumed type parameters.)
- If one non-KIND type parameter is deferred, must all of them be? (Again, I suggest not, as the comparable restriction does not hold for assumed shape and assumed type parameters.)

Assumed vs. Deferred Type Parameters

Consider a dummy argument that is declared as follows:

```
CHARACTER(*),POINTER :: DUMMY_VAR
```

There would appear to be two possible interpretations of this statement: This could indicate that the length of this dummy pointer is to be assumed from the declared length of pointer supplied as the actual argument, or it could indicate that the actual argument is an pointer with deferred length. The former interpretation would seem to be implied by Fortran 90/95, but the latter interpretation strikes me as being the one that is far more likely to be the one that would be useful. We could try to avoid this problem by using different declaration syntax for deferred parameters and assumed parameters, but I think the more acceptable solution may be to call this an assumed parameter, but allow the parameter to assume the fact that the parameter value has been deferred, so that, in effect, both interpretations are included.

Such semantics can be cleanly defined, and it doesn't matter for most code generation when the length is determined, but will this cause unacceptable problems in representation and procedure calling conventions, especially if one wishes to be object code compatible with existing Fortran 90/95 implementations? Should there be a notation for determining whether an assumed parameter has assumed a specific value or a deferred value (e.g., PRESENT(DUMMY_VAR%LEN))?

Looking Forward to the Edits

The hard part of the edits is the near universal replacement of the term "allocatable array" with a more general term like "allocatable object" or "data object with the ALLOCATABLE attribute". Otherwise, the edits should be relatively simple and localized to a few areas including declaration of ALLOCATABLE and POINTER objects, the ALLOCATE statement, pointer assignment, and argument association (to allow deferred parameter actual arguments to be associated with assumed parameter dummy arguments.

Ω