Date: 6 May 1998
To: J3
From: Van Snyder
Subject: Discussion paper – Explicitly typed allocations
References: 97-159, 98-124, 98-135

In paper 98-124, Richard Maine observed that array constructors suffer from the problem that their types and type parameters are inferred from their components. The solution proposed in that paper is to allow a complete *type-spec*, including specification of both kind and non-kind parameters, to preceed the array constructor.

The ALLOCATE statement similarly suffers in that the length of a character datum cannot be deferred until allocation. Furthermore, users will certainly want to be able to allocate:

- objects of parameterized derived types, and specify non-kind type parameters during allocation, and
- polymorphic objects with a type descendant from the declared type.

In parallel to the solution proposed in paper 98-124, this paper proposes to use a *type-spec*, including specification of non-kind parameters, to preceed an *allocate-object*, and its associated *allocate-shape-spec-list*, if any. (Allowing specification of kind parameters to be deferred until allocation would cause a tremendous performance problem.)

Exactly all deferred parameters must be specified during allocation. Actual type parameters to the *type-spec* correspond in positional order to deferred "dummy" type parameters. Keyword notation may also be used.

To make the declaration syntax for allocatable entities for which parameters in addition to dimension will be specified during allocation more uniform, the ":" notation should be allowed to indicate a specification that is deferred until allocation, or assumed from a dummy argument. The present syntax for parameterized derived types uses "*" for assumed type parameters of dummy arguments.

For example, to declare a rank-1 allocatable character array, in which the element size and array dimension are both to be specified during allocation, one should write

```
character(len=:), allocatable :: char_arr(:)
...
allocate ( character(len=16)( char_arr(23) ) )
```

If a facility to specify character length during allocation had been available, the mechanism to access command line information might well have been developed differently.

Paper 97-159 illustrates the kind of difficulties that would have been encountered in trying to access command line information by using procedures, but without the ability to specify character length during allocation. A solution of the kind adopted for command line information is, however, at best a clumsy solution for access to status error messages:

```
  ! Call the system's intrinsic routine to give access to error messages:
  call system_error ( message_number, my_processing_routine )
  ...
contains
```

```
subroutine my_processing_routine ( message )
  character(len=*) message(:)
  ! access other information by host or use association
  ...
end subroutine my_processing_routine
```

(This solution was not proposed in 97-159 because it is even uglier than the ugly solutions proposed therein. It would be the only intrinsic procedure that has a dummy procedure argument.)

It would be better to allow:

```
character(len=:), allocatable :: my_message(:)
...
! I want to do the allocation myself:
call system_message ( message_number, width=my_width, lines=my_lines )
allocate ( character(len=my_width)( my_message(my_lines) ) )
! If I don't allocate enough, the system_message routine truncates
call system_message ( message_number, message=my_message )
print * 'Something went wrong:'
print *, my_message
deallocate ( my_message )
...
! I want the system to do the allocation.
call system_message ( message_number, allocate_message=my_message )
! my_width = len(my_message); my_lines = size(my_message) ! but I don't care
print * 'Something went wrong:'
print *, my_message
deallocate ( my_message )
```

If one compares this solution to the preceeding one, and to the unbearably messy solutions in 97-159, the advantages are obvious.

Facilities to access system environment information, and the `MANGLE` function proposed in discussions of C interoperability, would suffer difficulties similar to those shown here and in 97-159 without the ability for character variable length to participate in allocation, and would enjoy similar benefits to those shown here if it were possible.

Compiler code generation strategies for objects for which specification of non-dimension parameters is deferred until they are allocated should be similar to, and no more difficult than, existing strategies for dummy arguments of character type having assumed length, and strategies that will be necessary for dummy arguments of parameterized derived type, with assumed parameters.