Date: 10 August 1998
To: J3
From: Van Snyder
Subject: Explicitly typed allocations - Rationale, Specs, Syntax
References: 98-146, 98-160

1 Rationale

This paper does not address a specific requirement or minor technical enhancement. Rather, it addresses issues of integration concerning parameterized derived types and polymorphic data.

In Fortran 95, character length cannot be deferred until allocation. In Fortran 2000, without change, one will in addition not be able to defer specifying parameters of parameterized derived types until allocation, nor will one be able to allocate a polymorphic object with a type extended from its declared type. This paper addresses those deficiencies.

2 Specs

Change the syntax of declarations to indicate that nonkind type parameters are deferred until allocation.

Extend the syntax of R623 *allocate-stmt* so that the type and type parameters, in addition to dimensions, can be specified.

Extend the semantics of `allocate` to allow specifying a type of a polymorphic object that is an extension of its declared type.

Extend the semantics of `allocatable` to scalars, which would be almost silly in Fortran 95, but makes good sense with parameterized derived type objects and polymorphic objects.

Extend the semantics of pointer assignment and argument association so that deferred parameters of the *pointer-object* or the dummy argument are assumed from those of the *target* or actual argument, respectively.

3 Syntax

In a declaration, allow ":" to stand for a deferred nonkind parameter, e.g.

```
character(len=:), allocatable :: char_arr(:)
```

One can defer zero or more nonkind parameters. Independently, one can defer zero or all dimensions.

Extend the syntax of `allocate` to specify the type and at least all deferred parameters, using an executable form syntactically identical to *type-spec* that does not constrain the expressions for "actual" kind parameters to be specification expressions. E.g.

```
call calculate_the_length_and_dimension ( n, m )
allocate ( character(len=n) :: char_arr(m) )
```

If a *type-spec* appears in an `allocate` statement, it is required to appear before the first *allocation*, and affects all of the *allocate-obj*s in the statement. Nonpolymorphic objects are required to have a type that is the same as the type specified by the *type-spec*. Polymorphic objects are required to have a declared type that is the same as the type specified by the *type-spec*, or a type from which the type specified by the *type-spec* is extended.

The "actual" parameters correspond by position or by keyword to the "dummy" parameters, in exactly the same way that actual procedure arguments correspond to dummy procedure arguments. Nondeferred parameters need not be specified, subject to the same kinds of rules as for optional arguments, e.g. if a value is specified by position for parameter k, it is required to specify values by position for parameters $1, ..., k-1$.

If a value is provided for a nondeferred parameter or dimension bound, it is required to be the same as the value specified for that parameter or dimension bound in the object declaration. Otherwise, an error condition exists, that must be signalled. It could be signalled by the compiler, by a nonzero status value, or by halting the program if no STAT= clause is present.

As in Fortran 95, if the bounds for one dimension are deferred, it is required to defer the bounds for all dimensions. If the bounds are deferred, it is required to specify them in an allocate statement or pointer assignment statement. If the bounds are not deferred, they may be omitted or specified in an allocate statement. If the bounds for one dimension are specified in an allocate statement, it is required to specify the bounds for all dimensions.