The following is a summary of the /DATA consideration of 98-146:

1. The subgroup supports the proposed functionality.

2. Declaration syntax – the subgroup discussed various alternatives:

   a. `CHARACTER(LEN=:)` (also `CHARACTER(:)` and `CHARACTER*(:)`): This alternative was proposed in 98-146 as an analog to the syntax for deferred-shape declarations. Objection: In deferred-shape declarations and elsewhere in the language, colon is used to separate two omitted values rather than to represent a single omitted value.

   b. `CHARACTER(LEN=*)`: The rationale for this alternative is that since the syntax for deferred-shape arrays is very similar to the syntax of assumed-shape arrays, the syntax of deferred-length pointers should resemble that of assumed-length variables. However, there are already assumed-length dummy pointers (because non-dummy `CHARACTER` pointers are declared with explicits lengths) and there are no assumed-shape dummy pointers, so we would need to reconcile the syntax that can be taken as either an assumed-length dummy pointer or a dummy deferred-length pointers. One possible reconciliation is for assumed-length dummies to also assumed the property of being deferred (and thus changeable). Thus, they would effectively become dummy deferred-length pointers when the corresponding actual arguments are deferred-length pointers. Remaining objections: Use of this combined semantic may be more error-prone than helpful. It may be difficult to implement this form of assumed-length length in a way that is object-compatible with implementations of the current assumed-length feature.

   c. `CHARACTER(LEN= )` (also `CHARACTER( )` and `CHARACTER*( )`): The rationale for this alternative is that in assumed-shape declarations, we simply omit the values, leaving the punctuation. Objections: This form looks strange and it might be confused with ordinary omission of the `LEN` type parameter (implying `CHARACTER(1)`).

   d. Others such as `CHARACTER(LEN=**)`, `CHARACTER(LEN=.)`, `CHARACTER(LEN=-)`, `CHARACTER(LEN=+)`, and `CHARACTER(LEN=_)`: These notations provide an explicit substitute for the omitted value, while avoiding semantic entanglements with assumed-length. The same notation could be allowed as an alternative to the existing assumed-shape notation. Objections: These notations have no precedent in the language. Some of them are subject to legibility concerns or concerns about precluding future extensions involving those notations (e.g., under some possible extensions, _ might be a legal variable name).

3. Allocation syntax: Although the proposed syntax appeared unambiguous, concerns were expressed that it placed an unnecessarily high burden on the parser and that the

connections to the syntax on which it was modeled were tenuous. As an alternative, the subgroup suggested the following:

```
ALLOCATE ( CHARACTER(LEN=16):: CHAR_ARR(23), CHAR_SCALAR )
```

The rationale (or perhaps rationalization) for this syntax choice was that since the ALLOCATE statement is filling in values deferred from the declaration of the variables, it is reasonable the syntax for doing so should resemble declaration syntax.

4. Declaration constraints: Currently, all values used in specifying shape must be deferred on an array pointer, but non-KIND type parameters must be specified. We wish to allow those non-KIND type parameters to be deferred. If one non-KIND type parameter is deferred, must all be? May the non-KIND type parameters be deferred and the shape specified? Should we revisit the decisions on deferring shape and allow some values to be specified (e.g., lower bounds or the both bounds on the inner dimensions) while others are deferred?

5. Allocation constraints: 98-146 proposes that only the deferred non-KIND type parameters be specified in the ALLOCATE statement. An alternative suggestion is that the specification of type parameters be complete, with the requirement that previously specified type parameters be given the same value. Another would be to allow such restatement of previously specified type parameters, but not require it. It would be possible to treat KIND and non-KIND type parameters in this regard (e.g., requiring KIND type parameters not to be restated while allowing restatement of non-KIND type parameters).

•