1. Proposals for procedure pointers and procedure variables offer similar functionality. After much discussion, /data is recommending the procedure pointer formulation because it more naturally offers the null pointer or null association concept.

2. Specifications for this facility are based on analogies with two existing language features.

   a. Dummy procedures. If a set of declarations of a name in conjunction with that name appearing in a dummy argument world make that name the name of a dummy procedure, then that same set of declarations in conjunction with that name being declared with the POINTER attribute will make that name the name of a procedure pointer. The class of procedures which can then be associated with that procedure pointer is exactly the class of procedures which could have been supplied as the corresponding actual argument in the dummy procedure case. Thus, procedure pointers cannot be associated with internal procedures, statement functions, generic procedures, most intrinsic procedures, etc. If the interface of the procedure pointer is explicit, the interface of the procedure being associated must also be explicit (with matching characteristics).

   b. Data-object pointers. The mechanism for establishing an association is the pointer assignment statement, a disassociated status can be created with the NULLIFY statement or the NULL intrinsic function, the one-argument form of the ASSOCIATED intrinsic function can be used to test for a null association status, and the two-argument form of the ASSOCIATED intrinsic function can be used to test for a particular association. However, we do not proposed to allow one to ALLOCATE or DEALLOCATE a procedure pointer.

      As with data object pointers, the default initial status for such pointers is "undefined". Just as a data-object pointer can become undefined if it is associated with a module variable and the module goes out of scope, a procedure pointer can become undefined if it is associated with a module procedure and the module goes out of scope.

      Procedure-pointer dummy arguments, function results, and derived type components would be allowed analogous with the data-object pointers.

3. There are a number of smaller details which may not be immediately obvious.

   a. No TARGET attribute for procedures is proposed.

   b. The description of the NULL intrinsic procedure must be extended to allow the optional argument to be a procedure.

   c. Allowing procedure pointers as components means that interface blocks may be nested inside type definitions.

d. There are no arrays of procedure pointers, but the usual workaround of having arrays of a derived type with pointer components works as well for procedure pointers as for data-object-pointers.

e. We recommend that procedure reference syntax be extended to allow the procedure to be specified as a structure component, e.g.

```
CALL PARENT%PROC_POINTER(ARGS)
```

On the other hand, we do not recommend a similar extension for functions that return procedure pointers. E.g., we would require

```
PP=>PPF(ARGS1)
CALL PP(ARGS2)
```

rather than extending to allow

```
CALL PPF(ARGS1)(ARGS2)
```

This is consistent with data-object pointers, which can be subscripted, substringed, etc. as components, but not as function results.

f. Note that for the two-argument form of ASSOCIATED, two pointers are nominally different if they are associated with the same module procedure but from different instances of the host module. However, it appears that such queries must always involve at least one pointer that is really undefined, so implementations are not required to track this.

4. Looking to the next phase, we expect most of the syntax to be obvious given the underlying analogies. The one area where it may not be straightforward is in the area of a combined specification of the POINTER attribute and the interface specifications. E.g.,

```
REAL,POINTER,EXTERNAL::FP ! function pointer with implicit interface
POINTER,EXTERNAL::SP ! subroutine pointer with implicit interface
INTERFACE
   SUBROUTINE SP2, POINTER ! subroutine pointer with explicit interface
   END SUBROUTINE
END INTERFACE
```

5. We identified two orthogonal but related issues:

a. Should there be a way to assign a symbolic name to a particular interface so procedure pointers can be declared to be of a particular procedure "type" or class rather than requiring the matching interfaces to be repeatedly declared in detail?

b. Should "procedureness" be made a basis for generic resolution (so, for example, one can distinguish between a procedure accepts a REAL variable from one that accepts a function that returns a REAL result)?

Ω