From: Kurt W. Hirchert X3J3/96-146 (Page 1 of 3)
Subject: Procedure Pointer Examples Meeting 138

The principle underlying procedure pointers declarations is that if

```
subroutine sub(NAME)
! declarations and usage patterns for NAME
…
```

results in NAME identifying a dummy procedure with certain properties, then

```
pointer::NAME
! same declarations and usage patterns for NAME
```

should result in NAME identifying a procedure pointer with analogous properties. Thus,

```
pointer::SP
…
CALL SP
```

results in SP being a procedure pointer than can be associated with subroutines and that has an implicit interface. Similarly,

```
pointer::FP
…
Y=FP(X)
```

or

```
pointer::FP
real,external::FP
```

results in FP being a procedure pointer that can be associated with real functions and that has an implicit interface, and

```
pointer::RFP
interface; real function RFP(X); end function; end interface
```

results in RFP being a procedure pointer that can be associated with real functions of a single real argument and that has an explicit interface. (The case of

```
pointer::P
external P
```

should be resolved in a similar fashion, but we have some disagreement on exactly how the corresponding dummy procedure case is interpreted.)

Such a procedure pointer can then be associated with an actual procedure using pointer assignment. For example,

```
RFP=>SIN
```

would associate the (specific) intrinsic function SIN with RFP. As with dta-object pointers, the right hand side could be another procedure pointer:

```
FP=>RFP
```

Note, however, that

```
RFP=>FP     ! Wrong !
```

is not legal because, analogous with dummy procedures, a procedure with an implicit interface may not be associated with a procedure pointer that has an explicit interface.

As with data-object pointers, one can make a procedure pointer testably disassociated with

```
nullify(PP)
```

or

```
PP=>NULL()
```

or even

```
real,pointer,external::PP=>NULL()
```

This is tested with the one-argument form of ASSOCIATED:

```
if (associated(PP)) …
```

As with data-object pointers, one can have procedure pointer dummy arguments and function results:

```
function MERGE_REAL_FUNCS(TP,FP,MASK)
pointer::MERGE_REAL_FUNCS,TP,FP
interface; real function MERGE_REAL_FUNCS(X); end function; end interface
interface; real function TP(X); end function; end interface
interface; real function FP(X); end function; end interface
logical::MASK

if (MASK) then
      MERGE_REAL_FUNCS=TP
else
      MERGE_REAL_FUNCS=FP
end if

end function MERGE_REAL_FUNCS
```

Although I believe the above is unambiguous for a compiler, it has been suggested that it might be clearer if we require the use of a RESULT variable, so the above would become

```
function MERGE_REAL_FUNCS(TP,FP,MASK) result(RP)
pointer::RP,TP,FP
interface; real function RP(X); end function; end interface
interface; real function TP(X); end function; end interface
interface; real function FP(X); end function; end interface
logical::MASK

if (MASK) then
      RP=TP
else
      RP=FP
end if

end function MERGE_REAL_FUNCS
```

As with data-object pointers, one can have a procedure pointer component in a derived type:

```
type REAL_FUNCTION_LIST
      type(REAL_FUNCTION_LIST),pointer::NEXT
      real,pointer,external::F
end type REAL_FUNCTION_LIST

…

type(REAL_FUNCTION_LIST),pointer::RFLP

…

allocate(RFLP)
RFLP%F=>SIN

…

Y=RFLP%F(X)
```

If the procedure pointer in the previous example is to have an explicit interface, the appropriate syntax is less obvious. One possibility is

```
type REAL_FUNCTION_LIST
      type(REAL_FUNCTION_LIST),pointer::NEXT
      interface
            pointer, &
            function F(X)
            end function F
      end interface
end type REAL_FUNCTION_LIST
```

Ω