Enhancements to allocation and pointers

Arjen Markus

Deltares

June 6, 2018

Automatic (re)allocation

Automatic (re)allocation occurs when the left-hand side is allocatable:

```
integer, dimension(:) :: x, y

allocate( x(4) )

x = [1, 2, 3, 4]

y = 2 * x
```

Note: Some compilers (or versions thereof) require an explicit option for this to work.

Intrinsic routine move_alloc

Transferring the allocated memory from one variable to another:

```
integer, dimension(:) :: x, y

allocate( x(4) )

x = [1, 2, 3, 4]

call move_alloc( x, y )

write(*,*) 'X: ', allocated(x)
write(*,*) 'Y: ', allocated(y)

===> X: F
     Y: T
```

The allocated memory now belongs to variable y.
Variable x becomes *unallocated*.

New: procedure pointers

Pointers to functions and subroutines:

```
interface
    subroutine find( data, value, idx )
        real, dimension(:)   :: data
        real                 :: value
        integer, intent(out) :: idx
    end subroutine find
end interface
procedure(find), pointer :: fp    ! The routine to point must adhere to
                                  ! this interface

fp => find_with_margin   ! Select a specific implementation

call fp( data, 1.0, idx )
```

Typed and source allocation

- Specify a type:

```
character(len=:), dimension(:), allocatable :: string
allocate( character(len=10):: string(25) )
```

- Specify a source:

```
character(len=10), dimension(:), allocatable :: &
     string1, string2
allocate( string1(25) ); string1 = A
allocate( string2, source = string1 )
        ! Copies values of string1 into string2
```

Note: each element of the array gets the same length

Typed and source allocation (2)

- Useful if you have heterogeneous data structures like linked lists or trees
- Connection to objected-oriented features
- With typed allocation you can create character strings of varying length

Also:

```
a = [ character(len=10):: 1, abc, d ]
b = [ type(mytype) :: ]  ! Zero-sized array!
```