Pointers

Fortran 90 introduced concept of pointers to overcome the issue of passing by reference vs. passing variables by value. Pointers can be defined at any data type using keyword `pointer` , and associated with them variables with keyword `target` , e.g.

```
type, pointer :: pa
type, target :: b

pa => b
```

operator `=>` above is used to assign the pointer to a given target. Contrary to C, Fortran pointers can be only associated with variables specified as `target` not any variable of the same type. Variable `pa` now can be used in any part of the program and will always link to `b` (receive and modify its value). Pointers can be unlined using `nullify()` function.

Pointers can be also defined as an array

```
real, pointer, dimension(:,:) :: pA
real, target, dimension(:,:), allocatable :: B
```

Example

```
program simple_pointer

  implicit none

  real, dimension(:,:), allocatable, target :: A
  real, dimension(:,:), pointer :: B
  integer :: i,j,n

  n = 5

  allocate( A(n,n) )

  A = 0.0
  forall( i=1:n) A(i,i) = i

  B => A

  do i=1,n
    print *,( B(i,j), j=1,n )
  end do

  nullify(B)

  deallocate( A )

end program simple_pointer
```

which returns

```
1.00000000       0.00000000       0.00000000       0.00000000       0.00000000
0.00000000       2.00000000       0.00000000       0.00000000       0.00000000
0.00000000       0.00000000       3.00000000       0.00000000       0.00000000
0.00000000       0.00000000       0.00000000       4.00000000       0.00000000
0.00000000       0.00000000       0.00000000       0.00000000       5.00000000
```

Pointers do not reference a whole target and can be only a slice of the memory

```
B => A
C => A(1:2,1:2)
```

Programmer can check if a given pointer is in use with `associated()` function which returns true if the pointer has been assigned with `=>` operator. Example

```
program pointer_functions

  implicit none

  integer, parameter :: n = 5
  real, dimension(n,n), target :: A

  real, dimension(:,:), pointer :: pt

  A = 1.0

  if( associated(pt) ) print *,'Pointer pt is in use'

  pt => A

  if( associated(pt) ) print *,'Pointer pt is in use'
  if( associated(pt, target=A) ) print *,'pt in use and points to A'

  nullify( pt )

end program pointer_functions
```

returns

```
 Pointer pt is in use
 pt in use and points to A
```

Allocating pointers

Pointer variables defined as arrays may have memory directly allocated to them just like arrays defined with word `allocatable` e.g.

```
program allocating_pointer

  implicit none

  integer :: i,n
  real, pointer, dimension(:) :: pt1
  real, pointer, dimension(:) :: pt2

  n = 5

  allocate( pt1(n) )

  forall( i=1:n ) pt1(i) = i

  print '(5f5.1)',( pt1(i), i=1,n )

  if( associated(pt2) ) print *,'pt2 is in use'
  pt2 => pt1
  if( associated(pt2) ) print *,'pt2 is in use'

  deallocate( pt1 )

end program allocating_pointer
```

Also, pointer can be used as a target. The above program returns

```
  1.0  2.0  3.0  4.0  5.0
 pt2 is in use
```