Pointers

A pointer is an animal pointing to some other variable.

```
real, pointer, dimension (:) :: p
```

Here `p` is a pointer that can point to an onedimensional array. No space is allocated for the array at this time.

The operator `=>` sets the pointer to point to an actual variable.

A pointer is not just an memory address of the variable, but a descriptor that contains also some other information. A pointer can also point to an array section:

```
real, dimension(100,100), target:: x
real, pointer, dimension (:) :: p1
real, pointer, dimension (:,:) :: p2

p1 => x(1,:)
p1 => x(:,10)
p1 => x(1, 1:91:10)

p2 => x
p2 => x(2:3, 10:20)
p2 => x(1:11:2, 10:100:10)
```

In Fortran a pointer is like a synonymous name for the object it points to.

N.B.!

```
p=p+1
```

In C the pointer moves forward to point to the next element.

In Fortran the value of the object pointed by `p` is incremented by one.

The association status of a pointer can be:

- Undefined: it does not point anywhere.
- Null: it points to 'nothing'. This can be used e.g. as a sign of the end of a linked data structure. Note that this is quite different from undefined.
- Associated: it points to a variable.

```
real, target :: x
real, pointer :: p
  ! p undefined
p => x
  ! p associated to x
nullify(p)
  ! p is null
```

If `p` is associated to a variable, the value of `associated(p)` is true.

Pointers can be used for dynamic memory allocation:

```
real, dimension(:), pointer :: p
...
allocate (p(1000))
...
deallocate(p)
```

This is convenient for handling linked data structures, like a binary tree:

```
type node
 character (len=100)  :: name
 type (node), pointer :: left, right
end type node
type (node), pointer :: root, l, r

allocate(root, l, r)
root%name='puun juuri'
root%left => l     ! link to the left subtree
root%right => r    ! link to the right subtree
```

Example: handling a linked list:

```
program linkedlist
! construct and print a list that will contain the input numbers
! in increasing order
implicit none
type node
 integer :: value
 type (node), pointer :: next
end type node
integer :: num
type (node), pointer :: first, current, p, q
! first points to the beginning of the list
! begin by creating an empty list
nullify(first)
! read numbers and add to the list
! finish when the given number is 0
do
 read (*,*) num
 if (num==0) exit
 allocate(current)            ! create a new record
 current%value=num            ! set its value field
 ! link the new record to a proper place in the list
```

```
if (.not.associated(first)) then
 ! the list is empty
 current%next=>first
 first=>current
else if (num <= first%value) then
 ! the number is smaller than the first number in the list
 ! the new record will be added at the beginning of the list
 current%next=>first
 first=>current
else
 ! otherwise proceed along the list
 ! the new record will be linked between the records
 ! pointed by p and q
 q=>first; p=>first%next
 do
  if (.not.associated(p)) exit  ! the list was exhausted
  if (num <= p%value) exit     ! a proper place was found
  q=>p; p=>p%next           ! move forward
 end do
 ! link the record between p and q
 current%next=>p
 q%next=>current
end if
```

```
end do
! the list is complete, print it
current=>first        ! 1. record of the list
! proceed until a null link is found
do
 if (.not.associated(current)) exit
 write (*,*) current%value
 current=>current%next  ! next record
end do
end program linkedlist
```

Potential problems with pointers:

Dangling reference: a pointer that points to a memory area that is no more used.

```
real, pointer :: p, q
...
allocate (p)
p=1.0
q=>p
deallocate(p)   ! q = ?
```

Inaccessible memory area:

```
real, dimension(:), pointer :: p
...
allocate (p(1000))
...
nullify(p)
```

Memory has been allocated for the array, but there is no way to access it any more. The space will become available only when the program terminates.