FORTRAN - POINTERS

In most programming languages, a pointer variable stores the memory address of an object. However, in Fortran, a pointer is a data object that has more functionalities than just storing the memory address. It contains more information about a particular object, like type, rank, extents, and memory address.

A pointer is associated with a target by allocation or pointer assignment.

Declaring a Pointer Variable

A pointer variable is declared with the pointer attribute.

The following examples shows declaration of pointer variables:

```
integer, pointer :: p1 ! pointer to integer
real, pointer, dimension (:) :: pra ! pointer to 1-dim real array
real, pointer, dimension (:,:) :: pra2 ! pointer to 2-dim real array
```

A pointer can point to:

- an area of dynamically allocated memory
- a data object of the same type as the pointer, with the **target** attribute

Allocating Space for a Pointer

The **allocate** statement allows you to allocate space for a pointer object. For example:

```
program pointerExample
implicit none

   integer, pointer :: p1
   allocate(p1)

   p1 = 1
   Print *, p1

   p1 = p1 + 4
   Print *, p1

end program pointerExample
```

When the above code is compiled and executed, it produces the following result:

```
1
5
```

You should empty the allocated storage space by the **deallocate** statement when it is no longer required and avoid accumulation of unused and unusable memory space.

Targets and Association

A target is another normal variable, with space set aside for it. A target variable must be declared with the **target** attribute.

You associate a pointer variable with a target variable using the association operator `=>` .

Let us rewrite the previous example, to demonstrate the concept:

```
program pointerExample
implicit none
```

```
   integer, pointer :: p1
   integer, target :: t1

   p1=>t1
   p1 = 1

   Print *, p1
   Print *, t1

   p1 = p1 + 4

   Print *, p1
   Print *, t1

   t1 = 8

   Print *, p1
   Print *, t1

end program pointerExample
```

When the above code is compiled and executed, it produces the following result:

```
1
1
5
5
8
8
```

A pointer can be:

- Undefined
- Associated
- Disassociated

In the above program, we have **associated** the pointer p1, with the target t1, using the => operator. The function associated, tests a pointer's association status.

The **nullify** statement disassociates a pointer from a target.

Nullify does not empty the targets as there could be more than one pointer pointing to the same target. However, emptying the pointer implies nullification also.

Example 1

The following example demonstrates the concepts:

```
program pointerExample
implicit none

   integer, pointer :: p1
   integer, target :: t1
   integer, target :: t2

   p1=>t1
   p1 = 1

   Print *, p1
   Print *, t1

   p1 = p1 + 4
   Print *, p1
   Print *, t1
```

```
   t1 = 8
   Print *, p1
   Print *, t1

   nullify(p1)
   Print *, t1

   p1=>t2
   Print *, associated(p1)
   Print*, associated(p1, t1)
   Print*, associated(p1, t2)

   !what is the value of p1 at present
   Print *, p1
   Print *, t2

   p1 = 10
   Print *, p1
   Print *, t2

end program pointerExample
```

When the above code is compiled and executed, it produces the following result:

```
1
1
5
5
8
8
8
T
F
T
952754640
952754640
10
10
```

Please note that each time you run the code, the memory addresses will be different.

Example 2

```
program pointerExample
implicit none

   integer, pointer :: a, b
   integer, target :: t
   integer :: n

   t= 1
   a=>t
   t = 2
   b => t
   n = a + b

   Print *, a, b, t, n

end program pointerExample
```

When the above code is compiled and executed, it produces the following result:

```
2  2  2  4
```