Tableaux automatiques

Tableaux utilisés localement (dans une sous-routine ou une fonction).

- Le système réserve la mémoire quand on rentre dans la sous-routine et libère la mémoire quand on sort de la sous-routine.
- Le rang du tableau doit être fixé à la compilation, mais les dimensions ne sont connues qu'à l'exécution.

Exemple :

```
1 SUBROUTINE echange(a, b)
2 REAL,DIMENSION(:,:)          :: a, b
3 REAL,DIMENSION(SIZE(a,1),SIZE(a,2)) :: work
4 work = a
5 a = b
6 b = work
7 END SUBROUTINE echange
```

Le tableau `work` est un tableau automatique.

Rappel : pour pouvoir appeler cette sous-routine, il faut écrire son interface dans le programme appelant.

Tableaux alloués dynamiquement

Pour pouvoir allouer dynamiquement un tableau, il faut lui ajouter le paramètre `ALLOCATABLE` :

```
real, dimension(:,:,:), allocatable :: A
```

Les instructions principales sont :

`ALLOCATE` réserve de la mémoire pour un (ou plusieurs) tableau(x)

```
ALLOCATE(A(n, 2*n, 4), B(n, n), STAT=inderr)
```

`DEALLOCATE` libère de la mémoire réservée par `ALLOCATE`

```
DEALLOCATE(A, STAT=inderr)
```

`ALLOCATED` teste si un tableau a été alloué

```
if (ALLOCATED(A)) then ...
```

Exemple :

```
1  REAL, DIMENSION(:,:,:), ALLOCATABLE :: T
2  INTEGER :: status
3
4  READ (*,*) n
5  ALLOCATE(T(n, 2*n, 2:n+1), STAT=status)
6  IF (status /= 0) THEN
7    WRITE(*,*) "erreur d'allocation"
8    STOP 3
9  ENDIF
10
11 T(n/2, n, 3) = 1.0
12 write(*,*) T
13
14 DEALLOCATE(T)
```

Pointeurs

Exemple :

```
1  ! pointeur vers un réel
2  real, pointer :: p, q
3  ! variable qui peut être pointée
4  real, target :: a
5
6  NULLIFY(p)
7  ALLOCATE(q)    ! alloue un réel pour q
8  q = 4.0; write(*,*) q
9  DEALLOCATE(q) ! désalloue
10
11 ! fait pointer le pointeur q vers a
12 q => a
13 ! fait pointer le pointeur p vers ce que pointe q
14 p => q
15 ! modifie la valeur de a
16 p = 3.4
```

États d'un pointeurs :

- **indéfini**, pour `p` : lignes 2 à 5, pour `q` : lignes 2 à 6;
- **associé** (après allocation ou association), pour `p` : lignes 14 à 16, pour `q` : lignes 7 à 8 et 12 à 16;
- **dissocié** (nul ou après libération de la mémoire), pour `q` : lignes 9 à 12.

Test si le pointeur `p` est associé :

```
IF (ASSOCIATED(p)) THEN
```

Test si le pointeur `p` est associé à la variable `v` :

```
IF (ASSOCIATED(p, v)) THEN
```

Test si les pointeur `p` et `q` sont associés à une même variable :

```
IF (ASSOCIATED(p, q)) THEN
```

Types utilisateurs (ou dérivés)

En fortran90, un type dérivé est une structure définie par l'utilisateur et pouvant contenir des scalaires, des tableaux, d'autres types dérivés.

Remarque :

Cela n'a rien à voir avec la dérivation dans les languages orientés-objet (C++, java, python, ...)

Chaque composante d'un type dérivé est identifiée par un nom.

Exemple :

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 \end{pmatrix}$$

```
type MatriceCreuse
   integer :: n, m, nnz
   integer, dimension(:), allocatable :: i, j
   real, dimension(:), allocatable :: coeff
end type MatriceCreuse
```

Utilisation :

```
type(MatriceCreuse) :: A
allocate(A%i(5), A%j(5), A%coeff(5))
A%nnz = 5; A%n = 3; A%m = 4
A%i     = (/ 1, 1, 2, 3, 3 /)
A%j     = (/ 1, 3, 2, 1, 2 /)
A%coeff = (/ 1.0, 2.0, 3.0, 4.0, 5.0 /)
deallocate(A%i, A%j, A%coeff)
```

Passage d'un type dérivé en argument d'une sous-routine

Il faut redéfinir les types dans toutes les fonctions qui

- reçoivent ou retournent des valeurs suivant ces types,
- définissent des variables locales suivant ces types.

Pratiquement, on définit les types dans un ou des modules.

Rappel :

Module : regroupement de constantes, variables, fonctions et sous-programmes, types dérivés

Exemple de définition de module :

```
module m
type MatriceCreuse
    integer :: n, m, nnz
    integer, dimension(:), pointer :: i, j
    real, dimension(:), pointer :: coeff
end type MatriceCreuse
end module m
```

Utilisation de ce module :

```
program test
use m
type (MatriceCreuse) :: A
! ...
call calcul(A)
end program test

subroutine calcul(B)
use m
type (MatriceCreuse) :: B
! ...
end subroutine calcul
```