Les Tableaux

Les tableaux permettent de manipuler plusieurs informations de même type, de leur mettre un indice : la $1^{ère}$ info, la $2^{ème}$ info, ..., la $i^{ème}$ info, ...

Ils sont stockés en mémoire centrale comme les autres variables, contrairement aux fichiers qui sont stockés sur le disque.

Une propriété importante des tableaux est de permettre un accès direct aux données, grâce à l'indice.

On appelle souvent *vecteur* un tableau en une dimension.

1 Le type `array`

1.1 Principe

Syntaxe

```
array [ I ] of T
```

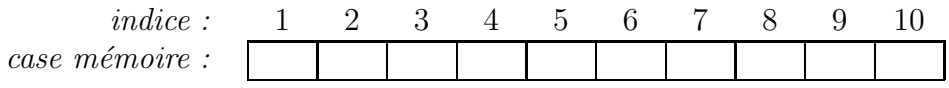
`I` étant un type intervalle, et `T` un type quelconque.

Ce type définit un tableau comportant un certain nombre de cases de type `T`, chaque case est repérée par un indice de type `I`.

Exemple

```
TYPE vec_t = array [1..10] of integer;
VAR  v : vec_t;
```

`v` est un tableau de 10 entiers, indicés de 1 à 10.



- ▷ À la déclaration, le contenu du tableau est indéterminé, comme toute variable.
- ▷ On accède à la case indice `i` par `v[i]` (et non `v(i)`).
- ▷ Pour mettre toutes les cases à 0 on fait

```
for i := 1 to 10 do v[i] := 0;
```

Remarque L'intervalle du `array` peut être de tout type intervalle, par exemple `1..10`, `'a'..'z'`, `false..true`, ou encore un intervalle d'énumérés `Lundi..Vendredi`.

On aurait pu déclarer `vecteur` comme ceci (peu d'intérêt) :

```
TYPE interv = 1..10 ; vec_t = array [ interv ] of integer;
```

1.2 Contrôle des bornes

Il est en général conseillé de repérer les bornes de l'intervalle avec des constantes nommées : si on décide de changer une borne, cela est fait à un seul endroit dans le programme.

L'écriture préconisée est donc

```
CONST vec_min = 1; vec_max = 10;
TYPE  vec_t = array [vec_min..vec_max] of integer;
```

Règle 1

Il est totalement interdit d'utiliser un indice en dehors de l'intervalle de déclaration, sinon on a une erreur à l'exécution.

Il faut donc être très rigoureux dans le programme, et ne pas hésiter à tester si un indice `i` est correct *avant* de se servir de `v[i]`.

Exemple Programme demandant à rentrer une valeur dans le vecteur.

```
CONST vec_min = 1; vec_max = 10;
TYPE  vec_t = array [vec_min..vec_max] of integer;
VAR   v : vect_t; i : integer;
BEGIN
    write ('i ? '); readln(i);
    if (i >= vec_min) and (i <= vec_max)
    then begin
           write ('v[', i, '] ? '); readln(v[i]);
         end
    else writeln ('Erreur, i hors intervalle ',
                  vec_min, '..', vec_max);
END.
```

Règle 2 Le test d'un indice `i` et de la valeur en cet indice `v[i]` dans la même expression sont interdits.

Exemple

```
if (i >= vec_min) and (i <= vec_max) and (v[i] <> -1) then ...
                                                     else ...;
```

Une expression est toujours évaluée en intégralité ; donc si `(i <= vec_max)`, le test `(v[i] <> -1)` sera quand même effectué, alors même que l'on sort du vecteur ! Solution : séparer l'expression en 2.

```
if (i >= vec_min) and (i <= vec_max)
then if (v[i] <> -1) then ...
                     else ...
else ... { erreur hors bornes } ;
```

1.3 Recopie

En Pascal, la *seule* opération globale sur un tableau est : recopier le contenu d'un tableau `v1` dans un tableau `v2` en écrivant : `v2 := v1;`

Ceci est équivalent (et plus efficace) que
`for i := vec_min to vec_max do v2[i] := v1[i];`

Il y a une condition : les 2 tableaux doivent être *exactement* de mêmes types, i.e issus de la *même déclaration*.

```
TYPE
    vecA = array [1..10] of char;
    vecB = array [1..10] of char;
VAR
    v1 : vecA; v2 : vecA; v3 : vecB;
BEGIN
    v2 := v1;  { legal car meme type vecA }
    v3 := v1;  { illegal, objets de types <> vecA et vecB }
```

2 Super tableaux

Quelques types un peu plus complexes à base de tableaux, et de combinaisons entre types.

2.1 Tableaux à plusieurs dimensions

Exemple : dimension 1 = vecteur ; dimension 2 = feuille excel ; dimension 3 = classeur excel [faire petit schéma].

On peut créer des tableaux à plusieurs dimensions de plusieurs manières :

Faire des schémas

- `v1 : array [1..10] of array [1..20] of real`
  $\longrightarrow$ Tableau de 10 éléments, chaque élément étant un tableau de 20 réels.
  On accède à l'élément d'indice `i` dans `1..10` et `j` dans `1..20` par `v1[i][j]`.
- `v2 : array [1..10, 1..20] of real`
  $\longrightarrow$ Tableau de 10×20 réels.
  On accède à l'élément d'indice `i` dans `1..10` et `j` dans `1..20` par `v2[i,j]`.

Exemple Mise à 0 du tableau `v2`.

```
VAR
    v2 : array [1..10, 1..20] of real;
    i, j : integer;
BEGIN
    for i := 1 to 10 do
      for j := 1 to 20 do
        v2[i,j] := 0.0;
END.
```

2.2 Tableaux de `record`

On peut créer des tableaux d'enregistrements, et des enregistrements qui contiennent des tableaux.

```
PROGRAM Ecole;
CONST
    MaxEleves = 35;
    MaxNotes  = 10;
TYPE
    note_t = array [1..MaxNotes] of real;

    eleve_t = Record
                age, nb_notes : integer;
                notes : note_t;
                moyenne : real;
              End;
    classe_t = array [1..MaxEleves] of eleve_t;
VAR
    c : classe_t;
    nb_eleves, i, j : integer;
BEGIN
    { ... }
    for i := 1 to nb_eleves do
    begin
      writeln ('Eleve n.', i);
      writeln ('  age   : ', c[i].age);
      write   ('  notes :');
      for j := 1 to c[i].nb_notes do write (' ', c[i].notes[j]);
      writeln;
      writeln ('  moy   : ', c[i].moyenne);
    end;
END.
```

• On a comme d'habitude le droit de faire une copie globale entres variables du même type :

```
VAR c1, c2 : classe_t;
    e : eleve_t;  i, j : integer;
BEGIN
    { copie globale de type classe_t }
    c2 := c1;
    { échange global de type eleve_t }
    e := c1[i]; c1[i] := c1[j]; c1[j] := e;
END.
```

• Exemple de passages de paramètres : on écrit une procédure affichant un `eleve_t`.

```
PROCEDURE affi_eleve (e : eleve_t);
VAR j : integer;
BEGIN
    writeln ('  age   : ', e.age);
    write   ('  notes : ');
    for j := 1 to e.nb_notes do write (e.notes[j]);
    writeln;
    writeln ('  moy   : ', e.moyenne);
END;
```

```
BEGIN
    { ... }
    for i := 1 to nb_eleves do
    begin
      writeln ('Eleve n.', i);
      affi_eleve (c[i]);
    end;
END.
```

`affi_eleve(e)` ne connait pas le numéro de l'élève ; l'appelant, lui, connait le numéro, et l'affiche avant l'appel.

On peut encore écrire une procédure `affi_classe` :

```
PROCEDURE affi_classe (c : classe_t ; nb : integer);
VAR i : integer;
BEGIN
    for i := 1 to nb do
    begin
      writeln ('Eleve n.', i);
      affi_eleve (c[i]);
    end;
END;

BEGIN
    { ... }
    affi_classe (c, nb_eleves);
END.
```