# 10 - Tableaux

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

# Tableau: définition

- Objet constitué de composants homogènes
  - Tous du même sous-type
- La valeur d'un objet tableau est constituée de la valeur de ses composants
- Caractérisation d'un objet tableau
  - Le type des éléments qu'il contient
  - Le type des indices qui permettent d'adresser les éléments du tableau
- Dimension d'un tableau: le nombre d'indices identifiant un élément du tableau
  - 1 dimension: «droite»
  - 2 dimensions: «plan»
  - 3 dimensions: «espace»...

# Déclaration d'un type tableau

```
decl_tab       type type_id1 is array (spec_indices) of type_id2 ;

spec_indices   spec_indice {, spec_indice }

spec_indice    valeur_discrète .. valeur_discrète
               type_id

type_id        identificateur
```

```
-- Un type tableau à une dimension
type Table is array (1 .. 10) of Integer;

-- Un type tableau à 2 dimensions
type Matrice_10_10 is array (1 .. 10, 1 .. 10) of Integer;

-- utilisation d'un type discret en indice
type Jour is (Lu, Ma, Me, Je, Ve, Sa, Di);
type Agenda is array (Jour) of Boolean;
```

# Déclaration d'un objet tableau sans passer par un type



- C'est possible mais attention
  - En cas de fort typage, deux objets tableaux ayant une déclaration identique sont de types différents

- Exemple

```
La_Table : array (1 .. 10) of Integer;
La_Matrice_10_10 : array (1 .. 10, 1 .. 10) of Integer;
L_Agenda : array (Jour) of Boolean;
Le_Tab_1, Le_Tab_2 : array (1..10) of Integer ; -- Deux types différents
```

# Accès aux éléments d'un tableau



```
-- définition: Programme d'exemple sur une matrice
with Ada.Text_Io, Ada.Integer_Text_IO, Construit_Matrice;
Use Ada.Text_Io, Ada.Integer_Text_IO;

procedure Essai_Matrice (N : in Positive) is

   type Matrice_N_N is array (1 .. N, 1 .. N) of Integer;
   M : Matrice_N_N;

   procedure Construit_Matrice (MM : out Matrice_N_N) is -- exercice ;-)

...

begin
   Construit_Matrice (M);
   Put_Line ("Voici la matrice construite");
   for I in 1 .. N loop
      for J in 1 .. N loop
         Put (M(I,J));
         Put (" ");
      end loop;
      New_Line;
   end loop;
end Essai_Matrice;
```

# Agrégats tableaux

- La notion de valeur immédiate pour un tableau doit être étendue
  - C'est un ensemble de valeurs immédiates
  - La valeur de chaque élément du tableau doit être définie
    - On ne peut définir d'agrégat partiel
- Il est intéressant de pouvoir représenter des valeurs immédiates composées

# Exemples d'agrégats-tableaux

Un agrégat-tableau peut se trouver dans n'importe quelle expression Ada

```
type Tableau is array (1 .. 10) of Integer;
T1 : Tableau := (1, 3, 5, 7, 9, 11, 13, 15, 17, 19);
T2 : Tableau := (1 => 1, 2 => 3, 3 => 5, 4 => 7, 5 => 9,
                 6 => 11, 7 => 13, 8 => 15, 9 => 17, 10 => 19);

type Matrice_3_4 is array (1 .. 3, 1 .. 4) of Integer;
type Matrice_4_4 is array (1 .. 4, 1 .. 4) of Integer;
M1 : Matrice_3_4 := ((0, 1, 3, 4),
                     (5, 6, 7, 8),
                     (9, 10, 11, 12));
M2:  Matrice_4_4 := (1 => (1 | 3 => 0, others => 1),
                     2..3 => (4, 5, 6, 6),
                     others => (others => 0));

type Matrice_6_6 is array (1 .. 6, 1 .. 6) of Integer;
Unite : Matrice_6_6 := (1 => (1 => 1, others => 0),
                        2 => (2 => 1, others => 0),
                        3 => (3 => 1, others => 0),
                        4 => (4 => 1, others => 0),
                        5 => (5 => 1, others => 0),
                        6 => (6 => 1, others => 0));
```

# Tranches de tableaux



- Cette notion permet de désigner un «sous-tableau»

```
type Tableau is array (1 .. 10) of Integer;
T1, T2 : Tableau;

…
T1 (1..5) := T2 (4..8); -- sur deux tableaux différents
T1 (1..3) := T1 (4..6); -- sur un même tableau (sans recouvrement)
T1 (1..5) := T1 (4..8); -- sur un même tableau (avec recouvrement)
```

- Remarque: une variable-tableau est sa propre tranche

```
T1 := T2;
```

# Tableaux non contraints



- Objectif: un type tableau dont la taille n'est définie qu'à l'utilisation
  - Au moment de la création d'une variable
    - Déclaration pour les variables-tableaux
    - Allocation de mémoire pour les pointeurs sur des tableaux

```
type Vecteur is array (Integer range <>) of Integer;
type Matrice is array (Integer range <>, Integer range <>) of
  Integer;

Vec2  : Vecteur (1 .. 2);   -- Vec2 et Vec10 sont du même type
Vec10 : Vecteur (1 .. 10);  -- (mais de sous-types différents)

M_2_2 : Matrice (1 .. 2, 1 .. 2);  -- M_2_2 et M_5_7 sont du même type
M_5_7 : Matrice (-1 .. 3, 2 .. 8); -- (mais de sous-types différents)
```

- Une variable-tableau-non-contraint s'utilise comme une variable-tableau
- Utile pour les passages de paramètres dans les sous-programmes

# Opérations prédéfinies pour les types tableaux



- Affectation, comparaison d'égalité
  - := et =, s'appliquent à un élément, une tranche ou le tableau
    - Les sous-types peuvent être différents si les deux parties de l'affectation sont de même longueur
- Concaténation (tableaux de dimension 1)
  - & s'applique à des tableaux, tranches ou des éléments de tableaux

# Note complémentaire sur les tableaux

- Certains langages offrent des opérations complémentaires associées aux tableaux
  - Ada => attributs prédéfinis

```
Nom_Variable'First(Valeur) ou Nom_Variable'First
Nom_Variable'Last(Valeur) ou Nom_Variable'Last
Nom_Variable'Range(Valeur) ou Nom_Variable'Range
Nom_Variable'length(Valeur) ou Nom_Variable'Length
```

Une dimension

Expressions statiques

# Exemple d'utilité des attributs de tableaux (1)



Obtenir des informations sur le dimentionnement d'un type non contraint

```
type Matrice is array (Integer range <>, Integer range <>)
      of Integer;

-- Addition entre deux matrices (version simple et sans tests)

function Plus (M1, M2 : Matrice) return Matrice is

   R : Matrice (M1'Range (1), M1'Range (2));

begin
   for I in M1'Range (1) loop
      for J in M1'Range (2) loop
         R (I, J) := M1 (I, J) + M2 (I, J);
      end loop;
   end loop;
   return R;
end Plus;
```

# Exemple d'utilité des attributs de tableaux (2)



Seconde version plus stable (intervalle d'indice de même taille)

```
type Matrice is array (Integer range <>, Integer range <>)
      of Integer;

-- Addition entre deux matrices (version plus sophistiquée)

function Plus (M1, M2 : Matrice) return Matrice is

   R : Matrice (0 .. M1'Length (1) - 1,
                0 .. M1'Length (2) - 1);

begin
   for I in 0 .. M1'Length (1) - 1 loop
      for J in 0 .. M1'Length (2) - 1 loop
         R (I, J) := M1 (M1'First(1) + I, M1'First(2) + J) +
                     M2 (M2'First(1) + I, M2'First(2) + J);
      end loop;
   end loop;
   return R;
end Plus;
```