

---

# Algorithmique & programmation

---

## Chapitre 2 : Vecteurs en Ada (première partie) Introduction

MCours.com

---

### Introduction

---

- En ada, un vecteur est de type **array**
- Les éléments d'un **array** ont le même type
  - scalaire
  - article, enregistrement, structure
  - vecteur
    - ⚠ un vecteur de vecteurs est un **tableau**
- Les indices d'un **array** peuvent être des éléments de n'importe quel type discret
  - des entiers (⚠ seul type utilisé en algorithmique)
  - des types énumérés
    - en particulier des caractères ou des booléens

# Vecteurs Contraints vs non Contraints

---

- On utilise les types **vecteurs contraints** pour
  - les déclarations de variables
    - À toute variable est associée de la place mémoire dont le compilateur a besoin de connaître la taille au moment de l'élaboration de la déclaration
    - Pour cette raison toute déclaration de variable ou de type de variable vecteur contraint le programmeur à préciser un intervalle d'indices

# Vecteurs Contraints vs non Contraints

---

- On utilise les types **vecteurs non contraints** pour
  - les déclarations de type
  - pour les paramètres formels des procédures et fonctions

## Vecteurs non contraints & paramètres formels

---

- On utilise des vecteurs non contraints comme paramètres formels dans une déclaration de sous-programme (procédure ou fonction)
  - c'est une facilité d'écriture pour désigner des valeurs (paramètres "donnés") ou des variables (paramètres "résultats")
  
- En effet ...
  - la déclaration d'une procédure ou d'une variable n'entraîne pas de réservation de mémoire pour les paramètres.
  
- Cela permet au programmeur de définir des paramètres vecteurs non contraints par leurs indices, autrement dit des vecteurs dont le nombre d'éléments peut être variable selon les appels.

## Déclaration d'un type vecteur

---

- Dans la déclaration d'un type vecteur (**array**), on peut
  - **contraindre** ou non l'intervalle des indices (**range**), autrement dit la **borne inférieure** et la **borne supérieure**
  
- Rappel : le **type des indices** peut être tout (sous-) type **discret** : **Integer**, **Character**, type énuméré.
  - ⚠ Ce type ne peut pas être **Float** ou un sous-type de **Float**
  
- Nous conviendrons de faire précéder tout nom de type vecteur par le préfixe **TV\_**

# Exemples de déclarations

---

## □ Types de vecteurs non contraints

```
type TV_Suite          is array (Integer range <>) of Float ;

type TV_Indicateurs is array (Natural range <>) of Boolean ;

type String           is array (Positive range <>) of Character ;
                        -- prédéfini
```

- Le symbole <> indique que l'intervalle des indices n'est pas contraint, c'est-à-dire quelconque dans l'intervalle d'indices du sous-type **Positive** ou **Natural**

# Exemples de déclarations

---

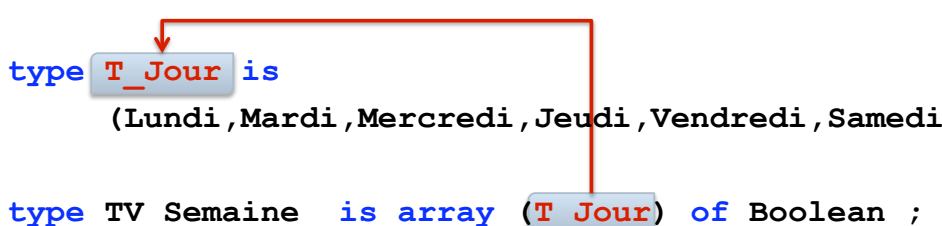
## □ Types de vecteurs contraints

```
type TV_Mesures is array (101..200) of Float ;

type TV_Ligne   is array (0..79) of Character ;

type T_Jour is
  (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche) ;

type TV_Semaine is array (T_Jour) of Boolean ;
```



# Exemples de déclarations

## □ Sous-types de vecteurs contraints dérivés de types vecteur non contraint

```
--type TV_Suite is array (Integer range <>) of Float ;
subtype TV_Echantillonnage is TV_Suite(1..365) ;

--type String is array (Positive range <>) of Character;
subtype TV_Spécial is String(128..255) ;
subtype TV_Truc is String(0..9) ;
-- Incorrect car 0 n'est pas du type Positive

--type TV_Semaine is array (T_Jour) of Boolean;
subtype TV_WeekEnd is TV_Semaine(Samedi..Dimanche) ;
```

## Attributs des (sous-)types vecteurs contraints

- TV\_Mesures' **First**
  - Borne inférieure des vecteurs de type TV\_Mesures : 101
- TV\_Mesures' **Last**
  - Borne supérieure des vecteurs de type TV\_Mesures : 200
- TV\_Mesures' **Length**
  - Nombre d'éléments des vecteurs de type TV\_Mesures : 100
- TV\_Mesures' **Range**
  - Intervalle d'indices des vecteurs de type TV\_Mesures : 101..200
  - TV\_Mesures' Range peut aussi être noté:
    - TV\_Mesures'First..TV\_Mesures'Last
    - $TV\_Mesures'Length = TV\_Mesures'Last - TV\_Mesures'First + 1$

## Attributs des (sous-)types vecteurs contraints

---

- L'utilisation de ces attributs n'a pas de sens pour les types de vecteurs non contraints
  
- Ces attributs ...
  - ... peuvent par contre s'appliquer aux **variables** de type **vecteurs**, ou aux **paramètres formels même de type non contraint**
  
  - ⚠ ... sont **indispensables** pour écrire **correctement** des sous-programmes ayant en paramètre des vecteurs
  
  - ⚠ **Attention** : en algorithmique les bornes des vecteurs sont notées dans les paramètres formels

## Déclaration d'une variable de type vecteur

---

- Lors de l'exécution, ...
  - ... l'élaboration d'une déclaration d'une variable ...
    - quel que soit son type,
  - ... nécessite que la **taille mémoire** (nombre d'octets) de cette variable puisse être **calculée**

***Ceci est une règle très générale,  
quel que soit le langage de programmation.***

- Par conséquent, en Ada,
  - ⚠ le **type** utilisé dans la déclaration d'une **variable vecteur** est **nécessairement contraint**

# Type non contraint `String`

## ❑ Exemples de déclarations

```
S1 : String ;  
-- incorrect car on ne peut déclarer une variable d'un type  
non contraint  
  
S2 : String(10..20) ;  
  
S3 : String := "Madame" ;  
-- s3 est du type contraint string(1..6)  
  
S4 : String(1..6) := "Madame" ;  
  
S5 : String(1..6) := "Monsieur" ;  
-- Incorrect car "Monsieur" Length est différent de s5 length
```

# Règle de bonne conduite



Il convient de faire dépendre le moins possible l'écriture d'un programme du nombre d'éléments des vecteurs

- ❑ Voici deux façons de réaliser cet objectif pour un type **TV\_Suite2N**

```
N : constant integer := 10 ; -- Nombre d'éléments des tableaux  
subtype TV_Suite2N is TV_Suite(2..2+N-1) ;  
-- 10 éléments numérotés de 2 à 11  
  
V : TV_Suite2N ;
```

```
N : constant integer := 10 ; -- Nombre d'éléments des tableaux  
subtype T_Indices is Positive range 2..2+N-1 ;  
subtype TV_Suite2N is TV_Suite(T_Indices) ;  
V : TV_Suite2N ;
```

# Règle de bonne conduite

---

- ❑ Une raison de plus pour déclarer des types pour les vecteurs, même quand ces types sont contraints :

```
V1: array(1..9) of Integer ;
V2: array(1..9) of Integer ;
```

Les types de **V1** et **V2** seront considérés comme **différents** et par exemple l'affectation **V1 := V2**; sera **refusée** par le compilateur.

- ❑ On résoudra facilement le problème en déclarant plutôt :

```
type TV_T9 is array(1..9) of Integer ;
V1, V2 : TV_T9 ;
      -- V1 et V2 sont bien de même type
```

# Opérations sur les vecteurs

---

- ❑ Dans les expressions, on peut désigner un **élément** ou une **tranche** d'un vecteur :

```
...
V,W : T_Mesures ;
      -- rappel : 100 Float numérotés de 101 à 200
...
V(150)      := 18.0 ;
W(103)      := V(104) ;
V(105..107) := W(103..105) ;
      -- les tranches doivent avoir la même longueur
...
```



# Opérations sur les vecteurs

---

- ❑ Les **agrégats** donnent un moyen pratique de noter des constantes et en particulier d'initialiser une variable vecteur :

```
V(103..107) := (103..105 => 12.0, others=>0) ;
```

- ❑ La **concaténation** (opérateur &) est applicable à tout type de vecteur :

```
V(I..J) := V(I..K-1) & 9.999 & V(K+1..J) ;
```

- ❑ Attention : les longueurs de l'expression et de V(I..J) doivent être égales.
- ❑ Les **comparaisons**, mais **seulement** l'égalité et la différence, peuvent être utilisées:

# Exemples de comparaisons

---

```
if V(I) = V(I+1) then .... end if ;
```

```
-- élément = élément ?
```

```
if V(I..I+K) = W(J..J+K) then .... end if ;
```

```
-- tranche = tranche ?
```

```
if V = W then .... end if ;
```

```
-- vecteur = vecteur ?
```

## Boucles *pour* de parcours de Vecteur

---

- La boucle **pour** (`for`) est particulièrement bien adaptée au parcours séquentiel total ou partiel d'un vecteur
- On peut faire ce parcours dans le **sens croissant** des indices :

```
for I in V'range loop
  Ecrire(V(I)) ;
end loop ;
-- écrit V(101), V(102), ..., V(200)
```

## Boucles *pour* de parcours de Vecteur

---

- On peut faire le parcours dans le **sens décroissant** des indices :

```
for I in reverse 107..109 loop
  Ecrire(V(I)) ;
end loop ;
--écrit V(109), V(108), V(107)
```