Algorithmique & programmation

Chapitre 2 : Vecteurs en Ada (seconde partie)

Paramètre de type vecteur

Intervalle d'indices inconnu avant exécution

Paramètres de type vecteur

- Rappel : le type d'une **variable vecteur** doit être **contraint**
  - L'intervalle des indices doit être défini
- Pour les **paramètres formels de type vecteur** des sous-programmes (fonctions & procédures), on peut utiliser des **vecteurs non contraints**
- **Avantage**
  - **définir des algorithmes** sur les vecteurs **indépendamment des longueurs de ces vecteurs et des valeurs des bornes de l'intervalle des indices**, et sans avoir à passer ces informations dans des paramètres supplémentaires
- Comment manipule-t-on les indices sur des vecteurs non contraints ?

On utilise les attributs **`First`**, **`Last`**, **`Length`**, et **`Range`**

Paramètre formel vecteur non contraint

- Écrivons une fonction de comparaison du nombre d'éléments de deux vecteurs (**V1** & **V2**) qui
  - rend **vrai** si et seulement si
    - **les deux vecteurs ont la même longueur**
- Code Ada

```
function MemeLongueur( V1, V2 : TV_Suite) return Boolean is
  -- retourne TRUE si même longueur
begin
  if V1'Length = V2'Length then
    return True ;    -- longueurs égales
  else
    return False ;   -- longueurs différentes
  end if ;
end MemeLongueur ;
```

Vecteurs non contraints

Utilisation de l'attribut longueur

Exemples d'appels

- Les paramètres effectifs sont ici des tranches d'un même vecteur W

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

W := (101..150 => 1.0; 151..200 => 2.0) ;

B := MemeLongueur(W(111..120), W(111..130)) ;
-- renvoie FALSE, longueurs différentes

B := MemeLongueur(W(111..120), W(160..169)) ;
-- renvoie TRUE
```

Paramètre formel vecteur non contraint

- Fonction de comparaison de deux vecteurs (V1 & V2) qui
  - rend vrai si et seulement si
    - les deux vecteurs ont la même longueur
    - chaque élément de V1 est plus petit que l'élément de même rang de V2

code planche suivante


```
function Estpluspetit(V1, V2: TV_Suite) return Boolean is
   -- retourne TRUE si même longueur et chaque élément
   -- de V1 plus petit que l'élément de même rang de V2

   InfV1 : Integer := V1'First;
   InfV2 : Integer := V2'First;
   K     : Natural;

begin
   if V1'Length = V2'Length  then
      K := 0;
      while K < V1'Length
                  and then V1(InfV1+K)) < V2(InfV2+K)) loop
         K := K + 1 ;
      end loop ;
      return K = V1'Length ;
   else
      return False ;  -- Longueurs différentes
   end if ;
end Estpluspetit ;
```

Exemples d'appels

- Les paramètres effectifs sont ici des tranches d'un même vecteur W

```
--type TV_Suite is array (Integer  range <>) of Float ;
W := (101..150 => 1.0; 151..200 => 2.0) ;

B := Estpluspetit(W(111..120), W(111..130)) ;
-- renvoie FALSE, longueurs différentes

B := Estpluspetit(W(111..120), W(160..169)) ;
-- renvoie TRUE

B := Estpluspetit(W(111..112), W(111..112)) ;
-- renvoie FALSE, valeurs égales
```


Vecteur à intervalle d'indices inconnu du programmeur

- Il arrive que l'on ait à déclarer des variables de type **`Vecteur`** dont les bornes sont inconnues lors de l'écriture du programme source
  - C'est le cas quand on laisse le soin à l'utilisateur du programme de les saisir manuellement au clavier
- On présente ici une façon de procéder dans un tel cas sur un exemple.
  - *L'usage de pointeurs et d'allocation dynamique de mémoire donneront plus tard une autre solution*.

Exemple

- Problème :
  - Pour une période d'années inconnue du programmeur on se propose de construire un **`Vecteur`** du nombre de jours de gel par an, de calculer la moyenne de ces jours de gel, puis de décider au vu du résultat si la période considérée est anormalement froide ou non.
- On va faire comme suit :
  - Déclarer un type de **`Vecteurs`** non contraints **`TV_Meteo`**
  - Demander à l'utilisateur de saisir des bornes **`AnnéeInf`** et **`AnnéeSup`** du **`Vecteur`** à considérer
  - Déclarer une variable **`Vecteur Vtemp`** obtenue en utilisant ces bornes

Point important :

- Afin que les valeurs des bornes soient connues au moment de sa déclaration, il faut déclarer, saisir et traiter le **`Vecteur Vtemp`** dans un **`sous-bloc`** **d'instructions**

Notion de Bloc d'Instructions

- En Ada un **bloc d'instructions** désigne un ensemble d'instructions placées entre **`begin`** et **`end`**
- Un bloc peut avoir (ou non) des variables locales que l'on déclare en les plaçant entre les mots **`declare`** et **`begin`**
- Un bloc peut avoir (ou non) un nom
  - Dans l'exemple on lui en donnera un

Spécification du paquetage P_Meteo

```
package P_Meteo is

   subtype T_NbJours is Natural range 0..366 ;

   type TV_Meteo is array(Integer range <>) of T_NbJours ;

   procedure Lire(V:out TV_Meteo) ;

   function Moyenne(V:in TV_Meteo) return Float ;
   -- délivre la moyenne des valeurs d'un Vecteur
   -- d'entiers

end P_Meteo ;
```


Corps du paquetage P_Meteo (début)

```
   with P_Esiut; use P_Esiut;

   package body P_Meteo is

   procedure Lire(V : out TV_Meteo) is
   -- le tableau V a été rempli avec des valeurs
   -- saisies au clavier
     begin
       Ecrire_Ligne("Nombre de jours de gel :") ;
       for I in V'range loop
         Ecrire("Année ") ; Ecrire(I); Ecrire(" : ") ;
         Lire(V(I)) ;
       end loop ;
     end Lire ;
   ...
```

Corps du paquetage P_Meteo (fin)

```
...
function Moyenne(V: in TV_Meteo) return Float is
-- résultat = la moyenne des éléments de V
    Total: Natural := 0 ;
  begin
    for I in V'range loop
      Total := Total + V(I) ;
    end loop ;
    return Float(Total)/Float(V'Length) ;
   end Moyenne ;

end P_Meteo ;
```


Utilisation de P_Meteo (début)

```
with P_Esiut; use P_Esiut;
with P_Meteo; use P_Meteo;

procedure Statsgel is

  subtype T_Année is Integer range 1900..2050 ;
                    -- limite l'intervalle possible
  AnnéeInf, AnnéeSup : T_Année ;
  LaMoyenne          : Float ;
  SEUIL: constant    := 20 ;

begin

  …

end Statsgel ;
```

```
procedure Statsgel is
  -- déclaration des sous-types, variables et constantes
begin
  -- lecture de AnnéeInf et AnnéeSup
  BlocMeteo:  -- étiquette de début de bloc interne, facultative
  -- élaboration, saisie et traitement du Vecteur contraint
  declare
    Stats: TV_Meteo(AnnéeInf..AnnéeSup) ;
    -- les bornes sont des variables lues
  begin
    Lire(Stats) ;
    LaMoyenne := Moyenne(Stats) ;
  end BlocMeteo ; -- fin du bloc interne
  -- affichage des résultats finaux
  Ecrire("La période de ") ; Ecrire(AnnéeInf,4);Ecrire(" à ") ;
  Ecrire(AnnéeSup,4) ; Ecrire(" est ") ;
  if LaMoyenne > Float(SEUIL) then
    Ecrire_Ligne("froide.") ;
  else
    Ecrire_Ligne("normale.") ;
  end if ;
end Statsgel ;
```

bloc principal de la procédure

sous-bloc d'instructions du bloc principal

Déclaration d'un sous-type dont les bornes sont fixées lors de l'exécution

Vecteurs à plusieurs dimensions

- On peut considérer qu'un livre est un vecteur de pages, chaque page étant un vecteur de lignes, chaque ligne étant un vecteur de caractères
- Toutes les pages ont un **même nombre** de lignes et toutes les lignes ont un même nombre de caractères.
- On dit qu'une telle structure est un vecteur multidimensionnel (ici 3 dimensions).
- On déclare en Ada une variable **Livre** par :

```
Livre : array(1..Nbpage,1..Nbligne,1..Nbcar) of Character ;
```

Vecteurs à plusieurs dimensions

- Le 5ème caractère de la 1ère ligne de la page 2 s'obtient alors par :

```
Livre(2,1,5);
```

- **`Livre'First(1)`**
  désigne l'indice de la première page du livre (**1**)
- **`Livre'Last(2)`** désigne
  l'indice de la dernière ligne des pages (**`Nbligne`**)
- **`Livre'Length(3)`** désigne
  le nombre de caractères par ligne (**`Nbcar`**)

Vecteurs à plusieurs dimensions

- Le parcours de ces Vecteurs s'effectue en utilisant plusieurs boucles imbriquées.
- Pour faire une saisie on écrirait :

```
for I in Livre'range(1) loop
  for J in Livre'range(2) loop
    for K in Livre'range(3) loop
      Lire(Livre(V(I,J,K)));
    end loop ;
  end loop ;
end loop ;
```