Algorithmique & programmation

ADA

Sous-types

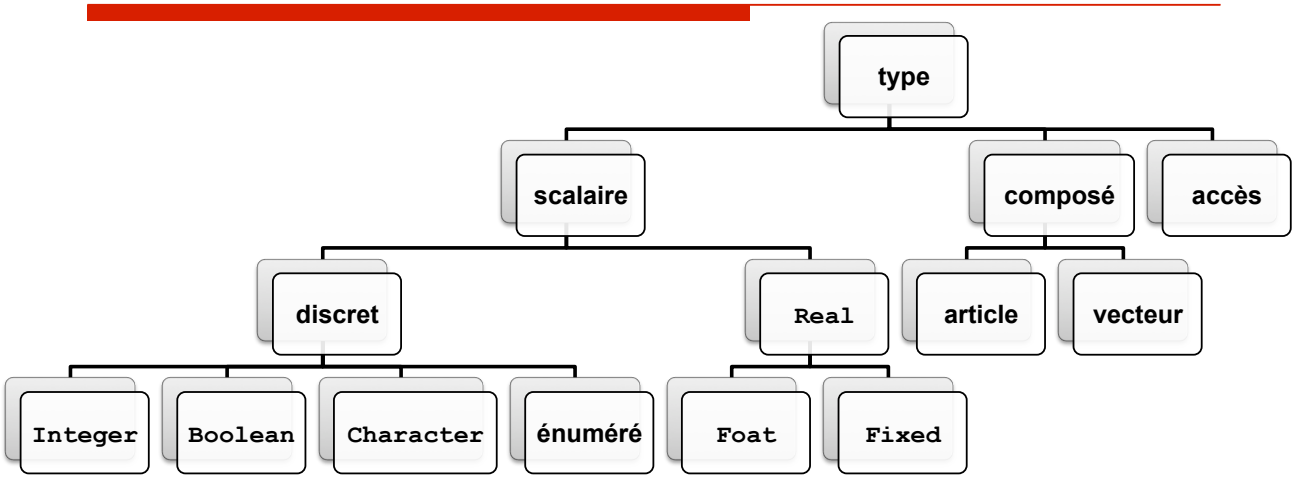
Types énumérés

Attributs

Rappel sur les types

- Un type est caractérisé par un ensemble de valeurs muni d'un ensemble d'opérations
  - le type **`Integer`** est l'ensemble des entiers muni des opérations `+, -, *, /, mod, **`

Hiérarchie des types



- Type discret
  - dont on peut numéroter les valeurs
- *Article : type structuré*
- *Vecteur : tableau (gestion statique de la mémoire)*
- *Accès : pointeur (gestion dynamique de la mémoire)*

Contenu de la section

- Les types scalaires, qui sont des ensembles de valeurs élémentaires
  - Types discrets
    - c'est à dire ceux dont on peut numéroter les valeurs:
    - `Integer`
    - types énumérés prédéfinis
      - `Boolean` et `Character`
    - types énumérés définis par le programmeur
  - Type `Real`

- *Les types composés (vecteurs, articles) et les types accès seront introduits par la suite.*

Notion de sous-type

- Un sous-type d'un type **`T`** est caractérisé par
  - un sous-ensemble des valeurs de **`T`**
  - et le même ensemble d'opérations que celui de **`T`**

- Exemple
  - le sous-type **`Natural`** prédéfini dans Ada est un sous-type du type **`Integer`**
    - c'est l'ensemble des entiers restreint aux entiers positifs ou nuls
    - et des opérations (héritées du type `Integer`) `+, -, *, /, mod, **`

Notion de sous-type

- Un sous-type d'un type **`T`**, pour être utilisé, doit être déclaré et donc nommé

- Le sous-ensemble de valeurs du sous-type est défini au moyen d'une contrainte
  - La contrainte est en général un intervalle de valeurs du type **`T`**
    - par exemple **`Natural`** est l'ensemble des **`Integer`** positifs ou nuls

- Syntaxe de déclaration

```
subtype nom_sous_type is nom_de_type contrainte;
```

Exemples de déclarations

```
-- Integer'Last désigne la plus grande valeur possible
-- pour le type Integer (cf. attributs)

subtype Natural is Integer range 0..Integer'Last;
   -- prédéfini dans Ada

subtype Positive is Integer range  1.. Integer'Last;
   -- prédéfini dans Ada

subtype T_Note      is Float     range  0.0..20.0;

subtype T_Chiffre   is Character range  '0'..'9';

subtype T_Heure     is Natural   range  0..23;

subtype T_Temperature is  Float range  -50.0..50.0;
```


Évaluation d'une expression

- Soit les instructions suivantes :

```
subtype T_Note is Float range  0.0..20.0;
A : T_Note := 13.0;
B : T_Note := 12.0;
Somme, Moyenne1, Moyenne2 : T_Note;
Somme   := A+B;
   -- erreur à l'exécution ! (Somme = 25)
Moyenne := (A+B) / 2.0 ;
   -- correct !  (Moyenne = 25.0 / 2.0 = 12.5)
```

- **`Somme := A+B;`**
  - **provoque une erreur** (CONSTRAINT_ERROR) à l'exécution, car **`Somme`** prendrait la valeur 25, qui n'appartient pas à `T_Note`
- **`Moyenne := (A+B) / 2.0;`**
  - **exécutée sans erreur** car, bien que le résultat intermédiaire A+B=25 n'appartienne pas à **`T_Note`**, le résultat final de l'expression, égal à 12.5, appartient bien à **`T_Note`**

Notion de type énuméré

- Définir un type par l'ensemble des valeurs qu'il peut prendre
- Par exemple
  - Définir les jours de la semaine par leur nom au lieu de les définir par des numéros ou des chaînes de caractères
- Pour être utilisé un type énuméré doit être déclaré

Exemples de déclarations

```
type Boolean is (False, True);
   -- prédéfini dans Ada

type T_Jour is
    (lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche);

type T_Couleur is
            (Blanc, Bleu, Rouge, Jaune, Vert, Brun, Noir);

type T_Niveau is (Seconde, Première, Terminale);

subtype T_Couleur1 is T_Couleur range Blanc..rouge;
```

Remarque

- Les valeurs d'un type énuméré (`lundi, seconde, etc.`) ne sont pas des chaînes de caractères !
  - ce sont des valeurs de type
    - la valeur `3` est un **`Integer`**
    - la valeur **`lundi`** est un **`T_jour`**
- Les valeurs d'un type énuméré sont ordonnées selon l'ordre où elles sont écrites dans la déclaration.
- Cet ordre est utilisé pour les comparaisons (`=, /=, <, <=, >, >=`)

```
type T_Jour is
            (lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche);
```

  - **`mardi < jeudi`** vaut **`true`**
  - **`mardi >= jeudi`** vaut **`false`**
- On procède comme d'habitude pour déclarer ou affecter une valeur à une variable de l'un de ces types :
  - **`Jour : T_Jour := lundi;`**
  - **`N : T_Niveau := terminale;`**

Entrées/Sortie de types énumérés

- Le package **`P_ESiut`** contient lui-même le package **`P_Enum`** qui permet de gérer les entrées-sorties des constantes des types énumérés.
- Mais comme il y a autant de sortes de types énumérés qu'on le désire, ce package n'est pas un package ordinaire, on dit que c'est un **package générique**.
- Le programmeur doit **instancier** ce package générique afin de créer un nouveau package spécialisé pour le type énuméré considéré
- Un package ainsi créé par instanciation du package génériques **`P_Enum`** est un package ordinaire qui comportent les primitives :

`lire(`*`variable_de_type_énuméré`*`);`

`ecrire(`*`constante_ou_variable_de_type_énuméré`*`);`

Exemple d'E/S sur les types énumérés

```
with P_ESiut; use P_ESiut;
procedure JourConge is
   package P_BooleanIO is new P_Enum(Boolean);
   -- Instanciation de P_Enum pour ES sur Boolean
   type T_Jour is
     (Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi,Dimanche);
   package P_JourIO is new P_Enum(T_Jour);
   -- Instanciation de P_Enum pour ES sur T_Jour

   Jour: T_Jour;
   Etat: Boolean;
begin
   Etat := True;
   Jour := dimanche;
   P_JourIO.Ecrire(Jour);
   P_BooleanIO.Ecrire(Etat);
end JourConge;
```


Attributs des types scalaires

- Permettent de connaître différentes informations sur les (sous-) types et expressions
- Des fonctions prédéfinies
  - applicables à plusieurs types : le type considéré doit être spécifié et Ada impose de placer le nom du type ou du sous-type en préfixe de la fonction (de l'attribut)
- Syntaxe

```
nom_du_type'attribut
nom_du_type'attribut(expression, ....)
```

Attributs des types scalaires (exemples)

- **`Integer'First`**
  - désigne la plus petite valeur de type `Integer`
- **`Integer'Last`**
  - désigne la plus grande valeur de type `Integer`
- Sur les machines que nous utilisons en TP, les entiers sont codés en complément à 2 sur 32 bits, et par conséquent :

`Integer'first` $= -2^{31} =$ **`-231 = - 2 147 483 648`**

`Integer'last` $= 2^{31}-1 =$ **`231-1 = + 2 147 483 647`**

Attributs des (sous-) types scalaires

- **`First`**, **`Last`** donnent la plus petite ou la plus grande valeur possible d'un type
- **`Image(expression)`** convertit la valeur de l'expression en une chaîne de caractères
- **`Value(expression)`** transforme une chaîne de caractères en une valeur
- Exemples

```
T_Jour'first            =====> lundi      --un T_Jour
T_Jour'last             =====> dimanche   --un T_Jour
T_Température'first     =====> -5.0       --un flottant
Float'Image(-5.0)       =====> "-5.0"     --une chaîne
Float'Value("-5.0")     =====> -5.0       --un flottant
T_Jour'Image(lundi)     =====> "lundi"    --une chaîne
T_Jour'Value("lundi")   =====> lundi      --un  T_Jour
Integer'Image(35+2)     =====> "37"       --une chaîne
Integer'Value("37")     =====> 37         --un entier
```

Attributs des (sous-) types discrets

- Soit **T** un type discret
- On peut utiliser les attributs suivants :
  - **T'Succ(*expression*)**
    - retourne la valeur suivante d'une valeur (équivalent à + 1 pour `Integer`).
  - **T'Pred(*expression*)**
    - retourne la valeur précédente d'une valeur (équivalent à - 1 pour `Integer`).
  - **T'Pos(*expression*)**
    - retourne la position (le numéro d'ordre, **à partir de 0**) de la valeur dans `T`
  - **T'Val(*position*)**
    - retourne la valeur du type correspondant à la position
- N.B. Ne pas confondre les attributs **Val** et **Value** …

Attributs des (sous-) types discrets

- Exemples

```
T_Jour'succ(lundi)       ======> mardi
T_Jour'succ(Dimanche)    ======> incorrect !

T_Jour'pred(Mardi)       ======> Lundi

T_Jour'pos(Lundi)        ======> 0
T_Jour'pos(Mardi)        ======> 1
T_Jour'pos(Dimanche)     ======> 6

T_Jour'val(0)            ======> lundi
T_Jour'val(1)            ======> mardi

Boolean'val(1)           ======> TRUE

Integer'val(3)           ======>  3  -- sans intérêt !
```