6 - Types & variables

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

PrC

Variables & types

- Variable, espace mémoire contenant une valeur
- Type: Ensemble de «valeurs licites» + Des opérations
  - Certaines opérations peuvent être définies implicitement
    - Arithmétiques pour les types entiers & réels
    - Assemblage (concaténation) pour les tableaux
    - Besoin minimum en général admis:

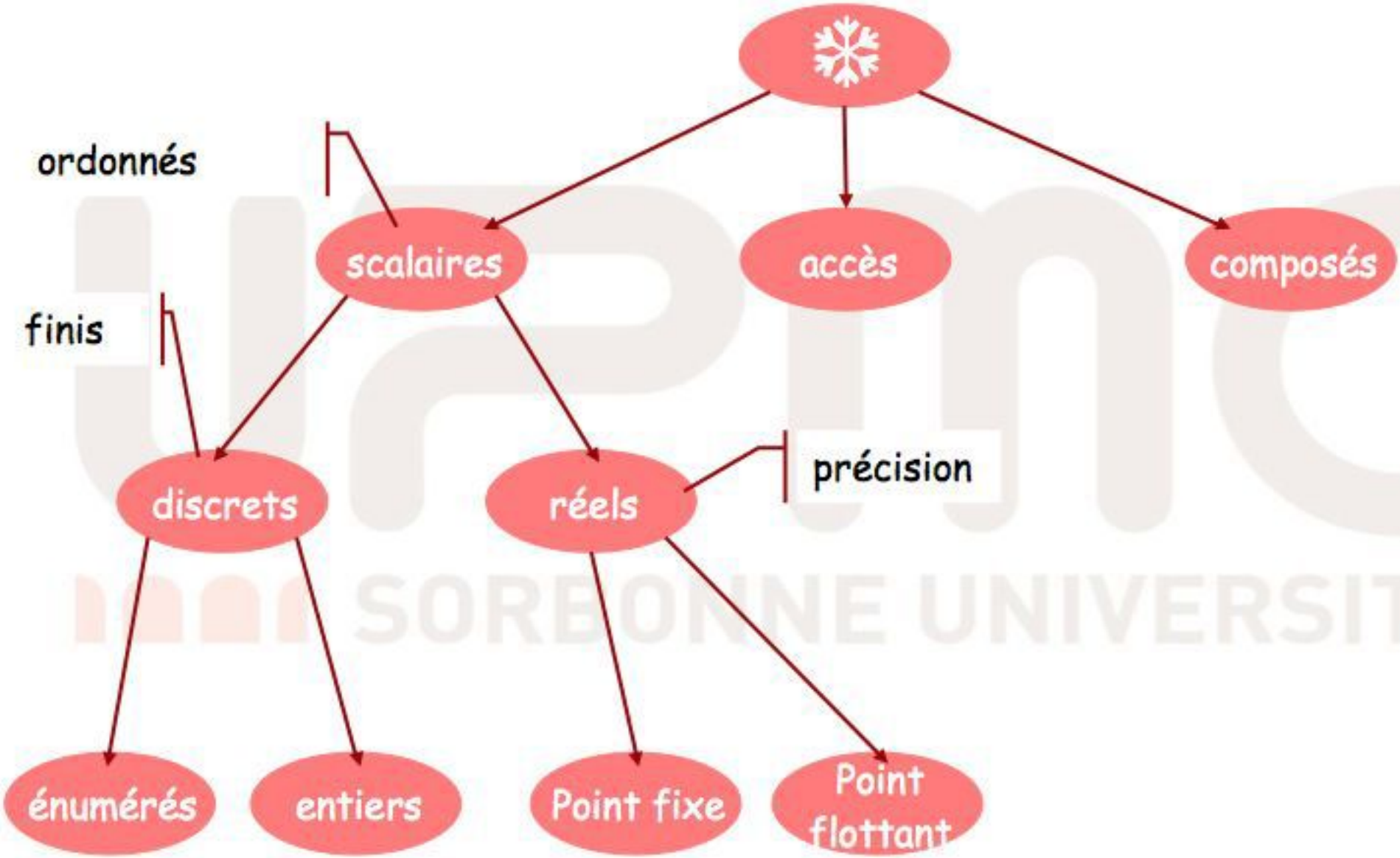
      affectation — recopie d'une zone mémoire

      comparaison d'égalité — comparaison de zones mémoires

      remarque: on souhaite parfois s'en passer!

- Opérations d'un type
  - Les opérations définies par le programmeur (s'il y a lieu)
  - Les opérations implicites si elles existent à un type

Hiérarchisation des types



Déclaration de type

Permet d'utiliser le type

Identificateur du nouveau type

```
type nom_de_type is déclaration_de_type;
```

Types prédéfinis (1)

- Objectif: disposer d'un mécanisme de type intégré au langage
  - Avantage: on ne les déclare pas
  - Inconvénient: on ne maîtrise pas leur représentation
- Booléen
  - Ensemble de deux valeurs `False`, `True` (False < True, op. booléens)
- Nombres entiers
  - `Integer`, `Long_Integer`, `Short_Integer` (opérateurs arithmétiques)
    Natural, ***Positive***
  - Méfiance: la représentation dépend du compilateur

Types prédéfinis (2)

- Nombre réels
  - Float (opérateurs arithmétiques)
  - Méfiance: la représentation dépend du compilateur
- Caractères
  - Character vu comme un type énuméré prédéfini
- Chaînes de caractères
  - String tableau de caractères (opérateur de concaténation &)

Le minimum pour manipuler des chaînes de caractères (type String)

- Déclaration d'une variable

```
La_Chaine : String(1..15);
-- chaîne de 15 caractères au maximum
```

- Utilisation de l'unité prédéfinie Ada.Text_IO
- Saisie d'une chaîne de caractères

```
La_Longueur : Natural;
Get_Line(La_Chaine,La_Longueur);
-- La_Longueur = nombre de caractères effectivement saisis
```

- Sélection d'une partie de la chaîne

```
La_Chaine(3) /= '2'
-- comparaison du 3ème caractère avec le caractère 2
Put_Line(La_Chaine(4..8));
-- affichage de la chaîne à partir du 4ème caractère et jusqu'au 8ème
```

- Ignorer des caractères tapés

```
Skip_Line;
-- Lit tous les caractères jusqu'à un caractère de fin de ligne compris
```

Plus de précisions plus tard (étude des tableaux)

Utilisation des types non prédéfinis

- Permet de «tailler» un type pour un usage particulier
  - Il faut le déclarer
  - Certains langages permettent un contrôle très fin sur la représentation (pas décrit dans le cadre de ce cours)
- Types «simples»
  - Énumérés
  - Entiers
  - Flottants
- Types «composés» (vus plus loin dans le cours)
  - Tableaux
  - Articles
  - Pointeurs

Types énumérés

- Liste d'identificateurs
  - Type couleur = {rouge, vert, bleu}
- Opérateurs associés
  - Affectation, comparaison d'égalité (et d'inégalité)
  - Comparaison d'ordre
    - Association d'un nombre entier à chaque valeur
    - Calqué sur l'ordre d'apparition des littéraux d'énumération
- Exemple

```
def_énum   type identificateur is (identificateur {, identificateur });

type Couleur is (Rouge, Vert, Bleu);
type Jour is (Lundi, Mardi, Mercredi,
              Jeudi, Vendredi, Samedi,
              Dimanche);
```

Types entiers

- Permet de préciser des caractéristiques sur les valeurs admises par une variable entière
  - Intervalles de valeurs
- Les types entiers sont toujours bornés par la représentation machine
- Exemple

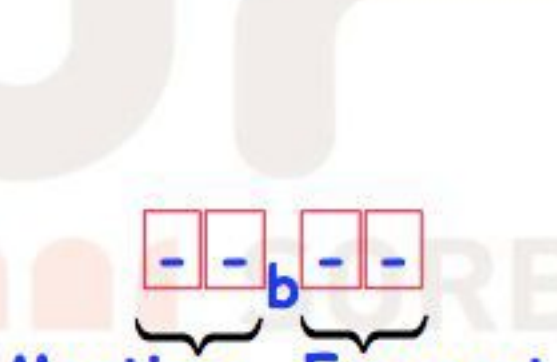
```
def_ent    type identificateur is range valeur1 .. valeur2;

type Un_Entier_Relatif is range -32268 .. 32267;
type Un_Entier_Naturel is range 0 .. 32267;
type Un_Entier_Positif is range 1 .. 32267;
```

Les types réels (principe)

- Virgule fixe
  - Valeur = <partie entière> · <Partie décimale>
  - Petits intervalles mais précision fixée

Les types réels (principe)

- **Virgule fixe**
  - Valeur = <partie entière> · <Partie décimale>
  - Petits intervalles mais précision fixée
- **Virgule flottante**
  - Valeur = Mantisse x $b^{exposant}$
  - Grands intervalles mais précision variable



$+99 \times 10^{99}$

$+1 \times 10^{-99}$

-99×10^{99}

Mise en œuvre

- Permet de préciser des caractéristiques sur la représentation d'un nombre réel
  - Précision
    - Partie entière, partie décimale
  - Intervalle de valeur
    - Mantisse, exposant
- Les types réels sont toujours bornés par la représentation machine
- Exemple

```
def_réel    type ident is delta expression range intervalle;
def_réel    type ident is digits expression [range intervalle];
def_réel    type ident is delta expression digits expression [range intervalle];
```

```
type Reel is digits 8;
type Probabilite is digits 5 range 0.0 .. 1.0;
type Volt is delta 0.125 range 0.0 .. 255.0;
```

Note complémentaire sur les types

- Manipulation des caractéristiques d'un type
  - Ada s'appuie sur la notion « d'attributs prédéfinis »

```
Nom_Type'First
Nom_Type'Last
Nom_Type'Image(Variable_Discrète)
Nom_Type'Value(chaine_de_caracteres)
--etc...
```

- Forme d'introspection
  - Permet de paramétrer le comportement de programmes en fonction des types

Déclaration de variables

Grammaire:

def_var identificateur {, identificateur }: identificateur [:= expression];

Exemples:

```
Valeur_De_N  : Entiers_Naturels;
Valeur_De_P  : Entiers_Naturels := 100;
Valeur_De_2P : Entiers_Naturels := 2 * Valeur_De_P;
F1, F2       : Nombres_Flottants := 0.0;
```

Remarque: une variable non initialisée a une valeur aléatoire