Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Approche à base de composants et concurrence

Filière: MR-GL2

Chapitre 4: La concurrence en Ada
Partie 3/5: Le typage en Ada

Réalisé par:
Dr. Sakka Rouis Taoufik

I. Introduction

Ada est un langage de programmation fortement typé

→ Ada est plus sévère sur ce point que Pascal.
→ Ada interdit le mélange de type à la compilation
→En Ada, toute variable, constante ou fonction est dotée d'un type précis.

Le type permet de définir :
- l'ensemble des valeurs que peut prendre la variable
- l'ensemble des opérations qu'on peut appliquer sur la variable.

I. Introduction

Outre que les types de base, ADA permet de définir :

- des types énumératifs
- des sous-types (ou restriction de type) : sous ensemble des valeurs d'un type
- des types dérivés : analogues aux sous-types mais incompatibles avec leur type parent
- des types numériques modulaires
- des types composites ou structurés : tableaux, record, …
- des types synonymes
- des types à discriminant : la détermination du type dépend d'un paramètre
- des types privés (private, limited private)
- des types de tâches (Task cf. Chapitre 5)
- des types de pointeurs d'allocation dynamique (access)

II. Types et sous-types

•Les déclarations de type définissent généralement des types nommés. Dans certains cas (déclaration interne à une autre déclaration) on peut avoir des types anonymes.

Exemples:

```
type Color is (Vert, Bleu, Rouge, Jaune, Violet, Blanc, Noir);
type Table is array (1 .. 12) of Integer;
```

•Un sous-type est obtenu par restriction du domaine de valeurs d'un autre type appel type de base.

Syntaxe:

```
subtype t1 is t_base2  constrainte ;
 --la contrainte dépend de la nature du type de base
```

Exemple:

```
subtype index is integer range 1..10 ;
```

III. Dérivation de type

•Un type dérivé est un nouveau type créé à partir d'un type existant (type parent).
•Un type dérivé est incompatible avec son type parent, contrairement à un sous-type.
•Un type dérivé peut servir à éviter des confusions entre entités prenant leurs valeurs dans le même ensemble mais de natures différentes.

Syntaxe:

type t1 **is** [abstract] [limited] new t_base2;

•**abstrait (abstract)**➔ on peut pas avoir des objets de ce type. Un type abstraite ne sert que comme type parent d'autres types.

•**limité (limited**) ➔ l'utilisation de l'affectation et des opérateurs d'égalité et d'inégalité sur des objets de ce type **est interdite**.

Intérêt du type limité : empêcher des affectations globales de types composés qui ne copient pas les valeurs des champs ou redéfinir l'égalité/inégalité entre objets composés.

III. Dérivation de type

Exemple : on veut gérer dans un programme des sommes en euros et en francs, sommes représentées dans les deux cas par des réels

```
type Euro is new Float;
type Franc is new Float;

sEuro : Euro := 20.0;
sFranc : Franc := 50.0;
s : Float := 10.0;

sEuro := sEuro + s; -- interdit par le compilateur
sEuro := sEuro + sFranc; -- interdit par le compilateur
```

IV. Les tableaux

• Pour définir un tableau en Ada, on joue sur deux paramètres:
- ➢ Le type index (1 ou plusieurs dimensions)
- ➢ Le type de base

•Exemples de types tableaux contraints :

```
type Table is array (1 .. 10) of Integer;
type Jour is (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche);
type Cours is (Cours_Magistral, TD, TP, Seminaire);
type Agenda is array (Jour) of Cours;
type Matrice is array (1..10,1..20) of Float;
```

•Exemples de types tableaux non contraints :

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

•Exemples d'utilisation d'un tableau non contraint :

```
subtype Vecteur1 is Vecteur ( -10..10);
subtype Matrice2  is Matrice (1..2, 1..2);
subtype Vecteur3 is Vecteur1 (1..2); -- interdit par le compilateur
subtype Matrice1 is Matrice (1..2, range <>); -- interdit par le compilateur
v1 : Vecteur1;
v2 : Vecteur (1..3);
v:Vecteur; -- interdit par le compilateur
```


IV. Les Tableaux

•Les principaux attributs des tableaux:

T'Firs ➔donne l'indice du première élément du tableau T
T'Last ➔ donne l'indice du dernière élément du tableau T
T'Length ➔ donne la taille du tableau T
T'range ➔ donne l'intervalle des indices du tableaux
T(n) ➔ donne la valeur de la case d'indice n

Exemple:

```
for i in T'range loop
        T(i) := 0
end loop;
```

V. Les enregistrements

• Les enregistrements (struct en C) permet de regrouper des informations étroitement liés et éventuellement hétérogènes (n'ont pas de même type)
•En Ada en peut distinguer deux types d'enregistrements:
➔sans discriminant et avec discriminant (paramétré)

Exemple d'enregistrement sans discriminant

```
-- on suppose qu'une type Nom_Mois existe
type Date is record
        Jour : Integer range 1 .. 31;
        Mois : Nom_Mois;
        Année : Integer range 0 .. 4000 :=2000 ; -- 2000 est une valeur par défaut
end record;

--Accès aux champs d'un enregistrement

    d : Date;
    d.Jour := 22; d.Mois := Mars; d.Année := 2007; -- notation qualifié ou pointé

    d2 : Date := (22,Mars,  2007);
    d3 : Date := (Jour => 22, Mois => Mars, Année => 2007);
```


V. Les enregistrements

Exemple d'enregistrement avec discriminant

```
Type Tableau is array (posistive range <>) of integer ;

type Pile (n: Positive) is record   -- n est appelé discriminant ou paramètre
        P: Tableaux (1.. n) ;
        Sommet : Naturel ;
end record;

P1: Pile(200) ;
P2:Pile (80) ;
P3: Pile ; -- erreur de compilation
```

Rq. Ada permet de donner un discriminant par défaut

```
type Pile2 (n: Positive:=50) is record   -- n est appelé discriminant ou paramètre
        P: Tableaux (1.. n) ;
        Sommet : Naturel ;
end record;

P1: Pile2 (200) ;
P3: Pile2 ; --ok
```

VI. Les pointeurs

•Les pointeurs sont des accès aux objets ou sous-programmes
•Un pointeur se définit à l'aide du type d'objet (ou de sous-programme) vers lequel il pointe.
•Les pointeurs en ADA obéissent à certaines règles pour éviter les erreurs :
➢un pointeur ne peut pointer que des objets d'un même type
➢un pointeur a toujours une valeur qui est soit null soit définie par allocation (opérateur **new**)

Syntaxe:

type t1 is acces t2 ;

Exemple:

```
type P_entier is access Integer;
ae : P_entier;
ae = new Integer;
```

On peut aussi préciser la valeur de la variable pointée à l'allocation par l'utilisation d'une expression qualifiée :

```
ae = new Integer'(21) ;
```


VI. Les pointeurs

•L'accès aux variables pointées se fait par l'opérateur **all**

```
type T_Pointeur is access Integer ;
Ptr1, Ptr2, Ptr3 : T_Pointeur ;
var1: integer;

Ptr1 := new integer ;
Ptr1 := new integer'(124) ;
Ptr2 := new integer ;
Ptr2.all := 421 ;
var1:= ptr2.all;
```

Utilisation

```
if Ptr1 = Ptr2   then …
if Ptr1 /= Ptr2  then ...

if Ptr1 < Ptr2   then …   -- erreur
```

VI. Les pointeurs

Ptr2.**all := 5 ; *--Ptr2 pointe sur "5"***
Ptr1 := Ptr2 ; *-- On modifie l'adresse contenue dans Ptr1 ! ! !*
--Ptr1 et Ptr2 pointent sur la même adresse
Ptr1.**all := 9 ; *--Ouais ! Ptr1 pointe sur "9" maintenant !***
--Sauf que du coup, Ptr2 aussi puisqu'ils
--pointent sur le même emplacement ! ! !
Ptr2.**all := 5 ; *--Ptr2 pointe sur "5" donc de meme pour ptr2***

Ch 4: La concurrence en Ada (Partie 3/5)

VI. Les pointeurs

Type element ; --définition incomplète, référence anticipé

Type pt_element **is access** element;

Type element **is record**
info: integer;
suivant: pt_element;
End record;

Tete: pt_element;

Tete**:=new** element;-- équivalent à l'allocation dynamique

Tete**.** info**:=** 12 ;-- initialiser le champ info à 12
Tete**.** suivant**:=**null;

VII. Exercices d'application

Exercice 1

Écrire un programme Ada qui lit deux nombres complexes C1 et C2 et qui affiche ensuite leur somme et leur produit.

NB. Utiliser une solution modulaire

On utilisera les formules de calcul suivantes :

$$(a + b\,i) + (c + d\,i) = (a + c) + (b + d)\,i$$

$$(a + b\,i) * (c + d\,i) = (a * c - b * d) + (a * d + b * c)\,i$$

Exercice 2

On suppose qu'un produit est caractérisé par son libellé, son prix d'achat et son prix de vente. Chaque produit est livré par un seul fournisseur. Un fournisseur est caractérisé par son code et son numéro de téléphone. On souhaite écrire un programme Ada qui permet de déminer et d'afficher la liste des produits hors stock (afficher juste le libellé et le code du fournisseur de chaque produit fini).