Université de Perpignan — Année 2003-2004
Licence 2e année Informatique — S. Graillat

Programmation impérative

TD n° 3 : Typage et instructions avec Ada

1 Types de données et déclaration

D'un point de vue pratique, un type de données (on dit plus simple un type) est une façon de structurer l'information manipulée dans un programme, d'indiquer au compilateur les valeurs admissibles pour certaines variables ainsi que les différentes opérations autorisées sur ces variables.

Vous allez voir qu'Ada est un langage fortemement typé. Cela offre deux principaux avantages :

- le code produit est "plus rapide" car les tests de cohérence ne sont exécutés qu'une fois à la compilation et non plus lors de chaque exécution.
- le programme est plus "sûr" car la plupart des erreurs de conception peuvent être détecter à la compilation.

1.1 Déclaration des variables et des constantes

La déclaration des variables est très simple. Il suffit de taper :

```
Nom_de_variable : Nom_du_type;
```

On peut aussi initialiser la variable en même temps en tapant :

```
Nom_de_variable : Nom_du_type := valeur;
```

Pour la déclaration des constantes, il s'agit de la même chose mais on rajoute le terme **constant**.

```
Nom_de_constante : constant Nom_du_type := valeur;
```

1.2 Déclaration des types

Toute déclaration de type doit commencer par le mot **type**.

```
type ENUM is (UN,DEUX); -- pour un type enuméré
type TAB is array(1..10) of INTEGER; -- pour un type tableau
type REC is
   record
      I : INTEGER;
```

```
      F : FLOAT;
   end record; -- pour un type article
type ACC is access REC; -- pour un type accès
```

Vous aurez noté la présence de deux tirets `--`. Il s'agit de commentaires dans le code source. Il commence par `--` et se termine à la fin de la ligne. Tout ce qu'il contient n'est pas lu par le compilateur.

1.3 Les principaux types prédéfinis

1.3.1 Le type INTEGER

Le premier type numérique est la représentation des entiers, INTEGER en Ada. Le type INTEGER correspond en principe à la taille d'un mot mémoire de la machine.

1.3.2 Le type FLOAT

Ada connait un type réel prédéfini, dit "flottant", qui dépend lui aussi de la machine. Le type principal est FLOAT.

1.3.3 Le type ARRAY

Il est possible de définir en Ada des tableaux. Les éléments d'une telle structure sont accessibles au moyen d'un ou plusieurs indices.

La définition d'un tableau comportera le type des indices et le type des éléments du tableau. Le type d'indices pourra être n'importe quel type discret (enuméré ou entier). Le type des éléments pourra être un type contraint (c'est-à-dire instanciable) quelconque. Par exemple :

```
type TAB is array(1..10) of INTEGER;
```

1.3.4 Les types enumérés

Les types énumérés sont déclarés en indiquant simplement les valeurs symboliques qu'ils peuvent prendre, par exemple :

```
type COULEUR is (BLEU, JAUNE, ROUGE);
type ETAT is (ARRET, MARCHE, HORS_SERVICE, EN_REPARATION);
type FEUX is (ROUGE, ORANGE, VERT);
```

Il résulte de la définition d'un type énuméré l'existence d'un ordre entre les éléments du type. Par exemple dans le type COULEUR on aura les relations suivantes : BLEU $<$ JAUNE $<$ ROUGE alors que dans le type FEUX on aura ROUGE $<$ ORANGE $<$ VERT.

En particulier le type BOOLEAN est prédéfini :

```
type BOOLEAN is (FALSE, TRUE);
```

Cette définition implique que FALSE est inférieur à TRUE. Les opérations prédéfinies sur ce type sont les opérateurs logiques `not`, `or`, `and`, `xor`. Les opérations prédéfinies sur tout type enuméré sont les suivantes : test d'égalité ($=$) et d'inégalité ($/=$), test d'ordre ($<$,$<=$,$>$,$>=$).

2 Les instructions conditionnelles et de boucle

2.1 Structure conditionnelle

```
if Condition then
   ...
elsif Autre_Condition then  -- sinon si
   ...
else -- sinon
   ...
end if;
```

2.2 Structure de cas

```
case Expression is
   when Valeur_1 => ...
   when Valeur_2 => ...
   when others => ...  -- branche eventuelle
end case;
```

2.3 Structure de boucle

```
loop
   ... -- suite d'instructions
end loop;
```

Il y a trois conditions d'arrêt possibles.

- le comptage :
  ```
  for INDICE in sous_type_enumeré loop
     ...  -- suite d'instructions
  end loop;
  ```
- Le "tant que" : la boucle tourne tant qu'une condition est vraie :
  ```
  while Condition loop
     ...  -- suite d'instructions
  end loop;
  ```
- La sortie "sauvage" :
  ```
  loop
     ...  -- suite d'instructions
     exit when Condition;
     ...  -- suite d'instructions
  end loop;
  ```

3 Exercices

1. Taper le progamme suivant.

```
procedure Test_Typage is

   a : INTEGER :=1;
   b,c,d : FLOAT;

begin
   b := a;
   c := 2;
   d := 2.0;
end Test_Typage;
```

Le compiler et regarder ce qui se passe. Expliquer les différentes erreurs lors de la compilation.

2. Écrire un programme ADA vous demandant d'entrer deux nombres réels et affichant le plus grand des deux.

3. Écrire un programme vous demandant d'entrer trois nombres réels et les affichant ensuite par ordre de grandeur croissante.

4. Écrire un programme résolvant une équation du second degré.

5. Écrire un programme affichant les K premiers multiples d'un nombre N.

6. Écrire un programme vous demandant de saisir des nombres strictement positifs et affichabnt au fur et à mesure le plus grand de ces nombres.

7. Écrire un programme calculant $N!$ et ce de manière itérative et récursive.

8. Écrire un programme calculant le PGCD de deux entiers a et b.

9. Écrire un programme affichant l'ensemble des diviseurs d'un entier N strictement positif.