Le modèle sémantique

Chapitre 5

Représentation Ada des valeurs

Un entier est représenté classiquement :

```
3       56       987
```

Un réel est représenté avec la notation pointée :

```
3.1415
```

Un caractère est représenté entre quotes pour le distinguer de tout autre identificateur :

```
'A'    '7'      '$'
```

Une chaîne de caractères est représentée entre guillemets pour la distinguer de tout autre identificateur ou mot clé :

```
"Hello World "
```

Un booléen est représenté par les symboles :

```
True, False
```

Expressions

Les valeurs constituent des ensembles.

L'environnement initial (qu'il est inutile d'importer explicitement) comporte tous les opérateurs sur les valeurs de ces ensembles de base.

Une expression est formée à partir des valeurs et opérateurs associées à cet ensemble.

Une expression crée ainsi une nouvelle valeur.

Exemple

```
(15+30)/2
"AU"&"SECOURS"
True and False
```

Programme

Un programme met en oeuvre un calcul sur un exécutant informatique.

Il manipule des objets informatiques représentant les différentes valeurs (autres que celles qui sont exprimées littéralement).

Ces objets peuvent être de nature différente :

- variable informatique
- constante
- fonction
- procédure
- type
- exception
- paquetage
- ...

Objets informatiques et identificateurs

Outre une valeur, les objets informatiques possèdent un type.
En Ada, ils possèdent un type et un seul.

Les objets Ada, pour être manipulés, doivent posséder un nom.

On parle d'identificateur pour signifier qu'un nom est construit en respectant une grammaire .

Exécution et contrôle

Un programme contient l'expression d'un calcul et les éléments qui permettent son exécution, les mécanismes de contrôle (boucles, séquence, conditionnelle, exceptions, appels de fonctions et de procédures)

Les mécanismes de contrôle permettent l'ordonnancement des instructions d'un programme.

Pour ordonnancer les instructions, il faut définir comment on continue un calcul..

On répond ainsi à la question : quelle instruction exécuter ensuite ?

Exécutant

Il est défini par :

- un modèle d'exécution implanté (mécanisme de gestion de la continuation)
- un mécanisme d'exécution des instructions

Formalisation d'une exécution

C'est une fonction de transformation :

`P*D->D`

Toute donnée est exécutable (évaluable)

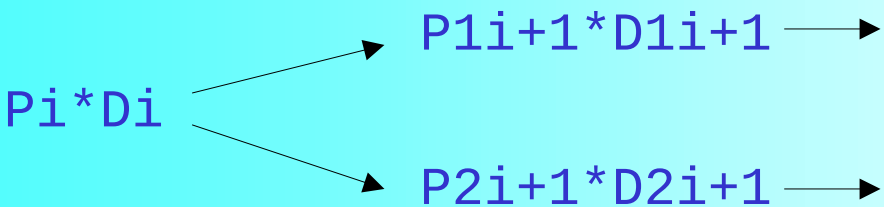
Un programme est une donnée particulière qui nécessite d'autres données pour s'exécuter.

Décomposition d'une exécution

En exécution séquentielle (ordre total)

P1*D1-> P2*D2->...->Pn*Dn->...

En exécution parallèle (ordre partiel restitué par la synchronisation des processus)



Etat d'un programme

L'état d'un programme correspond à la description d'une étape de son exécution. On parle d'état courant pour spécifier un état à un instant donné de l'exécution.

L'état d'un programme est constitué des objets informatiques manipulés par le programme à une étape donnée de l'exécution.

Plus formellement, l'état d'un programme est représenté par le couple formé de l'environnement (Env) et de la mémoire (Mem).

Environnement

L'environnement rassemble l'ensemble des identificateurs d'objets, utilisables au cours d'une exécution, associés à leur type et leur valeur.

C'est une liste de triplets (identificateur, type, valeur).

Identificateur	**Valeur**
variable	adresse
constante	valeur
exception	valeur d'exception
fonction/procédure	fermeture
paquetage	environnement
type	valeur de type

Mémoire

De manière simplifiée, la mémoire contient des valeurs référencées par des adresses. La mémoire est vue comme une liste de couples (adresse, valeur référencée).

Tout identificateur appartenant à l'environnement référence une valeur de la mémoire. Toute valeur de la mémoire non associée à un identificateur devient inaccessible.

Un programme ramasse-miettes ("garbage collector") récupère les zones mémoire allouées à des objets devenus inaccessibles.

Généralités sur les déclarations

Une déclaration d'objet permet :

- ✓ d'introduire son identificateur dans l'environnement.
- ✓ de préciser les conditions de son utilisation (son type)
- ✓ de définir une valeur initiale (facultativement)

Variables informatiques

Une variable informatique est un objet informatique dont la valeur peut changer au cours de l'exécution du programme.

A une adresse donnée correspond un ensemble de valeurs référencées possibles (notion de type) au cours de l'exécution d'un même programme .

Une variable peut être :

- anonyme (création dynamique)
- nommée (liée à un identificateur)

Déclaration d'une variable : syntaxe

```
<declaration_variable>::=
            <identificateur>:<type>:=<expression>;
```

Toute déclaration est réalisée dans un environnement Env donné.

Exemples

```
demain:Jour := mardi;
numero:Natural := 0;
meteo:Temps := pluvieux;
X:Integer;
```

Déclaration d'une variable : sémantique (1/3)

Soit l'état courant

Etat0 = (Env0, Mem0)

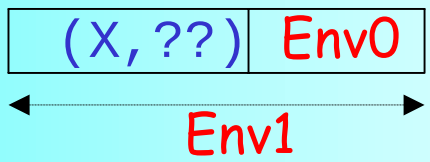
et la déclaration

```
X:Natural:=12;
```

✓ Extension de l'environnement Env0 avec le couple :

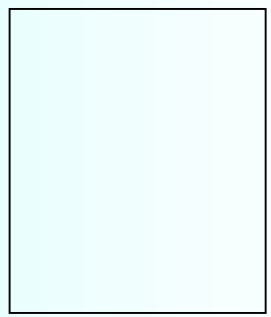
(X,??)

d'où



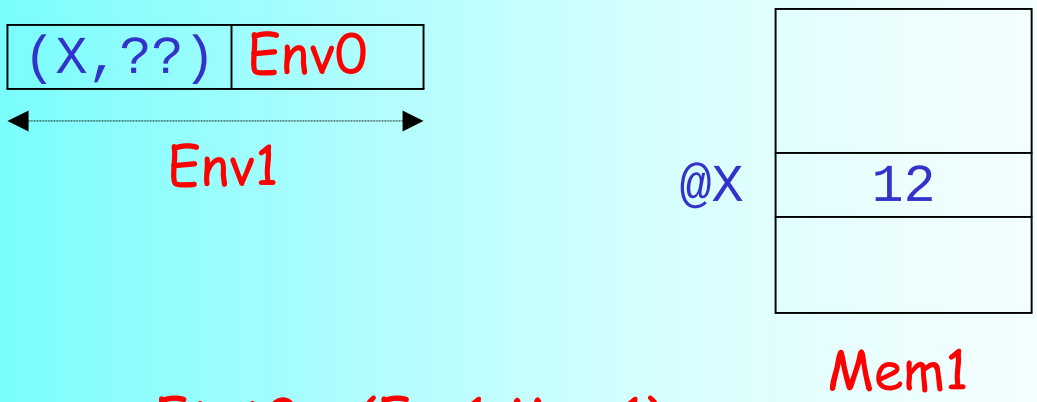
et

Etat1 = (Env1,Mem0)



Déclaration d'une variable : sémantique (2/3)

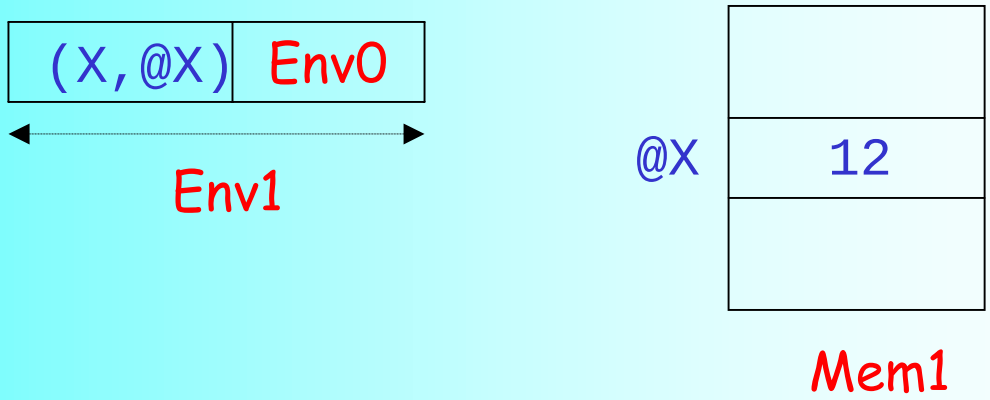
- ✓ Détermination du type et des contraintes
- ✓ Evaluation de l'expression dans ce nouvel état (Etat1) : l'expression vaut 12
- ✓ Adjonction du couple (@X,12) dans Mem0 qui devient Mem1



Etat2 = (Env1,Mem1)

Déclaration d'une variable : sémantique (3/3)

- ✓ Modification de Env1



Après évaluation de la déclaration,

Etat3 = (Env1,Mem1)

Déclaration d'une variable : exemple 1 (1/2)

Etant donné l'état :

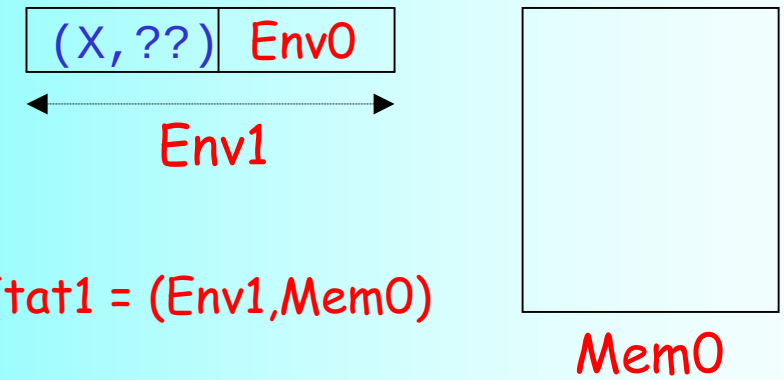
Etat0=(Env0,Mem0)

et la déclaration

```
X:Natural:=X+1;
```

Déclaration d'une variable : exemple 1 (2/2)

Extension de l'environnement Env0 avec le couple : (X, ??)



- ✓ Détermination du type et des contraintes
- ✓ Evaluation de l'expression X+1 dans Etat1

L'identificateur X n'étant lié à aucune valeur, l'évaluation est impossible. Une telle initialisation est donc interdite.

Déclaration d'une variable : exemple 2 (1/4)

Etant donné l'état :

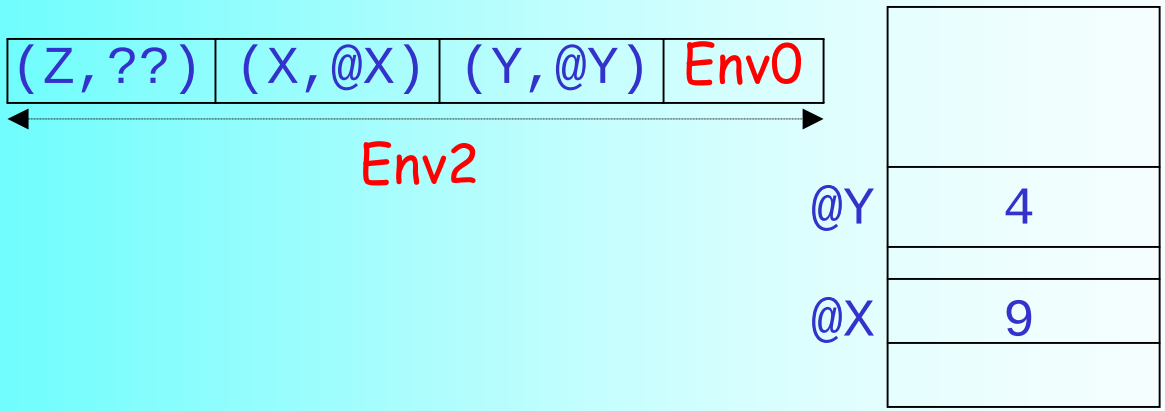
Etat1 = (Env1,Mem1)

et la déclaration :

```
Z:Natural:=Y-X;
```

Déclaration d'une variable : exemple 2 (2/4)

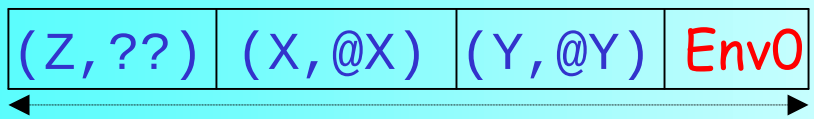
Extension de l'environnement Env1 avec le couple :



Etat2 = (Env2,Mem1)

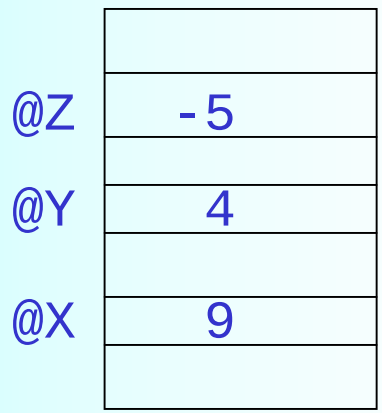
Déclaration d'une variable : exemple 2 (3/4)

- ✓ Détermination du type et des contraintes
- ✓ Evaluation de l'expression dans ce nouvel état (Etat2) : l'expression `Y-X` vaut `-5`
- ✓ Adjonction du couple `(@Z, -5)` dans Mem1 qui devient Mem2. Etat3 est le nouvel état



Env2

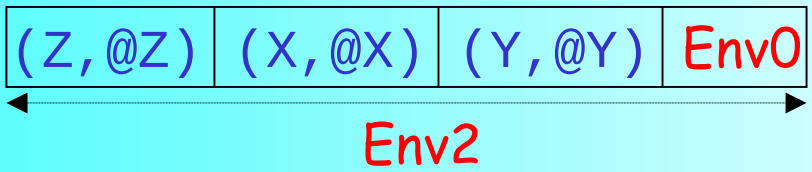
Etat3 = (Env2,Mem2)

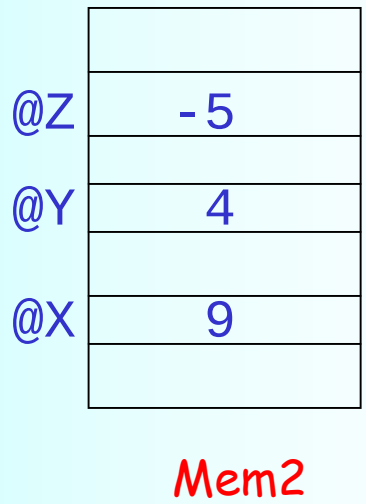


Mem2

Déclaration d'une variable : exemple 2 (4/4)

Modification de Env2





Après évaluation de la déclaration,

Etat4 = (Env2,Mem2)

Déclaration de constante

Un constante est un objet informatique dont la valeur ne change pas durant l'exécution d'un programme.

La déclaration d'une constante permet de nommer un objet et de l'ajouter à l'environnement courant.

Syntaxe

```
<déclaration_constante>::=
  <ident>:constant<type>:=<expression_constante>;
```

Exemples

```
PI:constant Float:=3.14159;
CLOVIS:constant Natural:=496;
e:constant Float:=2.712;
DEUX_PI:constant Float:=PI*2.0;

type ID is record
   nom:String(1..15);
   SS:String(1..12);
end record;
leo:constant ID:=( "Pelletier       " , "030862980567" );
```

Déclaration d'une constante : sémantique

- ✓ Ajout de l'identificateur, associé à une valeur indéfinie, à l'environnement
- ✓ Evaluation de l'expression constante
- ✓ Remplacement de la valeur indéfinie par la valeur évaluée dans l'environnement courant

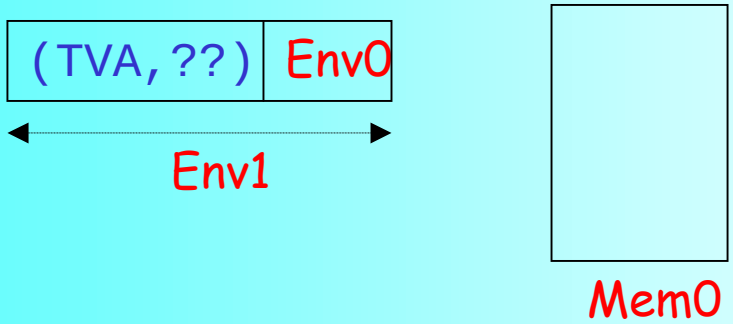
Exemple (1/2)

Soit, dans l'état (Env0,Mem0) la déclaration :

```
TVA:constant Float:=20.6;
```

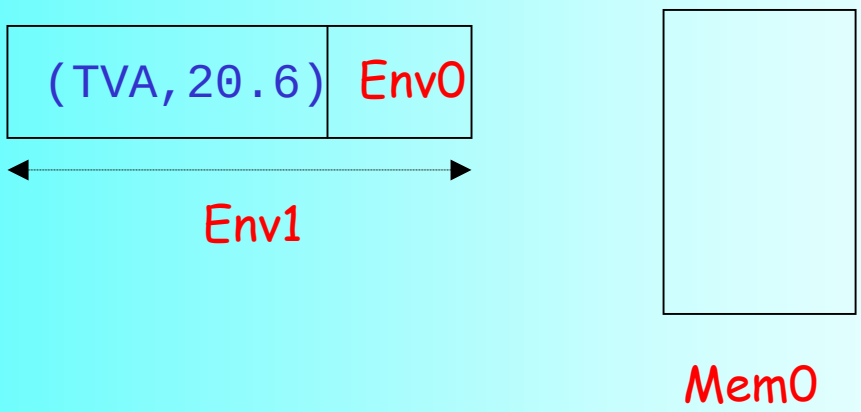
Elaboration en 3 étapes :

- Ajout de l'identificateur TVA associé à une valeur indéfinie dans Env0. On obtient le nouvel environnement Env1 :



Exemple (2/2)

- Evaluation de l'expression constante ---> `20.6`
- Remplacement de la valeur indéfinie (`??`) par la valeur (évaluée dans Env1)

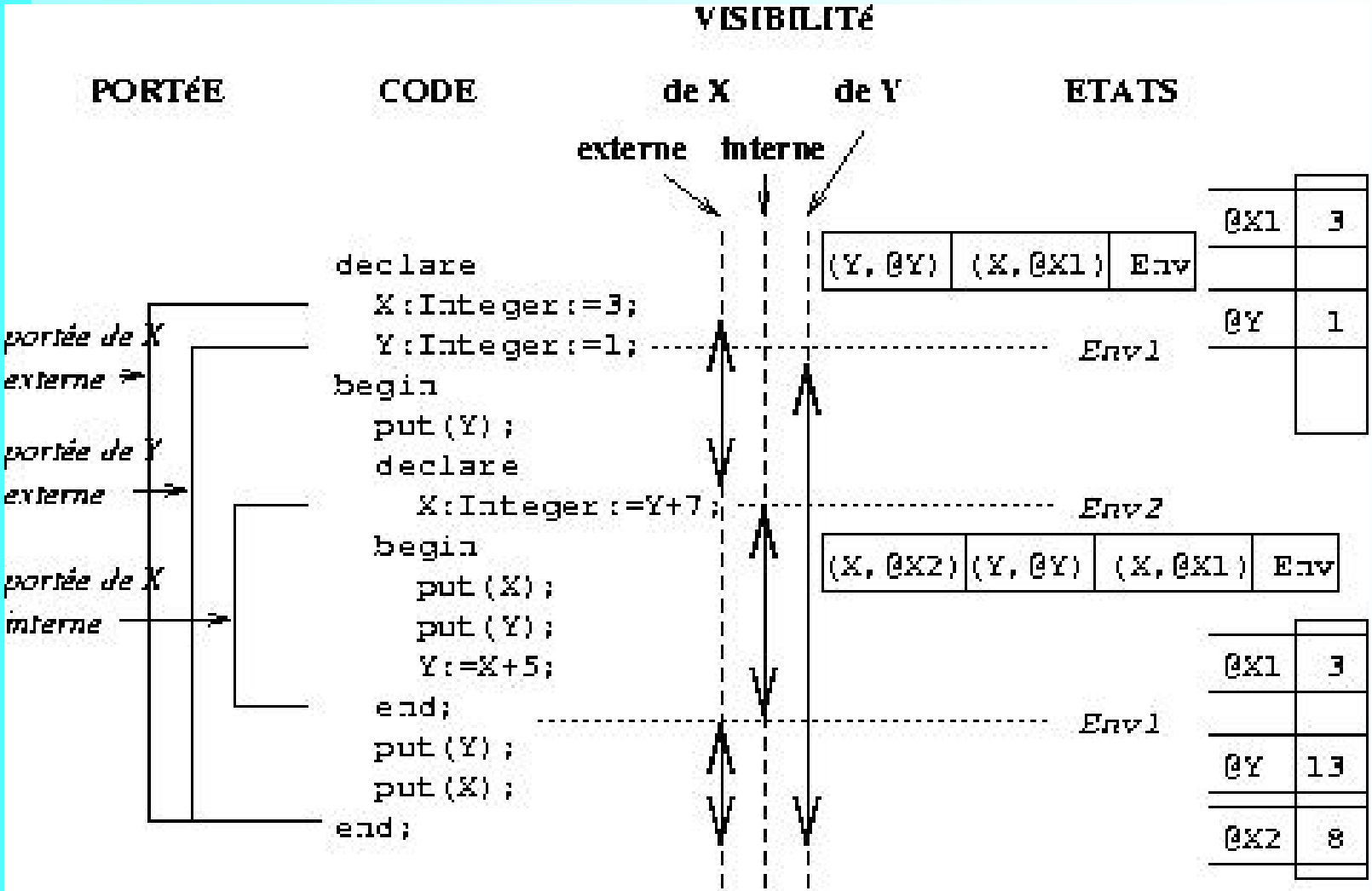


Portée et visibilité

La portée d'un objet informatique est la portion de texte à l'intérieur duquel il a une existence.

Sa visibilité ne couvre pas obligatoirement cette même portion. Il se peut qu'un objet de même identificateur le masque temporairement.

Exemple 1


VISIBILITÉ
PORTÉE
CODE
de X
de Y
ETATS
externe
interne
declare
X:Integer:=3;
Y:Integer:=1;
begin
put(Y);
declare
X:Integer:=Y+7;
begin
put(X);
put(Y);
Y:=X+5;
end;
put(Y);
put(X);
end;
portée de X externe
portée de Y externe
portée de X interne
Env1
Env2
Env1
(Y,@Y)
(X,@X1)
Env
(X,@X2)
(Y,@Y)
(X,@X1)
Env
@X1
3
@Y
1
@X1
3
@Y
13
@X2
8

Exemple 2 (1/3)

Dans l'exemple qui suit, nous avons remplacé `7+y` par `7+x`.

```
with Ada.Text_io;
use Ada.Text_io;
procedure exemple2 is
   x:Integer:=3;y:Integer:=1;
begin
   put( "y=" );put(Integer'image(y));
   declare
      x:Integer:=7+x;
   begin
      put( "x=" );put(Integer'image(x));
      put( "y=" );put(Integer'image(y));
   end;
   put( "x=" );put(Integer'image(x));
end exemple2;
```

Exemple 2 (2/3)

Dans ce cas une erreur est signalée par le compilateur :

```
object x cannot be used before end of its
   declaration
```

Pour comprendre cette erreur, il suffit de dérouler le mécanisme de la déclaration d'une variable appliquée à l'instruction :

```
x:Integer:=7+x;
```

Exemple 2 (3/3)

Evaluation de cette nouvelle déclaration de variable :

- ajout de `(x,??)` dans l'environnement courant qui devient le plus récent
- détermination du type (`Integer`)
- évaluation de l'expression `x+y`. L'évaluation de `x` s'avère impossible puisque la valeur qui lui est associée est indéfinie.

Sémantique des instructions

Affectation (1/3)

Syntaxe

<affectation>::= <expression_gauche>:=<expression droite>;

où

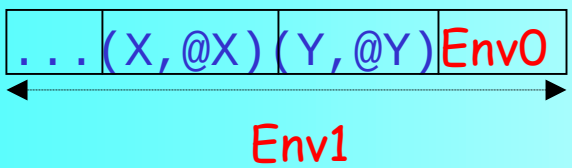
<expression_gauche> dénote une adresse

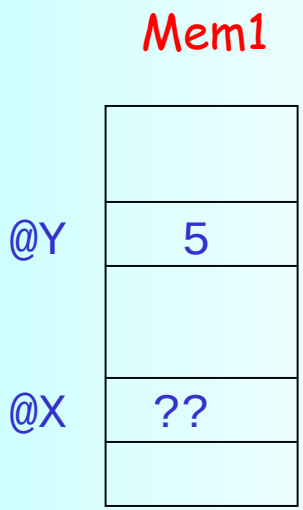
<expression droite> dénote une valeur

Sémantique : exemple

Evaluation de l'affectation : X:=Y+2;

dans (Env1,Mem1)





Affectation (2/3)

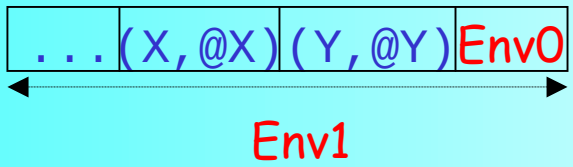
- L'évaluation de `<expression_gauche>` (c'est à dire `X`) dans cet état retourne l'adresse de `X` : `@X`

- L'évaluation de `<expression_droite>` (c'est à dire `Y+2`) dans cet état nécessite d'abord l'évaluation des 3 symboles `Y, +, 2` :
  - La valeur de `Y` est : `@Y`, la valeur de la variable est donc `5`
  - `2` est une constante littérale, elle s'évalue en elle-même: `2`
  - La valeur de `+` est sa fermeture, l'environnement d'exécution peut donc être constitué et son code binaire exécuté
  - Le résultat de l'évaluation de `Y+2` est : `7`

Affectation (3/3)

- La valeur obtenue est enregistrée à l'adresse de X

D'où le nouvel état : (Env1,Mem2)



Remarque

L'ordre d'évaluation des expressions gauche et droite est indéterminé.

Boucle for : syntaxe

```
<boucle_for>::=
for <ident_de_compteur> in [reverse] <intervalle> loop
   <suite _instructions>
end loop;
<intervalle>::=<ident_de_type>|<début> .. <fin>
```

où

`<début>` et `<fin>` sont des valeurs d'un type discret.

reverse est facultatif. Il indique un parcours inverse (dans l'ordre décroissant) de l'intervalle `<début> .. <fin>`

Boucle for : sémantique

Pour chaque valeur `val` de l'intervalle discret parcouru dans l'ordre croissant (**`in`**) ou décroissant (**`in reverse`**) :

1. adjonction de la liaison (`ident_de_compteur,val`) dans l'environnement courant
2. exécution de la suite d'instructions dans le nouvel état
3. suppression de la liaison (`ident_de_compteur,val`) dans l'environnement courant

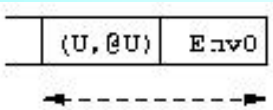
Remarques :

- L'identificateur de compteur est une constante pendant l'exécution de la suite d'instructions
- L'identificateur de compteur est local à la boucle. Il est détruit après l'exécution de la boucle.

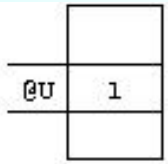
Exemple

```
-- dans l'état (Env1,Mem1)
for I in 1..3 loop
  U:=I*U;
end loop;
```

état initial



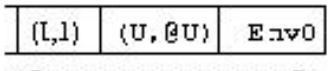
Env1



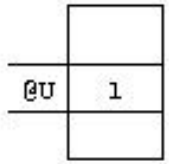
Mem1

```
for i in 1..3 loop
  U:=I*U;
end loop;
```

-- état avant le second
-- passage dans la boucle



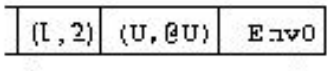
Env2



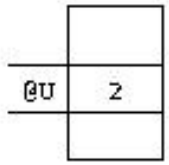
Mem2

```
for i in 1..3 loop
  U:=I*U;
end loop;
```

-- état avant le troisième
-- passage dans la boucle



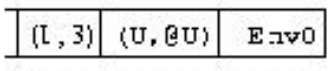
Env3



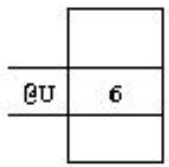
Mem3

```
for i in 1..3 loop
  U:=I*U;
end loop;
```

-- état avant la sortie
-- de boucle

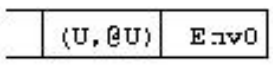


Env4

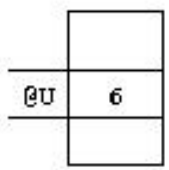


Mem4

-- état après la sortie de la
-- boucle
-- retour à l'environnement Env1



Env1



Mem5

Boucle tant que : sémantique (1/4)

La condition d'arrêt est de type booléen

La condition d'arrêt est évaluée avant chaque itération

Exemple

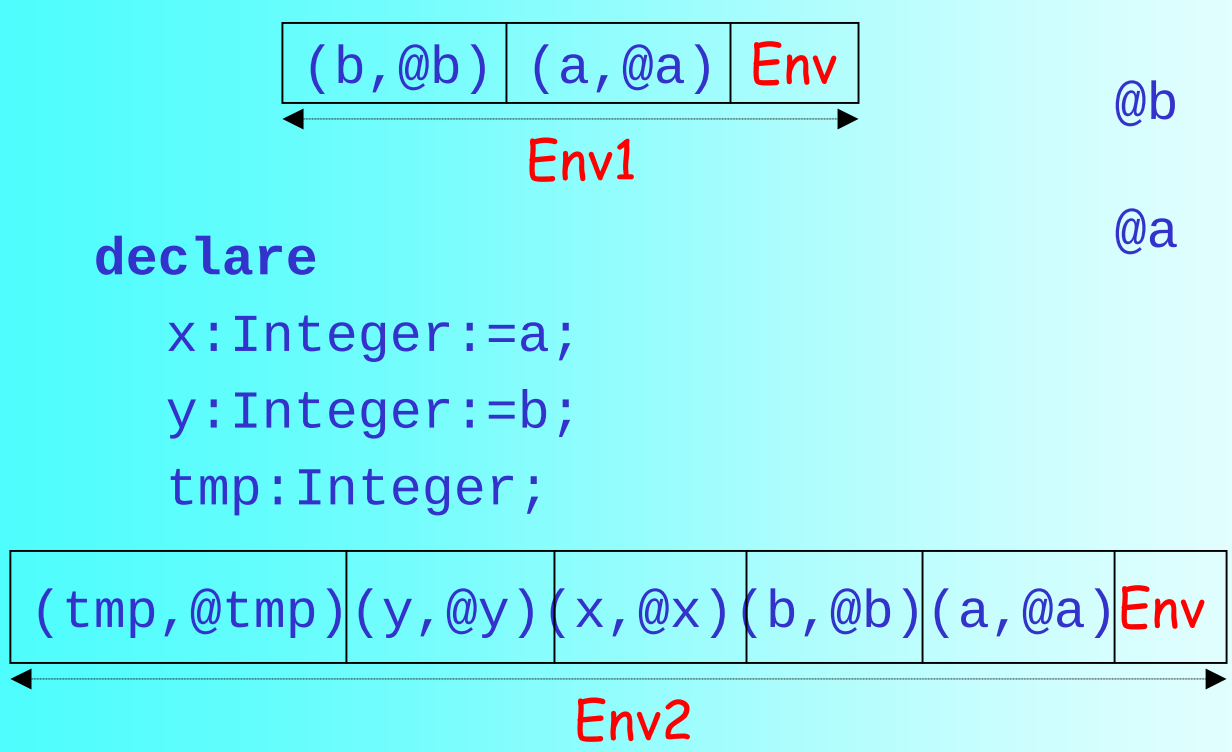
Calcul du PGCD de deux entiers naturels a et b par divisions successives

La boucle va permettre de construire une suite de valeurs pour les variables x et y respectivement initialisées à a et b.

chaque terme courant (x_i,y_i) est déterminée par le terme précédent : $(y_{i-1},x_{i-1}$ modulo $y_{i-1})$

Le calcul de la suite s'arrête lorsque x_k modulo $y_k=0$

Boucle tant que : sémantique (2/4)



Boucle tant que : sémantique (3/4)

```
begin
   while not(x mod y=0) loop
        tmp:=x;
        x:=y;
        y:=tmp mod y;
   end loop;
   put("PGCD(");put(a,2);put(",");put(b,2);put(")");
   put("=");put(y,2);new-line;
end;
```

Boucle tant que : sémantique (4/4)

L'exécution d'un pas d'itération ne modifie que la mémoire.

Au terme du premier passage,

```
x=28
y=7
```

la condition d'arrêt : `x mod y = 0` est donc satisfaite et le résultat est `y=7`

Résultat

```
a=399
b=28
PGCD(399,28)=7
```

Importation de module (1/2)

Tout programme s'exécute dans un environnement constitué d'objets informatiques. Par exemple, il est souvent utile de disposer des opérateurs arithmétiques `+, -, *, /, ...`, de commandes de lecture et d'écriture.

Tout programme bénéficie d'un environnement initial standard (les opérateurs arithmétiques en font partie).

Tout programme commence par la définition d'un environnement de travail de base. Celui-ci est construit par importation d'unités à partir de bibliothèques Ada.

Importation de module (2/2)

Syntaxe

```
<importation_module>::=with <nom_unite>;
```

Exemple,

```
with Ada.Text_io;
```

Le couple `(Ada.Text_io,val_Ada.Text_io)` appartient maintenant à l'environnement de l'unité importatrice.

L'environnement d'un module (unité) est constitué de 2 parties

Environnement de déclaration	Environnement d'importation

Environnement de déclaration et d'importation

Environnement de déclaration

C'est la liste des objets déclarés dans le module. Un module est lui-même un objet.

Environnement d'importation

Il est constitué de la liste des objets déclarés dans le module importé (appartenant donc à l'environnement de déclaration du module importé).

Remarque

L'environnement de déclaration a priorité sur l'environnement d'importation

Exemple 1

Soit l'environnement du module `M` :

environnement de déclaration	environnement d'importation
`(A,val_A1)(B,val_B1)(Ada.Text_io,val_T)`	`(B,val_B2)(A,val_A2)`

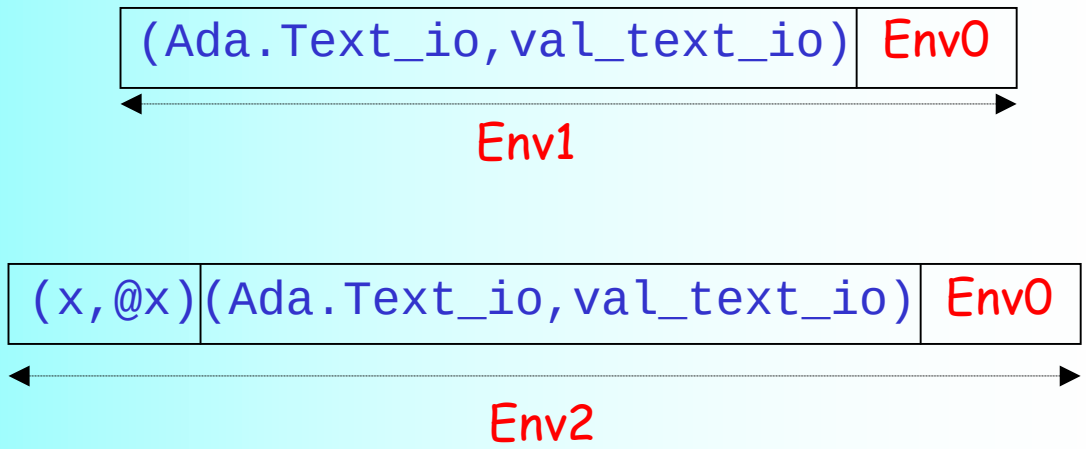
Il existe deux objets d'identificateur `A` et deux objets d'identificateur `B` dans l'environnement du module `M`, les valeurs associées à `A` et `B` dans le module `M` seront donc respectivement `val_A1` et `val_B1`, puisque l'environnement de déclaration a priorité sur l'environnement d'importation.

Exemple 2

```
------ évolution de l'environnement de déclaration -----
with Ada.Text_io;

procedure debut is
   x:Character:='$';
begin


   Ada.Text_io.put( "Hello World" );
   Ada.Text_io.new_line;
   Ada.Text_io.put( "-----------" );
   Ada.Text_io.new_line;
   Ada.Text_io.put_line(X);
end debut;
```



Exemple 3

```
------------- environnement de déclaration
with Ada.Text_io;
------------ environnement d'importation
use Ada.Text_io;
procedure debut is
  x:Character:='$';
---------- complète l'environnement de déclaration
begin
  Ada.Text_io.put( "Hello World" );
  Ada.Text_io.new_line;
  Ada.Text_io.put( "-----------" );
  Ada.Text_io.new_line;
  Ada.Text_io.put_line(X);
end debut;
```