Les procédures

Chapitre 7

Intérêt

- Créer une instruction nouvelle qui deviendra une primitive pour le programmeur

- Structurer le texte source du programme et améliorer sa lisibilité

- Factoriser l'écriture lorsque la suite d'actions nommée par la procédure intervient plusieurs fois

Différence entre fonction et procédure (1/2)

Une fonction peut être vue comme un opérateur produisant une valeur.

Une fonction n'a pas pour rôle de modifier l'état courant du programme en exécution.

Une bonne programmation interdit aux fonctions de réaliser des effets de bords.

On appelle effet de bord toute modification de la mémoire (affectation d'une variable, opération de lecture en mémoire) ou toute modification d'un support externe (disque, écran, disquette, etc).

Une fonction n'est pas une instruction, elle n'est donc pas en mesure de modifier l'état du programme. Une fonction réalise une simple opération dont le résultat peut être, par la suite, utilisé par une instruction.

Différence entre fonction et procédure (2/2)

Une procédure est une instruction composée qui peut prendre des paramètres et dont le rôle est de modifier l'état courant.
Les procédures ne retournent pas de résultat.

Les 3 aspects d'une procédure :

- Déclaration
- Déclaration du corps
- Appel

Déclaration de procédure

Le but de la déclaration d'une procédure est d'introduire dans l'environnement courant :

- son identificateur
- son type
- son mode de communication

Déclaration de procédure : syntaxe

```
<déclaration_proc>::=<en-tete_proc>;
<en-tete_proc>::=procedure <ident_proc> <liste_paramètres>
<liste_paramètres>::=vide|(<déclar_paramètres>)
<déclar_paramètres>::=
     <ident_param>:<mode> <type_param>{;<déclar_paramètres>}
```

où

```
<ident_param>::= identificateurs des paramètres formels
<mode> : := vide|in|out|in out
<type> ::= type des paramètres formels
```

Exemple 1

```
procedure permuter(a:in out Character;b:in out Integer);
```

Remarques

- La déclaration d'une procédure n'est pas obligatoire, la déclaration de son corps peut en tenir lieu.
- Une valeur par défaut peut être associée à chaque paramètre formel

Exemple 2

```
procedure accumuler
                (accu:in out Integer:=0;
                 x:in Tableau;
                 n:in Positive);
```

Type d'une procédure

Une procédure ne renvoie pas de valeur.
Son type est de la forme :

`t1*t2*...*tn->vide`

Le type de la procédure `permuter` est :

`Character*Integer->vide`

Le type de la procédure `accumuler` est :

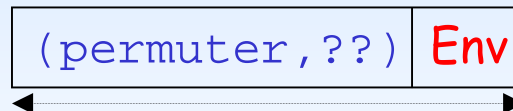
`Integer*Tableau*Positive->vide`

Déclaration de procédure : sémantique

```
procedure permuter(a:in out Character;b:in out Integer);
```

Mécanisme d'évaluation de cette déclaration de procédure dans l'environnement Env :

- Création de la liaison : `(permuter,??)`
- Extension de Env avec cette liaison : on obtient le nouvel environnement Env1



Env1

- Détermination du type des paramètres et du type du résultat. Le type de la fonction est :

`Character*Integer-->vide`

Déclaration du corps de la procédure **permuter**

```
procedure permuter(a:in out Character; b:in out Integer) is
   z:Integer;
begin
   z:=Character'pos(a);
   a:=Character'val(b);
   b:=z;
end permuter;
```

Le code exécutable de cette procédure associé au contexte de sa déclaration tient lieu de valeur liée à l'identificateur `permuter`.

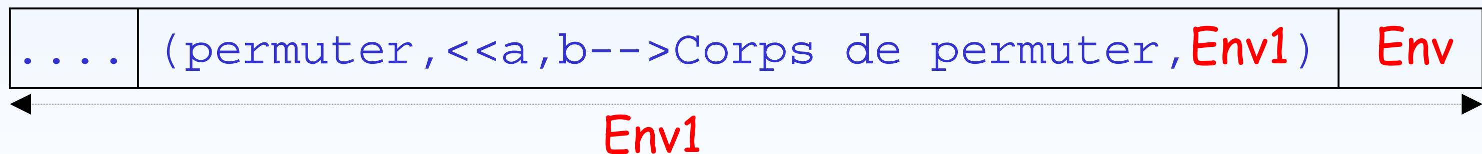
Déclaration du corps d'une procédure : sémantique

Mécanisme d'évaluation de cette déclaration dans Env1:

- Détermination de la fermeture `F` de `permuter`, dans un environnement Env1, à partir du corps de la procédure

  `F= <<a,b-->Corps de permuter,Env1>>`

- Modification de la liaison de l'environnement Env1



Appel de procédure

L'application d'une procédure à des paramètres effectifs constitue l'appel de cette procédure.

Syntaxe

```
<appel_procédure>::=
              <id_proc>(<liste_params_effectifs>)|<id_proc>;
<liste_params_effectifs>::=<param>{,<param>}
<param>::=<valeur>|<id_param>
```

Le type des paramètres effectifs doit être le même que celui des paramètres formels correspondants.

Si le paramètre formel est déclaré en mode **`in`**, toute expression peut être utilisée en tant que paramètre effectif correspondant.

Si le paramètre formel est déclaré en mode **`out`** ou **`in out`**, le paramètre effectif doit être une variable.

Association paramètre effectif/paramètre formel

- Liaison d'après l'ordre d'écriture

```
ident_proc(a1,a2,,an);
```

- Liaison en nommant le paramètre formel

```
ident_proc(pi=>ai,pj=>aj,,p2=>a2);
```

- Panachage

```
ident_proc(a1,a2,...,pj=>aj,,pi=>ai);
```

Passage de paramètre en mode in (1/3)

Soit le bloc,

```
declare
 y,z:Integer;
 procedure doubler(x:in Integer;t:out Integer) is
 begin
   t:=2*x;
 end doubler;
begin
 y:=3;
 doubler(y,z);
 put( "z=" );put(z,width=>3);
end;
```

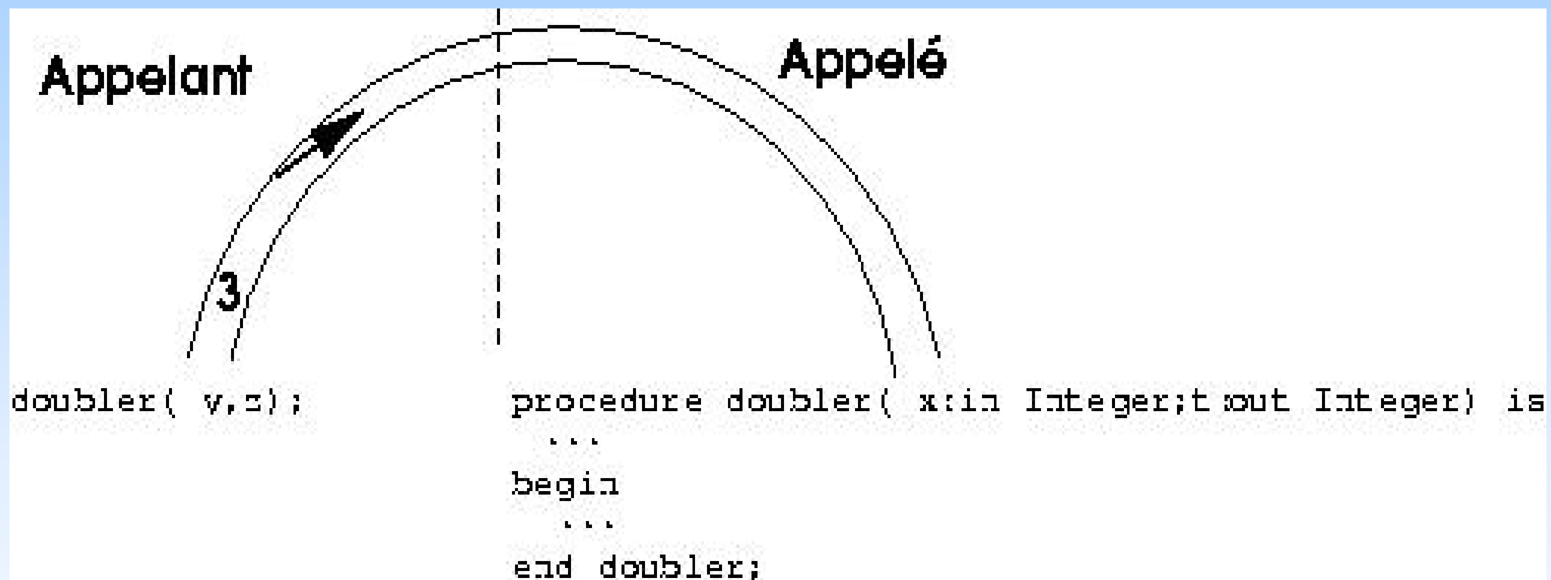
Passage de paramètre en mode `in` (2/3)

Le contrôle de type a été effectué. Les paramètres formels et effectifs correspondants ont donc le même type.

Transmission du paramètre passé en mode **`in`** :

- Etablissement d'un canal de communication entre le paramètre effectif et le paramètre formel
- Et d'un sens de la communication
- Copie de la valeur du paramètre effectif `y` et affectation de cette valeur au paramètre formel correspondant `x`
- Le paramètre formel `x` est une constante pendant toute l'exécution de la procédure

Passage de paramètre en mode in (3/3)



- Exécution de l'appelé
- Retour vers l'appelant :

  Le contrôle est passé à l'appelant derrière l'appel.

  Aucune valeur n'est transmise

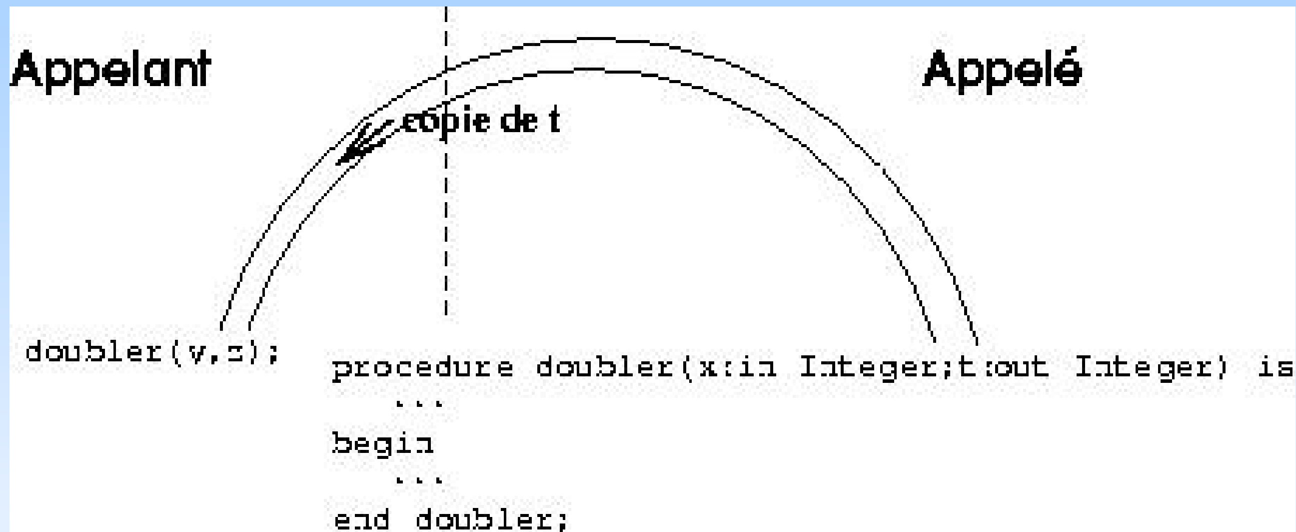
Passage de paramètre en mode `out` (1/2)

Le contrôle de type a été effectué. Les paramètres formels et effectifs correspondants ont donc le même type.

Transmission du paramètre passé en mode **`out`** :

- Etablissement d'un canal de communication entre le paramètre effectif et le paramètre formel
- Et d'un sens de la communication
- Le paramètre formel `t` est une variable dont la valeur initiale est indéfinie

Passage de paramètre en mode `out` (2/2)



- Exécution de l'appelé
- Retour vers l'appelant
- Copie de la valeur du paramètre formel `t` et affectation de cette valeur au paramètre effectif `z` (qui doit être une variable).
- Le contrôle est passé à l'appelant derrière l'appel.

Passage de paramètre en mode `in out` (1/3)

Soit le bloc,

```
declare
 x:Integer:=56; y:Integer:=57;
 procedure echange(u,v:in out Integer) is
   z:Integer;
 begin
   z:=u;u:=v;v:=z;
 end echange;
begin
 echange(x,y);
end;
```

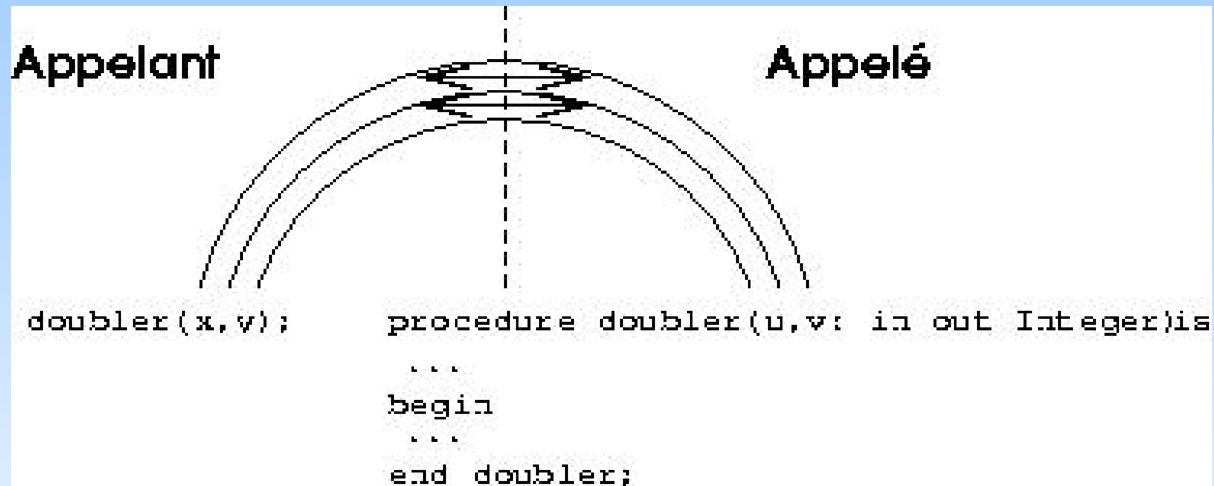
Passage de paramètre en mode `in out` (2/3)

Le contrôle de type a été effectué. Les paramètres formels et effectifs correspondants ont donc le même type.

Transmission du paramètre passé en mode **`in out`** :

- Etablissement d'un canal de communication entre le paramètre effectif et le paramètre formel
- Et d'un sens de la communication
- Les paramètres formels `u` et `v` sont des variables dont les valeurs initiales sont les copies des valeurs des paramètres effectifs `x` et `y`

Passage de paramètre en mode `in out` (3/3)



- Exécution de l'appelé
- Retour vers l'appelant
- Copie des valeurs des paramètres formels `u` et `v` et affectation de ces valeurs aux paramètres effectifs `x` et `y` (qui doivent être des variables).
- Le contrôle est passé à l'appelant derrière l'appel.

Passage de paramètres : par référence

Les passages en mode `in`, `out` et `in out` imposent une recopie de valeur.

Cette recopie peut être pénalisante lorsque les valeurs sont de taille importante (grande matrice).

Les compilateurs Ada ont la possibilité d'utiliser un autre mode de transmission de paramètres : le passage par référence pour pallier cet inconvénient.

Ce choix est transparent à l'utilisateur.

Exemple (1/2)

Soit le bloc,

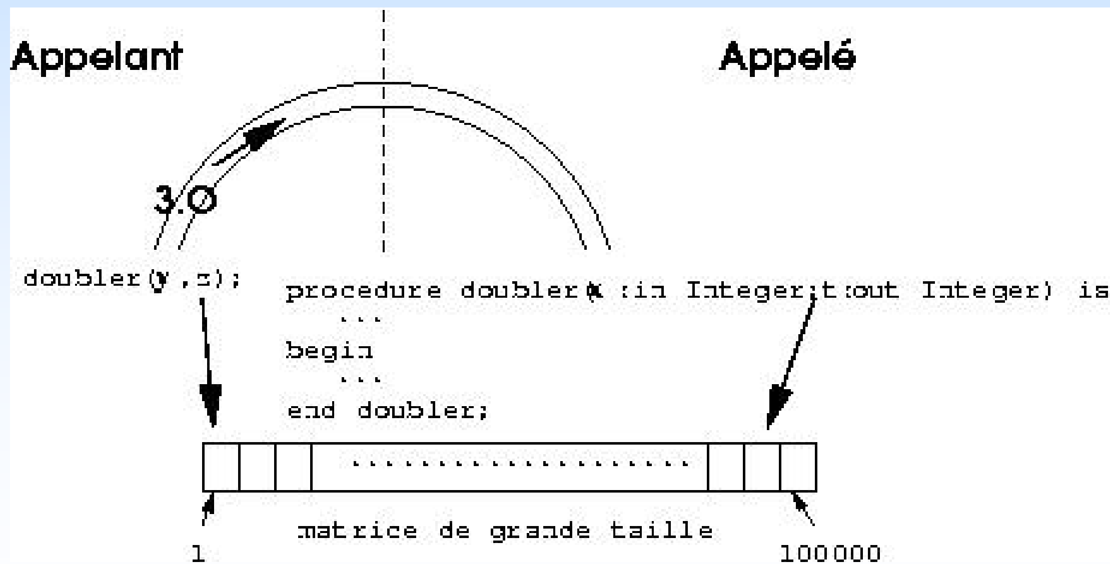
```
declare
 type Vecteur is array(Positive range<>) of Float;
 y:Float;z:Vecteur(1..100000):=(1.12,3.21,6.71,....,2.45);
 procedure doubler(x:in Float;t:in out Vecteur) is
 begin
   for i in t'range loop
     t(i):=t(i)*x;
   end loop;
 end doubler;
begin
 y:=3.0;
 doubler(y,z);
 put( "z=" );put(z);
end;
```

Exemple (2/2)

Après transmission des paramètres, le vecteur de dimension 100000 possède 2 synonymes :

- z dans l'environnement de l'appelant
- t dans l'environnement de l'appelé qui lui est lié

Conséquence, toute modification apportée à `t` dans l'appelé modifie aussi `z`.



Passage de paramètres : par valeur

- Ce mode de passage n'existe pas en Ada.
- On le trouve dans des langages impératifs comme C, C++, Pascal.
- Comme dans le mode `in`, une valeur est transmise de l'appelant vers l'appelé via un paramètre.
- La différence est que, dans l'appelé, ce paramètre n'est pas considéré comme une constante mais comme une variable.
- De même que pour le mode `in`, après retour vers l'appelant, ce type de passage garantit que le paramètre effectif conserve sa valeur d'appel.

Exemple

```
declare
  type Tableau is array(Positive range<>) of Integer;
  procedure init(x:in Positive;T:out Tableau) is
  begin
    for i in T'range loop
      T(i):=x;
      x:=x+1;
-- affectation illicite en Ada car x est une constante
-- mais affectation licite en Pascal ou C
    end loop;
  end init;
  Tab:Tableau(1..20);
begin
  init(1,Tab);
end;
```

Profil des paramètres d'une procédure

Deux procédures ont le même profil de paramètres si :

- elles ont le même nombre de paramètres
- à chaque position, les paramètres ont le même type

```
procedure P(a:in Character:='A';b:out Integer);
procedure Q(x:in Character;y:in out Integer);
```

Les procédures `P` et `Q` ont le même profil de paramètres

Les noms, mode de transmission et expression par défaut ne sont pas pris en compte

Masquage de procédures

Si deux procédures ont même profil et même identificateur, la dernière déclarée masque la précédente.

Une déclaration de procédure ne peut pas masquer une déclaration de fonction et vice versa.

Masquage

```
with Ada.Text_io;
use Ada.Text_io;
procedure masquage is
  a,b:Character;
  procedure permuter(x,y:in out Character) is
     z:Character;
 begin
     z:=x;x:=y;y:=z;
  end permuter;
  function permuter(x,y:Character) return Character is
  begin
     return x;
  end permuter;
begin
  put( "a=" );get(a);put( "b=" );get(b);
  permuter(a,b);put( "a=" );put(permuter(a,b));
end masquage;
```

Surcharge de procédures

Un identificateur de procédure est surchargé si :

- il identifie plusieurs procédures
- si ces procédures n'ont pas le même profil de paramètres

Un tel identificateur possède plusieurs sémantiques

```
procedure permuter(a:in out Character; b:in out Character);
procedure permuter(x:in out Integer; y:in out Float);
```

Le code de ces deux procédures sera différent puisque leur type est différent.

Ces deux déclarations sont dites surchargées.

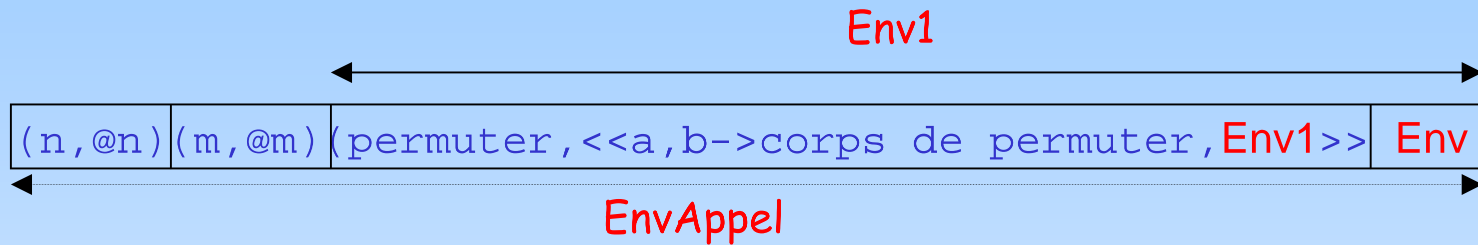
```
with Ada.Text_io;use Ada.Text_io;
procedure surcharge2 is
   x:Integer:=1;
   y:Integer:=0;
   z:Integer;
begin
   z:=x+y;
   put("z externe=");put(Integer'image(z));new_line;
   declare
      subtype bool is Natural range 0..1;
      function "+"(a,b:bool) return bool is
      begin
        if a=1 and b=1 then return 1; else return 0;end if;
      end "+";
      z:Integer;
   begin
      z:=x+y;
-- + sur les Integer est masqué
      put("z interne=");put(Integer'image(z));
   end;
end surcharge2;
```

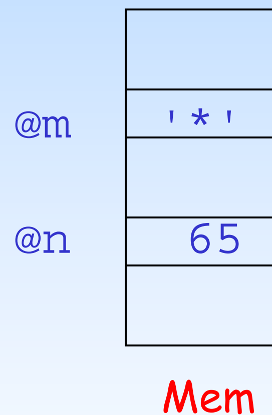
```
z externe = 1
z interne = 0
```

Sémantique de l'appel de la procédure **permuter**

```
with Ada.Text_io, Ada.Integer_Text_io;
use Ada.Text_io, Ada.Integer_Text_io;
procedure car_int is
   procedure permuter(a:in out Character; b:in out Integer) is
        z:Integer;
   begin
        z:=Character'pos(a);a:=Character'val(b);b:=z;
   end permuter;
  m:Character:='*';n:Integer:=65;
begin
  put( " m=" );put(m);put( "n=" );put(n,width=>2);new_line;
  permuter(m,n);
-- ou permuter(b=>n,a=>m);
  put( "permuter(m,n);" );new_line;
  put( " m=" );put(m);
  put( "n=" );put(n,width=>2);new_line;
end car_int;
```

Etat avant l'appel

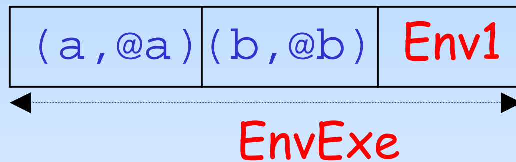


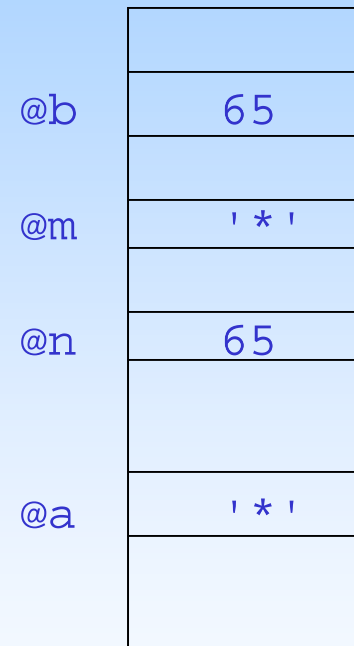


Etat avant l'appel `permuter(m,n) : (`*EnvAppel,Mem*`)`

Etat après transmission des paramètres

```
permuter(b=>n,a=>m);
```



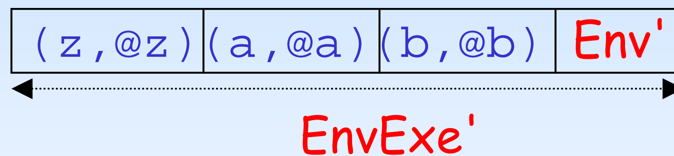


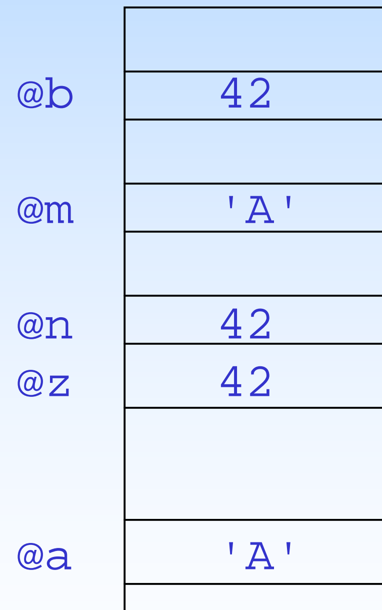
Etat initial de l'appelé (après transmission des paramètres) : (EnvExe,Mem1)

Etat à la fin de l'exécution de la procédure

Exécution du corps de la procédure

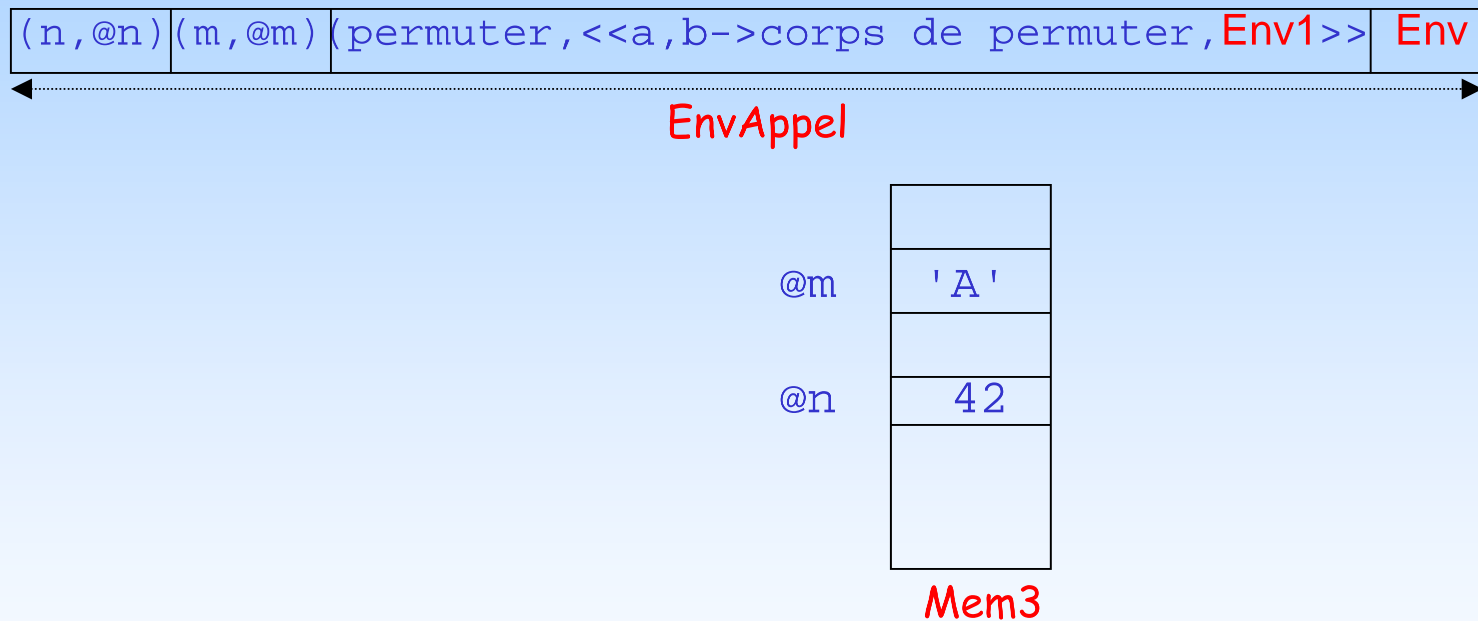
```
z:=Character'pos(a);
a:=Character'val(b);
b:=z;
```





Etat final de la procédure appelée : (EnvExe',Mem2)

Etat après retour vers l'appelant



Etat après retour vers l'appelant de la procédure : (EnvAppel',Mem3)

Procédures récursives

Comme les fonctions, les procédures Ada sont naturellement récursives.

Exemple : Les tours de Hanoï

3 pieux A,B,C sont plantés en terre. Initialement, sur le pieu A sont empilés des disques de taille décroissante. Les pieux B et C sont alors vides.

A l'état final tous les disques sont empilés sur le pieu C.

Les règles sont les suivantes :

- on ne peut déplacer qu'un disque à la fois et bien sûr, il ne peut être placé qu'en haut d'une pile (au sommet d'un pieu).
- il est possible d'utiliser le pieu B
- on peut toujours empiler un disque sur un autre à la condition que sa taille soit inférieure

La procédure **hanoi** (1/3)

La procédure `hanoi` construit et affiche la suite des déplacements de disques de manière récursive.

Les 3 pieux sont représentés par les caractères 'A, 'B', 'C'.

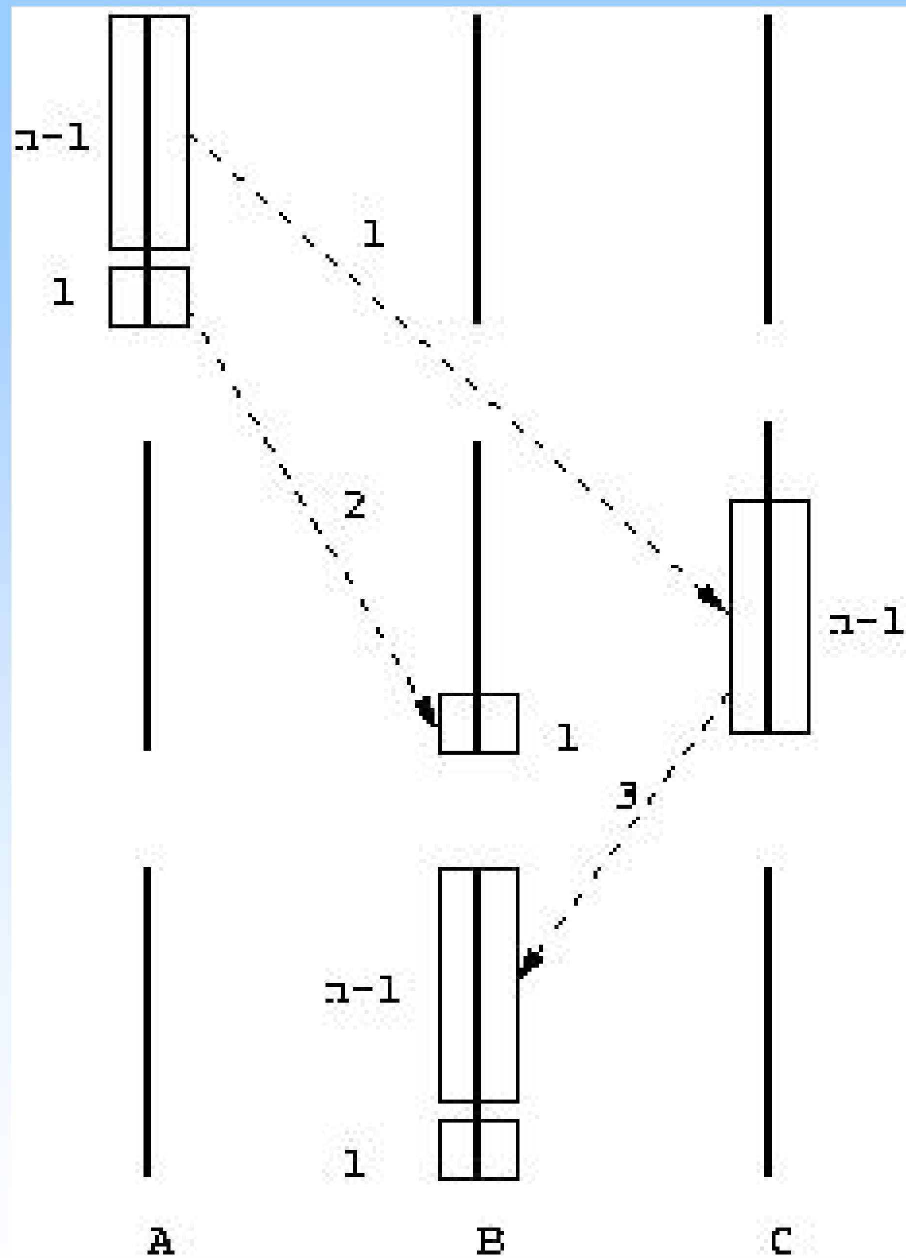
Les disques sont représentés par un entier qui simule leur taille.

La procédure **hanoi** (2/3)

```
procedure hanoi(n:in Natural;X,Y,Z:in Character) is
  begin
    if n/=0
    then
      hanoi(n-1,X,Z,Y);
      put( "disque : " );put(n,WIDTH=>2);put( " : " );
      put(X);put( " --> " );put(Y);new_line;
      hanoi(n-1,Z,Y,X);
    end if;
  end hanoi;
```

La procédure **hanoi** (3/3)

```
with Ada.Text_io;use Ada.Text_io;
with Ada.Integer_Text_io;use  Ada.Integer_Text_io;
procedure tours_hanoi is
  procedure hanoi(n:in Natural;X,Y,Z:in Character) is
-- voir transparent précédent
  end hanoi;
  A:Character:='A';
  B:Character:='B';
  C:Character:='C';
begin
  hanoi(3,A,C,B);
end tours_hanoi;
```


n-1
1
1
2
n-1
1
3
n-1
1
A
B
C

La procédure **hanoi** : résultats

Le résultat pour un pieu A où sont empilés 3 disques est :

```
disque :  1 : A --> C
disque :  2 : A --> B
disque :  1 : C --> B
disque :  3 : A --> C
disque :  1 : B --> A
disque :  2 : B --> C
disque :  1 : A --> C
```

La procédure `hanoi` : résultats

L'idée est, ici de dire :

Si aucun disque n'est empilé sur `A`, alors il n'y a rien à faire

Supposons que je sache déplacer les n-1 disques les plus petits de `A` vers `C` en utilisant `B`

Alors il suffit de placer le plus gros disque de `A` sur le pieu libre `B` et de recommencer ce que j'ai supposé savoir faire, c'est à dire déplacer les n-1 disques (cette fois de `C` vers `B` en utilisant `A`).