CH.6 ENVIRONNEMENTS D'EXÉCUTION

- 6.1 Les langages procéduraux
- 6.2 L'organisation de l'espace mémoire
- 6.3 Les stratégies d'allocation
- 6.4 L'accès aux noms non locaux
- 6.5 Le passage des paramètres
- 6.6 Les tables des symboles
- 6.7 L'allocation dynamique

6.1 Les langages procéduraux

Exemples : PASCAL ou C, le programme lui-même est une procédure.

Hypothèses :

• Déroulement séquentiel avec contrôle en un point spécifique ;

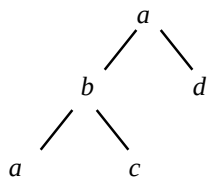
• Toute exécution de procédure rend le contrôle au point qui suit immédiatement le point d'où elle a été appelée.

Conséquence :

Si *a* et *b* sont des activations de procédures, leurs durées de vie sont disjointes ou imbriquées.

Possibilité de procédures récursives.

Représentation par un **arbre d'activation** :



Parcours en profondeur, avec priorité à gauche.

Les procédures actives sont conservées dans la **pile de contrôle**.

La liste des procédures actives est le chemin de la racine vers la dernière procédure activée.

La gestion en mémoire de la pile de contrôle permet d'implémenter les langages procéduraux.

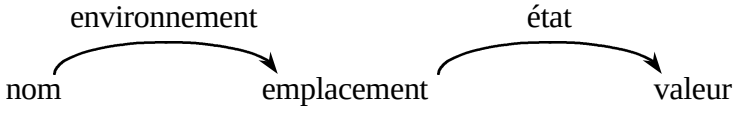
Portée d'une déclaration

Possibilité de déclaration multiple d'un même nom.

Résolu par les **règles de portées**.

Lors de la compilation (pas à l'exécution !) la table des symboles permet de gérer cette portée.

Liaison des noms



Liaison du nom = dynamique

6.2 L'organisation de l'espace mémoire

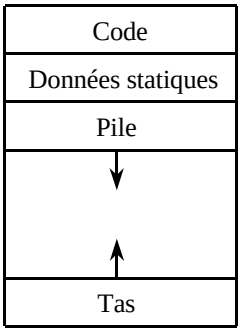
Destiné à l'**exécution**.
Dépend d'options fondamentales :

- Procédures récursives ;
- Devenir des valeurs associées aux noms locaux ;
- Références d'une procédure à des noms non locaux ;
- Passage des paramètres ;
- Possibilité de retourner une procédure comme résultat ;
- Possibilité d'allocation dynamique d'espace mémoire ;
- Modalité de libération de l'espace mémoire.

Exemple pour langages tels que FORTRAN, PASCAL ou C.

Le compilateur obtient un bloc de mémoire pour y placer :

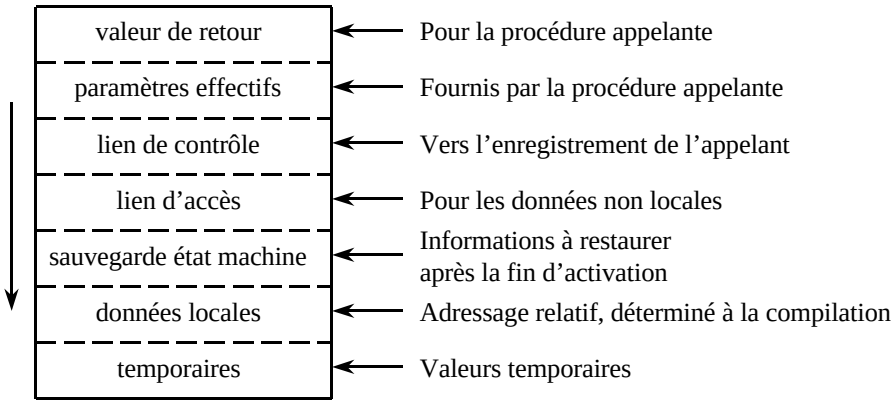
- Code cible ;
- Données ;
- Pile de contrôle.



Le code cible et certains emplacements de données sont **statiques**.

Le maximum d'informations est contenu dans la pile, sous forme d'**enregistrements d'activation**.

Enregistrement d'activation



Les tailles sont fixes, connues à la compilation, sauf pour des tableaux dont la taille dépend d'un argument (voir section suivante).

6.3 Les stratégies d'allocation

Allocation **statique**, dans la **pile**, dans le **tas** ("dynamique").

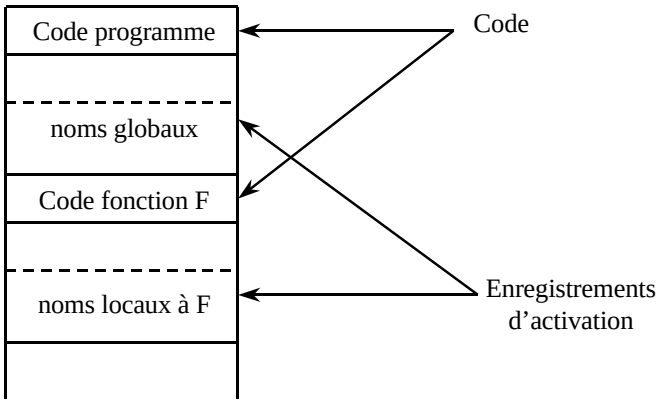
- **Allocation statique**

  Noms liés à des emplacements au fur et à mesure de la compilation ;
  Enregistrements d'activation rangés à une place fixée.
  En fin de compilation, toutes les allocations en mémoire sont mises en place.

  Avantages : aucun calcul d'adresse, taille minimale pour les données.

  Inconvénients : taille des données connue à la compilation, pas de création dynamique de structures, peu de possibilités de récursivité.

Remarque : ceci n'interdit pas les noms locaux ; ils sont alloués statiquement dans les enregistrements d'activation.



Accès aux noms locaux, voir plus bas.

Exemple : FORTRAN

- **Allocation dans la pile**

Pile de contrôle : les enregistrements d'activation sont empilés au début de l'activation et dépilés à la fin.

Les variables locales sont allouées dans les enregistrements d'activation. Elles sont perdues au dépilage.

Un registre Sommet Pile marque le sommet de la pile ; cas simple, si tous les enregistrements d'activation ont une taille connue à la compilation.

L'enregistrement au sommet de la pile correspond à l'activation qui possède le contrôle.

Le code de chaque fonction alloue les adresses des variables locales dans la pile relativement au début de l'enregistrement.

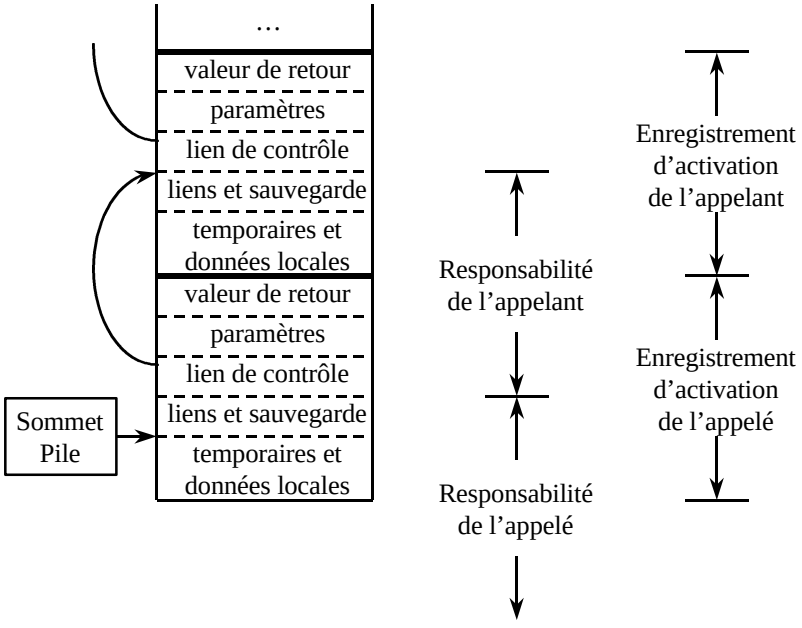
Protocole d'appel

Séquence d'appel : alloue et initialise un enregistrement d'activation ;
Séquence de retour : restaure l'état de la machine.

Implémentation dépendant de facteurs matériels et logiciels.

Réparti entre appelant et appelé sous forme de code. Essayer de solliciter au maximum l'appelé (optimise la quantité de code), et d'uniformiser les codes entre les diverses fonctions (plus simple et clair).

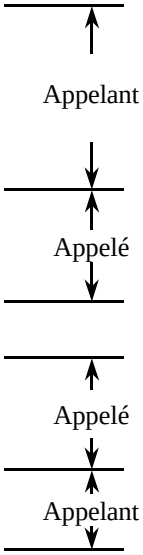
La répartition est donc sujette à des variantes.



Exemple : déroulement possible d'une séquence.

• Séquence d'appel
1.— Évaluation des arguments ;
2.— Stockage dans l'enregistrement de l'appelé de l'adresse de retour, du lien de contrôle et incrémentation de Sommet Pile ;
3.— Sauvegarde des registres et de l'état ;
4.— Initialisation des variables locales et exécution.

• Séquence de retour
1.— Placement de la valeur de retour ;
2.— Restauration de l'état et branchement à l'adresse de retour ;
3.— Utilisation de la valeur de retour et restauration de Sommet Pile.

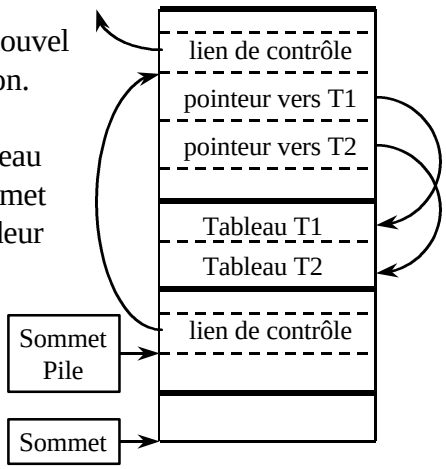


Données de taille variable

Elles sont stockées dans la pile et référencées par pointeur. Les emplacements correspondants ne font pas partie de l'enregistrement d'activation.

Sommet sert à empiler un nouvel enregistrement d'activation.

En fin d'activation, le nouveau Sommet est l'ancien Sommet Pile, diminué d'une grandeur connue à la compilation.



• **Allocation dans le tas**

Dans certains cas, il n'est pas possible de faire des allocations dynamiques :

Raisons dues à la sémantique du langage :
conservation de valeur des noms locaux,
chevauchement des activations de procédures.

Raisons de manipulations de données :
allocation dynamique de mémoire,
modification par un appelé de la taille d'une variable non locale.

Allocation faite dans le tas, avec problèmes de gestion de l'espace mémoire, enregistrements non adjacents, libération de l'espace, réorganisation de celui-ci.

6.4 L'accès aux noms non locaux

Comment traiter les noms non locaux ?
Règles de portée, dépendant des langages.

Deux possibilités :

• Portée statique, déterminable à la compilation (PASCAL, C, ADA)

• Portée dynamique, déterminée à l'exécution (LISP, APL)

En général, portée statique complétée par la règle de "l'englobant le plus imbriqué".

Dans ce cas, l'implémentation est différente en C, où les procédures ne sont pas imbriquées et en PASCAL.

Blocs

Notion introduite par ALGOL (1960).

Comportent une partie déclaration de variables locales et des instructions.

Ne doivent pas se chevaucher.

La portée de la déclaration d'une variable s'étend à tous les blocs inclus où cette variable n'est pas redéclarée.

Ceci permet d'implémenter les blocs dans la pile de contrôle comme des procédures sans paramètres et ne retournant pas de valeur, et de traiter l'accès aux noms non locaux comme dans les procédures.

Néanmoins, problèmes à régler si possibilité de sauts hors d'un bloc ou au milieu d'un bloc.

Portée statique–procédures non imbriquées (cas de C)

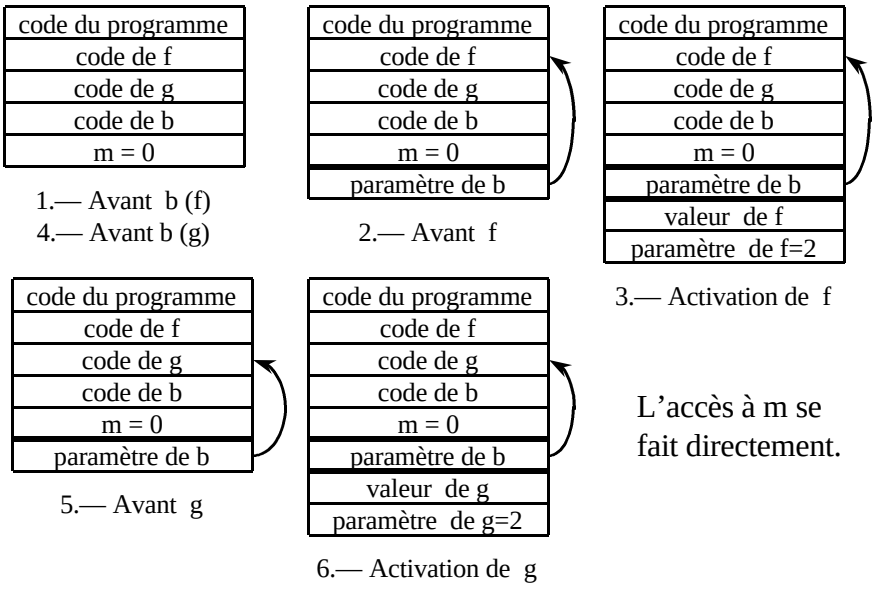
Pas de procédure imbriquée. Les noms non locaux sont donc déclarés en dehors de toute procédure. Ils peuvent être alloués statiquement. Toute référence à un nom est soit son adresse calculée statiquement, soit celle de l'enregistrement actif.

Ceci permet d'utiliser librement les procédures comme paramètres ou comme résultat (Les noms non locaux à celles-ci sont nécessairement à une adresse fixe).

Exemple : programme avec noms non locaux alloués statiquement.

```
program Passer (input, output) ;
   var m : integer ;

   function f (n : integer) : integer ;
      begin f := m + n end { f } ;

   function g (n : integer) : integer ;
      begin g := m * n end { g } ;

   procedure b (function h (n : integer) : integer) ;
      begin write (h (2)) end { b } ;

   begin
      m := 0 ;
      b (f) ; b (g) ; writeln
   end.
```

États successifs de la pile de contrôle :



1.— Avant b (f)
4.— Avant b (g)

2.— Avant f

3.— Activation de f

5.— Avant g

6.— Activation de g

L'accès à m se fait directement.

Portée statique–procédures imbriquées (cas de PASCAL)

On connaît à la compilation la **profondeur d'imbrication** de chaque procédure : 1 pour le programme, etc.

On ajoute un **lien d'accès** : si la procédure P est immédiatement imbriquée dans T dans le source, le lien d'accès d'un enregistrement d'activation de P référence le lien d'accès de l'enregistrement d'activation le plus récent de T.

Le code mettant en place les liens d'accès fait partie du protocole d'appel. Les grandeurs qu'il fait intervenir doivent donc être connues à la compilation.

Mise en place des liens d'accès :

Si P de profondeur *np* appelle X de profondeur *nx*.

Cas n°1 : $np < nx$:

X est nécessairement déclarée dans P. Le lien d'accès de l'enregistrement de X (appelé) référence celui de P (appelant).

Cas n°2 : $np \geq nx$:

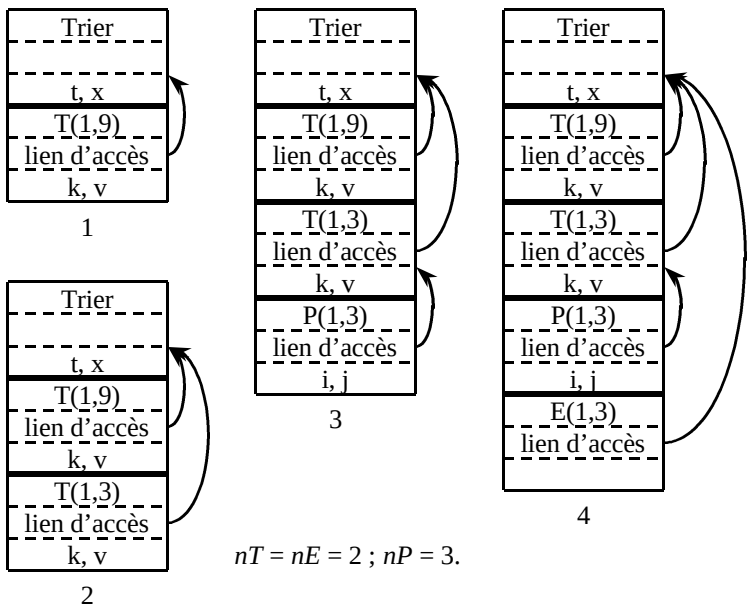
X et P ont mêmes procédures englobantes jusqu'au niveau $nx - 1$. Il faut donc remonter les niveaux d'imbrication à partir de l'enregistrement de P (appelant) en suivant $np - nx + 1$ liens. Cela donne la référence du lien d'accès de X (appelé).

Accès aux noms non locaux

Si P de profondeur *np* référence t de profondeur $nt \leq np$:

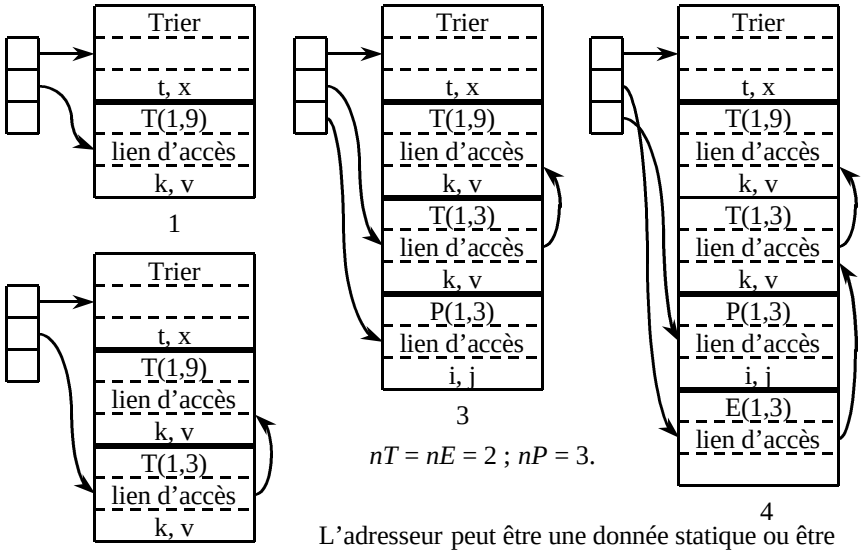
On suit $np - nt$ liens d'accès à partir de l'enregistrement d'activation de P (actif) pour obtenir l'enregistrement de la procédure où t est locale, puis on sait où se situe l'emplacement associé relativement au lien d'accès.

```
program Trier (input, output) ;
   var t : array [0..10] of integer ;
   x : integer ;

   procedure Echanger (i, j :integer) ;
      begin
         x := t [i] ; t [i] := t [j] ; t [j] := x
      end { Echanger } ;

   procedure TriRapide (m, n : integer) ;
      var k, v : integer ;

      function Partition (y, z : integer ) : integer ;
         var i, j : integer ;
            begin ... t ... v ... Echanger (i,j) ; ...
            end { Partition } ;

      begin ... end { TriRapide } ;

   begin ... end { Trier }.
```



nT = *nE* = 2 ; *nP* = 3.

On peut aussi gérer l'accès par un adresseur



nT = *nE* = 2 ; *nP* = 3.

L'adresseur peut être une donnée statique ou être dupliqué dans chaque enregistrement d'activation.

Passage d'une procédure en paramètre

```
program Param (input, output) ;

   procedure b (function h (n : integer) : integer) ;
      begin writeln (h (2)) end { b } ;

   procedure c ;
      var m : integer ;

      function f (n : integer) : integer ;
          begin f := m + n end { f } ;

      begin m := 0 ; b (f) end { c } ;

   begin c end.
```

Lorsque c appelle b, il fournit le paramètre f ; mais quand b devient active, f est normalement hors de portée de b. Donc c fournit le lien d'accès de f, qui est dans sa portée.

Portée dynamique

Tout nom non local conserve le même emplacement pour l'appelé qu'il avait pour l'appelant.

Peut être implémenté par

accès profond : on suit le lien de contrôle de l'enregistrement actif jusqu'au premier enregistrement contenant l'emplacement associé à ce nom (conserver les noms dans les enregistrements) ;

accès superficiel : valeur associée à chaque nom dans un emplacement alloué statiquement ; à une nouvelle activation, les noms locaux utilisent les valeurs correspondantes, et les anciennes valeurs sont sauvegardées dans l'enregistrement pour être restituées à la fin de l'activation.

Première solution simple, mais profondeur de recherche inconnue à la compilation.

Seconde solution plus rapide, mais complique les protocoles d'appel.

Exemple : portée statique–portée dynamique

```
program Dynamique (input, output) ;
   var r : integer ;

   procedure Montrer ;
      begin write (r) end ;

   procedure Double ;
      var r : integer ;
      begin r := 2 ; Montrer end ;

   begin r := 1 ;
      Montrer ; Double ; writeln ;
      Montrer ; Double ; writeln
   end.
```

Portée statique :

```
1 1
1 1
```

Portée dynamique :

```
1 2
1 2
```

6.5 Le passage des paramètres

Manière dont la procédure appelante transmet les paramètres de la procédure appelée.

Exemple :

```
program ValeurResultat ;
   var a : integer ;

   procedure Hasardeuse (x : integer)
      begin x := 2 ; a := 0 ; writeln (x) end ;

   begin
      a := 1 ; Hasardeuse (a) ; writeln (a)
   end.
```

Passage par valeur (Cas de C, PASCAL).

Paramètres traités comme des noms locaux à l'appelé.
Leur emplacement est dans l'enregistrement d'activation de l'appelé.

L'appelant détermine leur valeur et place le résultat dans l'enregistrement de l'appelé.

Les modifications de valeur des variables locales (et donc des paramètres formels) sont perdues.

La seule façon de modifier les paramètres est à travers des noms non locaux (effet de bord) ou par usage de pointeurs.

A l'exécution du programme ValeurResultat on obtient
2
0

Passage par référence (cas de PASCAL).

L'appelant calcule la valeur des paramètres. Le résultat est placé dans l'emplacement prévu (variable ou temporaire).

L'appelant passe l'adresse de cet emplacement à l'appelé.

A l'exécution du programme ValeurResultat on obtient

0

0

Passage par copie-restauration (certains FORTRAN).

L'appelant évalue les paramètres et place le résultat dans l'emplacement prévu (variable ou temporaire). Puis il passe à l'appelé la valeur comme paramètre et aussi l'adresse.

L'appelé agit comme dans le passage par valeur mais, avant de rendre le contrôle, il recopie la valeur finale des paramètres dans l'emplacement référencé par leurs adresses.

A l'exécution du programme ValeurResultat on obtient

2

2

Passage par nom (ALGOL)

La procédure est, en fait, une macro-instruction développée en ligne.

Aux noms locaux sont attribués de nouveaux noms et les paramètres formels sont remplacés littéralement par les arguments effectifs.

Intérêt théorique et historique, sauf pour les petites procédures, car évite les enregistrements d'activation.

Différent du passage par référence : dans ce dernier, les adresses des arguments effectifs sont fournies à l'appelé **avant** son exécution. Dans le passage par nom, elles ne sont utilisées que pendant l'exécution.

```
procedure Troquer (x, y : integer) ;
   var temp : integer ;
   begin
      temp := x ; x := y ; y := temp
   end ;
```

L'effet de **`Troquer (i, t [i])`** dans un appel par référence est bien d'échanger les valeurs de **`i`** et de **`t [i]`**.

Par contre, un appel par nom revient à faire :

```
temp := i ;
i := t [i] ;
t [i] := temp ;
```

Ceci place l'ancienne valeur de **`t [i]`** en **`i`** et l'ancienne valeur **`i0`** de **`i`** en **`t [t [i0]]`** .

6.6 Les tables des symboles

Utilisé par le compilateur pour garder trace de la portée et de la liaison des noms. Recherche dans la table à chaque occurence d'un nom dans le source.

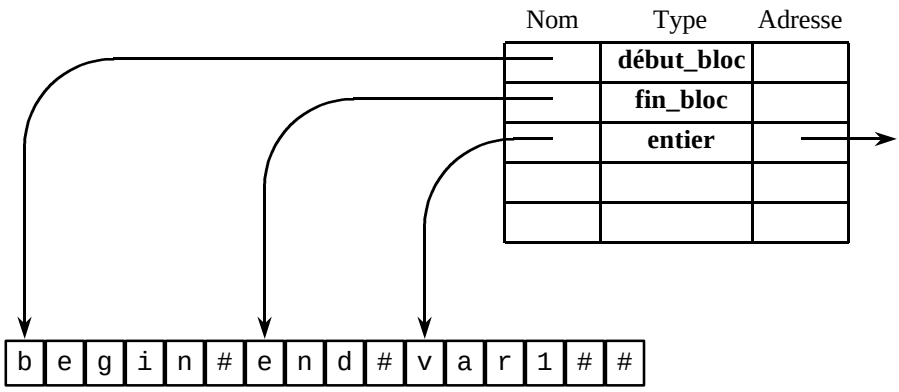
Normalement inutile pour l'exécution (le code produit ne contient que des adresses en mémoire). Utile néanmoins pour les gestionnaires d'erreur.

Par contre, présente à l'exécution dans le cas d'un langage interprété.

Nécessité de prévoir une gestion dynamique de la table des symboles.

Structure possible variée, souvent pointeurs vers tableau de caractères pour les noms et vers emplacements si nécessaire.

Mots clés parfois présents à l'initialisation.



Toutes les structures permettant une implémentation des tables sont possibles : listes linéaires, pile, table d'adressage dispersée.

Table initialisée par l'analyseur lexical. Parfois doit être repoussée à l'analyse syntaxique (si non unicité des noms de variables).

Informations sur la portée.

Problème des noms locaux et non locaux.

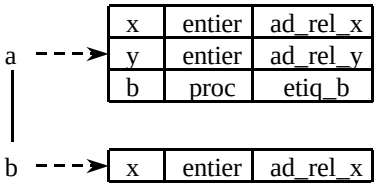
On peut fabriquer une table de symboles par procédure (les blocs étant considérés comme des procédures).

Les adresses sont des adresses relatives dans l'enregistrement d'activation pour les variables et des adresses dans le code pour les procédures.

On construit par ailleurs l'arbre représentant la structure d'imbrication des procédures. Les tables de symboles sont attachées aux nœuds. On peut donc appliquer les règles de portée.

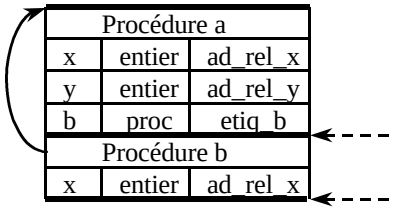
Attention :on est ici en phase de compilation, donc les enregistrements d'activation n'existent pas ; on indique en fait comment les construire ; cette information est utilisée dans la production de code.

```
procedure a ;
   var x, y : integer ;
   procedure b ;
      var x : integer ;
      begin ... end { b } ;
   begin ... end { a } ;
```



Une fois qu'on a fini de compiler une procédure, sa table des symboles n'a plus d'utilité que pour le gestionnaire d'erreurs.
Seuls les symboles des procédures en cours de compilation sont **actifs** (analogie avec la pile de contrôle à l'exécution).

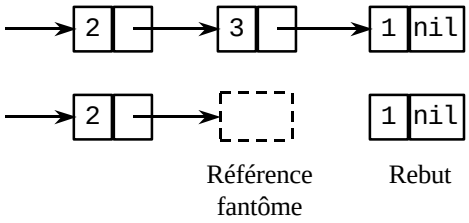
On peut ne gérer qu'une unique table des symboles, avec des algorithmes d'insertion, de recherche et de suppression des symboles inactifs.
Ceci est réalisable de la même manière que la gestion de la pile de contrôle, avec marqueurs et liens de portée.



6.7 L'allocation dynamique

Allocation dynamique explicite ou implicite.
Allocation implicite spécifiée par le compilateur, dans la pile ou dans le tas. Dans ce cas, la libération est aussi spécifiée par le compilateur.

Problèmes de l'allocation explicite :
- Rebut : zone allouée mais inaccessible ;
- Référence fantôme : accès à une zone libérée.



Allocation explicite de blocs :
Liste chaînée des blocs libres et gestion de la liste en fonction des allocations et libérations.
Simple à mettre en œuvre lorsque les blocs sont de taille fixe.
Lorsque les blocs sont de taille variable, problèmes de fragmentation de la mémoire : à la libération, regrouper les blocs libres contigus. Problème pour trouver un bloc libre pour l'allocation d'un bloc de taille donnée (première offre).

Libération explicite prévue par le programmeur.
En PASCAL ou C, seule est prévue la libération explicite, prévue par le programme ou produite par le compilateur.

Les langages produisant une forte allocation dynamique, par exemple à l'exécution, doivent pratiquer la récupération du rebut ("**garbage collection**"). Cas de LISP, MAPLE.

• Compteurs de référence :
Chaque bloc contient un compteur du nombre de blocs le référençant. Lorsque le compteur est à 0, ce bloc est libre.
Réciproque fausse en cas de circularité.
Nécessite la gestion de ces compteurs.

• Marquage :
Suspendre périodiquement l'exécution et parcourir tous les pointeurs (recherche des sommets accessibles dans un graphe).
Nécessite de conserver les références de tous les pointeurs vers le tas.

On peut profiter de cette interruption pour compacter le tas.