Chapitre 7
Environnements d'exécution

1. Considérations sur le langage source
2. Organisation de la mémoire
3. Stratégies d'allocation mémoire
4. Accès aux noms non-locaux
5. Passage de paramètres
6. Tables de symboles
7. Allocation dynamique

Introduction

Objectifs:

1. Description de la relation entre:

Programme source	Exécution
(statique)	(dynamique)
noms	adresses

2. Description du programme de gestion de l'exécution ('run-time support package' chargé avec le code généré).

Langages considérés: Fortran, Pascal, C, Java, LISP.

Aspects du langage source

Notions de base:

1. Définitions de procédures.
   (a) Identificateur (nom de la procédure)
   (b) Instruction (corps de la procédure)
2. procédures vs fonctions
3. Paramètres réels vs paramètres formels
4. Durée de vie d'une activation de procédure
5. procédure récursive
6. arbres d'exécution (voir exemple)
7. piles d'exécution
8. portée d'une déclaration: noms locaux vs globaux.

Exemple

```
program sort(input,output);
   var a :  array[0..10] of integer;
   procedure readarray;
      var i :  integer;
      begin
         for i:= 1 to 9 do read(a[i])
      end;
   function partition(y, z :  integer) :  integer;
      var i :  integer;
      begin ...
      end;
   procedure quicksort(m, n:  integer);
      var i :  integer;
      begin
         if (n>m) then begin
            i := partition(m,n);
            quicksort(m,i-1);
            quicksort(i+1,n)
         end
      end;
   begin
      a[0] := -9999; a[10] := 9999;
      readarray;
      quicksort(1,9)
```

```
end.
```

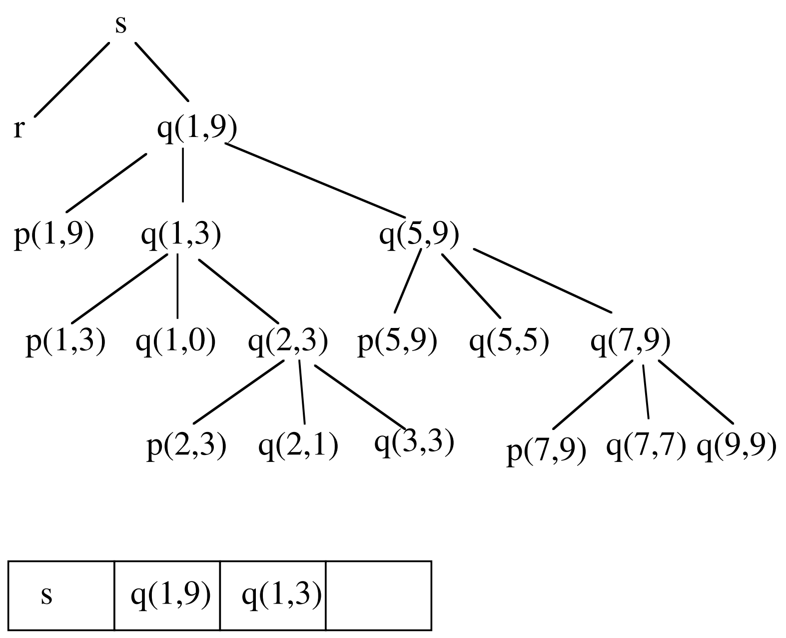


Figure 1: arbre et pile d'execution

Langage source (suite)

Passage en deux étapes des noms aux valeurs:

$$\text{nom} \xrightarrow{\text{environnement}} \text{adresse} \xrightarrow{\text{état}} \text{valeur}$$

Si l'environnement associe le nom x à l'addresse s (l-valeur), on dit que x est *lié* à s.

STATIQUE	DYNAMIQUE
définition de procédure	activation de procédure
déclaration d'un nom	liens du nom
portée d'une déclaration	durée de vie du lien

Traits distinctifs des langages

L'organisation d'un compilateur dépend des réponses à des questions sur le langage source du genre:

1. Récursivité des procédures?
2. Traitement des noms locaux en fin d'exécution d'une procédure.
3. Une procédure peut-elle utiliser des noms non-locaux?
4. Passage des paramètres?
5. Les procédures peuvent-elles être passées en paramètre?
6. Les valeurs peuvent-elles être des procédures?
7. L'allocation de mémoire peut-elle être faite de façon dynamique par programme?
8. La libération doit-elle être explicite?

Organisation de la mémoire

Zône utilisée en mémoire pour l'exécution d'un programme:

code
données statiques
pile
$\downarrow$ $\uparrow$
tas

1. Code: code exécutable (taille connue à la compilation)
2. Données statiques: adresses compilées dans le code (toutes les données en Fortran)
3. Pile: pile des activations de procédure (Pascal ou C).
4. Tas: emplacement de toutes les autres informations (allocation dynamique en Pascal).

Les adresses dans la pile sont des *adresses relatives* par rapport au sommet de la pile (place souvent dans un registre). La différence s'appelle l'*offset*.

Enregistrements d'activation

Bloc mémoire empilé à l'appel d'une procédure et dépilé à la fin de son exécution.

valeur rendue
paramètres d'appel
lien de contrôle
lien d'accés
sauvegarde de l'état machine
données locales
zône temporaire

1. Zône temporaire: évaluation des expressions.
2. Données locales: variables locales à la procédure.
3. état machine: sauvegarde de l'état du système au moment de l'activation de procédure (compteur d'instruction+ autres registres).
4. Lien d'accès: sert à trouver le lien pour les variables non-locales (utile en Pascal).
5. Lien de contrôle: pointe sur le registre d'activation appelant.
6. Paramètres d'appel: valeur des paramètres (en pratique souvent dans des registres).

7. Valeur rendue: valeur rendue à la procédure appelante (idem).

Placement des données locales

'Layout': calcul d'adresses fait à la compilation. Caractéristiques:

1. Unité minimale=octet ('byte') de 8 bits.
2. Mot machine= qlq octets.
3. Placement des objets: sur des octets consécutifs avec l'adresse du premier octet.
4. Placement fait à la lecture des déclarations locales à une procédure (exception pour les objets de taille variable).
5. Adresse relative: *offset*.
6. Particularités de l'adressage sur chaque machine, par exemple *alignement* des entiers placés à des adresses multiples de 4. Espace laissé libre='padding'.

Exemple

Placement fait par un compilateur C sur deux machines. Les deux sont adressables par octet seulement.

Sur la machine 1, quatre octets peuvent, par exemple être occupés par un caractère suivi d'un un petit entier.

Sur la machine 2, toute la mémoire est organisée en mots de 64 bits. Les adresses des mots sont sur 24 bits et 6 bits supplémentaires sont utilisés pour adresser les 64 positions à l'intérieur d'un mot.

TYPE	Taille (bits)		Alignement (bits)	
	M. 1	M. 2	M. 1	M. 2
`char`	8	8	8	64
`short`	16	24	16	64
`int`	32	48	32	64
`long`	32	64	32	64
`float`	32	64	32	64

Statégies d'allocation mémoire
1. Allocation statique

Caractéristiques:

1. Les liens sont permanents pendant toute l'exécution: les valeurs des noms qui sont locaux à une procédure persistent d'un appel au suivant.
2. Les positions des enregistrements d'activation de chaque procédure sont fixés à la compilation.
3. Les adresses des données sont connues à la compilation.

Inconvénients (avec seulement de l'allocation statique):

1. la taille des données doit être connue à la compilation.
2. La récursivité est limitée.
3. Pas de structures de données dynamiques.

Exemple

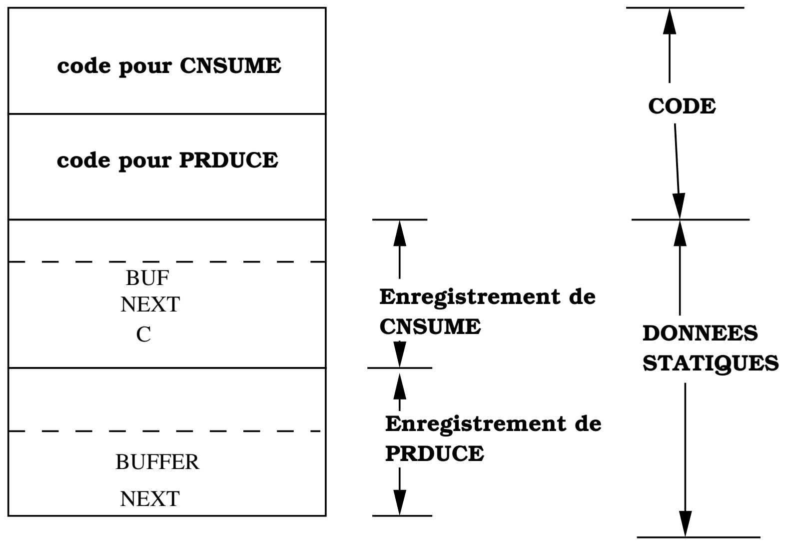
Programme Fortran 77 qui recopie les caractères de la donnée dans un tableau jusqu'à voir un blanc puis les écrit.

```
PROGRAM CNSUME
```

```
   CHARACTER * 50 BUF
   INTEGER NEXT
   CHARACTER C, PRDUCE
   DATA NEXT /1/, BUF /' '/
6     C = PRDUCE()
      BUF(NEXT:NEXT) = C
      NEXT = NEXT + 1
      IF ( C .NE. ' ' ) GOTO 6
      WRITE (*,'(A)') BUF
      END

CHARACTER FUNCTION PRDUCE()
   CHARACTER * 80 BUFFER
   INTEGER NEXT
   SAVE BUFFER, NEXT
   DATA NEXT /81/
      IF ( NEXT .GT. 80 ) THEN
         READ (*,'(A)') BUFFER
         NEXT = 1
      END IF
      PRDUCE = BUFFER(NEXT:NEXT)
      NEXT = NEXT+1
      END
```

L'appel de la fonction `PRDUCE` la première fois initialise `NEXT` à 81 et recopie dans `BUFFER` une ligne de texte. Les appels suivants retrouvent les anciennes valeurs des variables locales `BUFFER` et `NEXT` grâce à l'instruction `SAVE`. Si par exemple,



la donnée est

```
demain matin
```

Le résultat est

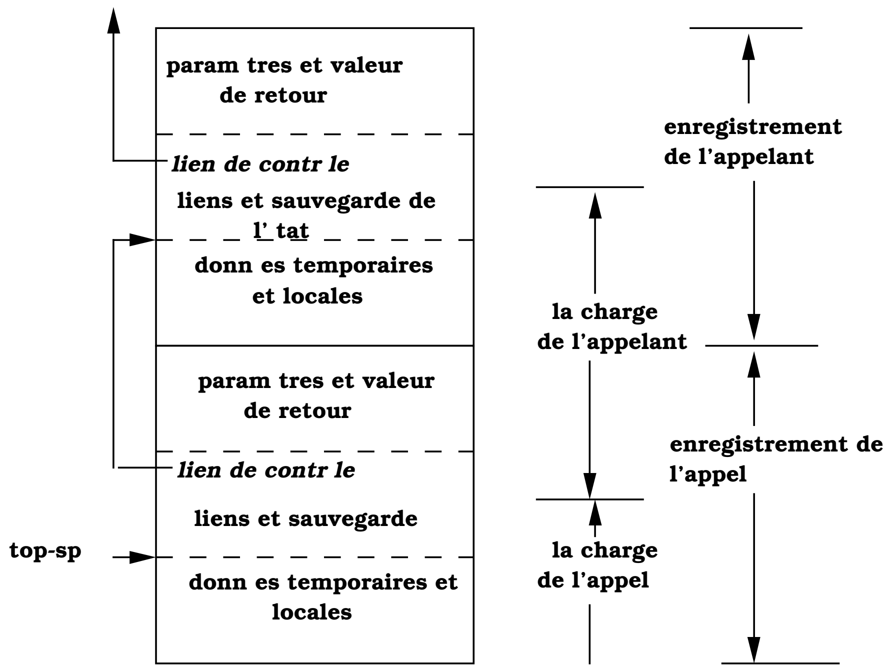
```
demain
```

Allocation en pile

Les enregistrements d'activation sont empilés à l'appel de procédure et dépilés en fin d'exécution $\rightarrow$ les valeurs locales sont perdues.

L'implémentation des appels de procédure se fait en créant des *séquences d'appel* :

1. 'calling sequence': alloue un enregistrement et lui fournit l'information correspondante.
2. 'return sequence': restaure l'état de la machine pour continuer l'exécution.



Division du travail entre appelant et appelé:

1. Séquence d'appel:
   (a) L'appelant évalue les paramètres.
   (b) Il enregistre une adresse de retour et la vieille valeur de *top_sp* dans l'enregistrement d'activation de l'appelé. Il incrémente ensuite *top_sp*.
   (c) L'appelé sauvegarde les valeurs des registres.
   (d) Il initialise ses données locales et commence l'exécution.
2. Séquence de retour:
   (a) L'appelé place une valeur de retour à côté de l'enregistrement de l'appelant.
   (b) Il restaure *top_sp* et les autres registres et saute à une adresse de retour ds le code de l'appelant.
   (c) L'appelant peut copier la valeur rendue par l'appelée.

allocation en pile (suite)
Données de longueur variable

Stratégie: l'enregistrement de la procédure contient des pointeurs vers les zones mémoire de taille variable. Les adresses relatives de ces pointeurs sont connues à la compilation.

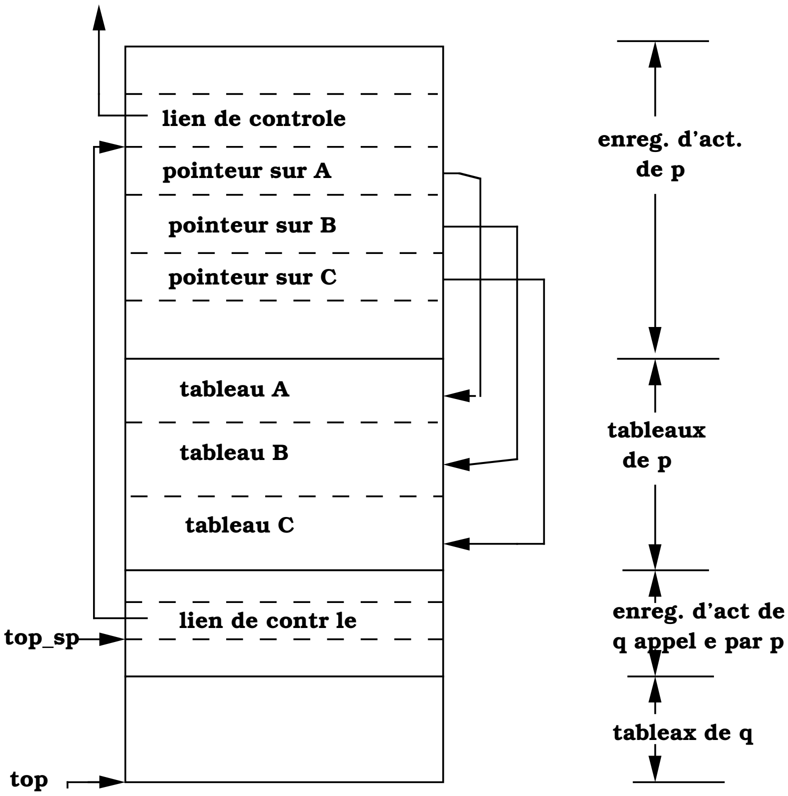


Figure 2: Exemple avec 3 tableaux locaux

Accès aux données sur la pile: à partir des bases fournies par les pointeurs *top* et *top_sp* (pour l'accès aux données locales).

Mise à jour de top et top_sp (à la compilation): quand `q` s'achève,

nouvelle valeur de $top = top\_sp -$ longueur des champs sauvegarde $-$ paramètres de `q`.

nouvelle valeur de $top\_sp =$ lien de contrôle.

Accès aux noms non-locaux

Déterminé par les règles de *portée des variables* (‘scope rules’) spécifiques au langage source. Deux types de règles:

1. Portée statique (ou lexicale): Pascal, C, Ada (portée déterminée par le texte du programme)
2. Portée dynamique: Lisp, APL, Snobol (portée déterminée par les activations en cours).

Structure de blocs

Bloc: { déclarations instructions}.
Règle du bloc le plus intérieur.

1. La portée d'une déclaration faite dans B est inclue dans B.
2. Si x n'est pas déclaré dans B, une occurrence de x dans B est dans le champ d'une déclaration de x dans un bloc plus grand B' tel que:
   (a) x est déclaré dans B'.
   (b) B' est le plus proche de B possible.

Deux implémentations possibles:

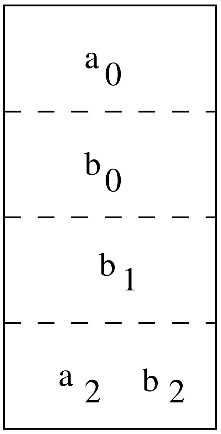
1. Allocation en pile (bloc=procédure sans paramètres).

2. Allocation globale (pour tout le corps d'une procédure).

Exemple: Blocs dans un programme C.

```
main()
{
      int a = 0;
      int b= 0;
      {
      .     int b= 1;
      .     {
      .     B2    int a = 2;
      .     .     printf("%d %d", a, b);
B0    B1    }
      .     {
      .     B3    int b = 3;
      .     .     printf("%d %d", a, b);
      .     }
      .     printf("%d %d", a, b);
      }
      printf("%d %d", a, b);
}
```

Implémentation globale:



Portée statique (sans procédures emboitées)

Les noms déclarés hors des procédures sont alloués de façon statique. Les autres noms sont accessibles à partir du pointeur *top*.

Les procédures peuvent facilement être passées en paramètre et rendues en résultat. Exemple en Pascal (sans procédures emboitées):

```
program pass(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.
```

En C :

```
#include <stdio.h>
int m;
int f(int n){
  return m+n;
}
int g(int n){
  return m*n;
}
int b(int (*h)(int)){
  printf("%d\n",h(2));
}
int main(void){
  m=0;
  b(f);b(g);
}
```

Portée statique (avec procédures emboitées)

Profondeur d'emboitement: le programme principal est à profondeur 1 et on ajoute 1 pour chaque emboitement. Pour un nom, profondeur de la procédure dans laquelle il est déclaré.

Lien d'accés: si la procédure p est emboitée directement dans q, le lien d'accés d'un enregistrement d'act. de p pointe sur le lien d'accés de l'enregistrement de q le plus récent (enchainement possible).

Règle pour trouver l'adresse d'un non-local a de profondeur n_a utilisé dans une procédure p de profondeur $n_p \geq n_a$:

1. remonter $n_p - n_a$ liens d'accès ($(n_p - n_a)$ peut être calculé à la compilation).
2. L'adresse de a dans l'enregistrement d'act. atteint est à un indice ('offset') fixe par rapport au lien d'accès.

Ainsi: l'adresse de a non-local dans p est donnée par le couple ($n_p - n_a$,offset dans l'enregistr. contenant a). Cette information peut être rangée dans la table des symboles.

Code pour calculer les liens d'accès (dans la séquence d'appel): Appel de x à prof. n_x par p à prof. n_p:

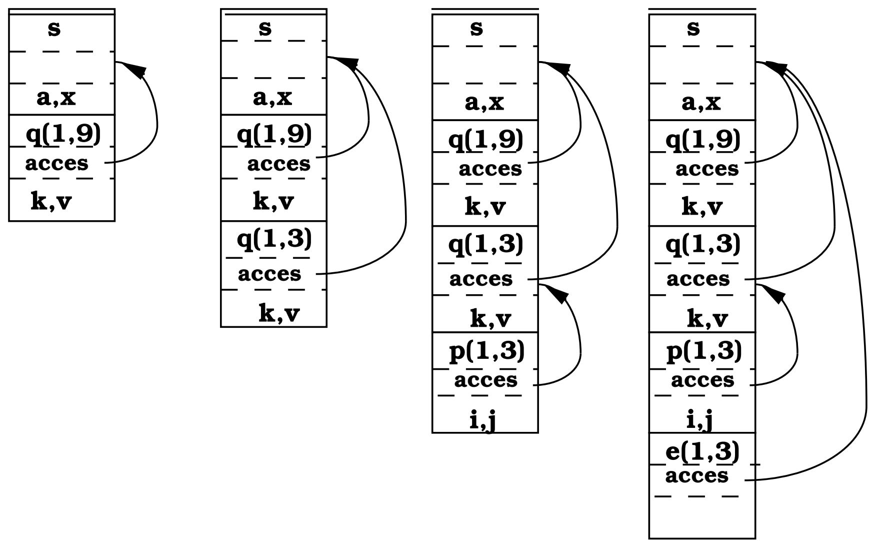
1. Si $n_p \leq n_x$ alors x est déclaré dans p et donc lien d'accès $\rightarrow$ enregistr. juste au-dessous dans la pile.
2. Si $n_p \geq n_x$ on suit $n_p - n_x + 1$ liens d'accès depuis l'enregistr. d'appel (calcul à la compil.).

Exemple: quicksort

```
program sort(input,output);
   var a : array[0..10] of integer;
       x : integer;
   procedure readarray;
      var i : integer;
      begin ... a ... end; {readarray}
   procedure exchange( i,j: integer);
      begin
         x:=a[i];a[i]:=a[j];a[j]:=x
      end {exchange}
   procedure quicksort(m, n: integer);
      var k, v : integer;
      function partition(y, z : integer) : integer;
         var i,j : integer;
         begin ... a ...
               ... v ...
               ... exchange(i,j); ...
         end {partition} ;

      begin ... end {quicksort}
   begin .. end {sort} .
```

Procedures en parametre

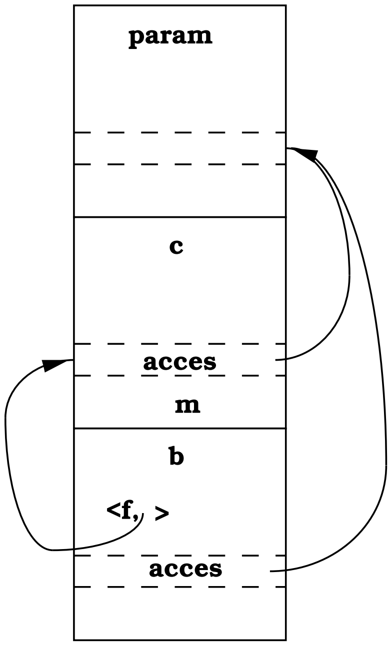


Exemple:

```
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.
```

Calcul du lien d'acces pour l'appel de f? Une



procedure passee en parametre doit voyager avec son lien d'acces.

Accès direct aux non-locaux

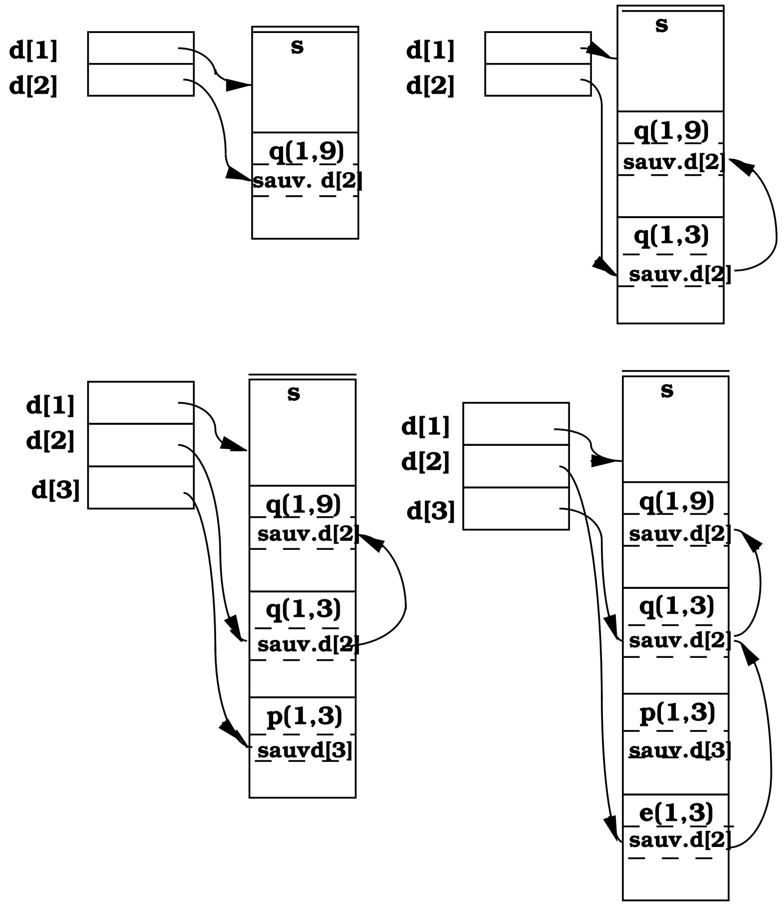
Il existe des mécanismes plus rapides d'accès aux non-locaux. Un tableau de pointeurs ('*display*') vers les enregistrements d'activation permet de le faire.

La place d'un non-local de niveau j est dans l'enregistrement sur lequel pointe $d[j]$.

Mise à jour du tableau des display: quand un nouvel enregistrement de niveau i est empilé, on effectue les opérations suivantes:

1. Sauver la valeur de $d[i]$ dans le nouvel enregistrement.
2. Faire pointer $d[i]$ sur le nouvel enregistrement.

Juste avant de dépiler l'enregistrement, $d[i]$ est remis à la valeur sauvée.


d[1]
d[2]
s
q(1,9)
sauv. d[2]
d[1]
d[2]
s
q(1,9)
sauv.d[2]
q(1,3)
sauv.d[2]
d[1]
d[2]
d[3]
s
q(1,9)
sauv.d[2]
q(1,3)
sauv.d[2]
p(1,3)
sauvd[3]
d[1]
d[2]
d[3]
s
q(1,9)
sauv.d[2]
q(1,3)
sauv.d[2]
p(1,3)
sauv.d[3]
e(1,3)
sauv.d[2]

Portée dynamique

Les liens des non-locaux ne changent pas quand on empile. Un non-local a dans la procedure appelée réfère au même emplacement que dans l'appelante.

Exemple

```
program dynamic(input,output)
   var r :  real
   procedure show
      begin write( r :  5:3) end;
   procedure small
      var r :  real;
      begin r := 0.125; show end;
   begin
      r := 0.250;
      show; small;writeln;
      show;small;writeln;
   end.
```

	portee statique		portee dynamique	
Resultats:	0.250	0.250	0.250	0.125
	0.250	0.250	0.250	0.125

Passage des paramètres

1. *Appel par valeur.* Les r-valeurs des paramètres d'appel sont passés à la procédure appelée (règle en C).

   Implémentation: Les paramètres formels sont traités comme des noms locaux. Les paramètres d'appel sont évalués par l'appelant.

   Effet: Ne modifie pas les valeurs dans l'enregistrement de l'appelant sauf à travers des noms non-locaux ou par des pointeurs passés par valeur.

   Exemple:

Pascal	C

`echange(i,j:`
`integer);`
`var x :  integer;`
`begin`
`x:=a[i];a[i]:=a[j];`
`a[j]:=x`
`end`
	`swap(x,y)`
	`int *x, *y;`
	`{ int temp;`
	`temp=*x;`
	`*x=*y;`
	`*y=temp;`
	`}`
	`main()`
	`{ int a=1,b=2;`
	`swap(&a,&b);`
	`}`

2. *Appel par référence* (ou par adresse). La l-valeur du paramètre d'appel est passée à la procédure appelée (`var` en Pascal).

   Exemple:

   ```
   program reference (input, output);
      var a, b:  integer;
      procedure swap( var x,y:  integer);
         var temp :  integer;
         begin
            temp:=x;
            x:=y;
            y:=temp
         end;
      begin
         a := 1; b := 2;
         swap(a,b);
      end.
   ```

3. *Appel par copy-restore* (Fortran). Les r-valeurs sont passées à l'appel mais à la fin de l'execution, les nouvelles r-valeurs sont substituées (dans les l-valeurs des paramètres)

4. *Appel par nom* (Algol). La procédure est traitée comme une macro: le corps est substitué à

l'appel avec substitution littérale des paramètres.

Exemple: `swap(i,a[i])` affecte la valeur `i` à `a[a[i]]`. Peut se comparer avec la macro C:

```
#define swapint(a,b); {int x=a; a=b; b=x;}
```