ÉCOLE
POLYTECHNIQUE
MONTRÉAL

LOG3210
Cours 9

Environnements d'exécution (Enregistrements d'activation)

Environnements d'exécution

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Un compilateur doit implémenter plusieurs concepts abstraits des langages de programmation:
  - Portée des variables (scope)
  - Polymorphisme
  - Fonctions et paramètres
  - etc.

- Pour ce faire, les compilateurs créent et gèrent des environnements d'exécution (run-time environments) qui seront responsables de gérer l'allocation de mémoire, les appels et retours de fonctions, les interactions avec l'OS, avec les systèmes I/O, etc.

Organisation de la mémoire

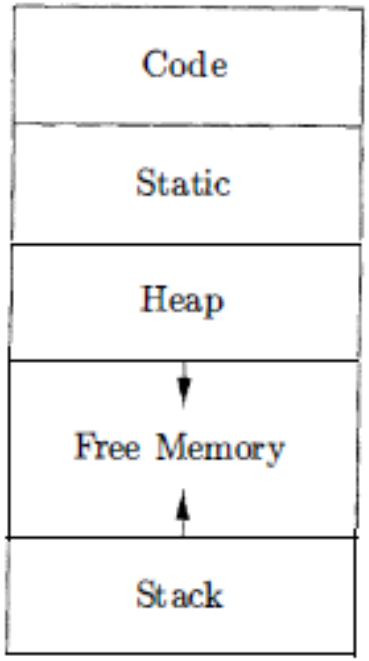
ÉCOLE POLYTECHNIQUE MONTRÉAL

- Du point de vue du compilateur, l'espace disponible pour l'exécution d'un programme est un bloc de mémoire contigüe: [0,…n], où 0 et n sont des adresses **logiques**.

- Le système d'exploitation est responsable de faire la correspondance entre les adresses **logiques** du programme et **physiques** de la machine.

Organisation de la mémoire (suite)

- D'un point de vue logique, la mémoire est subdivisée en différente sections.
  - Code: Contient le code exécutable du programme. Taille fixe, connue à la compilation.
  - Static: Constantes globales, données générées par le compilateur. Taille fixe, connue à la compilation.
  - Stack: Contient les structures qui « vivent » le temps d'une procédure (paramètres, variables locales, valeurs de retour, etc.) et qui sont détruites à la sortie. Taille variable, dynamique.
  - Heap: Contient les structures dont la « vie » ne dépend pas d'une procédure. Géré manuellement (malloc/free, new/delete) ou automatiquement (garbage collector). Taille variable, dynamique.



Arbre d'activation

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Pour bien comprendre pourquoi les appels / retours de procédures peuvent être modélisés par une pile, il est utile de considérer les arbres d'activation.

Exemple d'activations pour le programme quicksort

ÉCOLE POLYTECHNIQUE MONTRÉAL

```
int a[11];
void readArray() { /* Reads 9 integers into a[1], ..., a[9]. */
    int i;
    ...
}
int partition(int m, int n) {
    /* Picks a separator value v, and partitions a[m..n] so that
       a[m..p-1] are less than v, a[p] = v, and a[p+1..n] are
       equal to or greater than v. Returns p. */
    ...
}
void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    }
}
main() {
    readArray();
    a[0] = -9999;
    a[10] = 9999;
    quicksort(1,9);
}
```

Figure 7.2: Sketch of a quicksort program

```
enter main()
    enter readArray()
    leave readArray()
    enter quicksort(1,9)
        enter partition(1,9)
        leave partition(1,9)
        enter quicksort(1,3)
            ...
        leave quicksort(1,3)
        enter quicksort(5,9)
            ...
        leave quicksort(5,9)
    leave quicksort(1,9)
leave main()
```

Figure 7.3: Possible activations for the program of Fig. 7.2

Arbre d'activation de quicksort

ÉCOLE POLYTECHNIQUE MONTRÉAL

```
enter main()
    enter readArray()
    leave readArray()
    enter quicksort(1,9)
        enter partition(1,9)
        leave partition(1,9)
        enter quicksort(1,3)
            ...
        leave quicksort(1,3)
        enter quicksort(5,9)
            ...
        leave quicksort(5,9)
    leave quicksort(1,9)
leave main()
```

Figure 7.3: Possible activations for the program of Fig. 7.2

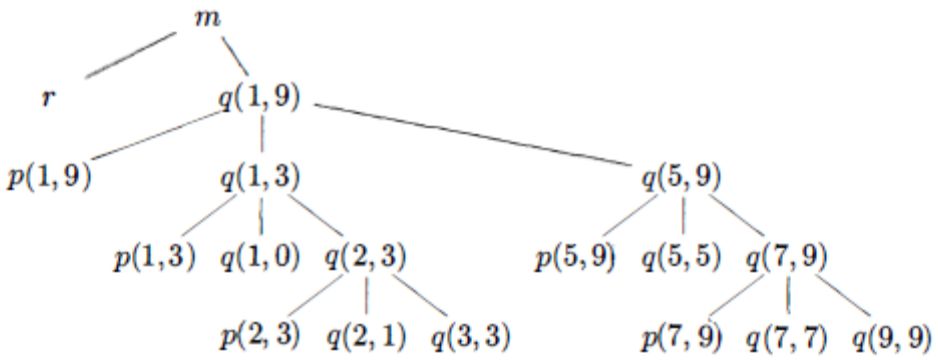


Figure 7.4: Activation tree representing calls during an execution of *quicksort*

Arbre d'activation et pile

ÉCOLE POLYTECHNIQUE MONTRÉAL

- L'utilisation d'une pile pour représenter les appels / retours de fonctions est possible à cause des relations suivantes entre l'arbre d'activation et le comportement d'un programme:
  - La séquence d'appels à procédures correspond à un traversement pré-ordre de l'arbre d'activation.
  - La séquence de retours correspond à un traversement post-ordre de l'arbre d'activation.
  - À un point donné *N* dans l'arbre, les activations valides correspondent à *N* et à tout ses ancêtres.

- Reportons-nous à l'arbre d'activation…

Enregistrements d'activation

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Lors d'un appel à fonction, un enregistrement d'activation (activation record) est généré et empilé sur la pile (stack).

- Un enregistrement d'activation correspond à un nœud dans l'arbre d'activation.

Structure d'un enregistrement d'activation

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Paramètres actuels: Les paramètres qui ont été passés lors de l'appel à fonction.
- Valeur de retour: Espace réservé pour contenir la valeur de retour (espace déterminé en fonction du type de la valeur).
- Lien de contrôle: Un pointeur vers l'enregistrement d'activation de l'appelant.

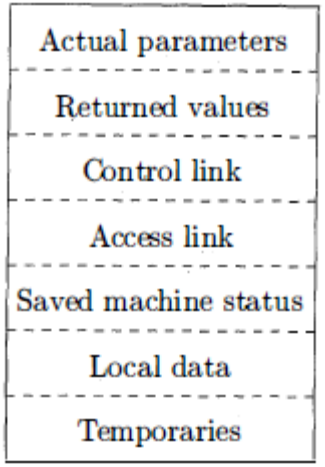


Figure 7.5: A general activation record

Structure d'un enregistrement d'activation (suite)

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Lien d'accès: Pointeurs vers des données situées dans d'autres enregistrements (utilisé pour les fonctions imbriquées).
- Statut sauvegardé: Information sur l'état de la machine juste avant l'appel (adresse de retour, valeurs de registres qui doivent être restaurées au retour, etc.)
- Données locales: Variables locales de la fonction
- Temporaires: Variables temporaires (ex. temporaires générées par l'évaluation d'une expression) qui ne peuvent être stockées dans un registre.

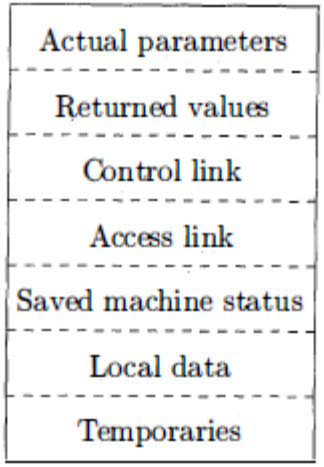
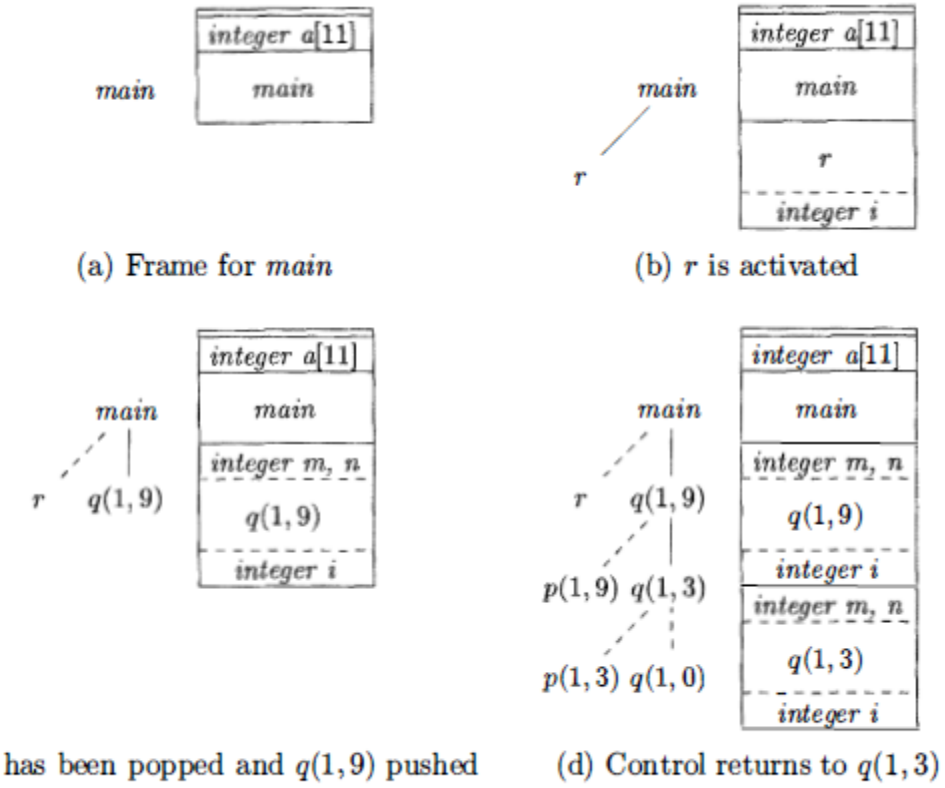


Figure 7.5: A general activation record

Pile d'enregistrements d'activation - Exemple



(a) Frame for *main*

(b) r is activated

(c) r has been popped and $q(1,9)$ pushed

(d) Control returns to $q(1,3)$

Figure 7.6: Downward-growing stack of activation records

Séquences d'appels

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Lors d'appels à fonctions, un enregistrement d'activation doit être généré et empilé et des données doivent être copiées dans certains de champs (ex. les paramètres actuels, pointeur vers l'enregistrement de l'appelant, etc.).

- Étant donné un appel à fonction d'un appelant (caller) vers un appelé (callee), certaines tâches peuvent être accomplies par l'appelant ou par l'appelé.

- Le partage des tâches peut varier d'un langage et d'un compilateur à l'autre.

- Le code associé aux mécanismes de création et gestion des enregistrements d'activation est défini comme étant la séquence d'appel (*calling sequence*).

Partage des tâches entre appelant et appelé

ÉCOLE POLYTECHNIQUE MONTRÉAL

- En général, si une fonction est appelée *n* fois, les instructions associées aux tâches de l'appelant seront générées *n* fois.

- Par contre, les instructions associées aux tâches de l'appelé seront générées 1 seule fois.

- Dans le contexte d'une séquence d'appel, il est donc généralement préférable de confier un maximum de responsabilités à l'appelé.

Design d'une séquence d'appels

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Quelques principes sont utiles pour designer une séquence d'appels:
  - Les valeurs passées de l'appelant à l'appelé sont mises au début de l'enregistrement de l'appelé afin qu'elles puissent être facilement générées par l'appelant.
    Simplifie aussi le support pour les nombres d'arguments variables (l'appelant connait le nombre d'arguments, les empile et l'appelé peut accéder aux arguments par positions relatives.)
  - Les éléments de taille fixe (lien de contrôle, lien d'accès, statut sauvegardé) sont mis au milieu de l'enregistrement. Si la structures de ces éléments est commune à tous les enregistrements, la tâche du compilateur et du débuggeur est largement simplifiée.

Design d'une séquence d'appels (suite)

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Les items dont la taille est connue de l'appelé, mais pas de l'appelant, sont placés à la fin de l'enregistrement et sont gérés par l'appelé.
  Par exemple, le nombre de variables temporaires requises par la méthode appelée pourrait ne pas être connu de l'appelant au moment où le code de l'appel est généré.

- Lorsque le contrôle est transféré de l'appelant à l'appelé, l'appelé doit savoir où se trouve le dessus de la pile, suite aux modifications faites par l'appelant. Ainsi, l'appelé peut aller chercher les valeurs des paramètres par exemple.
  Il est commun de confier à l'appelant la tâche de fixer la valeur du pointeur de dessus de la pile (*top_sp*) à l'adresse immédiatement après les éléments de taille fixe (lien de contrôle, lien d'accès, etc.)
  Ainsi, les paramètres, valeurs de retour, etc. se trouvent **avant** *top_sp* et les variables locales, temporaires, etc. se trouvent **après** *top_sp*.

Séquence d'appels (appel)

1. L'appelant évalue les paramètres actuels et les empile.
2. L'appelant stocke l'adresse de retour et l'ancien *top_sp* dans l'enregistrement de l'appelé.
3. L'appelant incrémente *top_sp* et transfert le contrôle à l'appelé.
4. L'appelé sauvegarde les valeurs de registre et autre données
5. L'appelé initialise ses données locales et s'exécute.

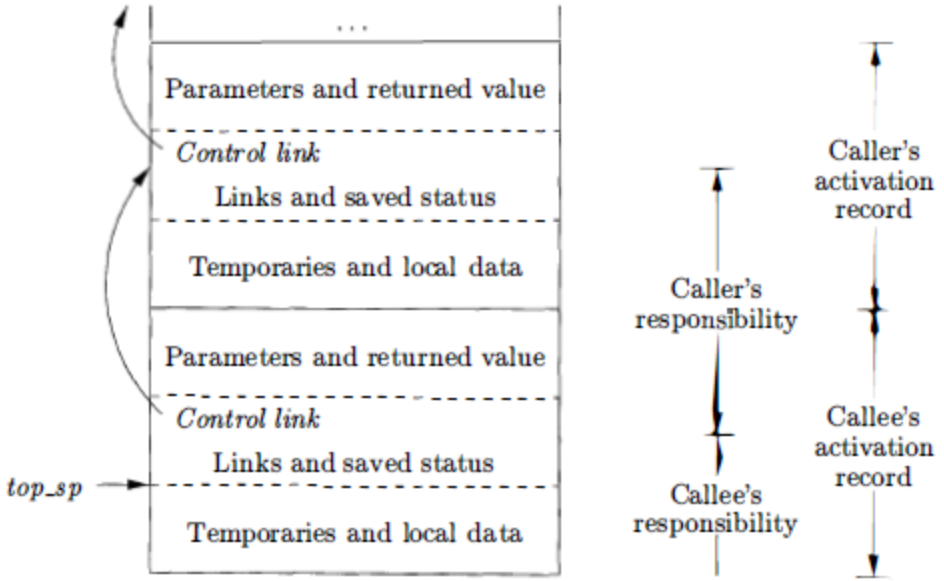


Figure 7.7: Division of tasks between caller and callee

Séquence d'appels (retour)

ÉCOLE POLYTECHNIQUE MONTRÉAL

1. L'appelé place la valeur de retour dans l'espace réservé.

2. En utilisant les informations stockées par l'appelant, l'appelé restore *top_sp* et les autres registres à leur valeur initiale.

3. L'appelé transfert le contrôle à l'adresse de l'instruction de retour, placée par l'appelant.

4. L'appelant va chercher la valeur de retour en calculant un décalage à partir de *top_sp*.

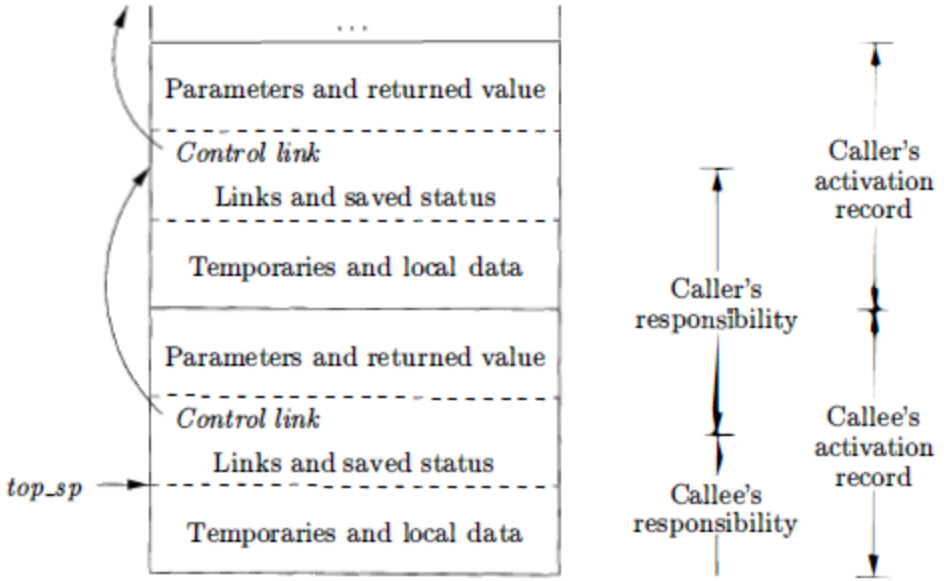


Figure 7.7: Division of tasks between caller and callee

Accès à des données non locales sur la pile

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Chaque appel à fonction génère un enregistrement d'activation sur la pile.
- L'accès aux données locales d'une fonction peut être fait en calculant un décalage à partir de *top_sp*.
- Il arrive toutefois qu'une fonction doive accéder à des données qui sont sur la pile, mais qui ne sont pas locales à la fonction.

Accès aux données non locales sans imbrication de fonctions

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Dans un langage comme le C, les variables sont locales à une seule fonction ou (XOR) globales.

- Une variable globale a une portée… globale! Elle peut donc être stockée dans un champ statique dont l'adresse est fixe, connue à la compilation et accessible de toutes les fonctions.

- Toutes autres variables sont locales et donc accessibles à partir de l'enregistrement d'activation.

Accès aux données non locales avec imbrication de fonctions

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Si l'imbrication de fonctions est permise et que les fonctions imbriquées peuvent accéder aux variables de leurs parents, il est possible que certaines données ne soient pas disponibles dans l'enregistrement d'activation d'une fonction.

```
1) fun sort(inputFile, outputFile) =
       let
2)         val a = array(11,0);
3)         fun readArray(inputFile) = ··· ;
4)               ··· a ··· ;
5)         fun exchange(i,j) =
6)               ··· a ··· ;
7)         fun quicksort(m,n) =
               let
8)                 val v = ··· ;
9)                 fun partition(y,z) =
10)                      ··· a ··· v ··· exchange ···
               in
11)                ··· a ··· v ··· partition ··· quicksort
               end
       in
12)        ··· a ··· readArray ··· quicksort ···
       end;
```

a réfère à 2

a réfère à 2

a réfère à 2, **v** réfère à 8, **exchange** réfère à 5

a réfère à 2, **v** réfère à 8, **partition** réfère à 9 **quicksort** réfère à 7.

a réfère à 2, **readArray** réfère à 3, **quicksort** réfère à 7.

Figure 7.10: A version of quicksort, in ML style, using nested functions

Liens d'accès

- Les liens d'accès doivent pointer vers l'enregistrement d'activation de la fonction *parent* (ex. le parent de *quicksort* est *sort*, le parent d'*exchange* est aussi *sort*.

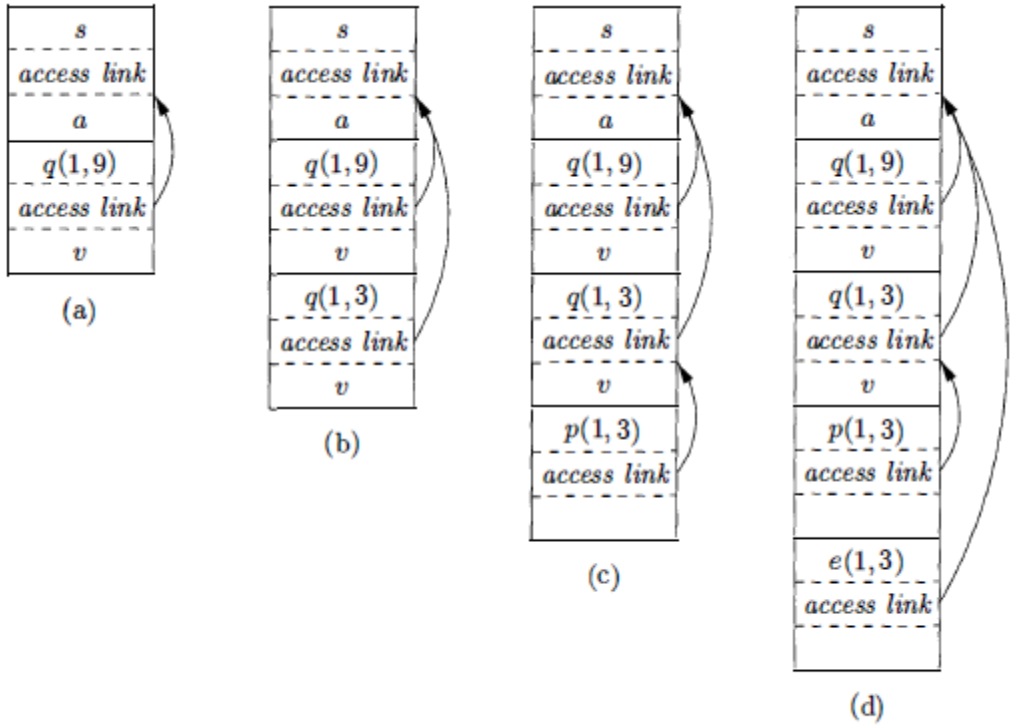


Figure 7.11: Access links for finding nonlocal data

Profondeur d'imbrication

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Afin d'être en mesure de retracer l'enregistrement d'activation qui contient les données non locales d'une fonction, nous introduisons la notion de profondeur d'imbrication.

- Toute fonction qui n'est pas imbriquée dans une autre fonction aura une profondeur d'imbrication de 1.

- Toute fonction imbriquée aura une profondeur d'imbrication de i+1 où i est la profondeur d'imbrication de son parent.

Construction des liens d'accès

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Étant donné un appel à fonction, le compilateur doit ajuster les liens d'accès afin qu'ils pointent vers les parents **immédiats** de la fonction appelée. Supposons un appel de *caller* à *callee*. Deux situations sont possibles:
  - *Callee* est à un niveau d'imbrication plus élevé que *caller* (ex. *partition* et *quicksort*). Dans ces cas, le niveau de *callee* doit être égal au niveau de *caller* + 1, sans quoi *callee* ne serait pas accessible à *caller*. Le lien d'accès pointe vers l'enregistrement d'activation de *caller*.
  - La profondeur d'imbrication $n_{callee} \leq n_{caller}$. Pour que cet appel soit valide, il doit exister une fonction *ancestor* de telle sorte que:
    - *callee* et *caller* sont imbriqués dans *ancestor*.
    - *ancestor* est le parent immédiat de *callee*.

Construction des liens d'accès (suite)

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Dans les cas où la profondeur d'imbrication $n_{callee} \leq n_{caller}$, le lien d'activation est créé selon l'algorithme suivant:

- À partir de l'enregistrement du *caller*, suivre $(n_{caller} - n_{callee} + 1)$ liens d'activation pour arriver à l'enregistrement *ancestor*.

- Faire pointer le lien d'accès du *callee* vers *ancestor*.