Chapitre III Allocation - Substitution

insérer le schéma

L'analyse syntaxico-sémantique a permis d'engendrer deux textes :

- Le dictionnaire qui contient les noms des objets manipulés par le programme

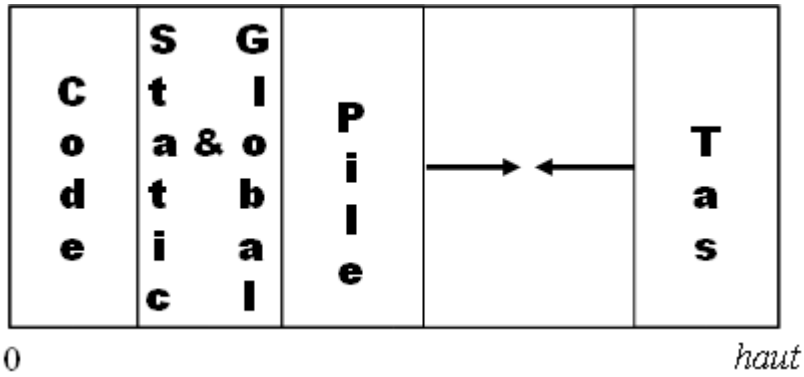
- Le texte intermédiaire correspondant aux instructions du programme

L'**allocation** a pour objet d'attribuer une place, i.e, une adresse en mémoire ou, d'une façon générale un mécanisme d'adressage, aux objets que le programme manipule. L'allocation opère donc sur la table des symboles et elle produit la TS « allouée », où à chaque objet, est associé une information d'adressage.
L'allocation peut effectuer certaines vérifications sémantiques touchant à la cohérence des déclarations des variables.

La substitution a pour objet de remplacer, dans les instructions, chaque occurrence d'un objet par son adresse, ou encore le mécanisme d'évaluation de son adresse. La substitution « travaille » sur le code intermédiaire en s'aidant du dictionnaire alloué produit par l'allocation. Elle produit un pseudo-code qui sera repris par la phase génération.

Schéma classique d'allocation mémoire

Espace logique d'adressage unique du programme (mémoire virtuelle) alloué par l'OS



- Les zones contenant le code, les données statiques et globales ont des tailles connues (par le compilateur par ex).
    - Le programme source les accède implicitement avec des étiquettes (noms).
- Le tas et la pile s'accroissent et rétrécissent durant le temps d'exécution du programme.
- meilleure utilisation de la mémoire si la pile et le tas s'accroissent chacun vers l'autre (Knuth).
- Codes & données séparés ou entrelacés
- on utilise l'espace d'adressage virtuel géré par le proc: on n'alloue pas dynamiquement cet espace.

Complément :

Tas : *heap*

Pile : *stack*

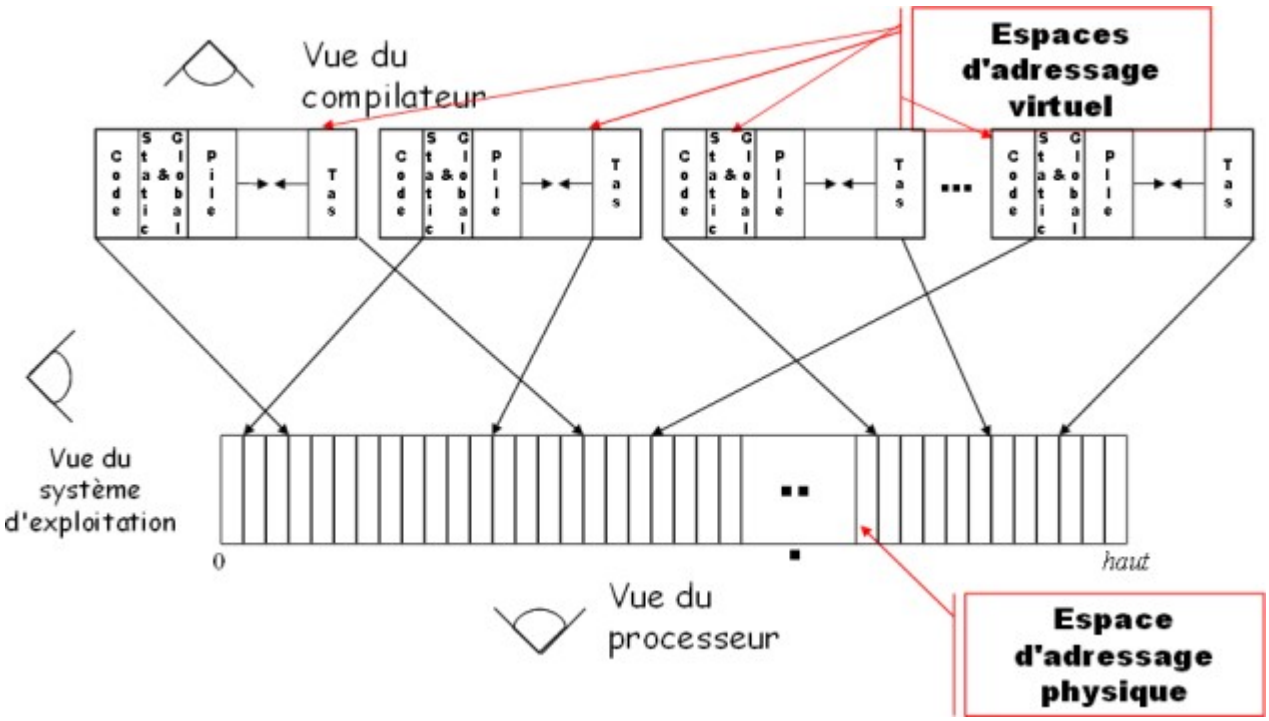
zone « Code » : contient le programme binaire (zone où l'on n'écrit pas par défaut).

zone « statique » : contient les données globales du programmes, i.e., celles qui sont en vie durant tout le temps d'exécution.

zone « tas » : utilisée pour stocker les données allouées dynamiquement.

zone « pile » : passage de paramètres et de contextes lors d'appels de fonctions, variables locales, gestion de la récursivité, etc.

Vue globale de la mémoire



Complément :

Un compilateur voit l'unique espace d'adressage virtuel du programme qu'il est en train de compiler.

Le système d'exploitation voit tous les espaces d'adressages des programmes et leur affectation à l'espace physique.

Le processeur lui ne voit que des adresses mémoire : un long tableau uni-dimensionnel.

1. Allocation

1.a. Variables simples

A une variable, on associe en général du point de vue de l'allocation, une longueur et un cadrage.
La longueur d'une variable est le nombre de cellules de mémoire servant à sa représentation. Cette longueur peut être constante ou variable.
Le cadrage d'une variable est une notion que beaucoup d'ordinateurs obligent à introduire. En effet, sur ces ordinateurs, pour qu'une variable soit directement utilisable pour un calcul, il est nécessaire qu'elle soit alignée sur une limite spécifique (de demi-mot, de mot,de double mot, ….).
Ainsi, nous avons pour chaque variable, deux informations : sa longueur et son cadrage.
Pour procéder donc à l'allocation de place mémoire, et puisqu'il s'agit de scalaires, donc d'entités indépendantes, on peut regrouper les variables en fonction de leur longueur et leur cadrage, de façon à perdre le moins d'espace mémoire possible. Les éléments de

longueur variable posent un problème quant à leur représentation. Les schémas les plus réalistes, dans les langages de programmation, précisent que la longueur d'un élément est variable, mais ne peut pas dépasser une certaine valeur (i.e , la longueur maximale). En fonction de cette longueur maximale, on se pose le problème de l'implémentation de ces éléments. Il existe deux types de méthodes :
- Les méthodes qui consistent à allouer systématiquement la longueur maximale.
- Les méthodes qui consistent à allouer uniquement la place nécessaire.

Le second type de méthode pose le problème de la gestion de la mémoire, il oblige quelques fois à utiliser le ramasse-miettes, ainsi est-il préférable de se tourner vers la 1ère à chaque fois que cela est possible.

1.b. Groupement Homogène

On entend par groupement homogène, tout groupement au niveau du langage d'objets de nature (et de longueur) identique.

Une première distinction doit être faite :
- Dimensions du groupement connues à la compilation (e.g Fortran)
- Dimensions du groupement non connues à la compilation (e.g Algol, Pascal, C, ….)

Cas des tableaux

L'adressage d'un élément d'un tableau à une seule dimension (vecteur) ne pose guère de problèmes, puisque les éléments d'un vecteur sont stockés séquentiellement en mémoire. Dans certains langages, on utilise les bornes pour le calcul de l'adresse de l'élément et le test de validité de l'indice. Le problème se complique dans le cas des tableaux multidimensionnels. Lorsque les bornes du tableau sont connues à la compilation, on peut directement bâtir le mécanisme d'adressage qui évite un grand nombre de calculs. Nous devons choisir avant tout un mode de représentation du tableau en mémoire. En effet, la mémoire est un tableau à une seule dimension.
On parlera alors de rangement ligne/ligne ou colonne /colonne.

Par exemple :

- par ligne (langage C) : les éléments d'une ligne de la matrice sont consécutifs en mémoire
- par colonne (Fortran) : les éléments d'une colonne de la matrice sont consécutifs en mémoire

Rangement ligne/ligne

Rangement colonne /colonne

Tableaux creux

2. Substitution

La substitution consiste à remplacer chaque objet par son adresse mémoire.