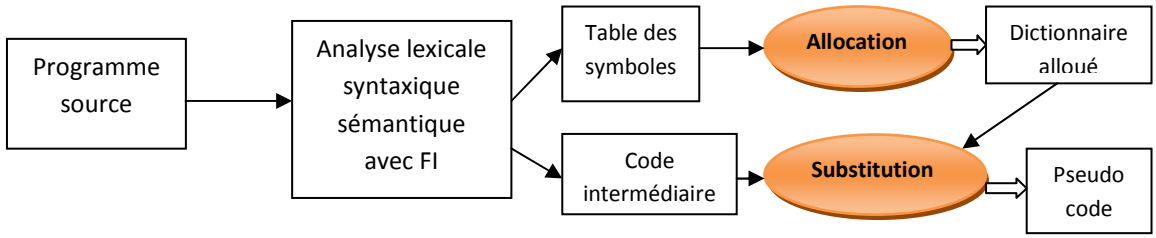
Chapitre 3 : Allocation / Substitution

L'analyse syntaxique /sémantique permet d'engendrer deux textes :

-Le dictionnaire (ou la table de symbole) 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 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 Table de symbole « «allouée », où à chaque objet est associée une information d'adressage.

L'allocation peut effectuer certaines vérifications sémantiques (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 par le mécanisme d'évaluation de son adresse.

La substitution utilise le code intermédiaire et le dictionnaire alloué produit par l'allocation. Elle produit un pseudo code qui sera repris par la phase génération de code objet.

Allocation

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 (demi-mot, mot, double mot).

Ainsi, nous avons pour chaque variable deux informations sa longueur et son cadrage. Pour procéder donc à l'allocation de mémoire et puisqu'il s'agit d'entités indépendantes, on peut regrouper les

variables en fonction de leur longueur et de 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 (c'est à dire la longueur maximale). En fonction de cette longueur maximale, on pose le problème de l'implémentation de ces éléments.

- La méthode consistant à allouer systématiquement la longueur maximale.
- Les méthodes qui consistent à allouer uniquement la place nécessaire (allocation dynamique).

Le second type de méthode pose le problème de la gestion de mémoire, il oblige quelquefois à utiliser le « ramasse-miettes », ainsi est- il préférable de se retourner vers la première méthode à chaque fois que cela est possible.

Groupement homogène

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

On peut distinguer deux cas :

- Dimension du groupement connu à la compilation (langage à allocation statique, par exemple langage FORTRAN).
- Dimension du groupement non connu à la compilation (exemple langage ALGOL).

L'adressage d'un élément d'un vecteur ne pose aucun problème puisque les éléments d'un vecteur sont stockés séquentiellement en mémoire. Dans certains langages, on utilise les bornes pour le calcul de l'élément et le test de validité de l'indice. Le problème se complique dans le cas des tableaux à plusieurs dimensions. Lorsque les bornes du tableau sont connues à la compilation, on peut directement bâtir le mécanisme d'adressage. Par contre, dans le cas où les bornes ne sont pas connues, il s'agit d'implémenter une routine qui fera l'allocation mémoire à l'exécution.

Avant tout, on doit choisir un mode de représentation de tableau en mémoire.

On parlera alors du rangement ligne /ligne ou colonne / colonne.

Rangement ligne /ligne

Exemple soit le tableau M [-2 :1,1 :4]

Donner le rangement des éléments du tableau M ligne/ligne.

Les éléments sont regroupés comme suit :

M[-2,1], M[-2,2], M[-2,3], M[-2,4]

M[-1,1], M[-1,2], M[-1,3], M[-1,4]

M[0,1], M[0,2], M[0,3], M[0,4]

M[1,1], M[1,2], M[1,3], M[1,4]

Nous remarquons que l'indice 2, qui varie le plus vite.

Calcul de l'adresse de l'élément M[i,j]

@M[i,j] := @base$_M$+**[** [(i-(-2))*(4-1+1)]+(j-1) **]***taille d'un élément

Généralisation : M[U1 :L1,U2 :L2]

@M[i1,i2] := @base$_M$+**[** [(i1-U1)*(L2-U2+1)]+(i2-U2) **]***taille d'un élément

Tableau à trois dimensions : M[U1 :L1,U2 :L2, U3 :L3]

@M[i1,i2,i3] := @base$_M$+**[** [(i1-U1)*(L2-U2+1)*(L3-U3+1)]+(i2-U2)*(L3-U3+1)+(i3-U3) **]***taille d'un élément.

Généralisation à plusieurs dimensions M[U1 :L1,U2 :L2, …, Un :Ln]

@M[i1,i2, …, in] := @base$_M$+**[** [(i1-U1)*(L2-U2+1)*…*(Ln-Un+1)]+(i2-U2)*(L3-U3+1))*…*(Ln-Un+1)+…+(in-Un) **]***taille d'un élément.

Soit : Dj=Lj-Uj+1 et D_{n+1}=1

@M[i1,i2, …, in] := @base$_M$+**[** [(i1-U1)*$\prod_{k=2}^{n} Dk$)]+(i2-U2)*$\prod_{k=3}^{n} Dk$+… +(in-Un) **]***taille d'un élément.

@M[i1,i2, …, in] := @base$_M$+**[** $\sum_{j=1}^{n}$(ij $- uj$)*$\prod_{k=j+1}^{n+1} Dk$) **]***taille d'un élément.

@M[i1,i2, …, in] :=@base$_M$+**[**$\sum_{j=1}^{n}$ ij*$\prod_{k=j+1}^{n+1} Dk$)**]***taille d'un élément - **[** $\sum_{j=1}^{n} uj$*$\prod_{k=j+1}^{n+1} Dk$) **]***taille d'un élément.

Cette formule met en évidence une partie variable (Var Part) et une partie constante (Const Part):

VarPart = **[** $\sum_{j=1}^{n}$ ij*$\prod_{k=j+1}^{n+1} Dk$) **]***taille d'un élément

ConstPart = **[** $\sum_{j=1}^{n} uj$*$\prod_{k=j+1}^{n+1} Dk$) **]***taille d'un élément

Afin de référencer un élément du tableau, nous avons besoin de vérifier que les indices appartiennent bien au domaine autorisé, il sera donc nécessaire de mémoriser les informations Ui, Li, Di et n (nombre de dimensions) dans un vecteur de renseignement.

n		
U1	L1	D1
U2	L2	D2
. . .	. . .	. . .
Un	Ln	Dn
Const Part		
@ Base		

U1
D1
. . .
Un
Dn
Const Part
@ Base

Vecteur de renseignement (cas1) *Vecteur de renseignement (cas 2)*

Dans le cas des langages à allocation statique (les bornes sont connus à la compilation), et ne sont pas variables. On peut engendrer le code correspondant au calcul de l'adresse d'un élément.

Dans le cas de l'allocation dynamique, les bornes sont inconnues, elles peuvent varier d'une exécution à une autre. Le compilateur intervient donc pour :

- Générer un code permettant le calcul des paramètres d'allocation : Ui, Li, Di et de la taille maximale.
- Insérer dans le code la macro instruction d'allocation Alloc (taille, adrbase).

La déclaration d'un tableau se traduit par : M [U1 :L1, U2 :L2, …, Un :Ln]

```
Debut
i :=1 ; (*nombre de la dimension du tableau*)
taille :=1 ;
constpart :=0 ;
Pour chaque dimension i du tableau
    Faire
        Calculer Ui ;
        Calculer Li ;
        Si (Ui>Li) Alors Ecrire ('Erreur : borne inférieure > borne supérieure)
                Sinon
                    Calculer Di
                    constpart :=constpart*Di+Ui
                    taille :=taille*Di
                    i :=i+1
                fsi
        Fait
    taille :=taille*taille d'un élément
    constpart :=-constpart
    Alloc(taille, adrbase) ;
Fin
```