Chapitre 4

Organisation des données à l'exécution

Nous nous intéressons dans ce cours à l'organisation des données, aux paramètres, au mode de leur passation et à la structure des programmes.
Il existe deux types d'organisation et d'allocation des données :
- Statique
- Dynamique (sur le tas, pile, ..)

4.1 Allocation statique

Dans l'organisation des données statique, toutes les données nécessaires à l'exécution du programme (ou fragment de programme) sont présentes en mémoire. L'espace alloué statiquement est déjà pré-réservé dans le fichier exécutable du programme, lorsque le système d'exploitation charge le programme en mémoire pour l'exécuter. L'ensemble des données est de taille constante. Cette taille est calculée lors de la compilation. Lors de l'exécution, aucune allocation statique ne peut avoir lieu.
Allouer statiquement de la mémoire pour un programme signifie :
- Prévoir l'espace mémoire nécessaire avant l'exécution du programme, en spécifiant la quantité nécessaire dans le code source.
- Réserver cet espace au moment de la compilation, dans le fichier binaire produit.
- Au chargement du programme en mémoire, juste avant l'exécution, l'espace réservé devient accessible.

L'avantage de ce type d'allocation est que la taille est connue dès la compilation, ce qui évite les coûts de l'allocation dynamique à l'exécution.

A la compilation : le compilateur calcule la taille mémoire nécessaire à la bonne exécution du programme.
A l'exécution : le système d'exploitation vérifie s'il dispose d'assez de mémoire pour exécuter le programme, dans le cas contraire, celle-ci est annulée.

4.1.1 Organisation de la mémoire

Pour exécuter un programme, il faut placer en mémoire :
- Le code cible du programme
- Les données (Allouées statiquement)
- Les données dynamiques

Allocation dans le cas du langage FORTRAN

Ce langage est destiné aux calculs scientifiques (problèmes d'analyse numériques, gros calculs, ...). Il possède les caractéristiques suivantes :
- Déclaration implicite des variables.
- Pas d'objets à structure dynamique : par exemple, les tableaux sont statiques, leurs bornes sont connues à la compilation.
- Les sous programmes et les fonctions (subroutine et function, ne sont pas récursifs, donc pas de notion de récursivité en Fortran.
- Possibilité d'attribuer plusieurs noms à une même zone mémoire : Ordre EQUIVALENCE
- Possibilité de déclarer des zones de données communes partageables entre programme-sous programmes : COMMON
- Un sous programme peut accéder, en plus de ses variables locales, aux variables déclarées dans les COMMON et aux paramètres effectifs.
- Un programme FORTRAN a la structure suivante :

Programme principal

stop
end
Sous programme1

return
end
Sous programme2

return
end

L'appel du Programme principal à un sous programme se fait à l'aide de l'instruction CALL.

Définition d'une zone de données

Une zone de données est un bloc de mémoire dans lequel on stocke les valeurs des objets. En Fortran, on connaît dès la compilation les adresses relatives de toutes les variables du programme et de la dimension de la zone de données. Il existe une zone de données pour le programme principal, pour chaque procédure et pour chaque bloc COMMON. La table des symboles doit noter pour chaque nom, la zone de données à laquelle il appartient, ainsi que que son déplacement dans la zone.
L'adresse de la zone de données active (programme ou sous programme) est stockée dans un registre appelé ACTIVAREA.

Zone de données du programme principal

Dans cette zone, on retrouve les variables du programme principal non déclarées dans un bloc COMMON, ainsi que les temporaires.

Zone de données du sous programme

Elle est organisée comme suit :

Paramètres implicites
paramètres effectifs
Variables simples, locales, tableaux temporaires

Paramètres implicites :
Ce sont des paramètres définissant le contexte à sauvegarder :
- @ de retour dans le programme appelant
- @ de la zone de données du programme principal
- Sauvegarde des contenus de certains registres

Paramètres effectifs : Ce sont des paramètres de passage par valeur, par référence, ...

Variables simples et les Temporaires

Appel de sous programme
Un appel de sous programme consiste à exécuter le sous programme en utilisant las paramètres effectifs spécifiés dans l'appel. A la fin de l'exécution du sous programme, il est nécessaire de retourner au programme appelant à l'instruction qui suit l'appel.

Call id ($< id - param - effectifs >$, se traduit par :

. Le calcul des paramètres effectifs et stockage de leur adresse (appel par référence) ou valeur (appel par valeur) dans une zone conventionnelle globale utilisée par tous les sous-programmes pour réaliser le passage des paramètres.

. Saut au début du code du sous-programme après avoir sauvegardé l'adresse de retour dans un registre.

Subroutine id ($< liste - param - formels >$), se traduit par :
. Recopie des paramètres implicites dans la zone de données (@retour, @zone de données programme appelant, ...).

. Recopie des paramètres effectifs.

Return se traduit par :

. Chargement des paramètres implicites.
. Saut à l'adresse de retour.

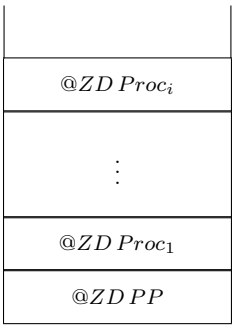
4.1.2 allocation dynamique

Dans cette organisation, les objets sont crées et supprimés en fonction des besoins du programme. De façon précise, à une unité particulière de programme (bloc ou procédure) est associé un ensemble de données. Ainsi, lorsque l'on débute l'exécution du bloc ou de la procédure, on dispose de l'ensemble des données nécessaires à l'exécution de cette unité de programme. Lorsque l'exécution du bloc ou de la procédure est achevée, on libère la place mémoire occupée par ces données devenues inutiles.

Dans ce type d'organisation des données, la zone de mémoire utilisée est gérée comme une pile, c'est le cas par exemple des langages C, C++, Algol, ... Un ensemble de cellules de mémoire utilisée est utilisé appelé Display.

Dans la pile, une zone dynamique est associée à chaque activation de la procédure. Cette zone dynamique contient les informations de longueur fixe, spécifiées dans la procédure, (exemple, variables de longueur fixe, vecteurs de renseignements, ...). Les éléments de longueur variable sont engendrés à partir du sommet courant de la pile.

A l'exécution, un certain nombre de procédures peuvent figurer dans l'état d'activation du programme : ces procédures ont été appelées, mais leur exécution n'est pas terminée, du fait qu'elles ont appelé une autre procédure. De toutes ces procédures, une seule est en cours d'exécution à un instant donné, c'est la procédure active, le niveau d'imbrication du programme principal est 0 implicitement. La procédure active peut accéder aux variables associées aux niveaux $0, 1, 2, ..., i-1$ et i. Le display actif (le display associé à la procédure active) aura alors la forme suivante :



Ainsi, une référence à une variable (de longueur fixe) associée à un niveau j sera de la forme : j,déplacement dans ce niveau.

Dans certaines réalisations, on utilise deux piles distinctes. L'une pour les displays et l'autre pour les données.

Cas du langage Algol :
Caractéristiques du langage :
- Déclaration dynamique des tableaux : les bornes peuvent être fonction d'une ou plusieurs variables du programme source. La taille du tableau varie donc en fonction des variables.
- Structure en blocs : Blocs imbriqués ou de même niveau.
Grammaire d'un bloc :
$< bloc > \rightarrow$ **begin** $< liste - decl >< liste - decl - proc >< liste - bloc >$
$< liste - inst >$**end**
$< liste - decl - proc > \rightarrow$ **Procedure** id $< liste - param >< bloc >$;

- Récursivité

La mémoire est gérée comme une pile. A chaque appel de procédure, une zone de données pour la procédure est alloué. A la fin de cette procédure, la zone est libérée.

Format d'une zone de données d'une procédure (display)

La zone est constituée d'une partie statique, dont la taille est déterminée à la compilation et d'une zone dynamique pour les tableaux.

Zone statique, elle est constituée de :

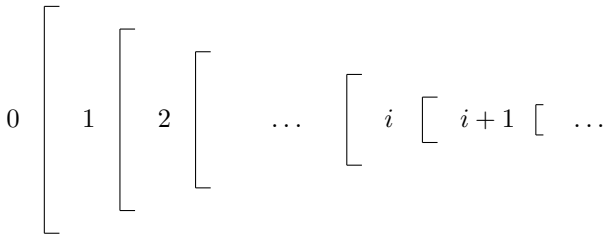
. Display : contient les adresses des zones de données auxquelles la procédure peut accéder.
. têtepile : (stacktop) : pointeur vers le dernier mot de la zone de données statique de la procédure.

. Paramètres implicites : servent pour la sauvegarde du contexte de la procédure appelante. Parmi des paramètres, on retrouve : - l'adresse de retour, le sommet de pile avant l'allocation de la zone de données de la procédure, adresse du display global, ...

. Paramètres effectifs : Zone réservée pour passer les valeurs ou les adresses des paramètres de passage.

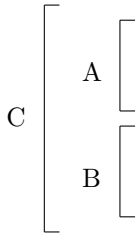
. Pour chacun des blocs de la procédure :
- Une variable têtepile, qui contiendra l'adresse du sommet de pile, lorsque ce bloc sera exécuté.
- Emplacements relatifs aux variables simples de ce bloc.
- Emplacements relatifs aux variables de travail nécessaires à ce bloc.

- Une procédure peut définir une autre.



- Une procédure de niveau i, peut accéder aux variables déclarées dans les procédures de niveaux inférieurs.

- Deux procédures déclarées dans un même bloc et qui sont de même niveau peuvent s'appeler



A peut appeler B et inversement, mais A(resp B) ne peut accéder aux variables de B (resp A).

Zone dynamique

Avant l'exécution du bloc, le stacktop est rempli. On alloue de l'espace pour les tableaux déclarés dans le bloc et on remplit les vecteurs de renseignements. L'espace est alors libéré dès que l'on sort du bloc.

Exemple

```
Procédure A(X,Y) ; Integer X, Y ;
L1 : Begin
      Real S, P ; array B[X :Y] ;
            L2 : begin Integer I,J ;
                  L3 : for i := X to Y step 1
                  do S := S+ B[I] ;
            L4 : end ;
            L5 : begin array C[1 :X] ;
                  L6 : begin Real E
```

```
                    L7 : ..... end ;
            end
      end ;
end
```

Donner les différents états de la pile aux différentes étiquettes L1 ... L7 :

Zone de données d'un Programme principal
Elle est composée de :
. têtepile
. zone pour les variables, vecteurs de renseignements et temporaires, du programme principal.
. Pour chaque bloc de programme principal,faire les mêmes opérations.

Calcul d'adresse des variables :
Pour calculer les adresses absolues des variables, on utilise une zone mémoire appelée activarea qui pointe vers le début de la zone de données active.

Dispactiv (ou activarea) identifie le display actif et la zone de données associés à la procédure active.

1. adresses des variables déclarées dans le programme principal
adresse absolue := activarea + (déplacement de la variable/au début de la zone)

2. variables déclarées dans une procédure
adresse absolue := activarea + (déplacement de la variable/début de la zône)

3. variable non locale, déclarée dans une procédure de niveau K, référencée par une procédure de niveau i :
activarea pointe vers le début de la zone de données active
adresse absolue := C(activarea+k) + (déplacement de la variable /début de la zone de données), k étant le niveau d'imbrication de la procédure.

Traitement des procédures :
. déclaration : procédure id(....)
. appel : id (...)
. fin : end ;

1. Déclaration de procédure se traduit par : . Empiler (pile des données, display de la procédure) . Empiler (pile des données, têtepile procédure) . Empiler (pile des données, param implicites) . Empiler (pile des données, param formels)
. activarea adresse de début de la zone
2. Appel . Calcul des paramètres effectifs
. Sauvegarde de l'adresse de retour
. Se brancher au début de la procédure

3. Retour . Récupération de l'adresse de la zone de données de la procédure appelante et l'adresse de retour, c'est à dire les paramètres implicites.
. Libérer la zone de données de la procédure.

. activarea adresse du programme appelant.
. Se brancher à l'adresse de retour.

Traitement d'un bloc : begin ..... end ;
- A la rencontre de "begin", le compilateur génèrera le code :
- Empiler dans la pile, le sommet de pile.
- les variables, ...

. A la rencontre de "end" :
. Libération de l'espace mémoire occupé par le bloc.

Traitement des tableaux
. Allocation dans la zone de données dynamique de l'espace pour les tableaux.
. Remplissage du vecteur de renseignement.
. Mise à jour du sommet de pile du bloc où le tableau est déclaré.

Remarque : La zone dynamique est libérée entre le sommet de pile courant jusqu'à
l'adresse du sommet de pile du bloc englobant ou sommet de pile de la procédure.

Récursivité

Etant donné que le langage algol permet la récursivité, nous ne pouvons pas allouer de l'espace pour les procédures avant de d'entamer l'exécution du programme Algol. L'allocation d'espace donc d'une procédure est effectuée à son exécution. Cet espace est géré comme une pile LIFO. A chaque appel de procédure, une zone de données est allouée, à la fin de la procédure, cette zone est libérée.