Chapitre 6 : Gestion dynamique de la mémoire à l’exécution

|  |
| --- |
| Zone code |
| Zone des données |

|TAS

v

^

| PILE

La zone des données contient :

* Objets statiques
  + Accès au début du programme et ne « bougent » plus
  + Objets dynamiques : gérés de manière automatique et crée de manière explicite, objets crées à la demande (malloc, new, …) et sont rangés dans le TAS. Ils peuvent être déplacé par le ramasse miette.
    - Attachés aux fonctions et procédures et ne changent pas de place dans l’environnement d’exécution de la fonction (ou procédure) Environnements : morceau de la pile contenant tout ce qui nécessaire à la fonction s’exécute (Paramètre, variables locales, …)

F(int m)

{

Int a,b ;

}

Void main()

{

Int a,b ;

F(3)  
 a = 2

}

PC @instruction suivante

©

SP

Base

|  |
| --- |
|  |
| b |
| a |
|  |
| @retour |
| n |
| b : 3 |
| a : 1 |

F(3) Environnement

Main()

PILE

Il faut laisser de la place pour les variables locales.

Les environnements sont séparés par un registre de base

Le compteur ordinal : adresse de retour : adresse de la prochaine instruction une fois que la fonction f est finie

Code\_main :

MOVE #1, @a

#3, @b @b : base courante (dyn) + depl (statique dans la TDS)

JSR (code\_f)

MOVE #2, @a

Code\_F :

Sauvegarde base de l’appelant (main)

= chainage dynamique

RTS

L’appelant empile les paramètres

Comme le calcul est base + depl, il faut changer de base.

Rappels :

* Environnement : alloué dynamiquement en PILE à chaque appel de fonction / procédure
* L’adresse (l’@) d’une variable n’est pas fixe (en PILE) => On utilise un mécanisme d’adressage basé
* @variable = base de l’environnement courant (dynamique) + déplacement (statique, dans TDS)
* A chaque sortie de fonction / procédure on libère la place.
* Avantages de cette gestion mémoire : Récursivité possible.
* Inconvénient : prévoir cette gestion dans le code généré.

F (int x, int y) 🡺 Paramètre formel

Main(){

A = 0

B = 0

F(A,B) 🡺 Paramètre effectifs

!!! Le déplacement est relatif à une machine particulière !!!

Il faut se brancher à l’étiquette du code de F. On met donc les paramètres effectifs à la place des paramètres formels dans la pile.

PC est un registre qui contient toujours l’adresse de la prochaine instruction à exécuter (à l’examen)

Adresse de retour : adresse de la prochaine instruction à exécuter.

Dès que l’on est dans l’environnement de l’application, il faut changer la base, elle repère un endroit dans l’environnement. On sauvegarde la base de l’appelant dans la pile

BASE

SP

|  |
| --- |
| B |
| A |
| Ch statique |
| Chainage  Dynamique |
| @ retour |
| Y 4 |
| X 5 |
|  |
|  |

: Zone de liaison

Si l’applenat à empiler les paramètres, il doit les dépiler en mettant à jour le sommet de pile (SP). C’est l’appelant qui sauvegarde l’ancienne base.

Accès aux objets (variiables) non locaux.

Programme PP : Région 1 Numéro imbrication : 1

| A : entier

| Procédure R() : Région 2 Numéro imbrication : 2

| | A, D: entier

| | P ()

| --

| Procedure P () : Région 3 Numéro imbrication : 2

| | B: entier

| | Procedure a () : Région 4 Numéro imbrication : 3

| | | c: entier

| | | c = 3, b = 2, a = 1, d = 4

| | a()

| ---

| R()

|  |
| --- |
|  |
| 4 |
| 3 |
| 1 |

PRO

Par Q, on passe par les régions 4, 3, 1 donc D n’est pas défini donc ce n’est pas correct sémantiquement.

PP ()

P ()

R ()

|  |
| --- |
|  |
|  |
| C 3 |
| N° Région 4 |
| Chainage dynamique |
| @retour |
| B |
| N° region 3 |
| Ch dynamique |
| @ Retour |
| D |
| A |
| N° Région |
| Chainage dynamique |
| @retour |
| 1 |
| N° régions 1 |
| Chainage dynamique NULL |
| @ retour NULL |

PP

En jaune : PP

C =3

MOVE = #3, @c

Base : environnement + depl

Il faut trouver @c (base de debut )

On refait une pile tenant en compte des numéro d’imbrication

PP ()

P ()

R ()

|  |
| --- |
|  |
| C 3 |
| N° Région 4 |
| Chainage statique |
| Chainage dynamique |
| @retour |
| B |
| N° region 3 |
| Chainage statique |
| Ch dynamique |
| @ Retour |
| D |
| A |
| N° Région |
| Chainage statique |
| Chainage dynamique |
| @retour |
| 1 |
| N° régions 1 |
| Chainage dynamique NULL |
| @ retour NULL |

PP

Chainage statique = contient la base de l’environnement du bloc de déclaration

🡪 Region 1

A = 1 Imbrication : 1

MOVE #1, @A

3 – 1 = 2 chainage statique à parcourir

Résumé :

* A chaque bloc est attaché un numéro d’imbrication (calculé et rangé dans la TDS)
* Retrouver la base du bloc d’une déclaration : les règles de portée garantissent que
  + Si I est déclarée dans le bloc X d’imbrication Nx
  + Si I est visible dans le bloc Y d’imbrication NY (alors Nx ≤ NY)
* => Il faut « remonter » NY – NX chainages statiques depuis le bloc courant en PILE

Comment mettre en place ce chainage sémantique ?

* Supposons que :
  + L’appelant est un bloc d’imbrication NX
  + L’appelé NY
* Le chainage statique du bloc appelé doit désigner (le dernier) le bloc d’imbrication de numéro NZ = NY – 1
* Depuis l’appelant, on trouve le bloc d’imbrication NZ = NY – 1 en parcourant NY – NZ chainages statiques
* => Il faut parcourir Nx – (NY – 1) = Nx – NY + 1 chainages statiques pour trouver le chainage statique de l’appelé

C’est l’appelant qui calcule le chainage statique.

C’est l’appelé qui met en place le chainage statique.

Tous les chainages (statiques et dynamiques) pointent sur l’octet de point faible des chainages dynamique.