Chapitre 5 – Représentation des objets en mémoire (à l’exécution)

Types de base :

* Entiers réels, caractère :
  + Dépend de la machine sur laquelle on se trouve
  + Codage sur 2,4, 8 octets en fonction du nb de bits
* L’arbre de l’objet ⬄ Son adresse[[1]](#footnote-1) d’implémentation

Dans le cas de « void » : On alloue aucun emplacement mémoire.

Les structures / unions

* Nb fixe de champs

Lors d’une une union entre 2 champs consiste à choisir un des deux champs.

Une structure réserve en mémoire, l’emplacement pour les deux champs en même temps.

La structure des tableaux  (3 types de tableaux) :

* Tableaux statiques
  + Tout est connu à la compilation
    - Dont sa taille
  + Int T[10] ;
* Tableaux dynamiques
  + Taille connue à l’exécution
    - Int T[N]  
      lire(N)
* Flexible (Algol68)
  + Peuvent changer de taille lors de l’exécution
  + On connait quand même sa dimension

Les chaines de caractères :

* Chainer 1 marqueur de fin de chaine
* Longueur + la chaine.

Implantation des tableaux (Différentes):

* Soit de manière contiguë
* Soit avec une succession d’indexations

## Implémentation des tableaux de manière contiguë (Tombe à l’examen)

X : entier

Flex T[10]

Y : entier

T[20]

|  |
| --- |
| Y |
| {10 |
| X |

### Cas tableau 1D

T = tabeau [ 1 …. S ]  d’entiers

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | TD(1) | TD(2) |  |  | TD(S) |  |  |  |
|  |  |  |  |  |  |  |  |  |

@ implémentation

@ implémentation :

* @ en mémoire à partir de laquelle on range les éléments du tableau

@(T[i]) = accès à un élément (Bi = Borne inférieur)

= @implémentation + (i – BI) + t

=@implémentation - (BI \* t) + (i \* t)

⬄ @ T[0]

@ de l’origine virtuelle Indépendant de i

### Cas des tableaux à 2 Dimensions

On suppose que l’on a un rangement ligne par ligne.

T[ -2 … 3, 2 … 5 ]



Ligne Colonne



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| T[-2,2] | T[-2,3] | T[-2,4] | T[-2,5] | T[-1,2] |  |  |  |  |  |

@T[i ,j ] = @impl + (i – BI1) + (BS2 – BI2 + ) \* t + ( j – BI2 ) \* t

Toutes les lignes Toutes les colonnes avant j

Avant la ième ligne

t : nombre d’octet nécessaire pour ranger les éléments de T[i,j]

= @impl – BI1 (BS1 – BI2 + 1 ) +t – BI2 \* t + i \* (BS2 – BI2 + 1 \* t) + j \* t

En rouge : @Origine virtuelle (@ov)

En jaune : enjambée (e1)

En Bleu : (e2)

Quand calculer ces différentes valeurs ?

* Cas du tableau statique
  + Ei
  + => Ces deux éléments sont rangés à compilation dans la TDS
    - Place nécessaire pour ranger le tableau en mémoire des éléments \* taille d’un élément
* Cas du tableau dynamique
  + On ne peut pas les calculer à la compilation mais uniquement à l’exécution
  + Comme pour le tableau statique mais cela se passe à l’exécution plutôt qu’à la compilation (Prévoir les instructions en assembleur pour les calculer) et sont rangés dans la pile (ou parfois dans le tas)
    - Nb éléments \* t + 3 \* n + 1 adresse (@OV)
      * 3\*n car (BIi, BSI, ei) (Borne Inf, Borne Sup, enjambée)
      * N : Correspond à la dimension du tableau
* Cas du tableau flexible
  + Calculé à l’exécution, comme les tableaux dynamiques mais recalculé à chaque changement de taille du tableau
  + Valeurs dans le TAS (en mémoire):
    - Nb éléments \* t + 3 \* n + 1 adresse (@OV)
      * 3\*n car (BIi, BSI, ei) (Borne Inf, Borne Sup, enjambée)
      * N : Correspond à la dimension du tableau
  + Les éléments d’un tableau flexible sont obligatoirement rangé dans le TAS

Exemple :

F()

{

Int a,b ;

Int T[BI1 , BS1, BI2 , BS2 ]

Reel c ;

}

* Statique

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
| T |  |
| B |  |
| A |  |
| . |  |
| . |  |

PILE

Réserver la place c’est incrémenter me sommet de la pile du nombre d’octets pour ranger le tableau

* Dynamique

|  |  |
| --- | --- |
|  | Les éléments de T |
| c |  |
|  |  |
| T | E1  BS1  BI1 |
| B |  |
| A |  |
| . |  |
| . |  |

La partie dynamique du tableaux (peuvent être mis dans le TAS)

Autant de fois que dimensions : partie statique du tableau dynamique

Partie statique du tableau dynamique

PILE

* Flexible

|  |  |
| --- | --- |
|  |  |
| c |  |
|  |  |
| T | E1  BS1  BI1 |
| B |  |
| A |  |
| . |  |
| . |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

Autant de fois que dimensions :

Partie statique du tableau dynamique

PILE TAS : Les éléments du tableau

Les dimensions du tableau sont quand même connues et ne varient pas au cours de l’exécution.

@ impl : Adresse où se trouve le premier élément du tableau.

## Représentation des tableaux par indexation successives

Idée :

* A chaque dimension correspond un indec qui donne accès aux dimensions suivantes :
  + L’accès à un élément se fait par une d’indirection

|  |
| --- |
|  |
| T[1, 0] |
| T[1, 1] |
| T[1,2] |
| T[1, 3] |

* + - Int T[1…4, 2…3]

|  |
| --- |
|  |
| T[1, .] |
| T[2, . ] |
|  |
|  |
|  |
|  |
|  |

En pile

|  |
| --- |
|  |
|  |
| @OV |
|  |
|  |
|  |
|  |
|  |

1

Eléments

2

T 3

4

|  |
| --- |
|  |
| T[2, 0] |
| T[2, 1] |
| T[2,2] |
| T[2, 3] |

Eléments

Accès à T[i,j] :

@ T[i,j]  = ((@ov + i) + j) où () signifie « contenu de »

@T[2,3] = ((@ov + 2) + 3)

1. En hexa [↑](#footnote-ref-1)