Les zones mémoires

Michael Mrissa

Université de Pau
et des Pays de l'Adour

Note de l'enseignant

Cours initialement créé par A. Aoun, A. Benzekri, J.-M. Bruel.
Nicolas Belloir est l'auteur originel de ce support, merci à lui.

Organisation

Définition

Un programme compilé est organisé en mémoire de la façon suivante :

1. **Un segment de données** (statique) : C'est l'emplacement mémoire de tous les objets auxquels le compilateur alloue une adresse fixe en mémoire que ce même objet conservera durant toute la durée de vie du processus.
2. **Un segment de code** : C'est l'emplacement où réside le code exécutable fourni par le compilateur.
3. **La pile, le tas** (dynamique) : C'est l'emplacement où le processus définit tous les objets dont la durée de vie est limitée dans le temps (par opposition au segment de données statique).

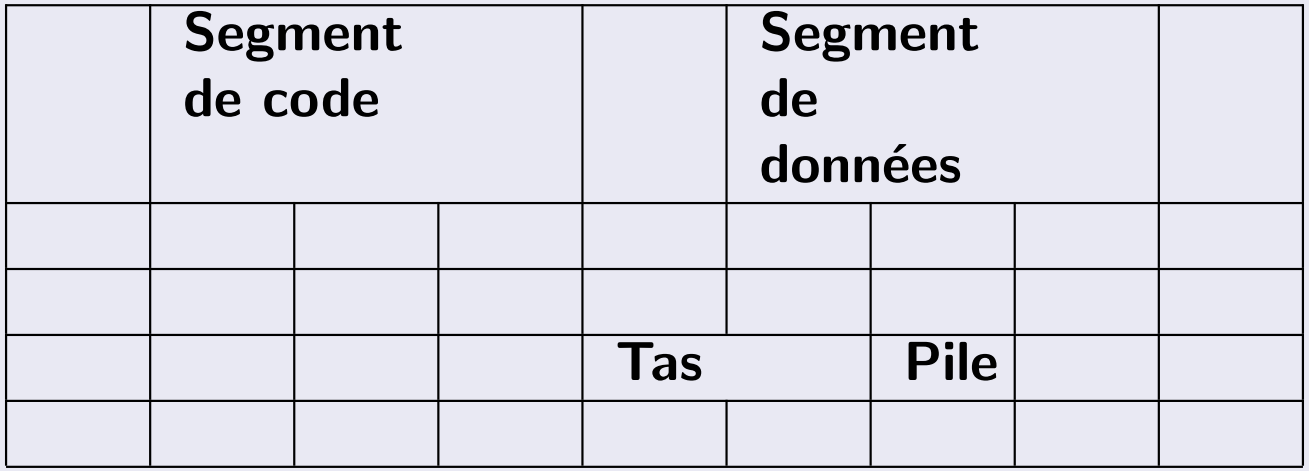
Définition

Pile : zone d'allocation des variables automatiques, des paramètres des fonctions, etc.

Tas : zone d'allocation dynamique.

Représentation schématique

Mémoire centrale



x	Page mémoire réservée pour le processus.
x	Page mémoire libre ou réservée pour un autre processus.

Variable

Une variable possède :

- Un nom,
- Un type,
- Une classe d'enregistrement. Celle-ci définit [a] :
  - le type d'espace mémoire (segment de données, pile, ...) dans lequel la variable sera allouée,
  - sa durée de vie,
  - sa zone de visibilité.

a. on ne parlera pas ici de la classe register.

Les variables locales

Visibilité

Les variables locales ne sont visibles qu'au niveau (fonction, bloc) où elles sont définies.

classe auto (par défaut)

Allocation

- La variable doit être allouée dans la pile au moment de l'appel à la fonction.
- L'emplacement mémoire réservé lors de l'allocation sera libéré lors de la sortie de l'objet où la variable est définie (fonction ou bloc).

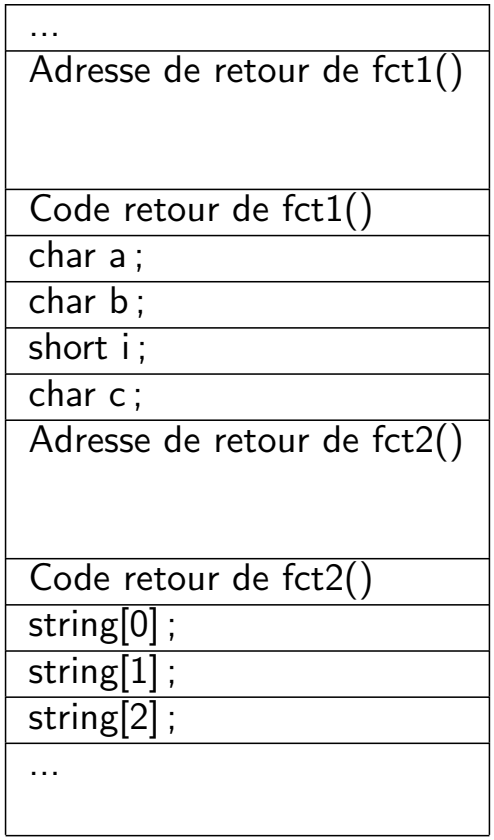
Example

```
char fct1 (char a, char b)
{
    short i;          /* variable locale short int */
    char c;           /* variable locale caractère */
    ...
    c = fct2 ();
    return ( c );       /* valeur retournée */
}

char fct2 () {
    char string [3];
    ...
    return ( string [0] );
}
```

Commentaire

Les variables déclarées dans cet exemple peuvent être représentées dans la pile comme suit :



classe static (variables rémanentes)

Définition

- Ces variables sont allouées dans le segment de données.
- Elles conservent donc leur valeur entre deux appels de la fonction dans laquelle elles sont définies.
- Leur domaine de visibilité reste malgré tout local à l'objet où elles sont définies.
- L'avantage d'un tel type de variable est de pouvoir initialiser la variable lors de sa déclaration.

Example

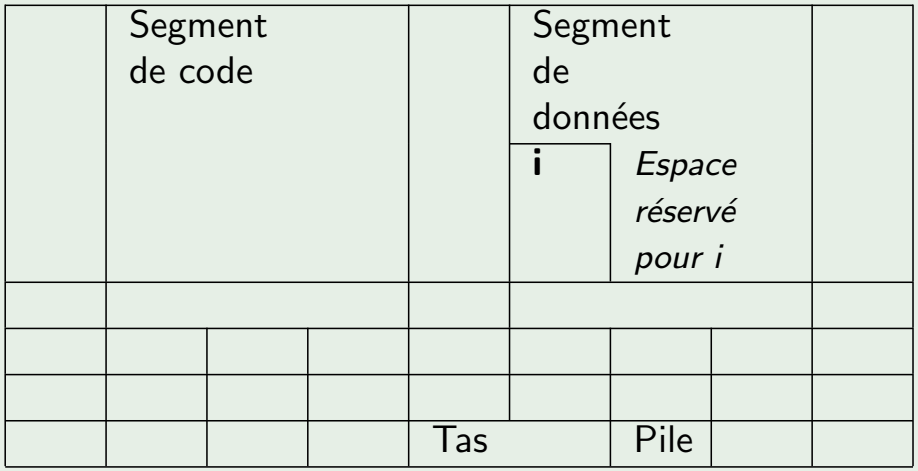
```
main () {
    int i;
    for ( i = 0; i < 7; i++)
        fct ();
}

void fct () {
    static int i = 5; /* visibilité réduite à fct */
    printf ("%d, ", i++); /* affiche : 5,6,7,8,9,10,11 */
}
```

Définition

- A l'intérieur d'un programme, il suffit de déclarer cette variable de niveau fichier, c'est à dire, en dehors de toute fonction.
- Si l'on fait référence à une variable déclarée à l'extérieur d'un programme, il faut utiliser la **classe** extern.

Example



Référence à la même variable i

Déclaration

```
#include <stdio.h>
int i ;
main()
...
```

Définition

```
#include <stdio.h>
extern int i ;
function()
— ...
```

Example

```
int i ;
main()
...
```

Résumé

Classe	**extern**	static	extern (importée)
Validité	le fichier	le fichier	le fichier
Accès	le programme	le fichier	globale importée
Allocation	à la déclaration	à la déclaration	non
Localisation	zone de données	zone de données	globale importée
Durée de vie	la tâche	la tâche	globale importée
initialisation	autorisée	autorisée	interdite

Example

```
fct(int i)
...
```

Résumé

Classe	**auto**	register
Validité	la fonction	la fonction
Accès	la fonction	la fonction
Allocation	à chaque appel	à chaque appel
Localisation	la pile	la pile
Durée de vie	la fonction	la fonction
initialisation	par l'appel	par l'appel

Example

```
main()
{
    int i ;
...
}
```

Résumé

Classe	**auto**	static	register	extern(importée)
Validité	le bloc	le bloc	le bloc	le bloc
Accès	le bloc	le bloc	le bloc	globale importée
Allocation	à l'entrée	à la déclaration	à l'entrée	non
Localisation	la pile	zone de données	la pile	globale importée
Durée de vie	le bloc	la tâche	le bloc	globale importée
Initialisation	autorisée	autorisée	autorisée	interdite