Langage C (Les Tableaux)

Les tableaux simples (à une dimension)

Définition

La définition d'un tableau nécessite trois informations :

- le type des éléments du tableau (rappelez-vous : un tableau est une suite de données de *même type*) ;
- le nom du tableau (en d'autres mots, son identificateur) ;
- la longueur du tableau (autrement dit, le nombre d'éléments qui le composent). Cette dernière doit être une expression *entière*.

```
type identificateur[longueur];
```

Comme vous le voyez, la syntaxe de la déclaration d'un tableau est similaire à celle d'une variable, la seule différence étant qu'il est nécessaire de préciser le nombre d'éléments entre crochets à la suite de l'identificateur du tableau.

Ainsi, si nous souhaitons par exemple définir un tableau contenant vingt `int`, nous devons procéder comme suit.

```
int tab[20];
```

Initialisation

Comme pour les variables, il est possible d'initialiser un tableau ou, plus précisément, tout ou une partie de ses éléments. L'initialisation se réalise de la même manière que pour les structures, c'est-à-dire à l'aide d'une liste d'initialisation, séquentielle ou sélective.

Initialisation séquentielle

Initialisation avec une longueur explicite

L'initialisation séquentielle permet de spécifier une valeur pour un ou plusieurs membres du tableau en partant du premier élément. Ainsi, l'exemple ci-dessous initialise les trois membres du tableau avec les valeurs 1, 2 et 3.

```
int tab[3] = { 1, 2, 3 };
```

Initialisation avec une longueur implicite

Lorsque vous initialisez un tableau, il vous est permis d'omettre la longueur de celui-ci, car le compilateur sera capable d'en déterminer la taille en comptant le nombre d'éléments présents dans la liste d'initialisation. Ainsi, l'exemple ci-dessous est correct et définit un tableau de trois `int` valant respectivement 1, 2 et 3.

```
int tab[] = { 1, 2, 3 };
```

Initialisation sélective

Initialisation avec une longueur explicite

Comme pour les structures, il est par ailleurs possible de désigner spécifiquement les éléments du tableau que vous souhaitez initialiser. Cette initialisation sélective est réalisée à l'aide du numéro du ou des éléments.

Faites attention ! Les éléments sont numérotés à partir de *zéro*. Nous y viendrons dans la section suivante.

L'exemple ci-dessous définit un tableau de trois `int` et initialise le troisième élément.

```
int tab[3] = { [2] = 3 };
```

Initialisation avec une longueur implicite

Dans le cas où la longueur du tableau n'est pas précisée, le compilateur déduira la taille du tableau du plus grand indice utilisé lors de l'initialisation sélective. Partant, le code ci-dessous créer un tableau de cent `int`.

```
int tab[] = { [0] = 42, [1] = 64, [99] = 100 };
```

Comme pour les structures, dans le cas où vous ne fournissez pas un nombre suffisant de valeurs, les éléments oubliés seront initialisés à zéro ou, s'il s'agit de pointeurs, seront des pointeurs nuls.

Également, il est possible de mélanger initialisations séquentielles et sélectives. Dans un tel cas, l'initialisation séquentielle reprend au dernier élément désigné par une initialisation sélective. Dès lors, le code ci-dessous définit un tableau de dix `int` et initialise le neuvième élément à 9 et le dixième à 10.

```
int tab[] = { [8] = 9, 10 };
```

Accès aux éléments d'un tableau

Le premier élément

```
#include <stdio.h>

void main()
{
    int tab[3] = { 1, 2, 3 };

    printf("Premier élément : %d\n", tab[0]);

}
```

Résultat

```
Premier élément : 1
```

Notez toutefois qu'il n'est pas possible d'affecter une valeur à une variable de type tableau (nous y viendrons bientôt). Ainsi, le code suivant est incorrect.

```
int t1[3];
int t2[3];

t1 = t2; /* Incorrect */
```

Les autres éléments

Pour accéder aux autres éléments, il va nous falloir ajouter la position de l'élément voulu à l'adresse du premier élément et ensuite utiliser l'adresse obtenue. Voici un exemple affichant la valeur de tous les éléments d'un tableau.

```
#include <stdio.h>

void main()
{
    int tab[3] = { 1, 2, 3 };

    printf("Premier élément : %d\n", tab[0]);
    printf("Deuxième élément : %d\n", tab[1]););
    printf("Troisième élément : %d\n", tab[2]););
}
```

Résultat

```
Premier élément : 1
Deuxième élément : 2
Troisième élément : 3
```

Parcours et débordement

Une des erreurs les plus fréquentes en C consiste à dépasser la taille d'un tableau, ce qui est appelé un cas de **débordement** (*overflow* en anglais). En effet, si vous tentez d'accéder à un objet qui ne fait pas partie de votre tableau, vous réalisez un accès mémoire non autorisé, ce qui provoquera un comportement indéfini. Cela arrive généralement lors d'un parcours de tableau à l'aide d'une boucle.

```c
#include <stdio.h>

void main()
{
    int tableau[5] = { 784, 5, 45, -12001, 8 };
    int somme = 0, i = 0;

    for (i = 0; i <= 5; ++i)
       {  somme = somme + tableau[i];     }

    printf("%d\n", somme);
}
```

Le code ci-dessus est volontairement erroné et tente d'accéder à un élément qui se situe au-delà du tableau. Ceci provient de l'utilisation de l'opérateur `<=` à la place de l'opérateur `<` ce qui entraîne un tour de boucle avec `i` qui est égal à 5, alors que le dernier indice du tableau doit être quatre.

N'oubliez pas : les indices d'un tableau commencent *toujours* à zéro. En conséquence, les indices valides d'un tableau de *n* éléments vont de 0 à n - 1.