C_facile : Introduction au langage C

Cas des tableaux bi-dimensionnels

Exemple

Considérons l'exemple :

```
#define MAX_L 2
#define MAX_C 3
int t[MAX_L] [MAX_C];
int i,j;
```

Cela correspond au dessin linéarisé suivant :

t						
indices	0	1	2	3	4	5
	t[0][0]	t[0][1]	t[0][2]	t[1][0]	t[1][1]	t[1][2]

Linéarisation d'un tableau à deux dimension

En effet, le tableau est linéarisé selon le schéma ligne/ligne.

Dans ce cas, t est l'identificateur d'un pointeur constant aussi. Mais ici c'est l'adresse d'une adresse.

C'est l'adresse de l'adresse de la première ligne. Et *t est donc l'adresse de la première ligne c'est en fait *(t+0).

Donc :

(t+i) est l'adresse de l'adresse de la ligne d'indice i

*(t+i) est l'adresse de la ligne i

*(t+i) + j est l'adresse du premier octet de la case j de la ligne i car on a un décalage de j entiers dans la ligne i

((t+i)+j) est donc la valeur de l'entier qui se trouve là,

et c'est la même chose que t[i][j]

donc, si vous avez bien compris le cours sur les tableaux :

t[1][2] est la même chose que :

*(t[1]+2)

((t + 1) + 2)

*((int *)t+5)

Qui sont des façons d'accéder à la case qui se trouve en ligne 1 colonne 2.

Explications particulière pour la notation « *((int *) t+5) » : il y a 3 colonnes et ((1*3)+2)=5 est bien le nombre d'entiers depuis le début du tableau qui correspond à la case d'indices (1,2) avec la linéarisation.

Mais il faut un « cast » car « t » est vu comme « un pointeur de pointeur vers entier» : il faut que le compilateur le "voit" comme un « pointeur vers entier » pour ne pas avoir de « warning » à la compilation.

Si on écrit directement l'expression (t+5) pour calculer l'adresse d'un entier (pointeur vers entier), le compilateur va l'interpréter comme l'accès à la ligne d'indice 5, ce qui peut être gênant.

Attention

Quand on écrit dans le programme :

```
&(t[i][j])
```

la machine calcule en fait l'expression mathématique suivante pour avoir l'adresse :

```
t + ( i x MAX_C + j) x Taille
```

avec Taille = sizeof(int)

C'est-à-dire la taille en nombre d'octets d'une case du tableau.

Et forcément quand on écrit :

```
t[i][j] = 0;
```

Le compilateur effectue quelque chose comme :

```
*(t + (i x MAX_C + j) x Taille) = 0
```

Un tableau à deux dimensions peut donc être vu comme un tableau de tableau à une dimension.

Considérons la déclaration suivante :

```
float machin[2][4] = {{10.0,9.0,8.0,7.0},
{51.0,52.0,53.0,54.0}};
```

Le compilateur le « voit » comme dans la figure suivante :

pointeur de ligne :

```
machin+0 -> 10.0 9.0 8.0 7.0
machin+1 -> 51.0 52.0 53.0 54.0
```

La valeur « machin[0] » est une valeur qui est un pointeur vers le premier octer de la ligne 0, ce qui correspond à *(machin +0).

Ce qu'il faut retenir :

- le compilateur à absolument besoin de connaître le nombre de colonnes car il a besoin de la valeur MAX_C pour effectuer ces calculs d'adresses
- l'usage des pointeurs pour accéder aux tableaux à deux dimensions est compliqué.

Conseil
Usage des tableaux multidimensionnels

N'utilisez pas les pointeurs avec des tableaux à 2 dimensions

Utilisez les notations

- tab[i][j] pour les opérations manipulant la valeur de la case (i,j)
- &(tab[i][j]) pour les opérations qui ont besoin de l'adresse de la case comme le « scanf » par exemple.

En effet :

scanf("%d",&t[i][j]); sera toujours plus clair que

scanf(%d",&*((int *)*(t+i)+j));

Les tableaux à « n » dimension (n>2) correspond à un tableau de tableaux de dimension n-1.

Nous ne présenterons pas en détail les tableaux à plus de 2 dimensions, sachez seulement que :

```
#define MAX_L 2
#define MAX_C 3
#define MAX_P 7
int t_3[MAX_L] [MAX_C] [MAX_P];
int i,j,k;
```

Déclare un tableau à 3 dimensions, et que :

```
t_3[i][j][k]= 7;
```

pour mettre la valeur 7 dans la case d'indices (i,j,k) ainsi que :

```
scanf("%d",&(t_3[i][j][k]));
```

pour saisir la case d'indices (i,j,k), fonctionnent parfaitement.

Et « t_3 » est l'identificateur d'une constante qui est vue comme une adresse d'adresse d'adresse d'entier par le compilateur ...

Exemple
Pour terminer

Comme nous l'avons vu il y a un lien étroit entre pointeurs et indices de tableau.

Considérons l'exemple suivant :

```
#include <stdio.h>
#define MAX 30
int main()
{
    float t_reels[MAX];
     int i,j;
     float * p1, *p2;
     p1 = &t_reels[3];
     p2 = &t_reels[7];
     printf("\n7-3 = %d",7-3);
```

```
     printf("\n(p2-p1) = %d ",(p2-p1));
     printf("\n(p2-p1) avec cast void = %d\n",(void *)p2-
(void *)p1);
     printf("\n\n");
}
```

Nous aurons à l'exécution l'affichage :

```
7-3 = 4
(p2-p1) = 4 (p2-p1) avec cast void = 16
```

En effet, "p1" et "p2" sont deux variables de type "pointeur vers flottant".

Elles sont initialisées avec deux adresses de flottant qui correspondent aux cases 3 et 7.

L'expression "(p2-p1)" effectue la soustraction de deux pointeurs.

Le compilateur sait que ce sont deux pointeurs vers des flottants. Il calcule donc le nombre de flottant qui séparent les deux adresses en utilisant l'information sur la taille d'un flottant.

Si l'on transforme les "pointeurs sur float" en "pointeur sur void" avec des cast, la différence donne 16 car dans ce cas on soustrait deux adresses d'octets. Sur cette machine, nous en déduisons qu'un flottant est codé sur 4 octets.