Chapitre 7 (suite) : Pointeurs et tableaux

En C, il existe une relation très étroite entre tableaux et pointeurs. Ainsi, chaque opération avec des indices de tableaux peut aussi être exprimée à l'aide de pointeurs. Le nom d'un tableau représente l'adresse de son premier élément. En d'autre termes:

&tableau[0] et tableau sont une seule et même adresse.

En simplifiant, nous pouvons retenir que le nom d'un tableau est un pointeur sur le premier élément du tableau.

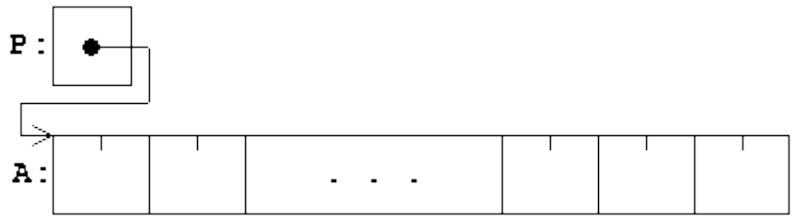
Exemple

En déclarant un tableau A de type int et un pointeur P sur int,

```
int A[10];
int *P;
```

l'instruction:

P = A; est équivalente à P = &A[0];



Ainsi, après l'instruction,

```
P = A;
```

le pointeur P pointe sur A[0], et

**(P+1) désigne le contenu de A[1]*

**(P+2) désigne le contenu de A[2]*

... ...

**(P+i) désigne le contenu de A[i]*

Si on travaille avec des pointeurs, les erreurs les plus perfides sont causées par des pointeurs malplacés et des adresses mal calculées. En C, le compilateur

peut calculer automatiquement l'adresse de l'élément P+i en ajoutant à P la grandeur d'une composante multipliée par i. Ceci est possible, parce que:

- chaque pointeur est limité à un seul type de données, et

- le compilateur connaît le nombre d'octets des différents types.

Exemple

Soit A un tableau contenant des éléments du type float et P un pointeur sur float:

```
float A[20], X;
float *P;
```

Après les instructions,

```
P = A;
X = *(P+9);
```

*X contient la valeur du 10-ième élément de A, (c.-à-d. celle de A[9]). Une donnée du type float ayant besoin de 4 octets, le compilateur obtient l'adresse P+9 en ajoutant 9 * 4 = 36 octets à l'adresse dans P.*

Rassemblons les constatations ci dessus :

Comme A représente l'adresse de A[0],

**(A+1) désigne le contenu de A[1]*

**(A+2) désigne le contenu de A[2]*

...

**(A+i) désigne le contenu de A[i]*

Attention !

Il existe toujours une différence essentielle entre un pointeur et le nom d'un tableau:

- Un pointeur est une variable, donc des opérations comme P = A ou P++ sont permises.

- Le nom d'un tableau est une constante, donc des opérations comme A = P ou A++ sont impossibles.

I. Allocation dynamique

Problème

Souvent, nous devons travailler avec des données dont nous ne pouvons pas prévoir le nombre et la grandeur lors de la programmation. Ce serait alors un gaspillage de réserver toujours l'espace maximal prévisible. Il nous faut donc un moyen de gérer la mémoire lors de l'exécution du programme.

Nous parlons dans ce cas de l'allocation dynamique de la mémoire.

II. La fonction malloc() et l'opérateur sizeof()

1. ***La fonction malloc** de la bibliothèque <stdlib> nous aide à localiser et à réserver de la mémoire au cours d'un programme.*

Si nous voulons réserver de la mémoire pour des données d'un type dont la grandeur varie d'une machine à l'autre, nous avons besoin de la grandeur effective d'une donnée de ce type. L'opérateur sizeof nous aide alors à préserver la portabilité du programme.

2. L'opérateur sizeof

sizeof <var>

fournit la grandeur de la variable <var>

sizeof <const>

fournit la grandeur de la constante <const>

sizeof (<type>)

fournit la grandeur pour un objet du type <type>

Exemple

Nous voulons réserver de la mémoire pour X valeurs du type int; la valeur de X est lue au clavier:

```
int X;

int *PNum;

printf("Introduire le nombre de valeurs :");

scanf("%d", &X);

PNum = malloc(X*sizeof(int));
```