

Série de TPI : Les pointeurs

Les pointeurs

Un pointeur est une variable qui permet de stocker l'adresse d'une autre variable. C'est à dire l'emplacement de cette dernière au niveau de la mémoire.

La déclaration

La syntaxe pour déclarer un pointeur est la suivante.

```
type *nom_du_pointeur;
```

Exemple : `int *ptr;`


L'astérisque peut être entourée d'espaces et placée n'importe où entre le type et l'identificateur. Ainsi, les trois définitions suivantes sont identiques.

```
int *ptr;
int * ptr;
int* ptr;
```

Notez bien qu'un pointeur est toujours typé. Autrement dit, vous aurez toujours un pointeur sur (ou vers) un objet d'un certain type (int, double, char, etc.).

Opérations sur les pointeurs

Un pointeur, comme une variable, ne possède pas de valeur par défaut, il est donc important de l'initialiser pour éviter d'éventuels problèmes. Pour ce faire, il est nécessaire de recourir à l'opérateur d'adressage (ou de référencement) : "&" qui permet d'obtenir l'adresse d'un objet. Ce dernier se place devant l'objet dont l'adresse souhaite être obtenue. Par exemple comme ceci.

<pre>int a = 10; int *p; // p est un pointeur sur un entier p = &a; *p=20; // *p : le contenu de la variable pointé par p</pre>		<pre>int a = 10; int *p = &a; *p=20;</pre>
-------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------	------------------------------------------------

*Faites bien attention à ne pas mélanger différents types de pointeurs ! Un pointeur sur **int** n'est pas le même qu'un pointeur sur **long** ou qu'un pointeur sur **double**. De même, n'affectez l'adresse d'un objet qu'à un pointeur du même type.*

```
int a;
double b;
int *p = &b; /* Faux */
int *q = &a; /* Correct */
double *r = p; /* Faux */
```

Allocation d'espace mémoire

La réservation d'espace d'un pointeur P sur un type quelconque se fait à travers la fonction "malloc(sizeof(int))", cette fonction permet de réserver un espace mémoire de même taille que le type de la variable pointée.

```
int *p;
p = malloc(sizeof(int));
```

```
p = malloc(sizeof(*p));
```


Libération d'espace mémoire

La Libération d'espace occupé par un pointeur P sur un type quelconque se fait à travers la fonction "Free(p)", cette fonction permet de récupérer (libérer) un espace mémoire déjà alloué. Cette opération ne renvoie rien, la mémoire pointée par P est restituée.

```
int *p;
p = malloc(sizeof(*p));
Free(p);
```

Tableaux et pointeurs

Tableaux et pointeurs étant de même type.

int A[5];	A[0] ⇔ *A	for(i=0;i<5;i++)		for(i=0;i<5;i++)
&A[0] ⇔ A	A[1] ⇔ *(A+1)	scanf("%d",&A[i]);		scanf("%d",A+i);
&A[1] ⇔ A+1	A[i] ⇔ *(A+i)			
&A[i] ⇔ A+i				

Structures et pointeurs

Les seules opérations sont :

- accès à un champ
- récupérer l'adresse de la structure (avec &)

```
struct date {
    int jour ;
    int mois ;
    int annee ;
    char nom_mois[9];
};
```

On peut déclarer des variables de type structure mais aussi des pointeurs pointant sur des structures.

struct date x,y, *pt; //pt est un pointeur pointant vers une structure de type date

Pour accéder par exemple au champ de la variable pointé par pt on écrit:

(*pt).jour

On peut aussi écrire :

pt->jour //se lit « pt flèche jour »

Exercice 1 — Ecrire un programme C qui utilise la notion de pointeur pour lire deux entiers et calculer leur somme.

Exercice 2 — Ecrire un programme C qui remplit un tableau d'entiers et calcule la somme de ses éléments en utilisant un pointeur pour son parcours.

Exercice 3 — Ecrire un programme C qui définit une structure permettant de stocker le nom, le prénom et l'âge d'une personne. Lit ensuite ces informations pour deux personnes et affiche le nom complet de la moins âgée d'entre elles en utilisant une seule fonction *printf* pour l'affichage du résultat.

Exercice 4 — Ecrire un programme C qui réserve l'espace mémoire à un tableau d'entiers de taille entrée par l'utilisateur, le lit puis l'affiche.