CHAPITRE 5 : LES POINTEURS

I. Introduction :

On dit souvent qu'on perd toute la puissance et la flexibilité du C si on n'utilise pas ou si on ne maîtrise pas les pointeurs. Le cas des pointeurs est souvent considéré comme délicat lors de l'apprentissage du C. En fait, ce n'est pas si compliqué que ça.

Un pointeur est une variable qui contient l'adresse en mémoire d'une autre variable. On peut avoir un pointeur sur n'importe quel type de variable. Pour récupérer l'adresse d'une variable, on la précède de l'opérateur **&**. Pour obtenir la valeur contenue par une zone mémoire pointée par un pointeur, on préfixe celui-ci par *****. Lorsqu'on déclare un pointeur, on doit obligatoirement préciser le type de la donnée pointée pour permettre au pointeur de connaître le nombre de blocs mémoire que prend la variable qu'il pointe. Un bloc représente un octet.

Quel est l'intérêt des pointeurs ? Principalement leur grande efficacité et flexibilité par rapport aux variables standards. Grâce aux pointeurs, on va pouvoir économiser des déplacements et copies de mémoire inutiles en précisant uniquement où se trouve la donnée.

II. Définition :

Un pointeur est une variable spéciale qui contient l'adresse mémoire d'une autre variable. Il permet d'accéder à cette autre variable indirectement.

III. Déclaration d'un pointeur :

La déclaration d'une variable de type pointeur se fait en utilisant l'opérateur ***** et en indiquant le type de la variable à laquelle correspond le pointeur.

```
int * pointeur_sur_entier;
```

(Où **int** désigne le type (pointé) et **pointeur_sur_entier** le nom du pointeur.
(*) est l'opérateur d'indirection (indique que la variable " **pointeur_sur_entier** " est un pointeur).

Exemple :

int *p ; p est un pointeur pointant sur un objet de type int

char *c ; c est un pointeur pointant sur un objet de type char

IV. Les opérateurs de base & et * :

Lors de sa déclaration, le pointeur ne contient encore aucune adresse. Il faut explicitement lui en attribuer une, à l'aide de l'opérateur **&** qui permet d'obtenir l'adresse d'une variable.

```
int a=5;
int * pointeur_sur_entier; //déclaration du pointeur
pointeur_sur_entier = &a; //le fait pointer sur l'adresse de entier
```

Pour accéder à la valeur de la variable désignée par le pointeur, on utilise l'opérateur *****. Ainsi, si on reprend l'exemple précédent, ***pointeur_sur_entier** vaut 5 ;

Exemple :

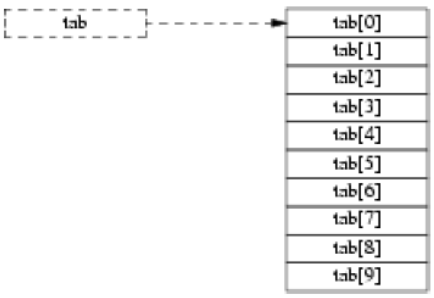
```
int X=4,Y ;
int *p,*q ;
p=&X ;  /* p contient l'adresse de X*/
printf(''%d'',*p) ;  /* affiche le contenu de la variable pointée par le pointeur p (c.-à-d. la valeur  de x).*/
Y=*p-1 ; /* Y vaut 3  */
*p+=1 ; /* incrémente  X de 1*/
(*p)++ ; /* incrémente aussi de 1 la variable pointée par p , X vaut 6*/

/* N.B : les parenthèses sont importantes. Car *p++ incrémente le pointeur p (l'adresse) et non pas la valeur de X. */
```

V. Pointeurs et tableaux :

Les notions de tableaux et de pointeurs sont étroitement liées, car l'identificateur (le nom) d'un tableau, lorsqu'il est employé seul (sans indices à sa suite), est considéré comme un pointeur (**constant**) sur le début du tableau.
Supposons, par exemple, que l'on effectue la déclaration suivante :

int tab[10] ;



- La notation **tab** est alors totalement équivalente à **&tab[0].**
- L'identificateur **tab** est considéré comme étant de type pointeur sur le type correspondant aux éléments du tableau, c'est-à-dire, ici, **int***. Ainsi, voici quelques exemples de notations équivalentes :

tab+1	**&tab[1]**
tab+i	**&tab[i]**
tab[i]	*** (tab+i)**

Pour illustrer ces nouvelles possibilités de notation, voici plusieurs façons de placer la valeur 1 dans chacun des 10 éléments de notre tableau **tab** :

```
int i ;
for (i=0 ; i<10 ; i++)
* (tab+i) = 1 ;
```

```
int i ;
int * p :
for (p=tab, i=0 ; i<10 ; i++, p++)
* p = 1 ;
```

V-1. Arithmétique des pointeurs :

Comme les pointeurs jouent un rôle si important, le langage C soutient une série d'opérations arithmétiques sur les pointeurs que l'on ne rencontre en général que dans les langages machines.

- Affectation par un pointeur sur le même type :

  Soient P1 et P2 deux pointeurs sur le même type de données, alors l'instruction **P1 = P2**; fait pointer P1 sur le même objet que P2

- Addition et soustraction d'un nombre entier ;

  Si P pointe sur l'élément tab[i] d'un tableau, alors

  P+n pointe sur tab[i+n]

  P-n pointe sur tab[i-n]

- Incrémentation et décrémentation d'un pointeur :

  Si P pointe sur l'élément tab[i] d'un tableau, alors après l'instruction

  P++; P pointe sur tab[i+1]

  P+=n; P pointe sur tab[i+n]

  P--; P pointe sur tab[i-1]

  P-=n; P pointe sur tab[i-n]

- Domaine des opérations :

L'addition, la soustraction, l'incrémentation et la décrémentation sur les pointeurs sont seulement définies à l'intérieur d'un tableau. Si l'adresse formée par le pointeur et l'indice sort du domaine du tableau, alors le résultat n'est pas défini.

Seule exception: Il est permis de 'pointer' sur le premier octet derrière un tableau (à condition que cet octet se trouve dans le même segment de mémoire que le tableau). Cette règle, introduite avec le standard ANSI-C, légalise la définition de boucles qui incrémentent le pointeur *avant* l'évaluation de la condition d'arrêt.

- Soustraction de deux pointeurs :

Soient P1 et P2 deux pointeurs qui pointent dans le même tableau:
P1-P2 fournit le nombre de composantes comprises entre P1 et P2.
Le résultat de la soustraction P1-P2 est
- négatif, si P1 précède P2
- zéro, si P1 = P2
- positif, si P2 précède P1
- indéfini, si P1 et P2 ne pointent pas dans le même tableau

Plus généralement, la soustraction de deux pointeurs qui pointent dans le même tableau est équivalente à la soustraction des indices correspondants.

- Comparaison de deux pointeurs :

On peut comparer deux pointeurs par <, >, <=, >=, ==, !=.
La comparaison de deux pointeurs qui pointent dans le même tableau est équivalente à la comparaison des indices correspondants. (Si les pointeurs ne pointent pas dans le même tableau, alors le résultat est donné par leurs positions relatives dans la mémoire).

Notes:

- * p++ est équivalent à *(p++) car les opérateurs unaires ' ++' et ' * ' sont évalués de droite à gauche.
- *(p + n); (nécessite les parenthèses car * est prioritaire sur +) incrémente (ou décrémente) l'adresse de ' n ' objets.