Les pointeurs en C

I. Définition : Pointeur

*Un **pointeur** est une variable spéciale qui peut contenir l'**adresse** d'une autre variable.*
En C, chaque pointeur est limité à un type de données. Il peut contenir l'adresse d'une variable simple de ce type ou l'adresse d'une composante d'un tableau de ce type.
Si un pointeur P contient l'adresse d'une variable A, on dit que
'P pointe sur A'.

Remarque

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

* Un ***pointeur*** est une variable qui peut 'pointer' sur différentes adresses.

* Le ***nom d'une variable*** reste toujours lié à la même adresse.

II. Les opérateurs de base

Lors du travail avec des pointeurs, nous avons besoin

- d'un opérateur 'adresse de' : **&** pour obtenir l'adresse d'une variable.
- d'un opérateur 'contenu de' : ***** pour accéder au contenu d'une adresse.
- d'une syntaxe de déclaration pour pouvoir déclarer un pointeur.

A. L'opérateur 'adresse de' : &

`&<NomVariable> fournit l'adresse de la variable <NomVariable>`

L'opérateur **&** nous est déjà familier par la fonction **scanf,** qui a besoin de l'adresse de ses arguments pour pouvoir leur attribuer de nouvelles valeurs.

Exemple

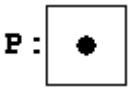
```
int N;
printf("Entrez un nombre entier : ");
scanf("%d", &N);
```

Attention !

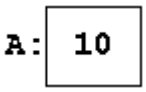
L'opérateur **&** peut seulement être appliqué à des objets qui se trouvent dans la mémoire interne, c.-à-d. à des variables et des tableaux. Il ne peut pas être appliqué à des constantes ou des expressions.

Représentation schématique

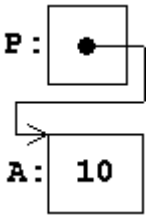
Soit P un pointeur non initialisé



et A une variable (du même type) contenant la valeur 10 :



Alors l'instruction `P = &A;` affecte l'adresse de la variable A à la variable P. Dans notre représentation schématique, nous pouvons illustrer le fait que 'P pointe sur A' par une flèche :

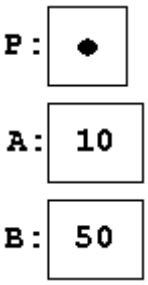


B. L'opérateur 'contenu de' : *

`*<NomPointeur> désigne le contenu de l'adresse référencée par le pointeur <NomPointeur>`

Exemple

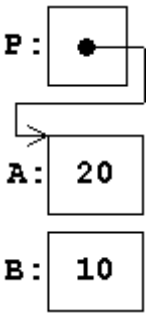
Soit A une variable contenant la valeur 10, B une variable contenant la valeur 50 et P un pointeur non initialisé :



Après les instructions,

```
P = &A;
B = *P;
*P = 20;
```

- P pointe sur A,
- le contenu de A (référencé par *P) est affecté à B, et
- le contenu de A (référencé par *P) est mis à 20.



C. Déclaration d'un pointeur

<Type> *<NomPointeur>
déclare un pointeur <NomPointeur> qui peut recevoir des adresses de variables du type <Type>

Une déclaration comme `int *PNUM;` peut être interprétée comme suit:

*"***PNUM** est du type **int**"*
ou
*"PNUM est un pointeur sur **int**"*
ou
*"PNUM peut contenir l'adresse d'une variable du type **int**"*

Exemple

Le programme complet effectuant les transformations de l'exemple ci-dessus peut se présenter comme suit :

```c
main()
{
    /* déclarations */
    short A = 10;
    short B = 50;
    short *P;
    /* traitement */
    P = &A;
    B = *P;
    *P = 20;
    return 0;
}
```

```c
main()
{
    /* déclarations */
    short A, B, *P;
    /* traitement */
    A = 10;
    B = 50;
    P = &A;
    B = *P;
    *P = 20;
    return 0;
}
```

Remarque

Lors de la déclaration d'un pointeur en C, ce pointeur est lié explicitement à un type de données. Ainsi, la variable PNUM déclarée comme pointeur sur **int** ne peut pas recevoir l'adresse d'une variable d'un autre type que **int**.

Nous allons voir que la limitation d'un pointeur à un type de variables n'élimine pas seulement un grand nombre de sources d'erreurs très désagréables, mais permet une série d'opérations très pratiques sur les pointeurs.

III. Les opérations élémentaires sur pointeurs

En travaillant avec des pointeurs, nous devons observer les règles suivantes :

Priorité de * et &

- Les opérateurs ***** et **&** ont la même priorité que les autres opérateurs unaires (la négation **!**, l'incrémentation **++**, la décrémentation **--**). Dans une même expression, les opérateurs unaires *, &, !, ++, -- sont évalués de droite à gauche.
- Si un pointeur P pointe sur une variable X, alors *P peut être utilisé partout où on peut écrire X.

Exemple

Après l'instruction **P = &X;** les expressions suivantes, sont équivalentes:

```
Y = *P+1    ⇔ Y = X+1
*P = *P+10  ⇔ X = X+10
*P += 2     ⇔ X += 2
++*P        ⇔ ++X
(*P)++      ⇔ X++
```

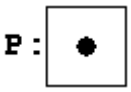
Dans le dernier cas, les parenthèses sont nécessaires :

Comme les opérateurs unaires * et ++ sont évalués *de droite à gauche*, sans les parenthèses le *pointeur* P serait incrémenté, *non pas l'objet* sur lequel P pointe.

On peut uniquement affecter des adresses à un pointeur.

Le pointeur NUL

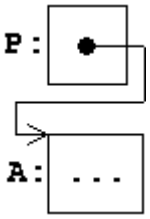
Seule exception : La valeur numérique 0 (zéro) est utilisée pour indiquer qu'un pointeur ne pointe 'nulle part'.




```
int *P;
P = 0;
```

Finalement, les pointeurs sont aussi des variables et peuvent être utilisés comme telles. Soit P1 et P2 deux pointeurs sur **int**, alors l'affectation P**1 = P2;** copie le contenu de P2 vers P1. P1 pointe alors sur le même objet que P2.

Résumons :



Après les instructions :

```
int A;
int *P;
P = &A;
```

`A` désigne le contenu de A
`&A` désigne l'adresse de A

`P` désigne l'adresse de A
`*P` désigne le contenu de A

En outre:

`&P` désigne l'adresse du pointeur P

`*A` est illégal (puisque A n'est pas un pointeur)

IV. Les pointeurs sur Tableaux

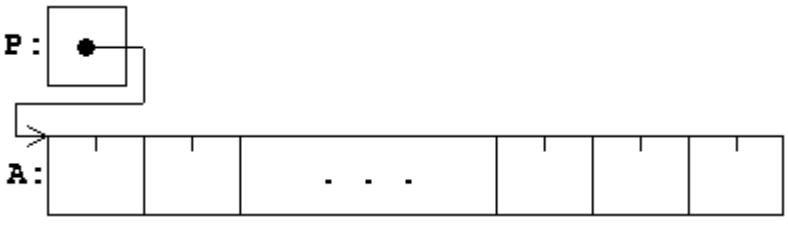
Nous pouvons retenir que *le nom d'un tableau est un **pointeur constant** sur le premier élément du tableau.*

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];`



Si P pointe sur une composante quelconque d'un tableau, alors P+1 pointe sur la composante suivante. Plus généralement,

`P+i` pointe sur la i-ième composante derrière P
`P-i` pointe sur la i-ième composante devant P.

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]

Sources d'erreurs

Un bon nombre d'erreurs lors de l'utilisation de C provient de la confusion entre soit contenu et adresse, soit pointeur et variable. Revoyons donc les trois types de déclarations que nous connaissons jusqu'ici et résumons les possibilités d'accès aux données qui se présentent.

Les variables et leur utilisation `int A;` déclare une ***variable simple*** du type **int**

`A` désigne *le contenu de A*
`&A` désigne l'adresse de A

`int B[];` déclare un ***tableau*** d'éléments du type **int**

`B` désigne *l'adresse de la première composante de B.* (Cette adresse est toujours constante)
`B[i]` désigne le contenu de la composante i du tableau
`&B[i]` désigne l'adresse de la composante i du tableau

en utilisant le formalisme pointeur:

`B+i` désigne l'adresse de la composante i du tableau
`*(B+i)` désigne le contenu de la composante i du tableau

`int *P;`
déclare un ***pointeur*** sur des éléments du type **int**.

`P` peut pointer
sur des variables simples du type **int** ou
sur les composantes d'un tableau du type **int**.
`P` désigne *l'adresse contenue dans P* (Cette adresse est variable)
`*P` désigne le contenu de l'adresse dans P

Si P pointe dans un tableau, alors

`P` désigne l'adresse de la première composante
`P+i` désigne l'adresse de la i-ième composante derrière P
`*(P+i)` désigne le contenu de la i-ième composante derrière P