LES POINTEURS

Il y a deux types de mémoire : la mémoire vive (permet de stocker de l'information de manière temporaire -RAM-) et la mémoire de masse (permet de stocker de l'information à long terme -disque dure-). Lors de l'exécution de nos programmes c'est dans la mémoire vive que se stock toutes nos variables et valeurs qui y sont associées. Lors de la déclaration d'une variable on réserve de la place en mémoire vive dans laquelle on stock de l'information grâce au caractère '='. On localise cette espace mémoire grâce à une adresse mémoire.
Tout d'abord un pointeur est une variable au même titre que les int, char, long … La particularité des pointeurs et que la valeur qui sera stocké dedans est une adresse mémoire. Mais l'adresse de quoi ? Et bien l'adresse d'une autre variable. Ainsi il y a plusieurs type de pointeurs : les pointeurs sur les int, les char ou même des tableaux.
Comme un pointeur est une variable, il suffit de les déclarer en début de fonction mais il faut connaitre le type de la variable que l'on veut « pointer » si on considère que « type » représente un type de variable (int, char, long …) on le fait de la manière suivante :

```
type    ma_variable;
type*   pointeur_sur_ma_variable
```

pour en déclarer plusieurs à la fois on peut faire :

```
type    ma_variable_1, ma_variable_2, …
type    *pointeur_sur_ma_variable_1,*pointeur_sur_ma_variable_2,…
```

On initialise un pointeur à l'aide la valeur NULL, cette valeur permet au pointeur de pointer sur rien.
Supposons maintenant que l'on veuille pointer sur un int qu'on appellera « nombre » et qu'on veuille stocker dans un pointeur l'adresse de « nombre ».
On commence par déclarer et initialiser notre variable et notre pointeur, qu'on appellera « pointeur_nombre » :

```
int     nombre;
int*    pointeur_nombre;
nombre = 0;
pointeur_nombre = &nombre;
```

Pour récupérer l'adresse d'une variable il suffit de placer le caractère '&' devant le nom de la variable ainsi « &nombre » permet d'avoir l'adresse mémoire de « nombre »
Ce qui est génial avec les pointeurs c'est qu'une fois qu'ils contiennent l'adresse d'une variable ils peuvent également aller chercher la valeur stockée à cette adresse et même la modifier ! On fait cela en ajoutant le caractère '*' au début du nom du pointeur. En reprenant l'exemple précédant si on fait :

```
*pointeur_nombre = 1;
printf  (« %d », nombre);
```

La valeur qui sera afficher sera 1 car nous avons demander à l'ordinateur de modifier la valeur stocké à l'adresse stocké par le pointeur !
Ceci est très utile lorsque vous désirez modifier une variable qui a été déclaré dans le main à l'aide d'une fonction définit au préalable car comme vous le savez une fois qu'une fonction est exécutée les arguments sont copiés et toutes les variables déclarées dans la fonction et les copies d'arguments sont supprimées.
Supposons que nous voulons créer une fonction qui multiplie un nombre par 2 en dehors du main on pourrait penser pouvoir faire :

Code 1:

```
int     multi_2 (int nb)
{
        nb = nb * 2;
        return 0;
}

int     main()
{
        int nb = 2;
        multi_2 (nb);
        printf (« %d », nb);
        return 0;
}
```

Code 2:

```
int     multi_2 (int nb, int* poiteur_nb)
{
        nb = nb *2;
        *pointeur_nb = nb;
        return 0;
}

int main ()
{
        même main que code 1
}
```

Si on teste le code 1 ça compile mais printf nous affiche 2 à l'écran car la fonction multi_2 créer une copie de nb donc utilise une nouvelle adresse et y stock la valeur de l'argument nb, puis affecte à la copie la valeur nb * 2 et une fois que c'est finit la copie de nb à été écrasée et on a appelé la fonction multi_2 pour rien ! Pour éviter ça, vous vous en doutez, il suffit d'utiliser nos chers pointeurs !
Dans le code 2 multi_2 fait une copie de nb et modifie cette copie, elle copie également pointeur_nb, mais la valeur qui se trouve dans cette copie est toujours l'adresse de nb déclarée dans le main (et c'est là que toute la magie opère). Ainsi en faisant *pointeur_nb on accède à l'espace mémoire réservé à l'adresse de la variable nb dans le main et non de la copie créer par multi_2 et on y stock la valeur de la copie de nb utilisé par la fonction multi_2. Ainsi on a bien modifié la valeur de nb dans le main ! Cet exemple est très simple et l'utilisation des pointeurs dans ce genre de cas peut être évité cependant je pense qu'il permet de bien comprendre l'utilisation des pointeurs.