

Les Pointeurs

Définition d'un pointeur

Un pointeur est une variable contient l'adresse d'une autre variable d'un type donné.

Comprendre ce qu'est une adresse

Lorsque l'on exécute un programme, celui-ci est stocké en mémoire, cela signifie que d'une part le code à exécuter est stocké, mais aussi que chaque variable que l'on a définie a une zone de mémoire qui lui est réservée, et la taille de cette zone correspond au type de variable que l'on a déclaré.

En réalité la mémoire est constituée de plein de petites cases de 8 bits (**un octet**). Une variable, selon son **type** (donc sa taille), va ainsi occuper une ou plusieurs de ces cases (une variable de type char occupera une seule case, tandis qu'une variable de type long occupera 4 cases consécutives). Chacune de ces « cases » (appelées **blocs**) est identifiée par un numéro. Ce numéro s'appelle **adresse**.

Comment connaît-on l'adresse d'une variable ?

La syntaxe suivante permet de récupérer l'adresse de cette variable :

`&Nom_de_la_variable`

Déclaration d'un pointeur

Un pointeur est une variable qui doit être définie en précisant le type de variable pointée, de la façon suivante :

`type * Nom_du_pointeur`

Le type de variable pointée peut être aussi bien un type primaire (tel que *int*, *char*...) qu'un type complexe (tel que *struct*...).

Remarque :

Grâce au symbole '*' le compilateur sait qu'il s'agit d'une variable de type *pointeur* et non d'une variable.

Initialisation d'un pointeur

Pour initialiser un pointeur, il faut utiliser l'opérateur d'affectation '=' suivi de l'opérateur d'adresse '&' auquel est accolé un nom de variable :

`Nom_du_pointeur = &nom_de_la_variable_pointee;`

Par exemple :

`int a = 2;`

```
char b;  
int *p1;  
char *p2;  
p1 = &a;  
p2 = &b;
```

Accéder à une variable pointée

Après avoir déclaré et initialisé un pointeur, il est possible d'accéder au contenu de l'adresse mémoire pointée par le pointeur grâce à l'opérateur '*'. Par exemple :

```
int a = 2;  
*p1 = 10;
```

Après ces deux instructions, le contenu des variables a sera 10

Pointeurs et tableaux à une dimension

Tout tableau en C est en fait un pointeur constant. Dans la déclaration
int tab[10];

tab est un pointeur constant (non modifiable) dont la valeur est l'adresse du premier élément du tableau. Autrement dit, tab a pour valeur &tab[0]. On peut donc utiliser un pointeur initialisé à tab pour parcourir les éléments du tableau.

Exemple :

```
#define N 5  
  
int tab[5] = {0, 1, 4, 20, 7};  
  
main()  
{  
    int i;  
  
    int *p;  
  
    p = tab;  
  
    for (i = 0; i < N; i++)  
  
    {  
        printf(" %d \n", *p);
```

```
p++;  
}  
}
```

On accède à l'élément d'indice i du tableau `tab` grâce à l'opérateur d'indexation `[]`, par l'expression `tab[i]`. Cet opérateur d'indexation peut en fait s'appliquer à tout objet `p` de type pointeur. Il est lié à l'opérateur d'indirection `*` par la formule $p[i] = *(p + i)$

Les pointeurs et tableaux se manipulent donc exactement de la même manière. Par exemple, le programme précédent peut aussi s'écrire

```
#define N 5
```

```
int tab[5] = {0, 1, 4, 20, 7};
```

```
main()
```

```
{  
int i;
```

```
int *p;
```

```
for (p = tab; p < tab + N; p++)
```

```
printf(" %d \n", p);
```

```
}
```

Allocation dynamique

Introduction

L'allocation dynamique de mémoire permet la réservation d'un espace mémoire pour son programme au moment de son exécution.

Les fonctions de l'allocation dynamique appartiennent toutes au fichier d'en-tête suivant : **stdlib.h**.

Fonction malloc

Cette fonction demande au système d'exploitation un bloc de taille `t` (en bytes) et renvoie un pointeur vers l'adresse du bloc alloué.

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

Allocation dynamique de tableaux :

Si nous voulons allouer un tableau de taille `n`, il faudra tout simplement allouer `n` blocs de taille `t`.

```
int * t;
```

```
t = (int*)malloc (n*sizeof(int));
```

```
#include <stdlib.h>

main()
{
    int n;

    int *tab;

    ...
    tab = (int*)malloc(n * sizeof(int));
    ...
    free(tab);
}
```