CHAPITRE 7 : Pointeurs en langage C

4. Pointeurs et Tableaux

- En C, il existe une relation très étroite entre tableaux et pointeurs. Ainsi, chaque opération avec des indices de tableaux peut aussi être exprimée à l'aide de pointeurs.

- Comme déjà mentionné (au chapitre 6), le nom d'un tableau représente l'adresse de son premier élément :
  - Tableau à un dimension (int T[N]) :
    - le nom T du tableau est un pointeur constant sur le premier élément (1er entier) du tableau
    - T et &T[0] contiennent l'adresse du premier élément (1er entier) du tableau.
  - Tableau à deux dimensions(int T[N][M]) :
    - le nom T est un pointeur constant sur le premier tableau (d'entiers).
    - T[i] est un pointeur constant sur le premier élément (1er entier) du ième tableau.
    - T et T[0] contiennent la même adresse mais leur manipulation n'est pas la même puisqu'ils ne représentent pas le même type de pointeur.

CHAPITRE 7 : Pointeurs en langage C

4. Pointeurs et Tableaux

4.1 Adressage et accès aux composantes d'un tableau à une dimension

- En déclarant un tableau A de type int (int A[N]) et un pointeur P sur des variables entière (int *P),
- l'instruction P = A crée une liaison entre le pointeur P et le tableau A en mettent dans P l'adresse du premier élément de A (de même P = &A[0]).
- A partir du moment où P = A, la manipulation du tableau A peut se faire par le biais du pointeur P. En effet :

p	pointe sur A[0]	*p	désigne A[0]
p+1	pointe sur A[1]	*(p+1)	désigne A[1]
….			
p+i	pointe sur A[i]	*(p+i)	désigne A[i]

où $i \in [0 , N-1]$

CHAPITRE 7 : Pointeurs en langage C

4. Pointeurs et Tableaux

4.1.1 Exemple (Lecture et Affichage d'un tableau matérialisé par un pointeur)

```
#include <stdio.h>
#define N 10

void main()
{
  float t[N] , *pt ;
  int i ;

  printf("Entrez %d entiers\n", N) ;

  pt = &t[0] ;       /* ou pt  = t */
  for (i = 0 ; i<N; i++)
    scanf("%f", pt+i) ;  /* pt+i pointe sur A[i] */

  printf("\nTableau lu : \n") ;
  for (i = 0 ; i<N ; i++)
    printf("%7.2f", *(pt+i)) ; /* *(pt+i)
équivalente à pt[i] */
}
```

```
 /* Autre Solution sans déclarer la variable i */
#include <stdio.h>
#define N 10

void main()
{
  float T[N] , *pt ;

  printf("Entrez %d entiers\n", N) ;

  for (pt = T ; pt<T+N; pt++)
    scanf("%f", pt) ;

  printf("\nTableau lu : \n") ;

  for (pt = T ; pt<T+N; pt++)
    printf("%7.2f", *pt) ;
}
```

CHAPITRE 7 : Pointeurs en langage C

4. Pointeurs et Tableaux

4.2 Adressage et accès aux composantes d'une matrice

- En déclarant une matrice A de type int (int A[M][N]) et un pointeur P sur des variables entières (int *P),
- l'instruction P = A[0] crée une liaison entre le pointeur P et la matrice A en mettent dans P l'adresse du premier élément de la première ligne de la matrice A (P = &A[0][0]).
- A partir du moment où P = A[0], la manipulation de la matrice A peut se faire par le biais du pointeur P. En effet :

p	pointe sur	A[0][0]	et	* p	désigne A[0][0]
p + 1	pointe sur	A[0][1]	et	* (p + 1)	désigne A[0][1]
….					
p + N	pointe sur	A[1][0]	et	* (p + N)	désigne A[1][0]
p + N + 1	pointe sur	A[1][1]	et	* (p + N + 1)	désigne A[1][1]
….					
p + i * N + j	pointe sur	A[i][j]	et	* (p + i * N + j)	désigne A[i][j]

où $i \in [0 , M-1]$ et $j \in [0 , N-1]$.

CHAPITRE 7 : Pointeurs en langage C

4. Pointeurs et Tableaux

4.2.1 Exemple (Lecture et Affichage d'une matrice matérialisé par un pointeur)

```
#include <stdio.h>
#define M 4
#define N 10

void main()
{
 short A[M][N] ;
 short *pt ;
 int i, j ;

 /* lecture d'une matrice */
 pt = &A[0][0] ;/* ou bien pt = A[0] ; */
 for (i = 0 ; i<M ; i++)
 {
    printf("\t ligne n° %d\n", i+1) ;
    for (j = 0 ; j<N ; j++)
      scanf("%i", pt + i * N + j ) ;
 }
```

```
    for (i = 0 ; i<M ; i++)
     {
        for (j = 0 ; j<N ; j++)
          printf("%d", *( pt + i * N + j ) ) ;
        printf("\n") ;
     }

    }
```

CHAPITRE 7 : Pointeurs en langage C

4. Pointeurs et Tableaux

4.3 Arithmétiques des pointeurs

Affectation par un pointeur sur le même type :

- Soient P1 et P2 deux pointeurs sur le même type de données.
- L'affectation : P1 = P2 ; fait pointer P1 sur le même objet que P2.

Addition et soustraction d'un nombre entier :

- Si P pointe sur l'élément A[i] d'un tableau, alors :
- P+n *pointe sur* A[i+n] et P-n *pointe sur* A[i-n]

Incrémentation et décrémentation d'un pointeur :

- Si P pointe sur l'élément A[i] d'un tableau, alors après l'instruction :
- P++ ; P *pointe sur* A[i+1]
- P += n ; P *pointe sur* A[i+n]
- P-- ; P *pointe sur* A[i-1]
- P -= n ; P *pointe sur* A[i-n]

Comparaison de deux pointeurs :

- On peut comparer deux pointeurs ***de même type*** par : <, >, <=, >=, == ou !=
- La comparaison de deux pointeurs qui pointent dans le même tableau est équivalente à la comparaison des indices correspondants.

CHAPITRE 7 : Pointeurs en langage C

4.4 Autres déclarations des pointeurs :

En C, il existe d'autres déclarations des pointeurs. En effet :

- **Tableau de pointeurs :**
  int *Tab[20] ;
  déclare un tableau Tab de 20 pointeurs d'entiers.

- **Pointeur de tableaux :**
  int (*pt)[30] ;
  déclare un pointeur pt sur des tableaux de 30 composantes.

- **Pointeur de pointeurs :**
  int **pt ;
  déclare un pointeur pt qui pointe sur des pointeurs d'entiers.

CHAPITRE 7 : Pointeurs en langage C

5. Allocation dynamique

- La déclaration d'un tableau définit un tableau "statique" (il possède un nombre figé d'emplacements). Il y a donc un gaspillage d'espace mémoire en réservant toujours l'espace maximal prévisible.

- Il serait souhaitable que l'allocation de la mémoire dépend du nombre d'éléments à saisir. Ce nombre ne sera connu qu'à l'exécution : c'est l'allocation dynamique.

CHAPITRE 7 : Pointeurs en langage C

5.1 Fonctions d'allocation dynamique de la mémoire

- En C, il existe 4 fonctions pour gérer l' allocation dynamiquement de la mémoire

Bibliothèque <**stdlib.h**>	
`char *malloc(taille)`	allocation d'un bloc
`char *calloc(taille, sizeof(type))`	allocation & initialisation d'un bloc
`char *realloc(char *, taille)`	modification de la taille d'un bloc
`void free(char *)`	libération d'un bloc

- Chacune des fonctions malloc, calloc ou realloc, prend une zone d'une taille donnée dans l'espace mémoire libre réservé pour le programme (appelé ***tas*** **ou** ***heap***) et affecte l'adresse du début de la zone à une variable pointeur.

- S'il n'y a pas assez de mémoire libre à allouer, la fonction renvoie le pointeur NULL.

CHAPITRE 7 : Pointeurs en langage C

5.2 Fonctions malloc et free

- **5.2.1 malloc**

<pointeur> = <type> malloc(<taille>);

- **<type>** est un type pointeur définissant la variable pointé.par <pointeur>
- **<taille>** est la taille, en octets, de la zone mémoire à allouer dynamiquement. <taille> est du type **unsigned int**, donc on ne peut pas réserver plus de 65536 octets à la fois
- La fonction **malloc** retourne l'adresse du premier octet de la zone mémoire allouée. En cas d'échec, elle retourne **NULL**.

- **5.2.2 free**

- Si on n'a plus besoin d'un bloc de mémoire réservé dynamiquement par **malloc**, alors on peut le libérer à l'aide de la fonction **free**.

free(<pointeur>);

- Libère le bloc de mémoire désigné par le pointeur <pointeur>

CHAPITRE 7 : Pointeurs en langage C

5.2.3 Exemple (Allocation dynamique, Saisie et Affichage d'un tableau)

```
#include <stdio.h>
#include <stdlib.h>

int  main()
{
  short  *pt;
  int N , i;
  printf("Entrez la taille N  du tableau \n") ;
  scanf("%d", &N) ;

  pt = ( short * ) malloc( N * sizeof( short ) );
  if (pt == NULL)
  {
    printf("Mémoire non disponible") ;
    system("pause") ;
    return 1;
  }
```

```
        printf("Saisie du tableau : ");
        for (  i = 0  ; i < N; i++)
              scanf("%d", pt + i ) ;

        printf("Affichage du tableau ") ;
        for ( i= 0  ; i < N; i++)
              printf("%d\t", *( pt + i ) ) ;

        free( pt );

        return 0;
}
```