Chapitre 3
Pointeur, Tableau, Allocation dynamique.

1- Les Pointeurs

Définitions

- ✓ **Un pointeur p est une variable qui sert à contenir l'adresse mémoire d'une autre variable.**
- ✓ **Si la variable v occupe plus d'une case mémoire, l'adresse contenue dans le pointeur p est celle de la première case utilisée par v.**
- ✓ **La variable v est appelée variable pointée.**
- ✓ **Si p contient l'adresse de v alors on dit que p pointe sur v.**
- ✓ **Le pointeur possède un type qui dépend du type des variables pointées.**
- ✓ **Un pointeur peut contenir aussi l'adresse d'une fonction, on parlera alors de pointeur de fonctions.**

Déclaration des pointeurs:

Une variable de type pointeur se déclare à l'aide du type de l'objet pointé précédé du symbole *.

Exemple:

*char *pc;*	**pc est un pointeur sur un objet de type char**
*int *pi;*	**pi est un pointeur sur un objet de type int**
*float *pr;*	**pr est un pointeur sur un objet de type float**

- ✓ **En dehors de la partie déclarative, l'opérateur * désigne en fait le contenu de la case mémoire pointée.**

- ✓ **L'adresse d'une variable peut être obtenue grâce à l'opérateur & suivi du nom de la variable.**

Arithmétique des pointeurs

- ✓ On peut essentiellement déplacer un pointeur dans un plan mémoire à l'aide des opérateurs d'addition, de soustraction, d'incrémentation, de décrémentation.
- ✓ On ne peut le déplacer que d'un nombre de cases mémoire multiple du nombre de cases réservées en mémoire pour la variable sur laquelle il pointe.

Affectation d'une adresse à un pointeur

- ➢ Lorsque l'on déclare une variable char, int, float .... un nombre de cases mémoire bien défini est réservé pour cette variable. En ce qui concerne les pointeurs, l'allocation de la case mémoire pointée obéit à des règles particulières :

Exemple:

*char *pc;*

- **Si on se contente de cette déclaration, le pointeur pointe « n'importe où ». Son usage, tel que, dans un programme peut conduire à un « plantage » du programme ou du système d'exploitation si les mécanismes de protection ne sont pas assez robustes.**
- **L'initialisation du pointeur n'ayant pas été faite, on risque d'utiliser des adresses non autorisées ou de modifier d'autres variables.**

Exemple d'affectation :

```
int  i=5;
int * pi;
pi=&i;
cout <<"valeur de i = " << i<< endl;
cout <<"adresse de i = " << &i<< endl;
cout <<"adresse où il pointe pi = " << pi<< endl;
```

Incompatibilité de types :

On ne peut pas affecter à un pointeur de typeT1 l'adresse d 'une variable d 'autre type T2

Exemple:

```
int  i=5;
char * pc;          erreur
pc=&i;
```

On peut cependant forcer la conversion.

A utiliser avec précaution ....

```
int  i=5;
char * pc;
pc= ( char *) &i;
```

Pointeur Void :

- ✓ Les pointeurs void sont un type particulier de pointeur
- ✓ Ils peuvent pointer sur une variable de n'importe quel type
- ✓ On ne peut pas utiliser l'opérateur d'indirection * sur un pointeur void. Il faut d'abord le convertir en un pointeur d'un type donné.

Exemples :

```
int  a=5, b=6;
int *  pi=&a;
void *  pv = &b;          //correct
pv=pi                     //correct
pi =pv                    // !!! Erreur!!!
pi=(int*) pv              //correct
cout <<*pv ;              // !!! Erreur!!!
cout <<*((int *) pv );    //correct
```

2- Les Tableaux

- ✓ **Les tableaux correspondent aux vecteurs et matrices en mathématiques. Un tableau est caractérisé par sa taille et par le type de ses éléments.**

Les tableaux à une dimension:

Déclaration: ***type*** **nom[*dim*];**

Exemples:

int **compteur[10];**

float **nombre[20];**

Cette déclaration signifie que le compilateur réserve *dim* places en mémoire pour ranger les éléments du tableau.

Remarque:

dim **est nécessairement une *EXPRESSION CONSTANTE*.**

Les tableaux à plusieurs dimensions:

- Tableaux à deux dimensions:

Déclaration:

```
type nom[dim1][dim2];
```

Exemples:

```
int compteur[4][5];
float nombre[2][10];
```

Initialisation des tableaux:

On peut initialiser les tableaux au moment de leur déclaration:

Exemples:

```
int liste[10] = {1,2,4,8,16,32,64,128,256,528};
float nombre[4] = {2.67,5.98,-8.0,0.09};
int x[2][3] = {{1,5,7},{8,4,3}}; / / 2 lignes et 3 colonnes
```

Les Tableaux et Les Pointeurs

- ✓ ***Les identificateurs de tableaux et de pointeurs sont similaires***
- ✓ ***L'identificateur d'un pointeur désigne l'adresse de la première case mémoire de la variable pointée***
- ✓ ***De même, l'identificateur d'un tableau désigne l'adresse de la première case mémoire du tableau***
- ✓ ***Il existe cependant une différence majeure:***
  - ***La valeur du pointeur peut être modifiée***
  - ***L'adresse du tableau ne peut pas être modifiée***
- ✓ ***L'identificateur d'un tableau peut être considéré comme un pointeur constant***

Exemple:

```
const int   MAX=7;
int tab[MAX]; int *p;
p=tab;
*p=0;                       //tab[0]=0
p++;
*p=10;                      //tab[1]=10
p=&tab[2];
*p=20;              //tab[2]=20
p=tab+3;
*p=30;                      //tab[3]=30
p=tab;
*(p+4)=40;              //tab[4]=40
tab[5]=50;                  //tab[5]=50
*(tab+6)=60;                //tab[6]=60
for  ( int n=0; n<MAX; n++)       cout << tab[n]<< "   " ;
```

Résultats d'exécution:

```
0   10   20   30   40   50   60
```

✓ **L'instruction telle que tab=p aurait provoqué une erreur à la compilation**

Exercice 1:

Un programme contient la déclaration suivante:

```
int tab[10] = {4,12,53,19,11,60,24,12,89,19};
```

Compléter ce programme de sorte d'afficher les adresses des éléments du tableau.

Exercice 2:

Un programme contient la déclaration suivante:

```
int tab[20] = {4,-2,-23,4,34,-67,8,9,-10,11, 4,12,-53,19,11,-60, 24, 12, 89,19};
```

Compléter ce programme de sorte d'afficher les éléments du tableau avec la présentation suivante:

```
   4    -2   -23    4   34
 -67     8     9  -10   11
   4    12   -53   19   11
 -60    24    12   89   19
```

3- L' allocation dynamique

Intérêt de l'allocation dynamique

- ✓ L'allocation dynamique permet une gestion plus flexible de la mémoire
- ✓ Un programme n'utilise la mémoire dont il a besoins qu'au moment du besoin
- ✓ Ainsi il évite de monopoliser de l'espace mémoire aux instants où il n'en a pas besoins et permet donc à d'autres programmes de l'utiliser
- ✓ La mémoire peut ainsi être partagée de manières plus efficaces entre plusieurs programmes.
- ✓ - Supposant qu'un programme ait besoin d'afficher 100 objets graphiques différents et que ces objets sont très volumineux (en espace mémoire).
  - Supposant aussi qu'un instant donné, le programme n'a besoin d'afficher simultanément et au maximum 10 objets parmi 100.
  - Supposant que la taille de la mémoire permet au maximum le stockage simultané de 30 objets graphiques
  - Allouer les 100 objets de façon statique est impossible puisque l'espace mémoire est insuffisant.
  - En revanche, déclarer dynamiquement 10 objets est tout à fait possible et le programme pourra alors s'exécuter sans problème pour 100 objets à des instants différentes.

L'Opérateur new

- ✓ L'opérateur new permet l'allocation de l'espace mémoire nécessaire pour stocker un élément d'un type T donné.
- ✓ Pour que l'opérateur new puisse connaître la taille de l'espace à allouer il faut donc lui indiquer le type T
- ✓ Si l'allocation réussit (il y a suffisamment d'espace en mémoire) alors l'opérateur new retourne l'adresse de l'espace alloué.
- ✓ Cette adresse doit être alors affectée à un pointeur de type T pour pouvoir utiliser cet espace par la suite
- ✓ Si l'allocation échoue (il n'y a pas suffisamment d'espace mémoire) alors l'opérateur new retourne NULL (0)

```
T *p;
P = new T      ou      T *p = new T
```

- ✓ **Il est possible d'indiquer après le type T une valeur pour initialiser l'espace ainsi alloué. Cette valeur doit être indiquer entre parenthèses : new T (val)**

Allocation d'un entier :

```
//Sans initialisation
    int *p = new int;
      *p =5;
cout <<*p; // la valeur affichée 5
```

```
//Avec initialisation
   int *p = new int (10);
cout <<*p; // la valeur affichée 10
```

- ✓ **On n'a pas vérifié ici si l'allocation a réussi. Dans la pratique, il faudra le faire systématiquement.**

- ✓ **Pour allouer dynamiquement un tableau d'éléments il suffit d'utiliser l'opérateur new [] en indiquant entre les crochets le nombre d'éléments**
- ✓ **Contrairement aux tableaux statiques où le nombre d'éléments devait être une constante (valeur connue à la compilation) ici le nombre d'éléments peut être variable dont la valeur ne sera connue qu'a l'exécution (la valeur saisie par l'utilisateur, lue à partir d'un fichier)**
- ✓ **On récupère alors du premier élément du tableau**

  T *p = new T[n]; // p contiendra l'adresse du premier élément

- ✓ **Il n'est pas possible ici d'indiquer les valeurs d'initialisation**

L'Opérateur delete

- ✓ **Pour que l'allocation dynamique soit utile et efficace, il est impératif de libérer l'espace réservé avec new dès que le programme n'en a plus besoin.**
- ✓ **Cet espace pourrait alors être réutilisé soit par le programme lui-même soit par d'autre programmes**
- ✓ **Libération d'un élément simple : *opérateur delete***

```
int  *p = new int;
     *p =5;
cout<< *p<<endl;
delete p;
```

- **Libération d'un tableau : opérateur delete []**

```
int  *p ,i,n;
 cout<<"Nombre delements ?: ";
 cin>> n;
 p= new int [n];
for (i=0;i<n;i++)
                        cin >>p[i];
for (i=0;i<n;i++)
                    cout<< p[i]*p[i]<<endl;
delete [] p;
```

Tableaux dynamiques à 2 dimensions:

• **Un tableau à deux dimensions est, par définition, un tableau de tableaux. Il s'agit donc en fait d'un pointeur vers un pointeur. Considérons le tableau à deux dimensions défini par :**

```
int tab[M][N];
```

• **tab est un pointeur, qui pointe vers un objet lui-même de type pointeur d'entier. tab a une valeur constante égale à l'adresse du premier élément du tableau, &tab[0][0].**
• **De même tab[i], pour i entre 0 et M-1, est un pointeur constant vers un objet de type entier, qui est le premier élément de la ligne d'indice i. l'élément tab[i] a donc une valeur constante qui est égale à &tab[i][0].**

• **Exactement comme pour les tableaux à une dimension, les pointeurs de pointeurs ont de nombreux avantages sur les tableaux multi-dimensionnés.**

• **On déclare un pointeur qui pointe sur un objet de type *type ** de la même manière qu'un pointeur, c'est-àdire**

***type **nom-du-pointeur*;**

• **Par exemple, pour créer avec un pointeur de pointeur une matrice à k lignes et n colonnes à coefficients entiers, on écrit :**

```
main()
{  int k, n;
   int **tab;
            tab = new int*[k];
            for (i = 0; i < k; i++)
                        tab[i] = new int[n];
                                ....
           for (i = 0; i < k; i++)
                                delete [] tab[i];
           delete [] tab;
}
```

- **La première allocation dynamique réserve pour l'objet pointé par tab l'espace mémoire correspondant à k pointeurs sur des entiers. Ces k pointeurs correspondent aux lignes de la matrice.**
- **Les allocations dynamiques suivantes réservent pour chaque pointeur tab[i] l'espace mémoire nécessaire pour stocker n entiers.**

Exemple de tableau de chaînes:

```
#include<iostream>
using namespace std;
int main(int argc, char**argv)
{
int  i, nbMots;
char ** tabMots;
cout<< "nombre de mots?: ";
cin>>nbMots;
tabMots= new char * [nbMots];       // Allocation du tableau de pointeurs

for  (i=0;i<nbMots;i++){
    tabMots[i]= new char;    // Allocation de l'espace nécessaire au ième mot
    cout<<"Mots" <<i+1<<"?: ";
    cin>>tabMots[i];              //  cin.getline(mot,20);     //Saisie du ième mot dans une variable temporaire

}
for (i=0;i<nbMots;i++)  cout<<tabMots[i]<<endl;
for (i=0;i<nbMots;i++) delete  tabMots[i];  //Liberation de l'espace de chaque mot
delete  [] tabMots;                //Liberation du tableau de pointeurs */
return 0;
}
```