Séance N°4

Pointeurs et Allocation dynamique

Notion de pointeur : *opérateurs & et **

On peut accéder à l'adresse mémoire de n'importe quelle variable en utilisant l'opérateur « adresse » : *&*.
Si *i* est un entier, *&i* renvoie l'adresse réelle en mémoire de la variable *i*.

Ex 1 – Testez le programme suivant :

```
//adresse d'une variable
#include <iostream>
using namespace std;

void main()
{
	int i;
	i = 2;
	cout<<" valeur de i : "<<i<<endl;
	cout<<" adresse mémoire de i : "<<&i<<endl;
}
```

On peut à l'inverse déclarer une variable qui désigne (ou pointe) une **adresse mémoire** en utilisant l'opérateur * (étoile) : on parle de **pointeur**. Une variable **pointeur** contient l'adresse où est rangée une valeur du type spécifié. Pour déclarer un pointeur sur une variable de type spécifié, on utilise la syntaxe suivante :

```
type *nom_pointeur ;	// nom_pointeur contient l'adresse où est rangée une valeur du type spécifié
```

nom_pointeur désigne donc une variable qui contient une adresse mémoire exprimée en base hexadécimale. Pour accéder à la valeur stockée dans cet emplacement mémoire on utilise à nouveau l'opérateur étoile : **nom_pointeur* désigne le contenu de la mémoire donc une valeur du type spécifié.

Ex 2 – Testez le programme suivant :

```
#include <iostream>
using namespace std;

void main()
{
	int i;		// Déclaration de la variable i comme entier
	i = 2;		// Initialisation de la variable i
	int *p;		// Déclaration de la variable pointeur p sur un entier
	p = &i;		// Initialisation de la variable p
	cout<<" valeur de i : "<<i<<endl;
	cout<<" valeur de p : "<<p<<endl;
	*p = 3;
	cout<<"valeur de i : " <<i<<endl;
}
```

Question : expliquer ce qui se passe lors de l'instruction ***p = &i;*** et pourquoi ****p = 3;*** modifie la valeur de i ?
Modifier le programme pour utiliser un *double* au lieu d'un *entier*.

Dans le cas des **tableaux**, on peut également accéder à la **valeur** ou à l'**adresse** d'un élément du tableau. Soit *t* une variable qui désigne le **nom** du tableau. La variable i est l'**indice** du tableau (toujours un **entier**) et permet d'accéder à la valeur *t[i]*.
L'opérateur *&* permet d'accéder à l'adresse de stockage de la valeur d'une variable. Ainsi *&t[0]* désigne l'adresse mémoire où est rangée la première valeur du tableau *t[0]* : l'adresse est exprimée en base hexadécimale (base 16) sous forme de 8 chiffres, parfois

précédés de 0x pour indiquer que la base est hexadécimale. Les valeurs ou éléments d'un tableau sont stockés continûment dans la mémoire à partir de *&t[0]*. *t* désigne à la fois le nom du tableau et la première adresse de stockage, en fait *t=&t[0]*.

Ex 3 – Testez le programme suivant :

```
#include <iostream>
using namespace std;

void main()
{
	int t[3];	// Tableau statique de 3 entiers

	for(int i=0; i<3; i++)
	{
	    cout<<" saisir la valeur t["<<i<<"] : "<<endl; cin >> t[i] ;
	}

	for(int i=0; i<3; i++)
	{
	    cout<< endl <<i<< " "<<&t[i]<< " "<<t[i]<<endl;
	}
}
```

Question : Combien d'octets séparent deux adresses de stockage *&t[i]* et *&t[i+1]* pour des entiers ?
Modifier le programme pour utiliser un tableau de *double*. Combien d'octets séparent deux adresses de stockage contigus pour un tableau de type *double* ?
Ajouter une ligne pour vérifier que *t=&t[0]*.

Allocation dynamique de mémoire : *instructions new et delete*

Les pointeurs permettent de **réserver un emplacement mémoire disponible** par l'instruction d'**allocation dynamique** new : cette instruction recherche dans la mémoire une zone libre, dont la **taille** (en octets) est précisée par le **type** de ce qui doit être rangé dans cette zone. L'adresse retournée par la fonction *new* correspond au **début de la zone trouvée** : elle est rangée dans une variable pointeur par l'instruction : *variable_pointeur = new type.*

S'il n'y a plus en mémoire de zone de taille suffisante d'octets contigus pour accueillir une valeur du type spécifié, l'allocation échoue et la fonction *new* renvoie la valeur **0**. Ceci permet de faire un test pour s'assurer qu'on peut bien utiliser la mémoire allouée. En fin de programme, la mémoire allouée dynamiquement par l'opérateur doit être libérée (ou désallouée) par l'opération delete.

L'allocation dynamique est particulièrement utile lorsque l'on veut utiliser des tableaux dont la taille est *a priori* inconnue, on parle alors de **tableau dynamique**. Les tableaux dynamiques font donc appel à la notion de **pointeur**.

Tableau dynamique :

C'est un tableau de **taille variable** pour lequel on **alloue dynamiquement de la mémoire** en fonction de sa **taille.**

Ex 4 – Testez le programme suivant :

```
#include <iostream>
using namespace std;

void main()
{
	int n;			// déclare une variable n de type entier
	cout<<endl<<" saisir la taille du tableau voulue : ";
	do  cin >> n; while (n < 0);

	int *t;			// déclare une variable pointeur t de type entier
	t = new int[n];		// alloue une zone mémoire correspondant à n entiers
```

```
    if(t)
    { // si l'allocation a réussi, le pointeur t contient la première adresse du tableau
        int somme=0;

        for(int i=0; i<n; i++)
        {
            cout<<" saisir la valeur "<<i<<" : ";
            cin>>t[i] ;
            somme += t[i] ;
        }

        for(int i=0; i<n; i++)
        {
            cout<<t[i]<<endl;
        }

        cout<< endl<<" somme = "<< somme;
        cout<< endl<<" moyenne = "<< somme/n;
        delete [] t;          // libération de l'espace mémoire réservée pour le tableau
    }
    else
        cout<<"l'allocation dynamique a échoué "<<endl;    // l'instruction new renvoie 0
}
```

Incrémentation d'un pointeur :

Remplacer la ligne d'instruction la dernière boucle *for(i=0; i<n; i++) cout<<t[i]<<endl;*
par les lignes suivantes :

```
int *p ;  // déclare un pointeur p sur un entier
p=t ;     // initialise le pointeur p à l'adresse du premier élément du tableau t

for(int i=0; i<n; i++)
{
    cout<<*p<<endl;     // affiche l'entier pointé par p
    p++ ;               // incrémente le pointeur p
}
```

Question : à chaque itération de la boucle for, de quelle taille mémoire incrémente-t-on la valeur de *p* par l'instruction *p*++ ?

TRAVAIL PERSONNEL

Pour mettre en application ce que j'ai appris, je fais les exercices suivants :

Exercice 1
Écrire un programme qui permet de saisir un nombre N de valeurs réelles dans un tableau, N étant choisi par l'utilisateur, puis classe les éléments du plus petit au plus grand et détermine et affiche le **minimum** et **maximum**.

Exercice 2
Ecrire un programme qui calcule les valeurs d'un polynôme. Le programme devra demander à l'utilisateur de saisir l'**ordre du polynôme** puis ces **coefficients** stockés dans un tableau dynamique. Ensuite, il demandera à saisir une valeur d'**abscisse**, et affichera la valeur du polynôme calculée pour cette abscisse.

BILAN PERSONNEL

Ce que j'ai appris aujourd'hui : (à compléter)

Vocabulaire informatique : (à compléter)