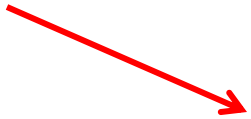
Pointeurs et références

- Toute variable est une référence à une adresse mémoire.
- Un pointeur est une variable spéciale qui stocke l'adresse d'une autre variable ou information.

- **Syntaxe:**

```
type *identificateur; // déclaration d'un pointeur
type *identificateur= &variable; //initialisation du pointeur
```



opérateur d'adresse (indirection)

- **Exemple:**

```
int *intptr;
double *reelptr;
int *pi1, *pi2, j, *pi3;
```

pi1, pi2 et pi3 sont des pointeurs d'entiers mais j est un entier.

Déréférencement et indirection

- Deux opérations permettant de récupérer l'adresse d'un objet et d'accéder à l'objet pointé. Elles sont respectivement appelées *indirection* et *déréférencement.*

- Ces opérateurs sont respectivement & et *.

A retenir:

***ptr et var = contenu de la variable (sa valeur).**

ptr et &var = adresse de la variable.

Exemple:

```
#include<iostream>
using namespace std;
main()
{int x=10;
int *px=&x;
cout<<"x= " <<x<<"\n";
cout<<" en utilisant le pointeur x= "
<<*px<<"\n";
 x*=2;
cout<< " en utilisant le pointeur x= "
<<*px<<"\n";
*px*=3;
cout<<"x= " <<x<<"\n";
return 0;}
```

Résultat de l'exécution:

```
x= 10
 en utilisant le pointeur x= 10
 en utilisant le pointeur x= 20
x= 60
```

Structures et pointeurs

- **Une fois qu'un pointeur a l'adresse d'une variable de type structure, il utilise l'opérateur -> pour accéder aux différents champs (de la structure).**
- **Syntaxe:**

  StructPtr -> membre

- **Exemple 1:**

```
struct point {double x;double y;} ;
point p;
point *ptr=&p;
ptr->x=23.35;
ptr->y=ptr->x+12.65;
```

Exemple 2:

```
struct Client
{int Age;
};
Client structure1;
Client *pstr = &structure1;
pstr->Age = 35; /* On aurait pu écrire
                   (*pstr).Age=35; */
```

Variables par référence

- **Les *références sont des synonymes* d'identificateurs. Elles permettent de manipuler une variable sous un autre nom que celui sous laquelle cette dernière a été déclarée.**

Note: Les références n'existent qu'en C++.

Syntaxe:

```
type &référence = identificateur ;
```

Exemple:

```
#include<iostream>
using namespace std;
main()
{int i=0; int &ri=i;
cout<<« en utilisant la variable, i= " <<i<<"\n";
cout<<" en utilisant la référence, ri= " <<ri<<"\n";
 ri+=2;
cout<<« en utilisant la variable, i= " <<i<<"\n";
cout<< " en utilisant la référence ri= " <<ri<<"\n";
return 0;}
```

Résultat de l'exécution:

```
en utilisant la variable, i= 0
en utilisant la référence, ri= 0
en utilisant la variable,i= 2
en utilisant la référence ri= 2
```

Lien entre les pointeurs et les références

- Les références permettent simplement d'obtenir le même résultat que les pointeurs avec une plus grande facilité d'écriture.

Exemple:

```
int *pi=&i ;
*pi+=2 ; // Manipulation de i via pi.
int i=0 ;
int &ri=i ; // ri est une référence de i.
ri+=2 ; // Manipulation de i via ri.
```

Exemple:

```
#include<iostream>
Using namespace std;
main()
{int i=0;
int &ri=i;
int *pi=&i;
cout<<"en utilisant la variable, i= " <<i<<"\n";
cout<<"en utilisant la référence, ri= " <<ri<<"\n";
cout<<"en utilisant le pointeur, *pi= " <<*pi<<"\n";
 ri+=2;        // Manipulation de i via ri.
// *pi+=2;  Manipulation de i via pi.
cout<<"en utilisant la variable, i= " <<i<<"\n";
cout<< "en utilisant la référence ri= " <<ri<<"\n";
 cout<<"en utilisant le pointeur, *pi= " <<*pi<<"\n";
return 0;}
```

RESULTAT DE L'EXECUTION

```
en utilisant la variable, i= 0
en utilisant la référence, ri= 0
en utilisant le pointeur, *pi= 0
en utilisant la variable,i= 2
en utilisant la référence ri= 2
en utilisant le pointeur, *pi= 2
```

```
en utilisant la variable, i= 0
en utilisant la référence, ri= 0
en utilisant le pointeur, *pi= 0
en utilisant la variable,i= 4
en utilisant la référence ri= 4
en utilisant le pointeur, *pi= 4
```

Conclusion:

Nous constatons que la référence ri peut être identifiée avec l'expression *pi, qui représente bel et bien la variable i.

Passage de paramètres

- **Par variable**

  ou

- **Par valeur**

Passage de paramètres par valeur

- **Aucune manipulation de la variable locale dans la fonction appelée ne modifie la variable passée en paramètre.**

- **Les modifications ne s'appliquent qu'à une copie de cette dernière.**

Exemple:

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

void test(int j) /* j est la copie de la valeur passée en paramètre */
{j=3; /* Modifie j, mais pas i. */
}

int main(void)
{
int i=2;
test(i); /*  contenu de i est copié dans j. i n'est pas modifié. i =2. */
cout<<"i= " <<i<<"\n"; // i=2
return 0;
}
```

Passage de paramètres par variable

- La deuxième technique consiste à passer non pas les valeurs des variables, mais les variables elles-mêmes.
- Il n'y a donc plus de copies, plus de variables locales.
- Toute modification du paramètre dans la fonction appelée entraîne la modification de la variable passée en paramètre.

Exemple: (valable en C et C++)

```
void test(int *pj) // test attend l'adresse d'un entier
{*pj=2; }                    /* l'entier est modifié */


int main(void)
{int i=3;
test(&i);  /* On passe l'adresse de i en paramètre */
 cout<<"i= "<<i; /* Ici, i vaut 2. */
return 0;}
```

Passage de paramètres par référence

Exemple: (valable en C++)

```
void test(int &i) /*  & signifie que l'on passe le
                      paramètre i par
                      référence*/ .
{i = 2; }       // Modifie le paramètre passé en référence.

int main(void)
{  int i=3;
   test(i);       // Après l'appel de test, i vaut 2.
    /* L'opérateur & n'est pas nécessaire pour appeler la
   fonction test. */
return 0;}
```

Récapitulation

Passage par valeur	Passage par variable	Passage par référence

int Fon1(int a, int b)
{
int nRet;
if(a==0)
a=10;
nRet = a + b;
return nRet;
}
	int Fon2(int * a, int * b)
	{
	int nRet;
	if(*a==0)
	*a=10;
	nRet = *a + *b;
	return nRet;
	}
		int Fon3(int &a, int &b)
		{
		int nRet;
		if(a==0)
		a=10;
		nRet = a + b;
		return nRet;
		}

Exécutez les trois fonctions tout en les améliorant:

```
int a, b, c, d, e;
a = 0;
b = 5;
c = appel de Fon1;
d = appel de Fon2;
e = appel de Fon3;
```

Arithmétique des pointeurs

- Il est possible d'effectuer des opérations arithmétiques sur les pointeurs.
- Les seules opérations valides sont l'addition et la soustraction.
- p + i = adresse contenue dans p + i*taille(élément pointé par p)
- p1 - p2 = adresse contenue dans p1 - adresse contenue dans p2
- Si p est un pointeur d'entier, p+1 est donc le pointeur sur l'entier qui suit immédiatement celui pointé par p.

A retenir: l'entier qu'on additionne au pointeur est multiplié par la taille de l'élément pointé pour obtenir la nouvelle adresse.

```
int a;          // création de la variable int a (sizeof(int) == 2)
int* p = &a; /* création du pointeur p sur int et initialisation à
                l'adresse de a */.
int* x = p++;    // création du pointeur int x (x == &a)
int* y = p;      // création du pointeur int y (y == &a + 2 octets)
int* z = ++p;    // création du pointeur sur int z (z == &a + 4 octets)
x = p--;         // (x == &a + 4 octets)
y = p;           // (y == &a + 2 octets)
z = --p;         // (z == &a)
// plus explicitement
x = p;           // (x == &a)
y = p + 1;       // (y == &a + 2 octets)
z = p + 1;       // (z == &a + 4 octets)
x = p;           // (x == &a + 4 octets)
y = p - 1;       // (y == &a + 2 octets)
z = p - 1;       // (z == &a)
```

Résultat d'une exécution

```
size of int est:2
EN A: val de p est:0x23bf25bc
EN A: val de x est:0x23bf25bc
EN A: val de y est:0x23bf25be
EN A: val de z est:0x23bf25c0
**************************************************
EN B: val de x est:0x23bf25c0
EN B: val de y est:0x23bf25be
EN B: val de z est:0x23bf25bc
**************************************************
en C: val de p est:0x23bf25bc
en C: val de x est:0x23bf25bc
en C: val de y est:0x23bf25be
en C: val de z est:0x23bf25be
**************************************************
en D: val de x est:0x23bf25bc
en D: val de y est:0x23bf25ba
en D: val de z est:0x23bf25ba
```

Pointeurs et tableaux

- **L'adresse du nième élément d'un tableau est calculée selon la formule :**

Adresse_n = Adresse_Base + n*taille(élément)

- **taille(élément) représente la taille de chaque élément du tableau.**
- **Adresse_Base l'adresse de base du tableau.**

N.B: Cette adresse de base est l'adresse du début du tableau, c'est donc à la fois l'adresse du tableau et l'adresse de son premier élément.

- **Si T est un tableau, alors &T[0] et T sont des expressions équivalentes.**

  ***(T)==T[0]**

  ***(T+1)== T[1]**

  ***(T+n)==T[n]**

- **Si M est une matrice, alors M et &M[0][0] sont des expressions équivalents.**

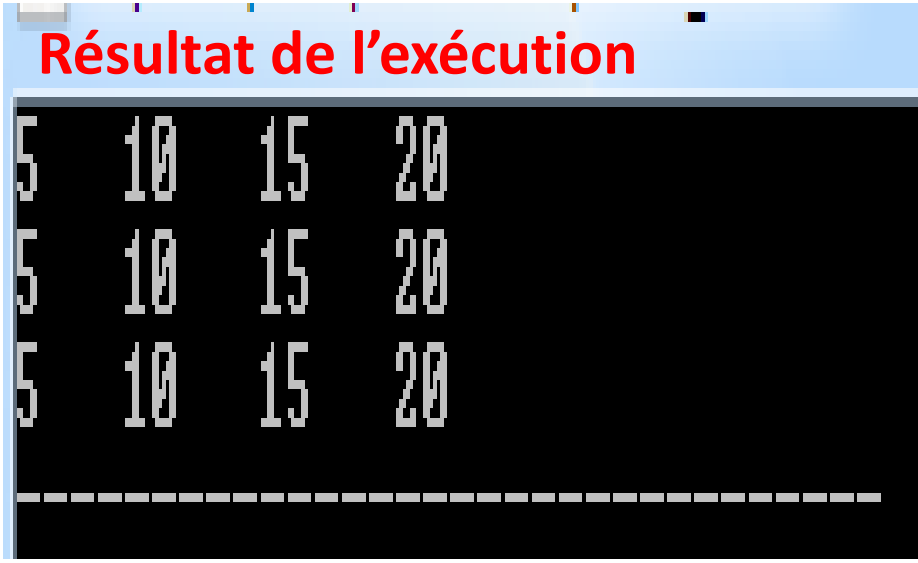
Exemple: **Accès aux éléments d'un tableau par pointeurs**

```
int tableau[100];

int *pi=tableau;

tableau[3]=5;       /* Le 4ème élément est initialisé à 5 */

*(tableau+2)=4;   /* Le 3ème élément est initialisé à 4 */

pi[5]=1;              /* Le 6ème élément est initialisé à 1 */
```

Exemple:

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

int main()
{ const int taille=4;
  int nombres[taille]={5,10,15,20};
 for (int i=0;i<taille;++i)
      cout<< nombres[i]<<"   ";
int *pnombres=&nombres[0];
cout<<endl;
for (int i=0;i<taille;++i)
      cout<< *(pnombres+i)<<"   ";
      cout<<endl;
for (int i=0;i<taille;++i)
      cout<< *(nombres+i)<<"   ";
return 0;
}
```

Résultat de l'exécution



Exemple de programme passant un tableau comme paramètres de fonctions

```
#include <iostream>
Using namespace std;
 const int MAX = 10;
 int Trouve_Min(int a[MAX], int taille);
 int Trouve_Max(int a[], int taille);
 main()
 { int tab[MAX];
 int n;
do {
 cout << "Préciser le nbre de données
 [2 à " << MAX << "] : ";
  cin >> n;  cout << "\n";
 } while (n < 2 || n > MAX);
 for (int i = 0; i < n; i++) {
 cout << "tab[" << i << "] : ";
 cin >> tab[i]; }
cout << "La plus petite valeur du tableau
     est : "  << Trouve_Min(tab, n) << "\n"
       << "La plus grande valeur du
     tableau est : "
<< Trouve_Max(tab, n) << "\n";
 return 0; }
```

```
int Trouve_Min(int a[MAX], int taille)
{ int petit = a[0];
for (int i = 1; i < taille; i++)
if (petit > a[i])
petit = a[i];
return petit;}
```

```
int Trouve_Max(int a[], int taille)
{int grand = a[0];
for (int i = 1; i < taille; i++)
if (grand < a[i])
grand = a[i];
return grand;}
```