**Définitions des Pointeurs**

Un pointeur est une variable permet de désigner – par un nom – un emplacement dans la mémoire. Il est aussi possible de désigner directement les cases mémoire par leur adresse. Un pointeur permet de stocker ces adresses mémoire.

Un pointeur est typé : on distingue les pointeurs sur un int, sur un double, sur une fraction etc.

**Comment Définir un pointeur ?**

type \* nom;

nom est un pointeur sur type, i.e. pouvant recevoir une adresse désignant un objet de type type.

* &V représente l’adresse mémoire d’une variable V.
* \*P désigne l’objet pointé par le pointeur P.

**Exemple 1**

i nt i , \* pi , k ;

i =3;

p i = &i ; // p i p o i n t e s u r i

k = (\* p i )+5; // i d e n t i q u e à : k=i+5

(\* p i ) = k+2; // i d e n t i q u e à : i = k+2

(\* p i )++; // i c i , i =11, k=8

**Déclaration**

* Déclaration d'un pointeur : int \* x;
* x est un pointeur vers un entier : x contient l'adresse en mémoire où est stocké un entier. Initialement le pointeur n'est pas initialisé : x vaut donc n'importe quelle adresse en RAM.

**L'opérateur &**

Si a est un entier, &a renvoie l'adresse réelle en mémoire de la variable a.

On peut donc écrire :

int a;

int \*x;

x=&a;

Dans ce bout de programme, on a copié l'adresse où est stockée en mémoire la variable a dans le pointeur x. Attention ! On a copié une adresse vers un entier, pas un entier.

On dit que x pointe vers la variable a.

**L'opérateur de déréférencement \***

Si x est un pointeur vers un entier, \*x sera l'entier pointé par x.  
Ainsi si on écrit :

int a=25;

int \*x;

x=&a;

\*x=25;

Comme x est un pointeur vers a, \*x désigne la variable a.

L'instruction \*x=25; copie un entier dans un autre et non une adresse.

**Exemple** 2 : un exemple d'utilisation de pointeurs

#include<iostream>

int main()

{

int a,b;

int \*x,\*y;

a=50;

b=80;

x=&a;

y=&b;

\*x=\*y;

cout << "a vaut : " << a << endl;

x=y;

\*x=1000;

cout << "b vaut : " << b << endl;

return 0;

}

**Explications**

* Dans cet exemple, on déclare deux entiers a et b et deux pointeurs vers des entiers x et y.
* a est initialisé à 50 et b à 80.
* L'instruction x=&a fait pointer x vers a.
* L'instruction y=&b fait pointer y vers b.
* L'instruction \*x=\*y copie l'entier pointé par y dans l'entier pointé par x c'est-à-dire copie b dans a. On a copié des entiers. a vaut donc 80.
* On affiche ensuite la valeur de a c'est-à-dire 80.
* L'instruction x=y copie le pointeur y dans le pointeur x : x pointe maintenant vers l'entier b (tout comme y d'ailleurs). On a copié des pointeurs.
* L'instruction \*x=1000 met la valeur 1000 dans l'entier pointé par x c'est-à-dire dans b. b vaut alors 1000.
* On affiche ensuite la valeur de b c'est-à-dire 1000.
* Dans cet exemple, il faut bien comprendre que parfois on copie des entiers, parfois on copie des pointeurs.

**Exécution**

Lorsqu'on exécute le programme voici ce qu'on obtient à l'écran :

a vaut : 80

b vaut : 1000

**Passage de paramètres par pointeurs**

* Lorsqu'on passe un pointeur en paramètre, on peut avoir un effet semblable au passage de paramètres par références.
* On préfère toutefois assez souvent le passage de paramètres par référence.
* Il faut connaître les 2 méthodes.

**Exemple 3 : passage de paramètres par pointeur**

#include<iostream>

using namespace std;

void minmax(int i, int j, int\* min, int\* max)

{

if(i<j) { \*min=i; \*max=j; }

else { \*min=j; \*max=i; }

}

int main()

{

int a, b, w, x;

cout << "Tapez la valeur de a : "; cin >> a;

cout << "Tapez la valeur de b : "; cin >> b;

minmax(a, b, &w, &x);

cout << "Le plus petit vaut : " << w << endl;

cout << "Le plus grand vaut : " << x << endl;

return 0;

}

**Explications**

* Dans cet exemple, on a une fonction minmax qui a comme paramètres 2 entiers i et j et 2 pointeurs vers des entiers min et max. Cette fonction trouve le plus petit de i et de j et le met dans l'entier pointé par min. Elle trouve le plus grand des 2 entiers et le copie dans l'entier pointé par max.
* Dans la fonction main(), on déclare 4 entiers a, b, w, et x. On demande à l'utilisateur de saisir au clavier les entiers a et b. Lors de l'appel de fonction minmax(a,b,&w,&x), on copie la valeur de a dans i, la valeur de b dans j. On copie la valeur de &w (un pointeur vers w) dans min et on copie &x (un pointeur vers x) dans max: min pointe donc vers w et max vers x. Lors de l'appel, on va donc récupérer dans w le plus petit des entiers a et b et dans x le plus grand de ces 2 entiers.

**Exécution**

Lorsqu'on exécute le programme voici ce qu'on obtient à l'écran :

Tapez la valeur de a : **25**

Tapez la valeur de b : **12**

Le plus petit vaut : 12

Le plus grand vaut : 25

**Exemple 4 : passage de paramètres par référence**

#include<iostream>

using namespace std;

void minmax(int i, int j, int& min, int& max)

{

if(i<j) { min=i; max=j; }

else { min=j; max=i; };

}

int main()

{

int a, b, w, x;

cout << "Tapez la valeur de a : "; cin >> a;

cout << "Tapez la valeur de b : "; cin >> b;

minmax(a, b, w, x);

cout << "Le plus petit vaut : " << w << endl;

cout << "Le plus grand vaut : " << x << endl;

return 0;

}

**Explications**

* + Au lieu d'utiliser un passage de paramètres par pointeur comme dans l'exemple 3, on peut bien sûr utiliser un passage de paramètres par référence.
  + Dans cet exemple, la fonction minmax possède 4 paramètres : 2 entiers i et j passés par valeur et 2 entiers min et max passés par référence. i et j sont les paramètres en entrée de la fonction minmax. min et max sont les paramètres en sortie de cette fonction.
  + Lors de l'écriture de la fonction minmax, on remarquera le symbole & placé après le type qui indique que le paramètre est passé par référence.
  + Lors de l'appel de minmax, on remarquera qu'il s'écrit minmax(a,b,w,x); sans symbole particulier. a et b sont passés par valeur et w et x sont passés par référence.

**Exécution**

Lorsqu'on exécute le programme voici ce qu'on obtient à l'écran :

Tapez la valeur de a : **20**

Tapez la valeur de b : **10**

Le plus petit vaut : 10

Le plus grand vaut : 20

**La taille des paramètres d'une fonction**

Lorsqu'on passe par valeur un paramètre à une fonction, ce paramètre est recopié juste avant l'appel. Si ce paramètre est une structure qui contient par exemple un tableau de taille importante, le temps nécessaire pour cette recopie peut être déraisonnable.

**Différentes solutions possibles**

Si un paramètre l de type T passé en paramètre à une fonction a une taille très importante, pour gagner du temps lors de l'appel on peut :

* passer ce paramètre par pointeur car la taille d'un pointeur est faible (en général 4 octets). Par contre cela donne la possibilité de modifier ce paramètre dans le corps de la fonction. Le paramètre sera de type \*T.
* passer ce paramètre par référence car la taille d'une référence est faible (en général 4 octets). Par contre cela donne également la possibilité de modifier ce paramètre dans le corps de la fonction. Le paramètre sera de type &T.
* ces 2 solutions sont problématiques car parfois il est intéressant d'empêcher le programmeur de modifier dans le corps de la fonction pour éviter les bugs.
* On peut également passer une référence vers une constante de type T : la taille du paramètre sera alors très petite (4 octets en général) et on peut empêcher le programmeur de modifier ce paramètre dans le corps de la fonction.
* Le paramètre de la fonction sera alors de type **const T &**.
* Cette solution sera désormais privilégiée pour passer à une fonction un paramètre de taille important en entrée uniquement.