

CHAPITRE 5

LES POINTEURS

Que ce soit dans la conduite de process, ou bien dans la programmation orientée objet, l'usage des pointeurs est extrêmement fréquent en C++.

L'OPERATEUR ADRESSE &

L'opérateur adresse & retourne l'adresse d'une variable en mémoire.

Exemple: **int i = 8;**
 cout<<"VOICI i:"<<i;
 cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL:"<<&i;

Exercice V 1: Exécuter l'exemple précédent, et indiquer les cases-mémoire occupées par la variable i.

LES POINTEURS

Définition: Un pointeur est une adresse mémoire. On dit que le pointeur pointe sur cette adresse.

DECLARATION DES POINTEURS

Une variable de type pointeur se déclare à l'aide du type de l'objet pointé précédé du symbole * (**opérateur d'indirection**).

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

Exercice V 1: Modifier l'exercice 1 comme ci-dessous, et conclure :

int i = 8;
int *p; // déclaration d'un pointeur
cout<<"VOICI i:"<<i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL:"<<&i;

p = &i; // p pointe sur l'adresse de i
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL:"<<p;

MANIPULATION DE LA CASE MEMOIRE

L'opérateur * désigne en fait le contenu de la case mémoire pointée.

Exercice V_1: Modifier l'exercice 1 comme ci-dessous, et conclure :

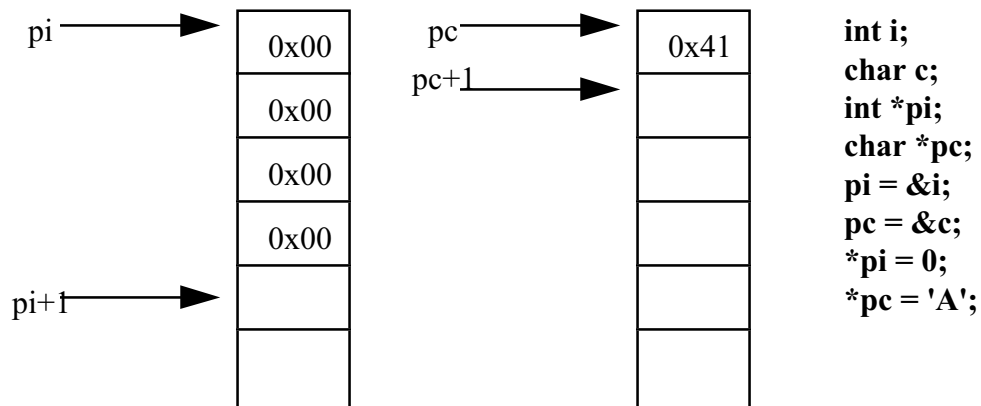
```
int i = 8;
int *p;
cout<<"VOICI i:"<<i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL:"<<&i;

p = &i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL:"<<p;
cout<<"\nVOICI i:"<<*p;

*p = 23;
cout<<"\nVOICI i:"<<i;
cout<<"\nVOICI i:"<<*p;
```

ARITHMETIQUE DES POINTEURS

On peut essentiellement **déplacer** un pointeur dans un plan mémoire à l'aide des opérateurs d'addition, de soustraction, d'incréméntation, de décréméntation. 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**.



Exemples:

```
int *pi;           // pi pointe sur un objet de type entier
float *pr;         // pr pointe sur un objet de type réel
char *pc;          // pc pointe sur un objet de type caractère

*pi = 421;         // 421 est le contenu de la case mémoire p et des 3 suivantes
*(pi+1) = 53;      // on range 53, 4 cases mémoire plus loin
*(pi+2) = 0xabcd;   // on range 0xabcd 8 cases mémoire plus loin
*pr = 45.7;        // 45,7 est rangé dans la case mémoire r et les 3 suivantes
pr++;              // incrémente la valeur du pointeur r (de 4 cases mémoire)
                  // manipulation déconseillée car elle déstructure le programme.
                  // pour assurer une maintenance facile du programme,
                  // il vaut mieux ne pas modifier la valeur initiale des pointeurs
*pc = 'j';          // le contenu de la case mémoire c est le code ASCII de 'j'
pc--;              // décrémente la valeur du pointeur c (d'une case mémoire)
                  // de même, cette manipulation est déconseillée.
```

AFFECTATION D'UNE VALEUR A UN POINTEUR ET ALLOCATION DYNAMIQUE

Lorsque l'on déclare une variable char, int, float un nombre de cases mémoire bien défini est **réservé** pour cette variable. Il n'en est pas de même avec les pointeurs.

Exemple:

```
char *pc;
```

Si on se contente de cette déclaration, le pointeur ne pointe « nulle part ». Son usage, tel que, dans un programme conduit à un « plantage » du système d'exploitation.

L'affectation des cases mémoires n'ayant pas été faite correctement, on risque d'utiliser des adresses non autorisées.

Exercice V_2 : Tester le programme ci-dessous et conclure

```
char *pc;
*pc = 'a';        // le code ASCII de a est rangé dans la case mémoire pointée par c
*(pc+1) = 'b';    // le code ASCII de b est rangé une case mémoire plus loin
*(pc+2) = 'c';    // le code ASCII de c est rangé une case mémoire plus loin
*(pc+3) = 'd';    // le code ASCII de d est rangé une case mémoire plus loin
cout<<"Valeurs : "<<*pc<<" "<<*(pc+1)<<" "<<*(pc+2)<<" "<<*(pc+3);
```

1ère méthode : utilisation de l'opérateur adresse :

C'est la méthode qui a été utilisée dans l'exercice V_1. Dans ce cas, l'utilisation de pointeurs n'apporte pas grand-chose par rapport à l'utilisation classique de variables.

2ème méthode : affectation directe d'une adresse :

Cette méthode est réservée au contrôle de processus, quand on connaît effectivement la valeur de l'adresse physique d'un composant périphérique.

Elle ne fonctionne pas sur un PC, pour lequel les adresses physiques ne sont pas dans le même plan mémoire que les adresses mémoires.

Il faut utiliser l'opérateur de "cast", jeu de deux parenthèses.

```
char *pc;  
pc = (char*)0x1000; // p pointe sur l'adresse 0x1000
```

3ème méthode : allocation dynamique de mémoire :

C'est la méthode la plus utilisée et qui donne pleinement leur intérêt aux pointeurs par rapport aux variables classiques.

Via l'opérateur **new**, natif dans le C++, on réserve de la place dans la mémoire pour le pointeur. L'adresse de base est choisie par le compilateur en fonction du système d'exploitation. Le programmeur n'a à se soucier que de la quantité de cases mémoire dont il a besoin.

Exercice V_2 : Modifier l'exercice V_2 comme ci-dessous :

```
char *pc;  
pc = new char[4]; // réservation de place dans la mémoire pour 4 char  
  
*pc = 'a'; // le code ASCII de a est rangé dans la case mémoire pointée par c  
*(pc+1) = 'b'; // le code ASCII de b est rangé une case mémoire plus loin  
*(pc+2) = 'c'; // le code ASCII de c est rangé une case mémoire plus loin  
*(pc+3) = 'd'; // le code ASCII de d est rangé une case mémoire plus loin  
cout<<"Valeurs : "<<*pc<<" "<<*(pc+1)<<" "<<*(pc+2)<<" "<<*(pc+3);  
  
delete pc; // libération de place
```

Remarques :

- L'allocation dynamique peut se faire n'importe quand dans le programme (avant d'utiliser le pointeur !).
- L'opérateur **delete** permet de libérer la place réservée quand elle devient inutile.
- L'allocation dynamique peut se faire dès la déclaration du pointeur avec la syntaxe suivante :

```
char *pc = new char[4];  
  
- Si on a besoin d'une seule cases mémoire, on écrira :  
char *pc; ou bien char *pc = new char;  
pc = new char ;
```

Réservation de place pour d'autres types de pointeurs :

```
char *pc;  
int *pi,*pj;  
float *pr;  
pc = new char[10]; // on réserve de la place pour 10 caractères, soit 10 cases mémoires  
pi = new int[5]; //on réserve de la place pour 5 entiers, soit 20 cases mémoire  
pj = new int; //on réserve de la place pour 1 entier, soit 4 cases mémoire  
pr = new float[6]; // on réserve de la place pour 6 réels, soit 24 cases mémoire  
  
delete pc; //on libère la place précédemment réservée pour c  
delete pi; //on libère la place précédemment réservée pour i  
delete pr; // on libère la place précédemment réservée pour r
```

Comme précédemment, l'allocation dynamique peut être faite dès la déclaration du pointeur :

```
int *pi = new int[5];
```

Exercice V 3:

adr1 et adr2 sont des pointeurs pointant sur des réels. Le contenu de adr1 vaut -45,78; le contenu de adr2 vaut 678,89. Ecrire un programme qui affiche les valeurs de adr1, adr2 et de leur contenu.

L'opérateur de "cast", permet d'autre part, à des pointeurs de types différent de pointer sur la même adresse.

```
Exemple: char *pc; // pc pointe sur un objet de type caractère  
int *pi; // pi pointe sur un objet de type entier  
pi = new int; // allocation dynamique pour i  
pc = (char*)i; // c et i pointent sur la même adresse, c sur un caractère
```

Exercice V 4:

adr_i est un pointeur de type entier; son contenu i vaut 0x41424344. A l'aide d'une conversion de type de pointeur, écrire un programme montrant le rangement des 4 octets en mémoire.

Exercice V 5:

Saisir un texte de 10 caractères. Ranger les caractères en mémoire. Lire le contenu de la mémoire et y compter le nombre de lettres e .

Exercice V 6:

Saisir 6 entiers et les ranger à partir de l'adresse adr_deb. Rechercher le maximum, l'afficher ainsi que sa position.

La place nécessaire dans la mémoire peut-être le résultat d'un calcul :

```
int taille;
char *image;
cout<<"\nQuelle est la taille de l'image ? (en octets)";
cin>>taille;
image = new char[taille];
```

Exercice V_7:

Reprendre l'exercice V_6 mais en demandant à l'utilisateur combien de nombres il souhaite traiter, et en réservant en mémoire la place nécessaire.

CORRIGE DES EXERCICES

Exercice V_3:

```
#include <iostream.h>
#include <conio.h>

void main()
{
float *adr1 = new float,*adr2 = new float;

*adr1 = -45.78;
*adr2 = 678.89;
cout<<"adr1 = "<<adr1<<" adr2 = "<<adr2<<" r1 = "<<*adr1<<" r2 = "<<*adr2;
delete adr1;
delete adr2;
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
getch();
}
```

Exercice V_4:

```
#include <iostream.h>
#include <conio.h>
void main()
{
char *adr_c;
int *adr_i = new int;

char u;
```

```

*adr_i = 0x41424344;
adr_c = (char*)adr_i;

u = *adr_c;
cout<<" CONTENU: "<<u<<"\n";

u = *(adr_c+1);
cout<<" CONTENU: "<<u<<"\n";

u = *(adr_c+2);
cout<<" CONTENU: "<<u<<"\n";

u = *(adr_c+3);
cout<<" CONTENU: "<<u<<"\n";

cout<<"Frapper une touche" ;
getch();
}

```

L'analyse de l'exécution de ce programme, montre que les microprocesseurs INTEL rangent en mémoire d'abord les poids faibles d'une variable.

Exercice V_5:

```

#include <iostream.h>
#include <conio.h>

void main()
{
char *adr_deb,c;
int i,imax,compt_e = 0;
adr_deb = new char[10]; // texte de 10 caracteres

// saisie et rangement du texte

cout<<"\nEntrer 10 caracteres separes par return:\n";
for (i=0;i<10;i++)
{
cout<<"caracteres numero "<<i<<": ";
cin>>c;
*(adr_deb + i) = c;
}

// lecture de la memoire et tri

```

```

for (i=0;i<10;i++)
{
    c = *(adr_deb+i);
    cout<<"\nCARACTERE:"<<c;
    if (c=='e') compt_e++;
}

// resultats
cout<<"\nNOMBRE DE e: "<<compt_e;
delete adr_deb;
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
getch();
}

```

Exercice V_6:

```

#include <iostream.h>
#include <conio.h>
void main()
{
    int *adr_deb,i,max,position;
    adr_deb=new int[6];

    // saisie des nombres
    cout<<"SAISIE DES NOMBRES:\n";
    for(i=0;i<6;i++)
    {
        cout<<"ENTRER UN NOMBRE: ";
        cin>>*(adr_deb+i);
    }

    // tri
    max = *adr_deb;
    position = 1;

    for(i=0;i<6;i++)
    {
        if(*(adr_deb+i)>max)
        {
            max = *(adr_deb+i);
            position = i+1;
        }
    }

    // resultats

```



```

cout<<"LE MAXIMUM EST "<<max<<" IL EST EN "<<position<<"EME
POSITION\n";
delete adr_deb;
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE";
getch();
}

```

Exercice V_7:

```

#include <iostream.h>
#include <conio.h>
void main()
{
int *adr_deb,i,max,position,nombre;

cout<<"Combien de nombres voulez-vous traiter ?";
cin>>nombre;
adr_deb=new int[nombre];

// saisie des nombres
cout<<"SAISIE DES NOMBRES:\n";
for(i=0;i<nombre;i++)
{
    cout<<"ENTRER UN NOMBRE: ";
    cin>>*(adr_deb+i);
}

// tri
max = *adr_deb;
position = 1;

for(i=0;i<nombre;i++)
{
if(*(adr_deb+i)>max)
{
    max = *(adr_deb+i);
    position = i+1;
}
}

// resultats
cout<<"LE MAXIMUM EST "<<max<<" IL EST EN "<<position<<"EME
POSITION\n";
delete adr_deb;

```

```
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE";  
getch();  
}
```