Chapitre V : Tableaux et Pointeurs

Objectif du chapitre :

Déclaration et utilisation des Tableaux

Sommaire :

Références :

Cours de C/C++ [Ouvrage] / aut. Casteyde Christian. - 2001. - Vol. Version 1.40.1 : p. 501.
Programmez avec le langage C++ [Ouvrage] / aut. Mathieu Nebra et Matthieu Schaller. - [s.l.] : le Livre du Zéro, 2015.
Site web : https://www.fresnel.fr/perso/stout/langage_C

1. Introduction

Les variables, manipuler par un langage de programmation peuvent être regroupées en 2 classes :

- Variable simple : Permette le stockage d'une seule valeur à la fois.

Variable complexe : Permette le stockage de plusieurs valeurs de même type comme les tableaux, de différents types comme les structures….

2. Tableaux

On appelle *tableau* une variable composée de données de même type, stockée de manière contiguë en mémoire (les unes à la suite des autres). Un tableau est donc une suite de cases (espace mémoire) de même taille. La taille de chacune des cases est conditionnée par le type de donnée que le tableau contient.

2. 1. Tableaux unidimensionnels

Un tableau unidimensionnel est un tableau qui contient des éléments simples (des éléments qui ne sont pas des tableaux). Un tableau unidimensionnel est donc une suite de « cases » de même taille contenant des éléments d'un type donné (de la longueur de la case en quelque sorte).

a- déclaration: type nom_tab [nbr_elts]

- " type" : définit le type des éléments que contient le tableau.
- " nom_tab" : représente le nom choisi par l'utilisateur du tableau.
- "nbr_elts" : nombre entier qui détermine le nombre de cases que le tableau doit comporter.

Exemple :

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

int main()
{
    int t[10], i;
    for(i=0; i<10; i++)
    {
        cout << "Tapez la valeur numero " << i << " : ";
        cin >> t[i];
    }
    for(i=0; i<10; i++) t[i] = t[i]+1;
    for(i=0; i<10; i++) cout << "La valeur numero " << i <<" est : "<< t[i] <<endl;
    return 0;
}
```

Explications

Dans ce programme, nous allons tout d'abord saisir une à une le contenu des 10 cases d'un tableau t. Ensuite nous allons effectuer un traitement simple sur ce tableau : nous allons incrémenter de 1 le contenu de chaque case. Finalement, nous afficherons le contenu final de chaque case du tableau.

Dans ce programme, nous commençons par définir un tableau t de 10 cases de type entier. La première case de

ce tableau sera t[0],… et la dernière t[9].

• La première boucle for permet de saisir une à une les cases du tableau :

for(i=0;i<10;i++) { cout<<"Tapez la

valeur numero "<<i<<" : "; cin>>t[i]; } remarque : la première valeur de i pour laquelle le corps du for sera

effectué sera i=0, la dernière i=9

• for (i=0;i<10;i++)t[i]=t[i]+1;

Dans cette boucle for, on augmente de 1 le contenu de chaque case du tableau.

• for (i=0;i<10;i++) cout<<"La valeur numero "<<i<<" est : " <<t[i]<<endl;

On affiche une à une le contenu des cases du tableau.

• **Exécution**

```
Tapez la valeur numero 0 : 5
Tapez la valeur numero 1 : 2
Tapez la valeur numero 2 : 50
Tapez la valeur numero 3 : 10
Tapez la valeur numero 4 : 20
Tapez la valeur numero 5 : 60
Tapez la valeur numero 6 : 80
Tapez la valeur numero 7 : 90
Tapez la valeur numero 8 : 10
Tapez la valeur numero 9 : 10
La valeur numero 0 est 6
La valeur numero 1 est 3
La valeur numero 2 est 51
La valeur numero 3 est 11
La valeur numero 4 est 21
La valeur numero 5 est 61
La valeur numero 6 est 81
La valeur numero 7 est 91
La valeur numero 8 est 11
La valeur numero 9 est 11
```

remarques :

- Les index des éléments d'un tableau vont de **0** à **nbr_elts -1** .

- La taille d'un tableau doit être connue statiquement par le compilateur. Impossible donc d'écrire int tab[n] et n est une variable inconnu.

- Il est recommandé de donner un nom à la constante nbr_elts par la directive du préprocesseur « define » : #define taille 10

int tab[taille]

b- initialisation : type nom-tab [taille]= {liste de valeurs }

exp : int tab [3]={ 3,84,0}

alors : tab [0]=_3 ;tab[1]=85 ;tab[2]=0 ;

c- exercice d'application : Soit un programme qui recherche le plus petit élément dans un tableau contenant 4 cases.

algorithme utilisé : le plus petit élément du tableau est stocké dans une variable **ppt**. On commence par initialiser **ppt** au premier élément du tableau (d'indice zéro). On parcourt alors tous les autres éléments du tableau en comparant l'élément courant à **ppt** et en mettant à jour la valeur de **ppt**.

solution :

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

int main()
{
    int t[4], i, ppt;
    for(i=0; i<4; i++)
    {
        cout << "Tapez la valeur numéro " << i << " : ";
        cin >> t[i];
    }
    ppt = t[0];
    for(i=1; i<4; i++) if(ppt>t[i]) ppt=t[i];
    cout << "La plus petite valeur est "<< ppt <<endl;
    return 0;
}
```

Explications

- Pour calculer le plus petit élement d'un tableau, on initialise ppt à t[0].

- On parcourt ensuite le tableau de la case 1 à la dernière case d'indice 3 en comparant t[i] à ppt. Si t[i] est plus petit que le plus petit courant ppt alors on copie t[i] dans ppt (t[i] devient alors le nouveau plus petit courant).

Dans le cas contraire, on ne modifie pas la valeur de ppt.

- On affiche finalement la valeur de ppt en dehors de la boucle.

Exécution

```
Tapez la valeur numero 0 : 9
Tapez la valeur numero 1 : 7
Tapez la valeur numero 2 : 11
Tapez la valeur numero 3 : 15
La plus petite valeur est 7
```

2. 2. Tableaux multidimensionnels

En C++, un tableau multidimensionnel est considéré comme un tableau dont ces éléments sont eux-mêmes des tableaux.

La syntaxe générale utilisée est la suivante : type nom_tab [tailles][taille2] ... [taile n]

exemple : si on veut déclarer un tableau de 5 lignes et 4 colonnes, il faut déclarer :

int a[5][4];

On accède alors à l'élément ligne i colonne j de la manière suivante : a[i][j]=99;

Dans cet exemple, i doit être compris entre 0 et 4 (bornes incluses) et j entre 0 et 3 (bornes incluses).

exemple :

```
#include <iostream>
using namespace std;
const int N = 2;
const int M = 3;

int main()
{
    int i, j;
    int t[N][M];

    for(i=0; i<N; i++)
      for(j=0; j<M; j++)
      {
        cout<<"Tapez t["<< i <<"]["<< j <<"] :";
        cin >> t[i][j];
      }

    cout<<"Voici le tableau :"<<endl;
    for(i=0; i<N; i++)
    {
      for(j=0; j<M; j++) cout<< t[i][j] <<" ";
      cout<<endl;
    }
    return 0;
}
```

Explications

• Dans cet exemple, on déclare un tableau t d'entiers comportant 2 lignes et 3 colonnes.

• Par 2 boucles imbriquées, on saisit un à un les 6 éléments du tableau.

• On affiche ensuite le contenu du tableau.

• **Exécution**

Tapez t[0][0] : 3

Tapez t[0][1] : 4

Tapez t[0][2] : 5

Tapez t[1][0] : 6

Tapez t[1][1] : 8

Tapez t[1][2] : 4

Voici le tableau :

3 4 5

6 8 4

3. Pointeurs

Un pointeur est invariable qui contient l'adresse d'une donnée ou d'un ensemble de données.

a- déclaration : 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.

b- 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. On dit que x pointe vers la variable a.

c-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;
```

d- exemple d'application

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

int main()
{
    int a;
    int *x,*y;

    a=90;
    x=&a;
    cout << *x << endl;

    *x=100;
    cout << "a vaut : " << a << endl;

    y=x;
    *y=80;
    cout << "a vaut : " << a << endl;

    return 0;
}
```

Explications

- dans cet exemple, on définit un entier a et deux pointeurs vers des entiers x et y.

a est initialisée à la valeur 90.

- après l'instruction x=&a, x pointe vers a. x contient l'adresse en mémoire où est stockée la variable a.

- l'instruction *x=100; modifie le contenu de la variable a et met la valeur 100 dans cette variable.

- l'instruction y=x copie le pointeur x dans le pointeur y. Après cette instruction, les deux pointeurs x et y pointent vers la même variable a.

- lorsqu'on écrit *y=80, on modifie alors le contenu de la variable a qui vaut alors 80.

- comme conclusion ; un pointeur permet de modifier indirectement le contenu d'une variable.

Exécution

```
90
a vaut : 100
a vaut : 80
```