Programmation C / C++

#3 Basiques (2)

par David Albert

GEMA
GRANDES ÉCOLES DES MÉTIERS D'AVENIR

2024

Table des matières

01

Pointeurs

Pointeurs

Définition

Un pointeur est une variable dont la valeur est une adresse mémoire.

Déclaration

La déclaration d'une variable de type pointeur se fait comme suit :

```
Type *nomVar;
```

i

On peut définir des pointeurs sur n'importe quel type.

- types de base : int, char, float, ...
- types composés : tableaux, structures, ...
- fonctions

Opérateurs

- l'opérateur d'indirection ***p** qui retourne le contenu (valeur de la variable pointée)
- l'opérateur d'adressage **&p** qui donne l'adresse d'une variable

Exemple

```
/* définit un pointeur sur un entier */
int *ptr;
int i=10;

/* initialisation du pointeur */
ptr=&i;

/* accès au contenu par l'opérateur
d'indirection (équivaut à i=i+1) */
(*ptr)++;
```

Pointeurs

Exercice

Que donne le programme suivant ?

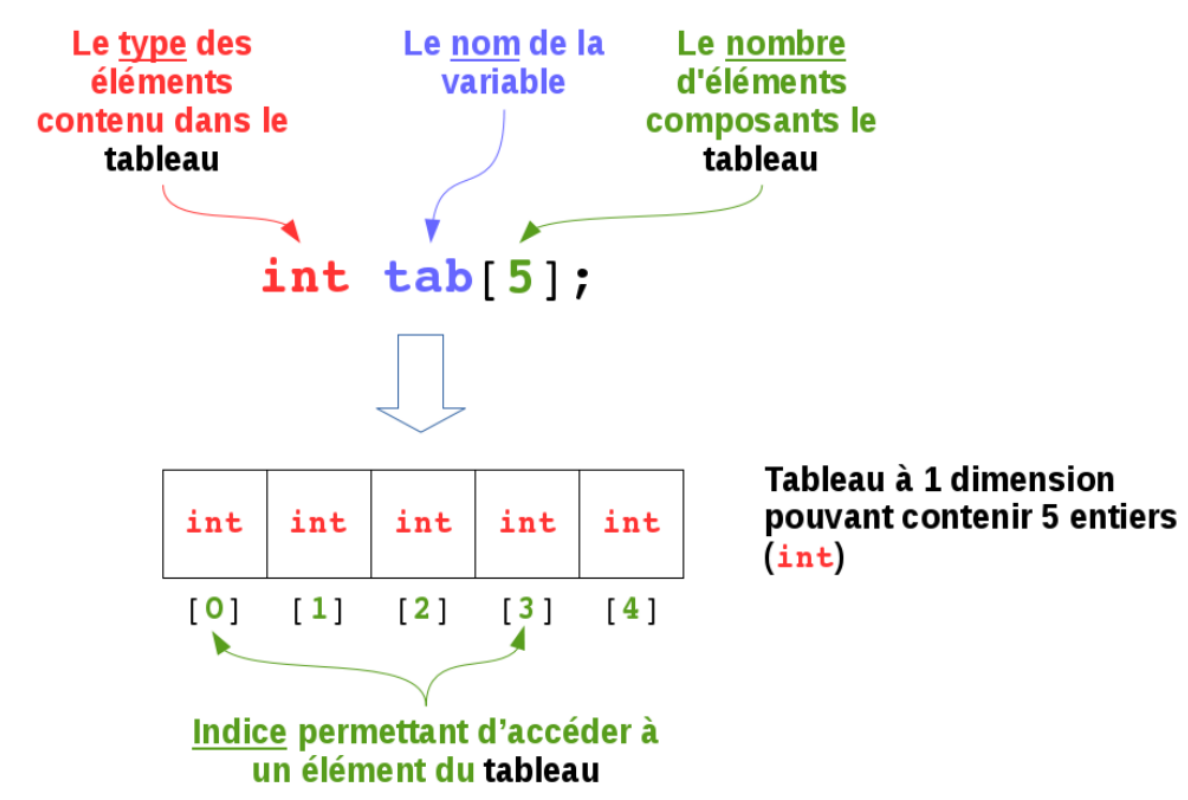
```cpp
#include <iostream>

using namespace std;

int main()
{
    int p1 = 10;
    int *p2 = &p1;

    cout << p1 << std::endl;

    *p2 += 10;
    cout << p1 << std::endl;
}
```

02

Les tableaux

Tableaux

Definition :
Un **tableau** est **un ensemble d'éléments de même type** désignés par un identificateur unique. Chaque élément est repéré par une valeur entière appelée **indice** (ou index) indiquant sa position dans l'ensemble.



!

L'indiçage démarre toujours à partir de 0.

Tableaux

Déclaration de tableaux

La forme habituelle de déclaration d'un tableau est la suivante :

`type identificateur[dimension1]... [dimensionn]`

Exemples

```
int notes[1000]; // un tableau de 1000 int non initialisé

float notes[1000]; // un tableau de 1000 float non initialisé

int notes[1000] = {0}; // un tableau de 1000 int initialisé à 0

float f[1000] = {0.}; // un tableau de 1000 float initialisé à 0

int coefficients[4] = { '1', '2', '2', '4' }; // un tableau de 4 entiers

// La dimension d'un tableau peut être omise si le compilateur peut en définir la valeur
float t[] = {2., 7.5, 4.1}; // tableau de 3 éléments
```

Tableaux à plusieurs dimensions

Déclaration de tableaux à plusieurs dimensions

```
// tableau à 2 dimensions de 2 lignes et 5 colonnes :
int m[2][5] = { 2, 6, -4, 8, 11, 3, -1, 0, 9, 2 }; // initialise avec des valeurs


// tableau à 2 dimensions pour stocker plusieurs chaînes de caractères
char noms[][16] = { {"robert"}, {"roger"}, {"raymond"}, {"alphonse"} };

int x[5][12][7]; // tableau a 3 dimensions, rarement au-delà de cette dimension
```

!

Segmentation fault

Le plus grand danger dans la manipulation des tableaux est d'accéder en écriture en dehors du tableau. Cela provoque un accès mémoire interdit qui provoquera une exception de violation mémoire (**segmentation fault**).

Tableaux et pointeurs

⚠️ Note Importante

L'identificateur du tableau ne désigne pas le tableau dans son ensemble, mais l'adresse en mémoire du début du tableau (l'adresse de la première case).

Conséquence 1

Soit un tableau `tab` :

```
int tab[] = {1, 2, 3};
```

Modification de la **première** valeur du tableau :

```
tab[0] = 5;
```

⟺

```
*tab = 5;
```

Modification de la **deuxième** valeur du tableau :

```
tab[1] = 3;
```

⟺

```
*(tab + 1) = 3;
```

Modification de la **troisième** valeur du tableau :

```
tab[2] = 1;
```

⟺

```
*(tab + 2) = 1;
```

Conséquence 2

i

Lorsque l'on passe un tableau en paramètre d'une fonction, il n'est pas possible de connaître sa taille et il faudra donc lui passer aussi sa taille.

Tableaux et pointeurs

Exercice

Que donne le programme suivant ?

```cpp
#include <iostream>

using namespace std;

int main()
{
    int tab[3] = {1, 2, 3};
    int *p3 = tab;

    cout << tab[0] << tab[1] << tab[2] << std::endl;
    cout << *p3 << std::endl;

    *(p3 + 1) = 10;
    cout << tab[0] << tab[1] << tab[2] << std::endl;
    cout << tab[1] << std::endl;
    cout << *(p3 + 2) << std::endl;

    return 0;
}
```

03

Chaînes de caractères

Les caractères

Type `char`

Exemples valeurs

`'A'`, `'C'`, `'\n'`, `';'`, `'!'`, `'0'`, `'1'`, ...

Déclaration

```
char a = 'A';
char newLine = '\n';
```

Représentation en machine

- Code ASCII
- unicode

Table ASCII

ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(	0010 1000
N	0100 1110	l	0110 1100	)	0010 1001
				space	0010 0000

Les chaînes de caractères

Définition : Un tableau de caractères.

Exemples :
`"Brice"`, `"la vie est belle !"`, `"#0$£ù%&-"`, `""`, `"\n\t\t\t\n"`

En C

Il n'existe pas de type spécifique pour traiter les chaînes de caractères.

Déclaration

```c
char chaine[100];
char salut[10] = {'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0' };
char salut[10] = "bonjour";
```

En C++

Il existe le type **string** pour manipuler des chaînes de caractères.

Type `string`

Déclaration

```cpp
string prenom = "Robert";
string nom("Guillon");
```

04

Fonctions

Les fonctions

Définition

Une fonction est une suite d'instructions qui peuvent être réutilisées à différents endroits dans notre programme.

Déclaration

Une fonction se caractérise par :

- un **nom**
- une **liste de paramètre(s)** et leurs types
- un **type de retour**

Exemple

```
// Le prototype de la fonction calculeNombreDeSecondes :
int calculeNombreDeSecondes(int heures, int minutes, int secondes);

// Soit :
// - son nom : calculeNombreDeSecondes
// - sa liste de paramètre(s) : elle reçoit 3 int
// - son type de retour : int
```

i

Si une fonction ne retourne aucune valeur. Alors le type de retour est **void**

```
void showMessage(string msg);
```

⇒ Ce type de fonction est appelée **procédure**

Les fonctions

En C/C++, il faut distinguer :

- la **déclaration** qui est une instruction fournissant au compilateur un certain nombre d'informations concernant une fonction

```
int plus(int, int); // fichier en-tête (.h)
```

- la **définition** qui revient à écrire le corps de la fonction

```
int plus(int a, int b) { return a + b; } // fichier source (.c ou .cpp)
```

- l'**appel** qui est son utilisation

```
int res = plus(2, 2); // fichier source (.c ou .cpp)
```

!

Pour être utilisée, une fonction doit être définit avant son utilisation. Sinon, le compilateur génèrera une erreur : ***'...' was not declared in this scope.***

Exemple

Dans le fichier ***temps.h***

```c
#ifndef __TEMPS_H__ // si l'étiquette __TEMPS_H__ n'est pas défini
#define __TEMPS_H__  // alors on définit l'étiquette __TEMPS_H__

int calculNbrSecondes(int heures, int minutes, int secondes);

#endif /* finsi __TEMPS_H__ */
```

i

La définition d'une macro `__TEMPS_H__` est très importante. Elle évite les déclarations multiples de la fonction `calculNbrSecondes`.

Dans le fichier ***temps.c***

```c
#include "temps.h"

// La définition de la fonction calculeNombreDeSecondes :
int calculNbrSecondes(int heures, int minutes, int secondes)
{
    return ((heures*3600) + (minutes*60) + secondes);
}
```

Dans le fichier ***main.c***

```c
#include "temps.h"

int main(){
    // Vous pouvez maintenant utiliser (appeler)
    // la fonction calculNbrSecondes :
    int s = calculNbrSecondes(1, 0, 0);
    return 0;
}
```

05

Types composés

Types composés

Les **types composés** permettent de regrouper des variables au sein d'une même entité.

- variables de même types : **tableaux**
- variables de types différents : **structures de données**

En C, il existe 3 types de structures de données :

- les structures (**struct**)
- les unions (**union**)
- les champs de bits

En C++, la notion de type **classe** est ajoutée. Elle permet de réaliser des programmes orientés objet (POO).

Structures

Une **structure** permet de grouper un certain nombre de variables de types différents au sein d'une même entité.

Exemple :

```
typedef struct
{
    int id;
    int age;
    float salary;
} Personne;
```

> i
>
> En C++, le **typedef** est inutile.

Accès aux membres

Pour accéder aux membres d'une structure, on utilise :

- l'opérateur d'accès **.** (point)
  pour une variable de type structuré

```
Personne jean = {9467, 37, 1140.6};

float salary = jean.salary;
```

- l'opérateur d'accès **->** (flèche)
  pour un pointeur sur un type structuré

```
Personne jean = {9467, 37, 1140.6};
Personne *p_jean = &jean;

float salary = jean->salary;
```

Union

Une **union** est conceptuellement identique à une structure mais peut, à tout moment, contenir n'importe lequel des différents champs.

i

Une union définit en fait plusieurs manières de regarder le même emplacement mémoire.

Exemple :

```c
typedef union mesureCapteur
{
    int iVal;
    float fVal;
    char cVal;
} Capteur;

int main()
{
    Capteur vitesseVent, temperatureMoteur;
    pressionAtmospherique.iVal = 1013; /* un int */
    temperatureMoteur.fVal = 50.5; /* un float */
    return 0;
}
```

Références

http://tvaira.free.fr/bts-sn/poo-c++/cours-poo-c++/cours-c-c++.pdf