Programmation Langage C
Les tableaux

Youssef ALJ

19 mars 2020

1 Tableaux en C
- Déclaration et initialisation
- Bonnes pratiques pour l'utilisation des tableaux

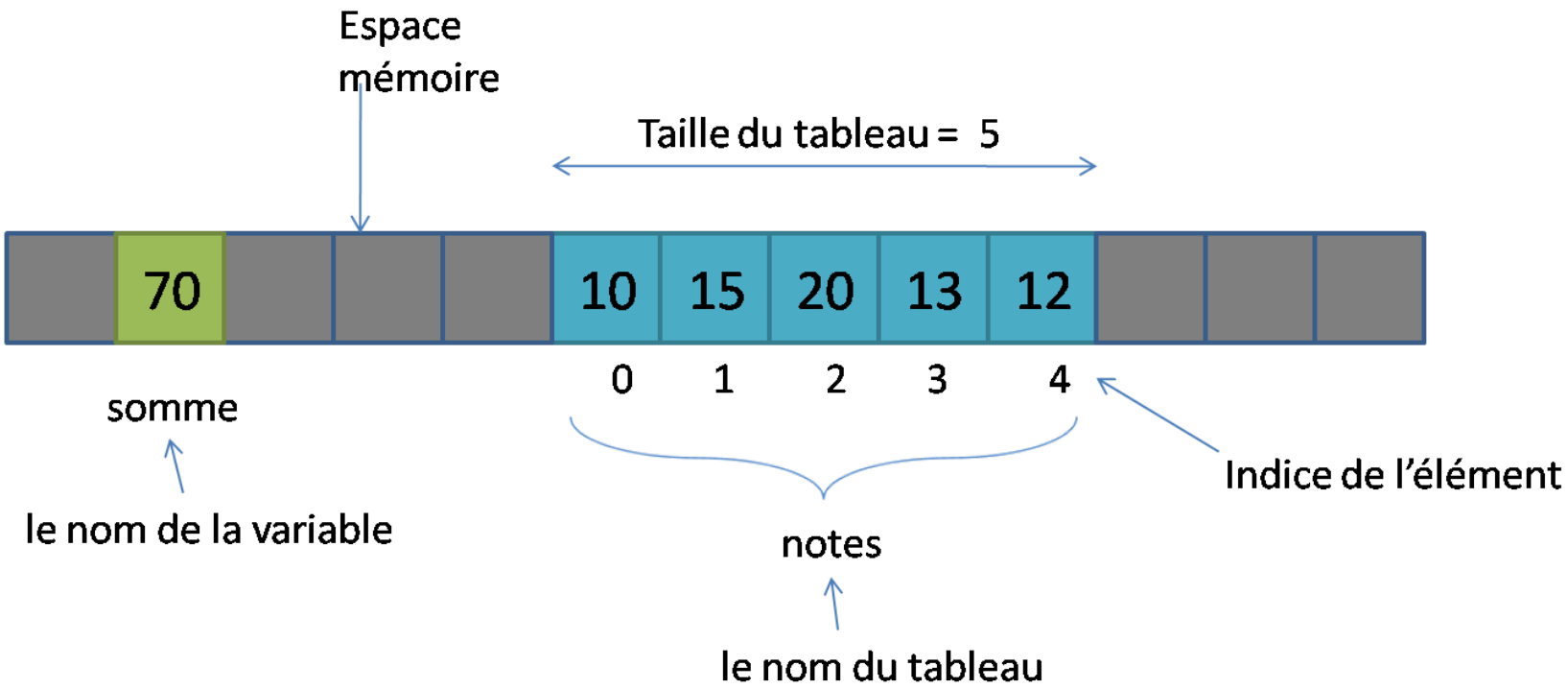
Pourquoi a-t-on besoin de tableaux ?

- On veut un programme qui demande à l'utilisateur de saisir les notes de 100 étudiants et qui calcule leur moyenne.
- La solution naïve : il faut déclarer 100 variables de types `float` et calculer leur moyenne.
- TROP COMPLIQUÉ !!!
- Code trop long et très facile de commettre des erreurs.
- Une solution : utilisation de tableaux.

Représentation des tableaux en mémoire

- Les tableaux en C sont représentés dans une zone continue de la mémoire (cf. slide suivant).
- Pour accéder à un élément particulier du tableau on utilise une valeur appelée **indice** (cf. slide suivant).
- l'indice d'un tableau commence par 0 et se termine par $N - 1$ où N est la taille du tableau (cf. slide suivant).
- Dans l'exemple du slide suivant l'indice du tableau varie de 0 à 4.

Exemple


Espace
mémoire
Taille du tableau = 5
70
10
15
20
13
12
0
1
2
3
4
somme
le nom de la variable
notes
le nom du tableau
Indice de l'élément

Déclaration d'un tableau

```
type_de_donnees nom_du_tableau[TAILLE];
```

Avec :

- `type_de_donnees` : est un type de variable à choisir parmi `int, float, char` etc.
- `nom_du_tableau` : est un nom qu'on donne à notre tableau.
- `TAILLE` : est une constante qui définit la taille du tableau.

Exemple :

```
int notes[5];
```

On distingue deux types d'initialisation : initialisation statique et initialisation dynamique.

- Initialisation statique :
  - On définit tous les éléments entre accolades pendant la déclaration.

    ```
    int notes[5] = {10, 15, 20, 13, 12};
    ```
  - On peut omettre la taille du tableau.
  - Le compilateur détermine automatiquement la taille du tableau en utilisant le nombre d'éléments donné.

    ```
    int notes[] = {10, 15, 20, 13, 12};
    ```
- Initialisation dynamique :
  - On peut affecter aux éléments du tableau des valeurs dynamiquement c'est à dire à l'exécution du programme. Pour cela : on déclare d'abord un tableau et on utilise cette syntaxe :
  - nom_tableau[indice] = valeur;
  - Exemple : notes[0] = 10;

Initialisation dynamique

```c
#include <stdio.h>

void main(){
    int indice;
    int notes[5];
    // on parcourt les elements du tableau
    for(indice = 0; indice < 5; indice++)
    {
        scanf("%d", &notes[indice]);
    }
}
```

Implémenter en C un programme qui demande à l'utilisateur de saisir 10 notes et qui calcule leur moyenne.

Implémenter en C un programme qui demande à l'utilisateur de saisir 10 notes et qui calcule leur moyenne.

```c
#include <stdio.h>

// Une constante qui represente la taille du tableau
#define SIZE 10

void main(){

    int notes[SIZE];
    int indice, somme=0;
    float moyenne;
    for (indice = 0; indice<SIZE; indice++){
        scanf("%d", &notes[indice]);
        somme += notes[indice];
    }
    // on calcule la moyenne
    // on fait un cast : on convertit somme en float
    // le compilateur convertit automatiquement SIZE en float
    // le resultat sera affecte a moyenne
    moyenne = (float) somme/SIZE;
    printf("La moyenne des notes saisies est %.2f", moyenne);
}
```

N.B.

- **Faire attention quand on accède aux éléments d'un tableau.**
- Pb : L'accès à un élément qui n'existe pas ne génère pas d'erreur à la compilation.
- Le comportement à l'exécution est imprévisible :
  - On peut soit avoir des valeurs aléatoires.
  - ou un arrêt brusque du programme (prgoram crash).

```c
// ce programme compile correctement
// mais affiche des valeurs bizarres
// a l execution

#include <stdio.h>

int main()
{
    int arr[2];
    printf("%d ", arr[3]);
    printf("%d ", arr[-2]);

    return 0;
}
```

Remarque 2

- On n'aura pas d'erreurs à la compilation de ce programme.
- On n'aura que des warning.
- Ce programme déclare un tableau de deux éléments.
- mais le remplit avec 5 éléments.

```c
#include <stdio.h>
int main()
{
    // declaration de tableau avec initialisation
    // avec plus de valeurs que sa taille.
    int arr[2] = { 10, 20, 30, 40, 50 };
    for (int i=0; i<5;i++){
        printf("%d\n",arr[i]);
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    // declaration de tableau
          avec initialisation
    // avec plus de valeurs
         que sa taille.
    int arr[2] = { 10, 20,
         30, 40, 50 };
    for (int i=0; i<5;i++){
        printf("%d\n",arr[i])
              ;
    }
    return 0;
}
```

La compilation de ce programme se fait sans erreur (seulement des avertissements) donc un exécutable est généré.

Résultat à la compilation :

```
ex4.c: In function 'main':
ex4.c:7:28: warning: excess elements in array initializer
    7 |     int arr[2] = { 10, 20, 30, 40, 50 };
      |                            ^~
ex4.c:7:28: note: (near initialization for 'arr')
ex4.c:7:32: warning: excess elements in array initializer
    7 |     int arr[2] = { 10, 20, 30, 40, 50 };
      |                                ^~
ex4.c:7:32: note: (near initialization for 'arr')
ex4.c:7:36: warning: excess elements in array initializer
    7 |     int arr[2] = { 10, 20, 30, 40, 50 };
      |                                    ^~
ex4.c:7:36: note: (near initialization for 'arr')
```

Résultat à l'exécution :

```
10
20
2
0
0
```

Remarque 3 : une démonstration

```c
// Un programme C qui montre que les elements d un tableau
// sont stockes dans des zones contigues en memoire

#include <stdio.h>
int main()
{
    // un tableau de 4 elements, si arr[0] est stocke
    // a l adresse x, alors arr[1] est stocke a x +sizeof(int)
    // arr[2] est stocke a x + 2*sizeof(int) etc
    int arr[5], i;
    printf("La taille d un entier en octet est %lu\n", sizeof(int));

    for (i = 0; i < 5; i++)
        // Notez l'utilisation de & pour acceder a ladresse
        // d une variable
        // utilisation de %p pour l affichage
        printf("Adresse de arr[%d] est %p\n", i, &arr[i]);

    return 0;
}
```

```
La taille d un entier est 4
Adresse de arr[0] est 000000000022FE30
Adresse de arr[1] est 000000000022FE34
Adresse de arr[2] est 000000000022FE38
Adresse de arr[3] est 000000000022FE3C
Adresse de arr[4] est 000000000022FE40
```

Remarque 3 : résultat de la démonstration

```
La taille d un entier est 4
Adresse de arr[0] est 000000000022FE30
Adresse de arr[1] est 000000000022FE34
Adresse de arr[2] est 000000000022FE38
Adresse de arr[3] est 000000000022FE3C
Adresse de arr[4] est 000000000022FE40
```

- Ce résultat montre que la taille (obtenue avec `sizeof()`) d'un entier en mémoire est 4 octets.
- Il montre aussi que les éléments d'un tableau sont stockés dans une zone contigues en mémoire. En effet on passe de l'ardresse représentée par le nombre hexadécimal 22FE30 à l'adresse 22FE34.
- L'adresse d'une variable s'obtient avec l'opérateur &
- Pour afficher une addresse dans `printf`, on utilise le spécificateur de format : `%p`.
- Exemple :

```c
int var;
printf("adresse de var en memoire est %p", &var);
```