Apprenez à programmer en C !

40 heures Moyenne

Mis à jour le 29/07/2019

Les tableaux

Ce chapitre est la suite directe des pointeurs et va vous faire comprendre un peu plus leur utilité. Vous comptiez y échapper ? C'est raté ! Les pointeurs sont partout en C, vous avez été prévenus !

Dans ce chapitre, nous apprendrons à créer des variables de type « tableaux ». Les tableaux sont très utilisés en C car ils sont vraiment pratiques pour organiser une série de valeurs.

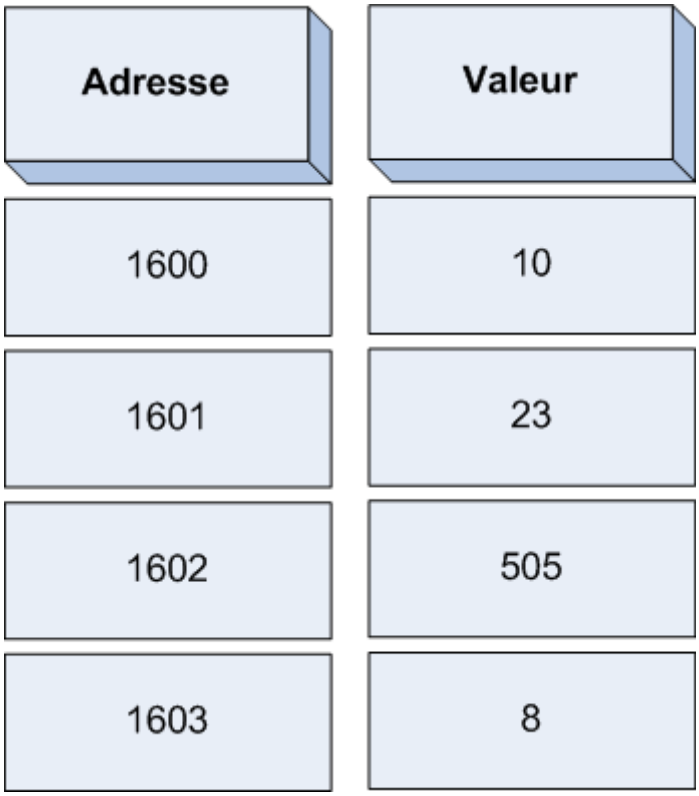
Nous commencerons dans un premier temps par quelques explications sur le fonctionnement des tableaux en mémoire (schémas à l'appui). Ces petites introductions sur la mémoire sont extrêmement importantes : elles vous permettent de comprendre comment cela fonctionne. Un programmeur qui comprend ce qu'il fait, c'est quand même un peu plus rassurant pour la stabilité de ses programmes, non ? ;-)

Les tableaux dans la mémoire

« Les tableaux sont une suite de variables de même type, situées dans un espace contigu en mémoire. »

Bon, je reconnais que ça ressemble un peu à une définition du dictionnaire. Concrètement, il s'agit de « grosses variables » pouvant contenir plusieurs nombres du même type (`long` , `int` , `char` , `double` ...).

Un tableau a une dimension bien précise. Il peut occuper 2, 3, 10, 150, 2 500 cases, c'est vous qui décidez. La fig. suivante est un schéma d'un tableau de 4 cases en mémoire qui commence à l'adresse 1600.



Lorsque vous demandez à créer un tableau de 4 cases en mémoire, votre programme demande à l'OS la permission d'utiliser 4 cases en mémoire. Ces 4 cases doivent être contiguës, c'est-à-dire les unes à la suite des autres. Comme vous le voyez, les adresses se suivent : 1600, 1601, 1602, 1603. Il n'y a pas de « trou » au milieu.

Enfin, chaque case du tableau contient un nombre du même type. Si le tableau est de type `int`, alors chaque case du tableau contiendra un `int`. On ne peut pas faire de tableau contenant à la fois des `int` et des `double` par exemple.

En résumé, voici ce qu'il faut retenir sur les tableaux.

- Lorsqu'un tableau est créé, il prend un espace contigu en mémoire : les cases sont les unes à la suite des autres.
- Toutes les cases d'un tableau sont du même type. Ainsi, un tableau de `int` contiendra uniquement des `int`, et pas autre chose.

Définir un tableau

Pour commencer, nous allons voir comment définir un tableau de 4 `int` :

c

```c
int tableau[4];
```

Voilà, c'est tout. Il suffit donc de rajouter entre crochets le nombre de cases que vous voulez mettre dans votre tableau. Il n'y a pas de limite (à part peut-être la taille de votre mémoire, quand même).

Maintenant, comment accéder à chaque case du tableau ? C'est simple, il faut écrire `tableau[numeroDeLaCase]`.

> Attention : un tableau commence à l'indice n° 0 ! Notre tableau de 4 `int` a donc les indices 0, 1, 2 et 3. Il n'y a pas d'indice 4 dans un tableau de 4 cases ! C'est une source d'erreurs très courantes, souvenez-vous-en.

Si je veux mettre dans mon tableau les mêmes valeurs que celles indiquées sur la fig. suivante, je devrai donc écrire :

c
```c
int tableau[4];

tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
```

> Je ne vois pas le rapport entre les tableaux et les pointeurs ?

En fait, si vous écrivez juste `tableau`, vous obtenez un pointeur. C'est un pointeur sur la première case du tableau. Faites le test :

c
```c
int tableau[4];

printf("%d", tableau);
```

Résultat, on voit l'adresse où se trouve `tableau` :

```
1600
```

En revanche, si vous indiquez l'indice de la case du tableau entre crochets, vous obtenez la valeur :

c
```c
int tableau[4];

printf("%d", tableau[0]);
```

```
10
```

De même pour les autres indices. Notez que comme `tableau` est un pointeur, on peut utiliser le symbole `*` pour connaître la première valeur :

c
```c
int tableau[4];

printf("%d", *tableau);
```

```
10
```

Il est aussi possible d'obtenir la valeur de la seconde case avec `*(tableau + 1)` (adresse de tableau + 1).
Les deux lignes suivantes sont donc identiques :

c

```c
tableau[1] // Renvoie la valeur de la seconde case (la première case étant 0)
*(tableau + 1) // Identique : renvoie la valeur contenue dans la seconde case
```

En clair, quand vous écrivez `tableau[0]`, vous demandez la valeur qui se trouve à l'adresse tableau + 0 case (c'est-à-dire 1600).
Si vous écrivez `tableau[1]`, vous demandez la valeur se trouvant à l'adresse tableau + 1 case (c'est-à-dire 1601).
Et ainsi de suite pour les autres valeurs.

Les tableaux à taille dynamique

Le langage C existe en plusieurs versions.
Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique, c'est-à-dire de tableaux dont la taille est définie par une variable :

c

```c
int taille = 5;
int tableau[taille];
```

Or cela n'est pas forcément reconnu par tous les compilateurs, certains planteront sur la seconde ligne. Le langage C que je vous enseigne depuis le début (appelé le C89) n'autorise pas ce genre de choses. Nous considèrerons donc que faire cela est interdit.

Nous allons nous mettre d'accord sur ceci : vous n'avez pas le droit d'utiliser une variable entre crochets pour la définition de la taille du tableau, même si cette variable est une constante ! Le tableau doit avoir une dimension fixe, c'est-à-dire que vous devez écrire noir sur blanc le nombre correspondant à la taille du tableau :

c

```c
int tableau[5];
```

> Mais alors... il est interdit de créer un tableau dont la taille dépend d'une variable ?

Non, rassurez-vous : c'est possible, même en C89. Mais pour faire cela, nous utiliserons une autre technique (plus sûre et qui marche partout) appelée **l'allocation dynamique**. Nous verrons cela bien plus loin dans ce cours.

Parcourir un tableau

Supposons que je veuille maintenant afficher les valeurs de chaque case du tableau.
Je pourrais faire autant de `printf` qu'il y a de cases. Mais bon, ce serait répétitif et lourd, et imaginez un peu la taille de notre code si on devait afficher le contenu de chaque case du tableau une à une !

Le mieux est de se servir d'une boucle. Pourquoi pas d'une boucle `for` ? Les boucles `for` sont très pratiques pour parcourir un tableau :

c

```c
int main(int argc, char *argv[])
{
    int tableau[4], i = 0;

    tableau[0] = 10;
    tableau[1] = 23;
    tableau[2] = 505;
    tableau[3] = 8;

    for (i = 0 ; i < 4 ; i++)
    {
        printf("%d\n", tableau[i]);
    }

    return 0;
}
```

```
10
23
505
8
```

Notre boucle parcourt le tableau à l'aide d'une variable appelée `i` (c'est le nom très original que les programmeurs donnent en général à la variable qui leur permet de parcourir un tableau !).

Ce qui est particulièrement pratique, c'est qu'on peut mettre une variable entre crochets. En effet, la variable était interdite pour la création du tableau (pour définir sa taille), mais elle est heureusement autorisée pour « parcourir » le tableau, c'est-à-dire afficher ses valeurs !
Ici, on a mis la variable `i`, qui vaut successivement 0, 1, 2, et 3. De cette façon, on va donc afficher la valeur de `tableau[0]`, `tableau[1]`, `tableau[2]` et `tableau[3]` !

> Attention à ne pas tenter d'afficher la valeur de `tableau[4]` ! Un tableau de 4 cases possède les indices 0, 1, 2 et 3, point barre. Si vous tentez d'afficher `tableau[4]`, vous aurez soit n'importe quoi, soit une belle erreur, l'OS coupant votre programme car il aura tenté d'accéder à une adresse ne lui appartenant pas.

Initialiser un tableau

Maintenant que l'on sait parcourir un tableau, nous sommes capables d'initialiser toutes ses valeurs à 0 en faisant une boucle !

Bon, parcourir le tableau pour mettre 0 à chaque case, c'est de votre niveau maintenant :

c

```c
int main(int argc, char *argv[])
{
    int tableau[4], i = 0;

    // Initialisation du tableau
    for (i = 0 ; i < 4 ; i++)
    {
        tableau[i] = 0;
    }

    // Affichage de ses valeurs pour vérifier
    for (i = 0 ; i < 4 ; i++)
    {
        printf("%d\n", tableau[i]);
    }

    return 0;
}
```

```
0
0
0
0
```

Une autre façon d'initialiser

Il faut savoir qu'il existe une autre façon d'initialiser un tableau un peu plus automatisée en C. Elle consiste à écrire `tableau[4] = {valeur1, valeur2, valeur3, valeur4}`. En clair, vous placez les valeurs une à une entre accolades, séparées par des virgules :

c

```c
int main(int argc, char *argv[])
{
    int tableau[4] = {0, 0, 0, 0}, i = 0;

    for (i = 0 ; i < 4 ; i++)
    {
        printf("%d\n", tableau[i]);
    }

    return 0;
}
```

```
0
0
0
0
```

Mais en fait, c'est même mieux que ça : vous pouvez définir les valeurs des premières cases du tableau, toutes celles que vous n'aurez pas renseignées seront automatiquement mises à 0.

Ainsi, si je fais :

```c
int tableau[4] = {10, 23}; // Valeurs insérées : 10, 23, 0, 0
```

... la case n° 0 prendra la valeur 10, la n° 1 prendra 23, et toutes les autres prendront la valeur 0 (par défaut).

Comment initialiser tout le tableau à 0 en sachant ça ?
Eh bien il vous suffit d'initialiser au moins la première valeur à 0, et toutes les autres valeurs non indiquées prendront la valeur 0.

```c
int tableau[4] = {0}; // Toutes les cases du tableau seront initialisées à 0
```

Cette technique a l'avantage de fonctionner avec un tableau de n'importe quelle taille (là, ça marche pour 4 cases, mais s'il en avait eu 100 ça aurait été bon aussi).

> Attention, on rencontre souvent `int tableau[4] = {1};`, ce qui insère les valeurs suivantes : 1, 0, 0, 0.
> Contrairement à ce que beaucoup d'entre vous semblent croire, on n'initialise pas toutes les cases à 1 en faisant cela : seule la première case sera à 1, les autres seront à 0. On ne peut donc pas initialiser toutes les cases à 1 automatiquement, à moins de faire une boucle.

Passage de tableaux à une fonction

Vous aurez à coup sûr souvent besoin d'afficher tout le contenu de votre tableau. Pourquoi ne pas écrire une fonction qui fait ça ? Ça va nous permettre de découvrir comment on envoie un tableau à une fonction (ce qui m'arrange).

Il va falloir envoyer deux informations à la fonction : le tableau (enfin, l'adresse du tableau) et aussi et surtout sa taille !
En effet, notre fonction doit être capable d'initialiser un tableau de n'importe quelle taille. Or, dans votre fonction, vous ne connaissez pas la taille de votre tableau. C'est pour cela qu'il faut envoyer en plus une variable que vous appellerez par exemple `tailleTableau`.

Comme je vous l'ai dit, `tableau` peut être considéré comme un pointeur. On peut donc l'envoyer à la fonction comme on l'aurait fait avec un vulgaire pointeur :

c

```c
// Prototype de la fonction d'affichage
void affiche(int *tableau, int tailleTableau);

int main(int argc, char *argv[])
{
    int tableau[4] = {10, 15, 3};

    // On affiche le contenu du tableau
    affiche(tableau, 4);

    return 0;
}

void affiche(int *tableau, int tailleTableau)
{
    int i;

    for (i = 0 ; i < tailleTableau ; i++)
    {
        printf("%d\n", tableau[i]);
    }
}
```

```
10
15
3
0
```

La fonction n'est pas différente de celles que l'on a étudiées dans le chapitre sur les pointeurs. Elle prend en paramètre un pointeur sur `int` (notre tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).
Tout le contenu du tableau est affiché par la fonction via une boucle.

Notez qu'il existe une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un `int *tableau`, mettez ceci :

c

```c
void affiche(int tableau[], int tailleTableau)
```

Cela revient exactement au même, mais la présence des crochets permet au programmeur de bien voir que c'est un tableau que la fonction prend, et non un simple pointeur. Cela permet d'éviter des confusions.

J'utilise personnellement tout le temps les crochets dans mes fonctions pour bien montrer que la fonction attend un tableau. Je vous conseille de faire de même. Il n'est pas nécessaire de mettre la

taille du tableau entre les crochets cette fois.

Quelques exercices !

Je ne manque pas d'idées d'exercices pour vous entraîner ! Je vous propose de réaliser des fonctions travaillant sur des tableaux.

Je donne juste les énoncés des exercices ici pour vous forcer à réfléchir à vos fonctions. Si vous avez du mal à réaliser ces fonctions, rendez-vous sur les forums pour poser vos questions.

Exercice 1

Créez une fonction `sommeTableau` qui renvoie la somme des valeurs contenues dans le tableau (utilisez un `return` pour renvoyer la valeur). Pour vous aider, voici le prototype de la fonction à créer :

c

```c
int sommeTableau(int tableau[], int tailleTableau);
```

Exercice 2

Créez une fonction `moyenneTableau` qui calcule et renvoie la moyenne des valeurs. Prototype :

c

```c
double moyenneTableau(int tableau[], int tailleTableau);
```

La fonction renvoie un `double` car une moyenne est souvent un nombre décimal.

Exercice 3

Créez une fonction `copierTableau` qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
Prototype :

c

```c
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

Exercice 4

Créez une fonction `maximumTableau` qui aura pour rôle de remettre à 0 toutes les cases du tableau ayant une valeur supérieure à un maximum. Cette fonction prendra en paramètres le tableau ainsi que le nombre maximum autorisé (`valeurMax`). Toutes les cases qui contiennent un nombre supérieur à `valeurMax` doivent être mises à 0. Prototype :

c

```c
void maximumTableau(int tableau[], int tailleTableau, int valeurMax);
```

Exercice 5

Cet exercice est plus difficile. Créez une fonction `ordonnerTableau` qui classe les valeurs d'un tableau dans l'ordre croissant. Ainsi, un tableau qui vaut `{15, 81, 22, 13}` doit à la fin de la fonction valoir

`{13, 15, 22, 81}`.

Prototype :

```c
void ordonnerTableau(int tableau[], int tailleTableau);
```

Cet exercice est donc un peu plus difficile que les autres, mais tout à fait réalisable. Ça va vous occuper un petit moment.

> Faites-vous un petit fichier de fonctions appelé `tableaux.c` (avec son homologue `tableaux.h` qui contiendra les prototypes, bien sûr !) contenant toutes les fonctions de votre cru réalisant des opérations sur des tableaux.

Au travail ! :-)

En résumé

- Les **tableaux** sont des ensembles de variables du même type stockées côte à côte en mémoire.
- La taille d'un tableau doit être déterminée avant la compilation, elle ne peut pas dépendre d'une variable.
- Chaque case d'un tableau de type `int` contient une variable de type `int`.
- Les cases sont numérotées via des **indices** commençant à 0 : `tableau[0]`, `tableau[1]`, `tableau[2]`, etc.

À L'ASSAUT DES POINTEURS

LES CHAÎNES DE CARACTÈRES

Le professeur

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...

Livre PDF