

Tableaux

2018

Florent Gluck – Florent.Gluck@hesge.ch

Version 0.6

Tableaux (1)

- C offre uniquement des tableaux statiques :
 - Un tableau est un « bloc » de mémoire contigüe associé à un nom :
 - taille **fixe** déterminée à la déclaration du tableau ;
 - la taille ne peut pas être changée.
 - Pas d'assignation de tableaux.
 - Un tableau déclaré dans une fonction ou un bloc est « perdu » à la sortie de celle/celui-ci :
 - Un tableau local à une fonction ne doit **jamais** être retourné (aussi valable pour toute variable locale) !

Tableaux (2)

- Les éléments d'un tableau sont accédés avec `[i]` où `i` est l'index de l'élément `i` dans le tableau.
- Le premier élément du tableau commence **toujours à l'index 0 !**
- Lorsqu'un tableau est déclaré, la taille de celui-ci doit toujours être spécifiée, sauf s'il est initialisé lors de sa déclaration :

```
int tab[5]; // valeurs du tab. non initialisées
int vals[] = {1,2,3}; // tableau de 3 éléments
int v = vals[2];
int w = vals[5]; // ATTENTION !! Pas de message
                  // d'erreur du compilateur C !
```

Quiz

```
int a1[5]; // OK ?
int a2[] = { 1, 2, 3 }; // OK ?
int a3[4][5]; // OK ?
int [] a4; // OK ?
int a5[]; // OK ?

int[] function(void) { // OK ?
    int array[5]; // OK ?
    return array; // OK ?
}

void foo(int a[]) { // OK ?
    a[3] = 0; // OK ?
}

void bar(void) {
    int a[5]; // OK ?
    foo(a); // OK ?
    a = a5; // OK ?
}
```

**Déclarations
correctes ?**

Exemple

```
int a1[5]; // OK
int a2[] = { 1, 2, 3 }; // OK
int a3[4][5]; // OK
int [] a4; // Erreur
int a5[]; // Erreur

int[] function(void) { // Erreur
    int array[5]; // OK
    return array; // Erreur
}

void foo(int a[]) { // OK
    a[3] = 0; // OK
}

void bar(void) {
    int a[5]; // OK
    foo(a); // OK
    a = a5; // Erreur
}
```

Quiz

Quels sont les bugs dans le code ci-dessous ?

```
#include <stdio.h>

int main(void) {
    char i;
    char a1[] = { 100,200,300,400,500 };
    char a2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    a2[10] = 42;

    for (i = 0; i < 5; i++) {
        printf("a1[%d] = %d\n", i, a1[i]);
    }

    return 0;
}
```

Quiz

Quels sont les bugs dans le code ci-dessous ?

```
#include <stdio.h>

int main(void) {
    char i;
    char a1[] = { 100, 200, 300, 400, 500 };
    char a2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    a2[10] = 42;

    for (i = 0; i < 5; i++) {
        printf("a1[%d] = %d\n", i, a1[i]);
    }

    return 0;
}
```

Tableaux et sizeof

L'opérateur `sizeof` renvoie la taille en bytes d'un type ou d'une variable.

Exemple :

```
void func() {  
    int t[4];  
    int n = sizeof(t); // valeur de n ?  
}
```


Tableaux et sizeof

L'opérateur `sizeof` renvoie la taille en bytes d'un type ou d'une variable.

Exemple :

```
void func() {  
    int t[4];  
    int n = sizeof(t); // valeur de n ?  
}
```

Tableaux et sizeof

L'opérateur `sizeof` renvoie la taille en bytes d'un type ou d'une variable.

Exemple :

```
void func() {  
    int t[4];  
    int n = sizeof(t); // valeur de n ? 16 et non 4!  
}
```

Pour déterminer la taille d'un tableau:

```
void func() {  
    int t[4];  
    int n = sizeof(t)/sizeof(t[0]);  
}
```

Attention! Ceci n'est pas valable pour un tableau passé en argument à une fonction !

Tableaux et arguments

Lorsqu'un tableau est passé en argument à une fonction, c'est **l'adresse du tableau** qui est passée à la fonction:

```
void func(int tab[]) {  
    int n = sizeof(tab); // valeur de n ?  
}
```

Tableaux et arguments

Lorsqu'un tableau est passé en argument à une fonction, c'est **l'adresse du tableau** qui est passée à la fonction:

```
void func(int tab[]) {  
    int n = sizeof(tab); // valeur de n ? 8!  
                        // (sur arch 64-bits)  
    for (int i = 0; i < n; i++) ...  
}
```

D'où la nécessité de **toujours** aussi passer la longueur du tableau !

```
void func(int tab[], int n) {  
  
    for (int i = 0; i < n; i++) ...  
}
```

Tableaux et arguments

Lorsqu'un tableau est passé en argument à une fonction, c'est **l'adresse du tableau** qui est passée à la fonction:

```
void func(int tab[]) {  
    int n = sizeof(tab);  
    for (int i = 0; i < n; i++) ...  
}
```

A éviter absolument !

D'où la nécessité de **toujours** aussi passer la longueur du tableau !

```
void func(int tab[], int n) {  
    for (int i = 0; i < n; i++) ...  
}
```