

Tableaux

Programmation séquentielle en C, 2026-2027

Orestis Malaspinas (A401)

2026-10-05

Informatique et Systèmes de Communication, HEPIA

Les tableaux statiques

Qu'est-ce qu'un tableau statique ?

- Une **collection** d'éléments du **même type**
- alloués de façon **contigüe** (en bloc) en mémoire, sur la **pile** s'il est local à une fonction
- dont la taille est fixée et **ne peut pas** être changée.

Remarques

- Les éléments d'un tableau sont accédés avec `[i]` où `i` est l'index de l'élément.
- Le premier élément du tableau est à l'index `0` !
- Lorsqu'un tableau est déclaré, la taille de celui-ci doit être spécifiée, sauf s'il est initialisé lors de sa déclaration (ou s'il s'agit d'un paramètre de fonction).
- On ne doit **jamais retourner l'adresse** d'un tableau local à une fonction !

Syntaxe des tableaux

```
float tab1[5]; // tableau de floats à 5 éléments
               // ses valeurs sont indéfinies

int tab2[] = {1, 2, 3}; // tableau de 3 entiers,
                       // taille inférée

int val = tab2[1]; // val vaut 2 à présent

int w = tab1[5]; // index hors des limites du tableau
                 // comportement indéfini!
                 // pas d'erreur du compilateur
```

Initialisation des tableaux

```
int t1[5] = {1, 2};    // {1, 2, 0, 0, 0}
                        // le reste est mis à zéro
int t2[5] = {0};       // {0, 0, 0, 0, 0}
int t3[5];             // valeurs indéfinies!
int t4[2] = {1, 2, 3}; // trop d'éléments: avertissement
```

Itérer sur les éléments d'un tableau

```
#define N 10
int x[N];
for (int i = 0; i < N; ++i) {
    x[i] = 0;
}
int j = 0;
while (j < N) {
    x[j] = 1;
    j += 1;
}
int k = 0;
do {
    x[k] = -1;
    k += 1;
} while (k < N);
```

Représentation des tableaux en mémoire

Les éléments d'un tableau sont :

- ... contigus en mémoire (les uns à la suite des autres),
- ... accessibles directement par leur index

Exemple :

```
char tab[4] = {79, 91, 100, 88};
```

char	79	91	100	88	...	...
addr	2000	2001	2002	2003	...	...

Qu'est-ce que `tab` ?

```
tab;      // 2000, l'adresse du 1er élément
&tab[0]; // 2000 == tab
tab[0];  // 79
sizeof(tab); // 4
```

Taille d'un tableau

```
int tab[] = {4, 8, 15, 16, 23, 42};  
sizeof(tab);    // 24 (6 * sizeof(int))  
sizeof(tab[0]); // 4  
int n = sizeof(tab) / sizeof(tab[0]); // 6 éléments
```

- Ne fonctionne **que** là où le tableau est déclaré (voir plus loin).
- Valeurs pour un `int` de 4 octets.

Ce qu'on ne peut pas faire avec un tableau

```
int a[3] = {1, 2, 3};  
int b[3];  
b = a;           // erreur de compilation  
if (a == b) {    // compile (avec un avertissement),  
}                // mais ne compare pas les éléments!
```

Ce qu'on ne peut pas faire avec un tableau

```
int a[3] = {1, 2, 3};  
int b[3];  
b = a;           // erreur de compilation  
if (a == b) {    // compile (avec un avertissement),  
}               // mais ne compare pas les éléments!
```

```
for (int i = 0; i < 3; ++i) {  
    b[i] = a[i]; // copie élément par élément  
}
```

Les tableaux comme arguments de fonctions (1/4)

```
void foo(int n, int tab[n]) { // n optionnel
    for (int i = 0; i < n; ++i) {
        tab[i] *= 2;
    }
}

int main() {
    int a[] = { 1, 2, 3, 4, 5 };
    foo(5, a);
}
```

- Que valent les éléments de `a` après `foo()` ?

Les tableaux comme arguments de fonctions (1/4)

```
void foo(int n, int tab[n]) { // n optionnel
    for (int i = 0; i < n; ++i) {
        tab[i] *= 2;
    }
}

int main() {
    int a[] = { 1, 2, 3, 4, 5 };
    foo(5, a);
}
```

- Que valent les éléments de `a` après `foo()` ?
- `{ 2, 4, 6, 8, 10 }`

Les tableaux comme arguments de fonctions (2/4)

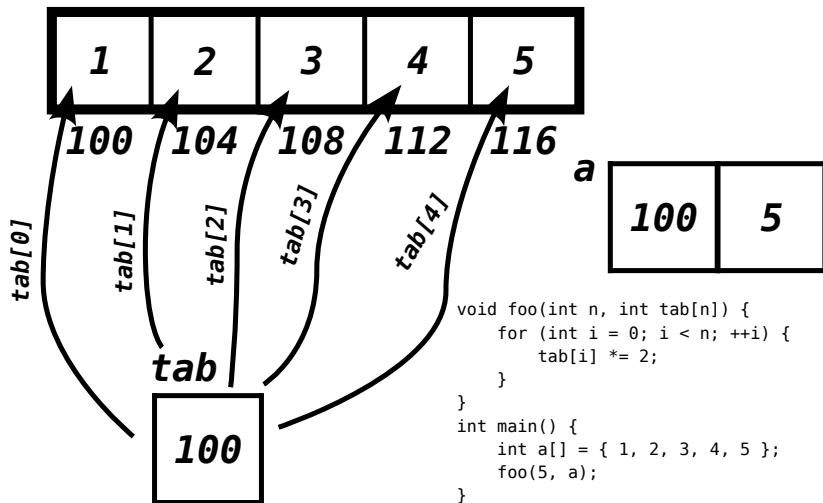


Figure 1 – Tableaux et fonctions

Les tableaux comme arguments de fonctions (3/4)

```
void foo(int n, int tab[n]) { // n optionnel
    for (int i = 0; i < n; ++i) {
        tab[i] *= 2;
    }
}

int main() {
    int a[] = { 1, 2, 3, 4, 5 };
    foo(5, a);
}
```

- `tab[ ]` est transformé en *pointeur* vers le **1er élément** du tableau
- Le pointeur est **copié** quand on le passe en argument
- On peut donc accéder en *lecture/écriture* aux éléments de `tab[ ]`

Les tableaux comme arguments de fonctions (4/4)

- Un tableau en argument est le pointeur vers sa première case.
- Pas moyen de connaître sa taille : `sizeof()` inutile.
- Toujours spécifier la taille d'un tableau passé en argument.

```
void foo(int tab[]) { // équivalent à int *tab
    // que vaut sizeof(tab)?
    for (int i = 0; i < ?; ++i) { // taille?
        printf("tab[%d] = %d\n", i, tab[i]);
    }
}

void bar(int n, int tab[n]) { // n optionnel
    for (int i = 0; i < n; ++i) {
        printf("tab[%d] = %d\n", i, tab[i]);
    }
}
```

sizeof d'un tableau en argument

```
void foo(int tab[]) {  
    printf("%zu\n", sizeof(tab)); // 8: taille d'un pointeur!  
}  
  
int main() {  
    int a[] = {1, 2, 3, 4, 5};  
    printf("%zu\n", sizeof(a)); // 20: taille du tableau  
    foo(a);  
}
```

- gcc avertit: `-Wsizeof-array-argument`
- D'où la nécessité de passer la taille en argument.
- Valeurs sur une machine 64 bits.

Quiz : OK ou erreur?

```
void decl() {  
    int a1[5];  
    int a2[] = { 1, 2, 3 };  
    int [] a3;  
    int a4[];  
}  
  
int[] function() {  
    int array[5];  
    return array;  
}  
  
void foo(int a[]) {  
    a[3] = 0;  
}  
  
void bar() {  
    int a[5];  
    foo(a);  
    a = a[5];  
}
```

Quiz : OK ou erreur? (solution)

```
void decl() {  
    int a1[5];           // OK  
    int a2[] = { 1, 2, 3 }; // OK  
    int [] a3;           // Erreur  
    int a4[];           // Erreur  
}  
  
int[] function() {      // Erreur: on ne peut pas retourner un tableau  
    int array[5];       // OK  
    return array;       // Dangereux: adresse d'un tableau local!  
}  
  
void foo(int a[]) {     // OK  
    a[3] = 0;           // OK  
}  
  
void bar() {  
    int a[5];           // OK  
    foo(a);             // OK  
    a = a[5];           // Erreur  
}
```