C 7 : tableaux

Juliusz Chroboczek

26 octobre 2022

Un *tableau* est une structure de données agrégée (composée de plusieurs sous-données) qui est *uniforme* (tous les éléments ont le même type) et *indexée par des entiers* (les éléments du tableau sont numéroté par des entiers). L'intérêt principal des tableaux est qu'on peut les parcourir à l'aide de boucles.

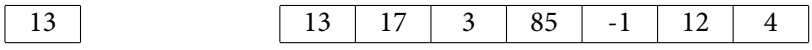
La notion native de tableau en C est extrêmement primitive et souffre de nombreuses limitations. Cependant, elle suffit pour implémenter nous-mêmes des notions plus utiles de tableaux, comme nous le verrons au prochain cours.

1 Syntaxe et sémantique

Un tableau nommé `a` dont le type de base est *T* et la taille est *n* est déclaré à l'aide de la notation « *T* `a[`*n*`];` ». Par exemple, pour déclarer un tableau d'entiers de taille 42, il fait écrire :

```
int a[42];
```

Un tableau de taille *n* contient *n* cases, numérotées de 0 à $n - 1$, dont chacune se comporte comme une variable du type de base du tableau : on peut lire sa valeur ou y affecter une valeur. Si une variable peut être vue comme une case, un tableau est une suite de *n* cases (figure 1).



(a) (b)

FIGURE 1 — Une variable entière (a) et un tableau d'entiers (b)

La case numéro *i* du tableau `a` est dénotée par `a[`*i*`]`. Par exemple, l'instruction suivante affecte 13 à la deuxième case de `a` :

```
a[1] = 13;
```

De même, l'instruction suivante recopie la valeur de la deuxième case dans la troisième :

```
a[2] = a[1];
```

Les tableaux sont primitifs La notion de tableau utilisée en C est extrêmement primitive : un tableau est simplement une suite de données, dont la longueur doit être connue *a priori*. Il n'est donc jamais correct de manipuler un tableau sans avoir stocké sa longueur quelque part : le concept utile n'est pas le tableau `a`, mais la paire (`a`, `n`), où `n` est la longueur du tableau.

Cette notion est différente de la notion plus habituelle de *tableau Pascal*, qui « connaît » sa longueur. Par exemple, en Java, Python ou Go, si `a` est un tableau, alors la notation `a.length` ou `len(a)` permet d'obtenir sa longueur. Nous verrons au prochain cours comment implémenter les tableaux Pascal en C.

2 Tableaux et boucles

Comme les tableaux sont uniformes et indexés par des entiers, il est possible de les parcourir à l'aide d'une boucle définie. Si `a` est un tableau de taille 42, le fragment de code suivant affecte 57 à chacune de ses cases.

```
int i;
for(i = 0; i < 42; i++)
    a[i] = 57;
```

Remarquez l'inégalité stricte dans l'expression de contrôle : les cases de `a` sont numérotées de 0 à 41, pas 42.

3 Tableaux et fonctions

Un tableau ne peut pas être passé par valeur à une fonction en C : un tableau peut être de grande taille, et le passer par valeur demanderait de copier une quantité potentiellement importante de données. Ce qui est possible, par contre, c'est de passer une *référence* au premier élément du tableau ; nous verrons les détails de ce mécanisme lors du cours sur les pointeurs.

Un paramètre formel qui recevra une référence à un élément d'un tableau de type de base T doit être déclaré comme ayant le type T`*` (pointeur sur le type T). Le paramètre effectif correspondant est simplement le nom du tableau.

Comme les tableaux en C ne connaissent pas leur taille, il n'est presque jamais utile de passer une référence à un tableau toute seule. Dans la plupart des cas, nous passerons une paire composée d'une référence et d'une taille :

```
int somme(int *a, int n) {
    int s = 0;
    int i;
    for(i = 0; i < n; i++)
        s = s + a[i];
    return s;
}
```

Passage par références et modification Comme les tableaux sont passés par référence, si une fonction modifie le tableau à travers la référence, l'effet est visible à la fonction appelante. Comparez la fonction suivante, qui ne fait rien :

```
void badswap(int x, int y) {
    int z;
    z = x;
    x = y;
    y = z;
}
```

à la fonction suivante, qui échange les deux premiers éléments d'un tableau :

```
void goodswap(int *a, int n) {
    int z;
    if(n < 2)
        return;
    z = a[0];
    a[0] = a[1];
    a[1] = z;
}
```

Tableaux et valeurs de retour De même qu'il n'est pas possible de passer un tableau à une fonction, il n'est pas possible de retourner un tableau d'une fonction : il faut retourner une référence au premier élément, ce qui n'est généralement pas correct si le tableau a été alloué sur la pile. On peut contourner cette limitation en allouant le tableau dans l'appelant et en le remplissant dans une fonction.

```
void fill(int *a, int n) {
    for(int i = 0; i < n; i++)
        a[i] = i * 2;
}
```

Nous verrons lors du cours sur l'allocation dynamique comment allouer un tableau dans une fonction de manière à pouvoir retourner une référence à ce dernier.

4 Digression : déclarations C99

Depuis la révision C99 du langage C, on peut déclarer une variable au milieu d'un bloc. Dans certaines communeautés, cependant, ces déclarations en milieu de bloc sont considérées comme étant de mauvais style.

On peut aussi déclarer une variable dans l'instruction d'initialisation d'une boucle `for` ; dans ce dernier cas, la variable n'est déclarée que dans l'entête et le corps de la boucle :

```
for(int i = 0; i < 42; i++)
    a[i] = 57;
```

5 Tableaux de taille variable

En C99, la taille d'un tableau n'a plus besoin d'être une constante : il est légal d'utiliser comme taille une expression entière arbitraire. Par exemple, on peut allouer un tableau de taille donnée par l'utilisateur.

```
int n;
scanf("%d", &n);
int a[n];
for(int i = 0; i < n; i++)
    scanf("%d", &a[i]);
printf("%d\n", somme(a, n));
```

Attention, un tableau de taille variable est alloué sur la pile, ce qui peut poser des problèmes sur les systèmes où la taille de cette dernière est limitée, par exemple Mac OS X. Il faudra attendre le cours sur l'allocation dynamique pour pouvoir allouer de grands tableaux de façon portable.