Cours 5 : Les tableaux

Guillaume Aubian

Ce document a été grandement inspiré par – et largement copié sur – le document correspondant du responsable précédent de ce cours, Juliusz Chroboczek, avec son accord.

Un tableau est une structure de données qui est uniforme (tous les éléments ont le même type) et indexée par des entiers. 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 faut écrire :

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

Un tableau de taille n contient n cases, numérotées de 0 à $n-1$, et 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 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];
```

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 quasiment 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.

La syntaxe pour les fonctions est la suivante :

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

Remarquez qu'ici, comme souvent, on accompagne le tableau en argument de sa taille.

3 Sous le capot

En réalité, dans l'expression

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

`a` est un pointeur vers l'emplacement de la première case du tableau, c'est-à-dire l'adresse (en mémoire) de la première case du tableau. Quand on écrit `a[0]`, on lit ce qu'il y a dans la case correspondante, et quand on écrit `a[1]` on lit le contenu de la case suivante.

C'est très important pour comprendre pourquoi ce code est fautif (en ceci que si on a définit deux variables `a` et `b` et qu'on appelle `badswap(a,b);`, les valeurs de `a` et `b` restent inchangés) :

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

alors que celui-ci est correct :

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

On pourra alors corriger `badswap` en demandant de fournir cette fois l'adresse des variables en arguments :

```
void fixedswap(int x[], int y[]) {
    int z;
    z = x[0];
    x[0] = y[0];
    y[0] = z;
}
```

Et pour récupérer l'adresse d'une variable, on utilise `&nomdelavariable`. Vous connaissez cette notation : c'est ainsi que fonctionne `scanf` !