Tableau unidimensionnel [tb]
Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 18 mai 2018

Table des matières

C - Tableau unidimensionnel (Cours)

Mots-Clés Tableau unidimensionnel, Parcours d'un tableau ■
Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif ■
Difficulté ●●○ (2 h) ■

Introduction

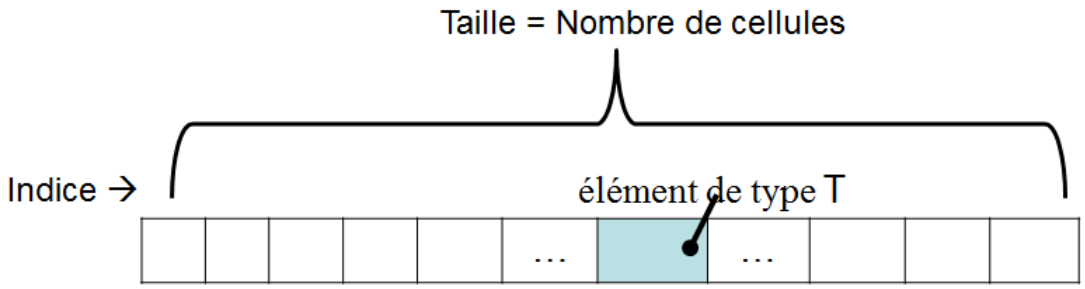
De par leur praticité, les **tableaux** sont omniprésents en algorithmique et programmation : une variable regroupant sous le même nom plusieurs valeurs de même type accessibles par leur position.

Ce module donne les définitions et notations de **tableau sur une dimension**puis décrit la transmission des tableaux ainsi que ses parcours (complet, partiel, imbriqué).

1 Définitions et notations

1.1 Qu'est-ce qu'un tableau ?

Tableau, Type du tableau, Taille
Un **tableau** (sous-entendu monodimensionnel ou linéaire) est une collection **homogène indicée** sur $\mathbb{N}$, c.-à.d. une séquence d'éléments de même type portant tous le même nom mais se distinguant les uns des autres par un indice. Le **type du tableau** est le type de ses éléments. Sa **taille** ou *capacité* est le nombre (strictement positif) de ses éléments.



Il n'y a pas de trou
Tous les éléments existent entre le premier et le dernier indice.

Indice (ou index ou rang)
Entier donnant la **position** d'un élément dans la séquence. Cet indice varie entre la position du premier élément et la position du dernier élément, ces positions correspondant aux bornes de l'indice.

Un Indice = Une case
A chaque valeur de l'indice ne correspond qu'une et une seule case du tableau, donc un élément.

Taille statique v.s. variable
La taille d'un tableau ne peut pas être modifiée pendant son utilisation. On qualifie ce genre de tableau de **taille statique** (nombre fixe d'éléments). Nous verrons ultérieurement qu'il existe des tableaux de **taille variable** et appelés vecteurs ou tableaux dynamiques. Ce module ne traite que des premiers.

Taille logique v.s. physique
Comme on utilisera un tableau plus grand que le nombre utile de ses éléments, on parle aussi de taille **logique**, dite aussi **effective** (le nombre d'éléments effectivement utilisés), que l'on oppose à la taille **physique** (la taille maximale du tableau).

En C/C++

Un tableau n'est pas une valeur : il ne peut pas être affecté à une variable, ni retourné par une fonction. Comme on le verra, un tableau, excepté lors de sa définition ou du calcul de sa taille, n'est pas manipulé globalement mais élément par élément.

1.2 Déclaration et initialisation d'un tableau

C/C++ **Déclaration d'un tableau**

```
TypeElement nomTab[taille];
```

Explication

Déclare une variable dimensionnée. Avec : `TypeElement` le type (simple ou non) des éléments constitutifs du tableau, `nomTab` l'identifiant et `taille` son nombre d'éléments. La taille doit être une valeur entière positive (littéraux ou expressions constantes).

C/C++ **Déclaration et initialisation**

```
TypeElement nomTab [taille] = {val1, ..., valN}; // taille explicite
TypeElement nomTab [] = {val1, ..., valN}; // taille de la liste
```

Explication

Déclare (voir supra) et initialise un tableau. Les `valI` sont des valeurs littérales ou expressions constantes de type compatible `TypeElement` initialisant séquentiellement chacun des éléments du tableau. Dans le cas (2) la longueur de la liste détermine le nombre d'éléments. Dans le cas (1), si la liste d'initialisation contient moins d'éléments que la `taille` spécifiée, les éléments manquants seront initialisés par défaut au zéro du type `TypeElement`.

1.3 Exemple : Déclaration et initialisation

Exemple C

```
int tab[8];
enum {NTEMPMAX=366};
double temp[NTEMPMAX];
char message[25];
```

Explication

La variable `tab` est un tableau de 8 cases de type entier, `temp` un tableau de `NTEMPMAX` cases de réels et `message` un tableau de 25 caractères.

Exemple C

```
int chif1[]   = {1,2,3,4,5,6,7,8,9,10};
int chif2[10] = {3,3,3};

double x1[]  = {0.25,0,0,-0.5,0};
double x2[5] = {0.0};

char coulr1[] = {'B','L','E','U'};
char coulr2[] = {"BLEU"};
```

Explication

Ces initialisations produisent l'affectation des valeurs suivantes aux éléments de ces tableaux :

```
chif1[0]=1  chif2[0]=3  x1[0]=0.25  x2[0]=0.0  coulr1[0]='B' coulr2[0]='B'
chif1[1]=2  chif2[1]=3  x1[1]=0.0   x2[1]=0.0  coulr1[1]='L' coulr2[1]='L'
chif1[2]=3  chif2[2]=3  x1[2]=0.0   x2[2]=0.0  coulr1[2]='E' coulr2[2]='E'
chif1[3]=4  chif2[3]=0  x1[3]=-0.5  x2[3]=0.0  coulr1[3]='U' coulr2[3]='U'
chif1[4]=5  chif2[4]=0  x1[4]=0.0   x2[4]=0.0
   coulr2[4]='\0'
chif1[5]=6  chif2[5]=0
chif1[6]=7  chif2[6]=0
chif1[7]=8  chif2[7]=0
chif1[8]=9  chif2[8]=0
chif1[9]=10 chif2[9]=0
```

1.4 Accès indiciel

Accès indiciel

```
tab[k]
```

Explication

Accède à la case d'indice `k` d'un tableau `tab`.
Le temps d'accès à l'élément est fixe.

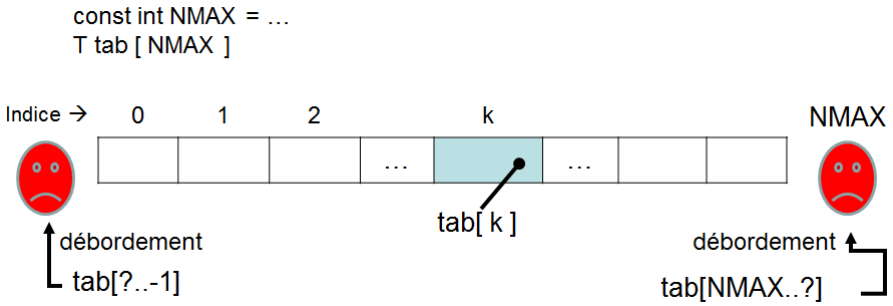
Numérotation des cases

Chaque langage de programmation possède sa propre convention.

- **alg :** Les cases sont numérotées de 1 (par défaut) à `TMAX` (taille du tableau).
- **C/C++, Java, Python :** Ils commencent à indicer un tableau à partir de 0. Ce principe est dit l'**indexation en base 0**.
- **Basic :** Il débute la numérotation à partir de 1 ou 0.
- **Ada :** Il permet de numéroter les cases à partir d'une valeur quelconque.

Dépassement des bornes

Les langages contrôlent le débordement des bornes d'un tableau et déclenchent une erreur qui généralement arrête le programme. A l'exception du langage C/C++ qui n'effectue aucun contrôle.



1.5 Exemple : Accès indiciel

C/C++ **Exemple C/C++**

Soit le tableau déclaré ainsi :

```
int tab[100];
```

Explication

Il est interdit d'utiliser `tab[-1]` ou `tab[100]`. De plus, chaque élément `tab[j]` (avec `j` dans [0..99]) doit être manié avec la même précaution qu'une variable simple, c.-à-d. qu'on ne peut utiliser un élément du tableau qui n'aurait pas été préalablement affecté ou initialisé.

2 Tableaux et paramètres

2.1 Synonyme de type

Synonyme de type

Alias d'un type existant (lorsqu'un nom de type est trop long ou est difficile à manipuler).

Synonyme de type

```c
typedef TypeExistant TypeAbrege;
```

Explication

Désigne l'identifiant `TypeAbrege` comme étant un synonyme du type `TypeExistant`.

Typedef = Définition

`Typedef` N'introduit pas de nouveau type mais un **nouveau nom** pour le type.

2.2 Tableau et paramètres

Le passage par valeur a pour conséquence de créer une copie de l'entité passé en paramètre effectif. Dans le cas d'un tableau, cette opération est coûteuse en temps et en mémoire.

L'algorithmique ne se préoccupe pas de cet aspect technique. Par contre il convient de tenir compte des caractéristiques du langage de programmation pour réaliser le passage des paramètres le plus adéquat.

L'autre point particulier est qu'il serait utile de pouvoir appeler le même module avec des tableaux de « tailles » différentes. Pour permettre cela, il convient de passer également la taille logique `n` en paramètre.

Définitions C

Soient les définitions suivantes :

```c
enum {TMAX = ...};
typedef T Tableau[TMAX]; // avec T un type quelconque
```

Tableau et paramètres

Le prototype des modules sera :

```c
... ssprg(T tab[TMAX], int n, ...) // tab en modification
... ssprg(Tableau tab, int n, ...) // autre écriture

... ssprg(const T tab[TMAX], int n, ...) // tab en lecture seule
... ssprg(const Tableau tab, int n, ...) // autre écriture
```

Explication

La transmission des tableaux se fait **toujours** par adresse, d'où la perte de son nombre d'éléments et la possibilité de les modifier.

Lors d'un **passage par adresse constant**, le compilateur vérifie que le paramètre n'est accédé qu'en lecture : ceci garantit efficacité et sécurité.

2.3 Tableau et fonction

Tableau et fonction
Une fonction **ne peut pas** fournir un résultat qui soit un tableau.

2.4 Exemple : Saisie et affichage d'un tableau

La saisie et l'affichage sont deux opérations de base d'un tableau. Nous allons donc écrire deux procédures utilitaires `saisirTab` et `afficherTab`. Voici d'abord l'algorithme :

Programme C

```c
#define TMAX 20
typedef int Tableau[TMAX];
int main(void)
{
  Tableau tab;
  int nelems = saisirTab(tab);
  afficherTab(tab,nelems);
}
```

Explication

L'algorithme déclare un tableau d'entiers `tab` de taille maximale `TMAX`. La procédure `saisirTab` a pour effet de demander le nombre d'éléments, de saisir les valeurs dans `tab` puis elle renvoie le nombre d'éléments saisis, ce dernier étant mémorisé dans l'entier `nelems`. La procédure `afficherTab` affiche les `nelems` éléments de `tab`.

Exemple d'exécution

```
Nombre d'éléments dans [1..20]? 10
t[0]? 45
t[1]? 4
t[2]? 1
t[3]? -56
t[4]? 22
t[5]? 34
t[6]? 49
t[7]? 12
t[8]? 0
t[9]? -27
[45 4 1 -56 22 34 49 12 0 -27 ]
```

Fonction saisirTab

```c
int saisirTab(Tableau t)
{
  printf("Nombre de valeurs dans [1..%d]? ",TMAX);
  int n;
  scanf("%d",&n);
  while(!(1<=n && n<=TMAX))
  {
     printf("ERREUR dans [1..%d]? ",TMAX);
     scanf("%d",&n);
  }
  int ix;
```

```
  for (ix=0; ix<n; ++ix)
  {
    printf("t[%d]? ",ix);
    scanf("%d",&t[ix]);
  }
}
```

Explication

La fonction `saisirTab` effectue une saisie sécurisée du nombre d'éléments dans `n` (entier) compris dans `[1..TMAX]` puis saisit un à un `n` entiers et les stocke dans un `ITableau t` et renvoie `n`.

Procédure afficherTab

```
void afficherTab(const Tableau t,int n)
{
  printf("[");
  int ix;
  for (ix=0; ix<n; ++ix)
  {
    printf("%d ",t[ix]);
  }
  printf("]\n");
}
```

Explication

La procédure `afficherTab` affiche à la queue-leu-leu (séparés par un espace) les `n` éléments d'un `ITableau t`.

3 Parcours d'un tableau

Les tableaux interviennent dans de nombreux problèmes : il est primordial de savoir les parcourir en utilisant des algorithmes **corrects, efficaces et lisibles**.

Cette section examine les situations courantes et quelles solutions conviennent. On peut aussi envisager d'autres parcours (à l'envers, une case sur deux, ...) mais ils ne représentent aucune difficulté nouvelle.

Déclarations

```c
const int TMAX = ...;
typedef T Tableau[TMAX]; // avec T un Type quelconque
Tableau tab;
```

3.1 Parcours complet

Pour parcourir **complètement** un tableau, la répétitive `Pour` est le moyen le plus simple comme dans l'algorithme suivant où « `traiter` » va dépendre du problème concret posé : afficher, modifier, sommer...

Parcours complet

```c
int j;
for (j=0; j<n; ++j)
{
  traiter tab[j];
}
```

3.2 Parcours partiel

Certains algorithmes se contentent de parcourir successivement les différents éléments du tableau jusqu'à rencontrer un élément satisfaisant une certaine condition. Par exemple :

- On cherche la présence d'un élément et on vient de le trouver.
- On vérifie qu'il n'y a pas de 0 et on vient d'en trouver un.

Parcours complet, boucle TantQue

Un tel **parcours partiel** est le plus souvent basé sur une répétitive conditionnelle. La première étape est donc de transformer le `Pour` en `TantQue` ce qui donne :

```
int j=0;
while (j<n)
{
  traiter tab[j];
  ++j;
}
```

Remarque

On peut à présent introduire le test d'arrêt. A la fin de la boucle, une contrainte est qu'on voudra savoir si oui ou non on s'est arrêté prématurément et, si c'est le cas, à quel indice. Il existe essentiellement deux solutions, avec ou sans variable booléenne. En général, la solution [A] sera plus claire si le test est court.

(A) Parcours partiel, Sans variable booléenne

```
int j=0;
while (j<n && test_sur_tab[j]_dit_que_on_continue)
{
  traiter tab[j];
  ++j;
}
if (j>=n)
{
  //on est arrivé au bout
}
else
{
  //arrêt prématuré à l'indice j
}
```

Ne testez pas tab(.j.) à l'extérieur de la boucle

Car `j` n'est peut-être pas valide.

(B) Parcours partiel, Avec variable booléenne

```
int j = 0;
bool trouve = false;
while (j<n && !trouve)
```

```
{
  if (test_sur_tab[j]_dit_que_on_a_trouvé)
  {
    trouve = true;
  }
  else
  {
    ++j;
  }
}
//tester le Booléen Pour savoir Si arrêt prématuré
```

Attention

Choisissez un nom de booléen adapté au problème et initialisez-le à la bonne valeur. Par exemple, si la variable s'appelle « `bcontinue` » :

- Initialisez la variable à `Vrai`.
- Le test de la boucle est « ...`Et bcontinue` ».
- Mettez la variable à `Faux` pour sortir de la boucle.

3.3 Parcours imbriqué

Certains algorithmes sur les tableaux font appel à des **boucles imbriquées**. La boucle principale sert généralement à parcourir les cases une à une, tandis que le traitement de chaque case dépend du parcours simple d'une partie du tableau (par exemple toutes les cases restantes) ce qui correspond à la boucle interne.

Parcours imbriqué

```
int j,k;
for (j=0; j<n; ++j)
{
  for (k=...)
  {
    traiter tab[k];
  }
}
```

4 C/C++ - Spécificités

4.1 Tableau variable

Avant la norme C99, tous les tableaux déclarés étaient de dimension fixe. Il n'était pas possible de paramétrer la taille d'un tableau en donnant une valeur au paramètre au moment de l'exécution du programme. La norme C99 a introduit la notion de tableau dont la dimension est une variable.

Exemple : C

Ainsi il est possible d'écrire :

```c
#include <stdio.h>
void ssp(int a)
{
   int t[a];
   printf("(ssp)taille t = %d\n",sizeof(t));
}
int main()
{
   int j = 5;
   int tab[j];
   printf("taille tab = %d\n",sizeof(tab));
   printf("n? ");
   int n;
   scanf("%d",&n);
   int tab2[n];
   printf("taille tb2 = %d\n",sizeof(tab2));
   ssp(10);
}
```

Exemple d'exécution :

```
taille tab = 20
n? 5
taille tb2 = 20
(dans spp)taille t = 40
```