

Utilisation des tableaux en C

L. Mounier

UGA - M2 CCI - PL1

8 octobre 2025

Plan

- 1 Les Tableaux en C
- 2 Représentation contiguë de séquence
- 3 Chaînes de caractères
- 4 Tableaux à plusieurs dimensions

Définir un tableau

Tableau = constructeur de type

- collection **bornée** d'éléments de **même type**
- **accès direct** aux éléments
⇒ index défini sur un **intervalle discret** [a..b]
- intervalle discret = entiers, caractères, type énuméré, ...

Notation algorithmique :

E : un type {type d'un élément }

Tab : tableau sur [a..b] de E

En C :

```
#define N ...
```

```
E Tab[N] ; // Tab = tableau sur [0..N-1] d'elem. de type E
```

Remarque : N fixé à la compilation → tableau **statique**

Opérations sur les tableaux (1)

Valeur initiale :

```
int T[10] = {0, 3, -9, 7, 8} ;  
    // initialise les 5 premieres valeurs  
    // (valeurs suivantes initialisees a 0)  
  
char nom[8] = {'m', 'a', 'r', 't', 'i', 'n'} ;  
    // on verra une meilleure solution
```

Accès aux éléments :

$T_i \rightsquigarrow T[i]$, avec i dans l'intervalle $[0..N-1]$

Opérations sur les tableaux (2)

Affectation :

```
int T1[10] ;  
int T2[10] ;  
T1 = T2 ;      // interdit ...
```

↪ **schéma de parcours** pour copier T2 dans T1 élém. par élém.

Comparaison

```
int T1[10] ;  
int T2[10] ;  
if (T1 == T2) ...    // ne compare pas les 2 tableaux !
```

↪ **schéma de recherche** pour comparer T1 et T2 élém. par élém.

Schéma de parcours en C

→ Appliquer un même traitement à tous les éléments de T

Notation algorithmique : pour i allant de a à b : traiter T_i

En C :

```
#define N ...  
int T[N] ;  
unsigned int i ;  
  
for (i=0 ; i<N ; i++) {  
    traiter T[i] ...  
}
```

Exemple : incrémenter tous les éléments d'indice pair de T

Schéma de recherche en C

→ $\exists$ un élément vérifiant une propriété P dans T ?

Notation algorithmique :

$i \leftarrow a$

tantque ($i \leq b$ et puis non $P(T_i)$) : $i \leftarrow i+1$

{si $i \leq b$ alors $P(T_i)$ }

En C :

```
#define N ...  
int T[N] ;  
unsigned int i ;  
i = 0 ;  
while (i<N && !(P(T[i]))) {  
    i = i+1 ;  
}  
// si i<N alors P(T[i])
```

Schéma de recherche en C (autre solution)

Instruction for et instruction break :

```
#define N ...  
int T[N] ;  
unsigned int i ;  
for (i=0 ; i<N ; i++) {  
    if ( P(T[i]) ) break ;  
}  
// si i<N alors P(T[i])
```

Exemple : recherche d'un élément de valeur négative

Valeurs d'index sur un intervalle (discret) quelconque

Principe général : Transformer $[a..b]$ en $[0..N-1]$

- se ramener à un intervalle d'indice sur les **entiers**
- appliquer une translation vers $[0..N-1]$
 $N = b-a+1$ (nbre d'éléments), $[a..b] \rightsquigarrow [0..b-a]$
- $T_i \rightsquigarrow T[i-a]$

Exemple :

T : tableau $[5..23]$ de caractères ;

$T_x \leftarrow 'c'$;

```
#define N 19    // 23-5+1
char T[N] ;
T[x-5] = 'c'
```

Plan

- 1 Les Tableaux en C
- 2 Représentation contiguë de séquence
- 3 Chaînes de caractères
- 4 Tableaux à plusieurs dimensions

Le problème

Représenter une **séquence** de p éléments :

$$S = [e_0, e_1, \dots, e_{p-1}] \quad (\text{avec } e_i \text{ de type } E)$$

$\hookrightarrow$ représentation **contiguë** dans un tableau T : T_i contient e_i

Pbs :

- dimensionner T : choisir $N > p$ pour toute évolution de $S \dots$
- gérer la longueur courante de S

Deux solutions :

- mémoriser explicitement la longueur de S
- utiliser une marque de fin

Solution avec longueur explicite

Séquence :

le type $\langle T : \text{tableau sur } [0..N-1] \text{ de } E, L : \text{un entier sur } 0..N \rangle$

```
#define N ...
```

```
typedef ... E ; // type d'un element
```

```
typedef struct {
```

```
    E T[N] ;
```

```
    unsigned int L ; // longueur de la sequence S
```

```
} Sequence ;
```

```
Sequence S ;
```

Exercices :

- parcours / recherche dans S
- ajouter un élément en début/fin de S
- supprimer l'élément d'indice i de S

Marque de fin

MarqueFin : la constante ... de type E

Séquence : le type tableau sur $[0..N-1]$ de E

```
#define MarqueFin ...  
#define N ... // prevoir une case pour la marque de fin !  
  
typedef E Sequence[N] ;  
  
Sequence S ;
```

Choix de la marque de fin ?

- un élément de type E
- ne pouvant pas être confondu avec un élément de S

Plan

- 1 Les Tableaux en C
- 2 Représentation contiguë de séquence
- 3 Chaînes de caractères
- 4 Tableaux à plusieurs dimensions

Représentation en C

Une chaîne de caractères est un tableau de caractères :

```
#define N ...  
char S[N] ;
```

Avec :

- L'utilisation par défaut d'une marque de fin : `'\0'`
- Une notation pour les **constantes** :
 `"ceci est une chaine"`
- Un format d'entrées-sorties : `%s`
 `char T[10] ; scanf("%s", T) ;`
- une librairie d'opérateurs prédéfinis : `<string.h>`

La librairie <string.h>

Affectation :

```
char s1[N], s2[N] ;  
strcpy(s1, s2) // s2 est affectee a s1
```

La taille de s1 doit être suffisante ...

Comparaison :

```
char s1[N], s2[N] ;  
int r ;  
r = strcmp(s1, s2)
```

r négatif si $s1 < s2$, positif si $s1 > s2$ et nul si $s1 = s2$.

Longueur :

```
char s1[N] ;  
unsigned int l ;  
l = strlen(s1) // longueur de s1 (marque de fin exclue)
```

Plan

- 1 Les Tableaux en C
- 2 Représentation contiguë de séquence
- 3 Chaînes de caractères
- 4 Tableaux à plusieurs dimensions**

Définition

→ Généralisation directe des tableaux à 1 dimension
(tableaux de tableaux)

Notation Algorithmique :

L, C : des constantes de type entier > 0

T : un tableau sur [1..L] de tableaux sur [1..C] d'entiers

En C :

```
#define L ...    // nbre de lignes
#define C ...    // nbre de colonnes
int T[L][C] ; // T tableau sur [0..L-1, 0..C-1] d'entiers
```

Accès aux éléments : $T_{i,j} \rightsquigarrow T[i][j]$

Exemple

Image 2D en noir et blanc :

tableau 2 dimensions de “pixels” (noirs ou blancs)

```
#define L ...  
#define H ...  
typedef enum {Noir, Blanc} Pixel ;  
Pixel Image[H][L] ;
```

Calcul de l'image “inverse vidéo” :

```
int i, j ;  
for (i=0; i<H ; i++) {  
    for (j=0 ; j<L ; j++) {  
        T[i][j] = (T[i][j] + 1) % 2 ;  
    } ;  
}
```