Langage *C* : tableaux

Sébastien Jean

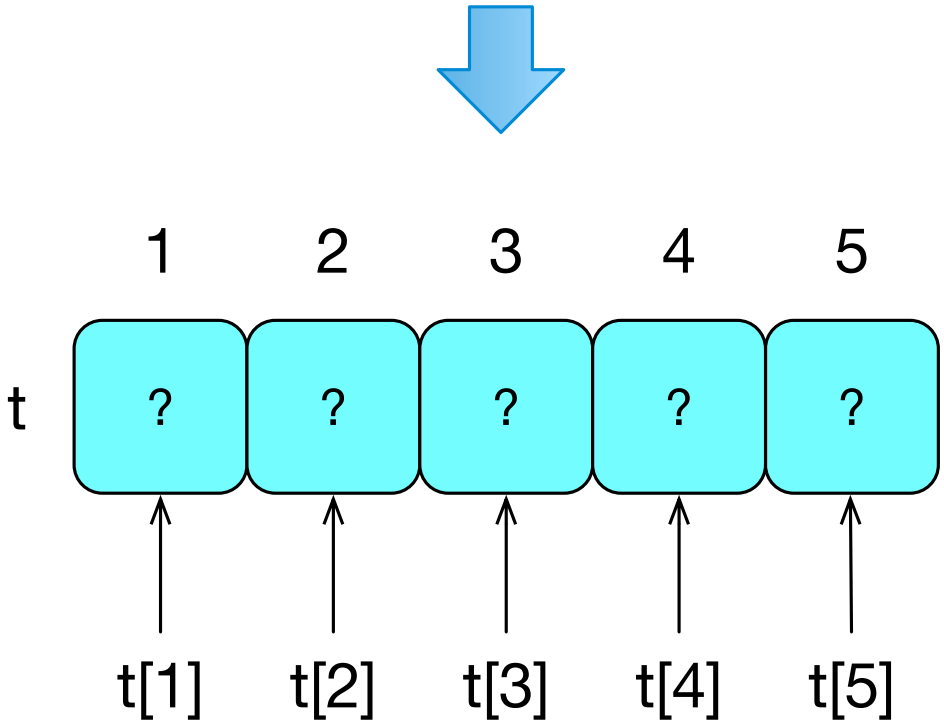
IUT de Valence
Département Informatique

v1.0, 5 novembre 2025

IUT Valence
Université Grenoble Alpes

Tableaux : *Pseudo-code* (rappel)

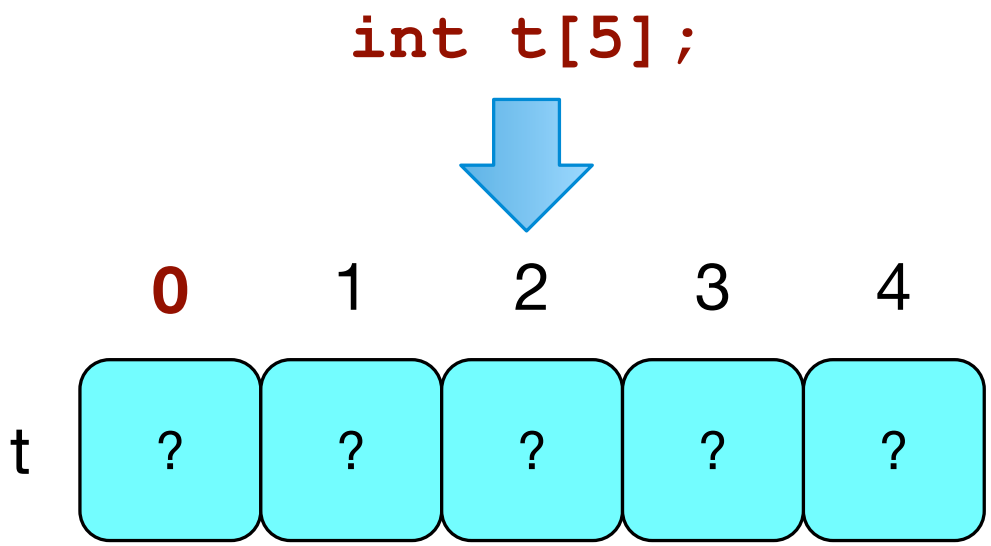
VARIABLE t : tableau d'entiers [5]



IUT Valence
Université Grenoble Alpes

Tableaux 1D en C : variables

- Déclaration d'une variable suivant la syntaxe `type nom[taille]`

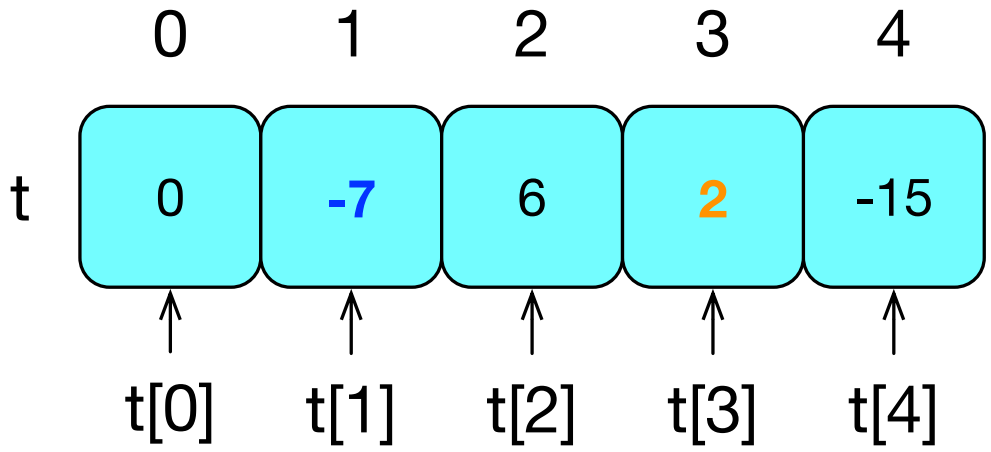


- La déclaration fait exister le tableau (et ses cases)
- Les indices valides sont sur l'intervalle `[0, taille-1]`

IUT Valence Université Grenoble Alpes

Tableaux 1D en C : lecture/affectation

- Désignation d'une case suivant la syntaxe `nom[indice]`

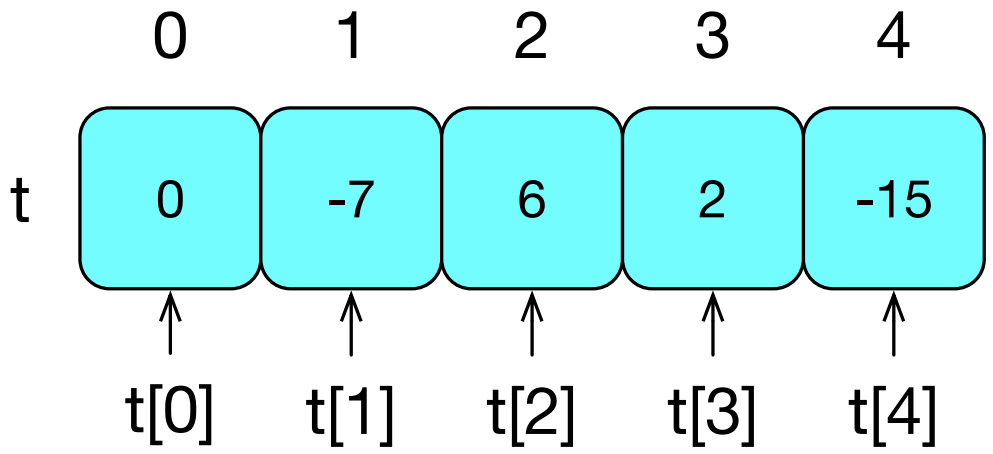


- L'expression `t[1]` vaut le contenu de la 2e case (ici `-7`)
- L'affection de `2` à la 4e case s'effectue via `t[3] = 2`

IUT Valence
Université Grenoble Alpes

Tableaux 1D en C : initialisation

- Initialisation après déclaration
  - case par case, avec ou sans boucle `for` (indice de `0` à `taille-1`)
- Initialisation pendant la déclaration
  - Liste entre accolades de toutes les valeurs, dans l'ordre
  - Ex. : `int tab[3] = {3, 2, 1};`



IUT Valence Université Grenoble Alpes

Passage de tableau 1D en paramètre

- Paramètre de type tableau 1D dans la signature d'une fonction
  → **pas d'indication obligatoire de la taille**
- Appel de fonction avec un paramètre de type tableau
  → **variable de type tableau** passée comme valeur
  - N.B. : La **réaffectation des cases d'un tableau est permanente et visible au retour de la fonction** (cf. plus tard)

Code C

```c
int f(int t[]) {
  ...
}

int main() {
  int tab[3] = {3, 2, 1};
  printf("%d\n", f(tab));
}
```

Interlude : Création/clonage d'un projet Gitlab

- **Créer un projet** `TableauxC` sur Gitlab (avec un `README`)
- **Cloner** le projet depuis *VsCode* et ouvrir le dépôt
- **Modifier** le fichier `README.md` pour indiquer à quoi sert ce projet
- N.B. : pour chacun des exercices `ExerciceX` suivant :
  - **Ecrire le programme** dans `ExerciceX/src/main.c` et le compiler dans `ExerciceX/build/ExerciceX`
  - Rédiger un jeu d'essai dans un fichier `ExerciceX/Essai`

IUT Valence Université Grenoble Alpes

Exercice 1 : spécification de la fonction

Enoncé du problème

Etant donnés un tableau d'entiers `t` et sa taille `n`, on souhaite calculer la moyenne des éléments du tableau.

Spécification du problème

- **Donnée d'entrée** : *t*, tableau d'entiers (tableau rempli de valeurs)
- **Donnée d'entrée** : *n*, entier (nombre de valeurs de `t`)
- **Donnée de sortie** : *moy*, réel (moyenne des éléments de `t`)
- **Pré-condition** : $n \geq 1$
- **Post-condition** : `moy` est égal à la valeur moyenne des éléments de `t`.

Signature de la fonction

- **moy_tab** (t : tableau d'entiers, n : entier) : réel

IUT Valence Université Grenoble Alpes

Exercice 1 : *pseudo-code*

```
FONCTION moy_tab (t : tableau d entiers,  n : entier)
                 : réel

 VARIABLE somme  : entier
 VARIABLE indice : entier

 somme ← 0

 POUR indice de 1 A n PAR PAS DE 1

  somme ← somme + t[indice]

 FIN POUR

 RETOURNER somme / n

FIN FONCTION
```

Exercice 1 : traduction en C

Code C

```c
#include <stdio.h>

float moy_tab(int[] t, int n) {

  int somme = 0;

  for (int indice=0; indice<n; indice=indice+1) {
    somme = somme + t[indice];
  }

  // (float) somme
  // construit un flottant à partir d'un entier

  return ((float) somme) / n;
}
```

Exercice 1 : traduction en C

Code C

```c
 ...

int main() {

    int t[5] = {-7, 5, 4, 0, -12};
    float moyenne = moy_tab(t, 5);
    printf("la moyenne du tableau est %f\n", moyenne);
    return 0;
}
```

IUT Valence Université Grenoble Alpes

Exercice 1 : traduction en C

Exemple de jeu d'essai

```
t = {-7,5,4,0,-12},   n = 5   -> -2.000000
t = {1},              n = 1   -> 1.000000
t = {1,2,3},          n = 3   -> 2.000000
t = {0,0,0},          n = 3   -> 0.000000
```

IUT Valence
Université Grenoble Alpes

Exercice 2 : spécification

Enoncé du problème

Etant donnés un tableau d'entiers `t` et sa taille `n`,
on souhaite **afficher une représentation du contenu du tableau**
sous la forme `{v1, v2, ... , vn}`.

Spécification du problème

- **Donnée d'entrée** : t, tableau d'entiers (tableau rempli de valeurs)
- **Donnée d'entrée** : n, entier (nombre de valeurs de `t`)
- **Donnée de sortie** : aucune (affichage seulement)
- **Pré-condition** : $n \geq 1$
- **Post-condition** : l'affiche produit est de la forme `{ v1, ... , vn}`, avec `vi` représentant la valeur de la case d'indice `i` de `t` (pour `i` de `1` à `n`)

Signature de la fonction

- **affichage_tab** (t : tableau d'entiers, n : entier) : (aucun)

Exercice 2 : *pseudo-code*

```
FONCTION affichage_tab (t : tableau d'entiers,
                        n : entier) : (aucun)

 VARIABLE indice: entier

 afficher("{")
 POUR indice de 1 A n PAR PAS DE 1
  afficher(t[indice])
  SI indice < n ALORS
   afficher(", ")
  FIN SI
 FIN POUR
 afficher("}")
 afficher_saut_ligne()

FIN FONCTION
```

Exercice 2 : traduction en C

Code C

```c
#include <stdio.h>

void affichage_tab(int t[], int n) {
    printf("{");
    for (int indice = 0; indice < n; indice = indice +
        1) {
        printf("%d", t[indice]);
        if (indice < n-1) {
            printf(", ");
        }
    }
    printf("}\n");
}
 ...suite après
```

Exercice 2 : traduction en C

Code C

```
 ...

int main() {

    int t[5] = {-7, 5, 4, 0, -12};
    affichage_tab(t, 5);
    return 0;
}
```

IUT Valence
Université Grenoble Alpes

Exercice 2 : traduction en C

Exemple de jeu d'essai

```
{-7, 5, 4, 0, -12}  -> {-7, 5, 4, 0, -12}
{1}                 -> {1}
```

Exercice 3 : spécification

Enoncé du problème

Etant donnés un tableau d'entiers `t`, sa taille `n` et une valeur `v`, on souhaite **trouver l'indice de la première occurrence de `v` dans `t` si elle existe**.

Spécification du problème

- **Donnée d'entrée** : *t*, tableau d'entiers (tableau rempli de valeurs)
- **Donnée d'entrée** : *n*, entier (nombre de valeurs de `t`)
- **Donnée d'entrée** : *v*, entier (valeur à trouver)
- **Donnée de sortie** : *indice*, entier (l'indice recherché)
- **Pré-condition** : $n \geq 1$
- **Post-condition** : $1 \leq$ `indice` $\leq$ `n` si la case d'indice `indice` de `t` vaut `v` et qu'aucune autre case d'indice inférieur à `indice` ne vaut `v`, `indice` = `-1` si aucune case de `t` ne contient la valeur `v`.

Signature de la fonction

- **premier_indice** (t : tableau d'entiers, n : entier, v : entier) : entier

Exercice 3 : *pseudo code*

```
FONCTION premier_indice (t : tableau d'entiers,
                         n : entier, v : entier) : entier

 VARIABLE indice    : entier

 POUR indice de 1 A n PAR PAS DE 1

  SI t[indice] = v ALORS
   RETOURNER indice

 FIN POUR

 RETOURNER -1

FIN FONCTION
```

Exercice 3 : traduction en C

Code C

```c
#include <stdio.h>

int premier_indice(int t[], int n, int v) {
    for (int indice = 0; indice < n; indice = indice +
        1) {
        if (t[indice] == v) {
         return indice;
        }
    }
    return -1;
}
```

...suite après

Exercice 3 : traduction en C

Code C

```c
 ...

int main() {

    int t[5] = {-7, 5, 4, 4, -12};
    printf("%d\n", premier_indice(t, 5, 4));
    return 0;
}
```

IUT Valence Université Grenoble Alpes

Exercice 3 : traduction en C

Exemple de jeu d'essai

```
t = {-7, 5, 4, 4, -12},     n = 5, v = -7     -> 0
t = {-7, 5, 4, 4, -12},     n = 5, v = 4      -> 2
t = {-7, 5, 4, 4, -12},     n = 5, v = -12    -> 4
t = {-7, 5, 4, 4, -12},     n = 5, v = 1      -> -1
t = {1},                    n = 5, v = 1      -> 0
t = {1},                    n = 5, v = 0      -> -1
```

IUT Valence Université Grenoble Alpes

Tableau comme résultat

- **Il n'est pas possible de spécifier int[]** dans la signature d'une fonction pour retourner un tableau
  - N.B. : cela est possible avec des pointeurs (cf. plus tard)
- Par contre, il est possible de **passer en paramètre un tableau** qui servira de **résultat et qui sera rempli dans la fonction**

Code C

```
void que_des_zeros(int t[], int n) {
 for (int i=0; i<n;i++) {
  t[i] = 0;
 }
}
 ...suite après
```

Tableau comme résultat (fin)

- Comme les modifications des cases d'un tableau passé en paramètre sont permanentes, il suffit de le remplir dans la fonction.

Code C

```c
 ...

int main() {

 const int TAILLE = 6;
 int tab[TAILLE];

 que_des_zeros(tab, TAILLE);

 for (int i=0; i<TAILLE; i++) {
  printf("%d ", tab[i]};
 }
}
```

Exercice 4 : spécification

Enoncé du problème

Etant donnés un tableau d'entiers `t` et sa taille `n`, on souhaite **produire un tableau d'entiers de même taille qui contient les mêmes valeurs que `t` mais dans l'ordre inverse**.

Spécification du problème

- **Donnée d'entrée** : *t*, tableau d'entiers (tableau rempli de valeurs)
- **Donnée d'entrée** : *n*, entier (nombre de valeurs de `t`)
- **Donnée de sortie** : *tinv*, tableau d'entiers (le tableau produit)
- **Pré-condition** : $n \geq 1$
- **Post-condition** : `tinv` est de même taille que `t`, possède les mêmes valeurs mais dans l'ordre inverse.

Signature de la fonction

- **tab_inverse** (t : tableau d'entiers, n : entier) : tableau d'entiers

Exercice 4 : *pseudo code*

```
FONCTION tab_inverse (t : tableau d'entiers,
                      n : entier) : tableau d'entiers

 VARIABLE tinv  : tableau d'entiers [n]
 VARIABLE indice: entier

 POUR indice de 1 A n PAR PAS DE 1

  tinv[n+1-indice] ← t[indice]

 FIN POUR

 RETOURNER tinv

FIN FONCTION
```

Exercice 4 : traduction en C

Code C

```c
#include <stdio.h>

void tab_inverse(int[] tab, int[] inv, int n) {

 for (int i=0; i < n; i=i+1) {
  inv[n-i-1] = tab[i];
 }
}
 ...suite après
```

Exercice 4 : traduction en C

Code C

```
 ...

int main() {

 int tab[5] = {-7, 5, 12, 0, 8};
 int tab_inv[5];

 tab_inverse(tab, tab_inv, 5);

 affichage_tab(tab, 5);
 affichage_tab(tab_inv, 5);

 return 0;
}
```

Université Grenoble Alpes

Exercice 4 : traduction en C

Exemple de jeu d'essai

```
t = {1},          n = 1    -> {1}
t = {1, 2},       n = 2    -> {2, 1}
t = {1, 2, 3},    n = 3    -> {3, 2, 1}
```

IUT Valence
Université Grenoble Alpes