Langage *C* : tableaux 2D/3D

Sébastien Jean

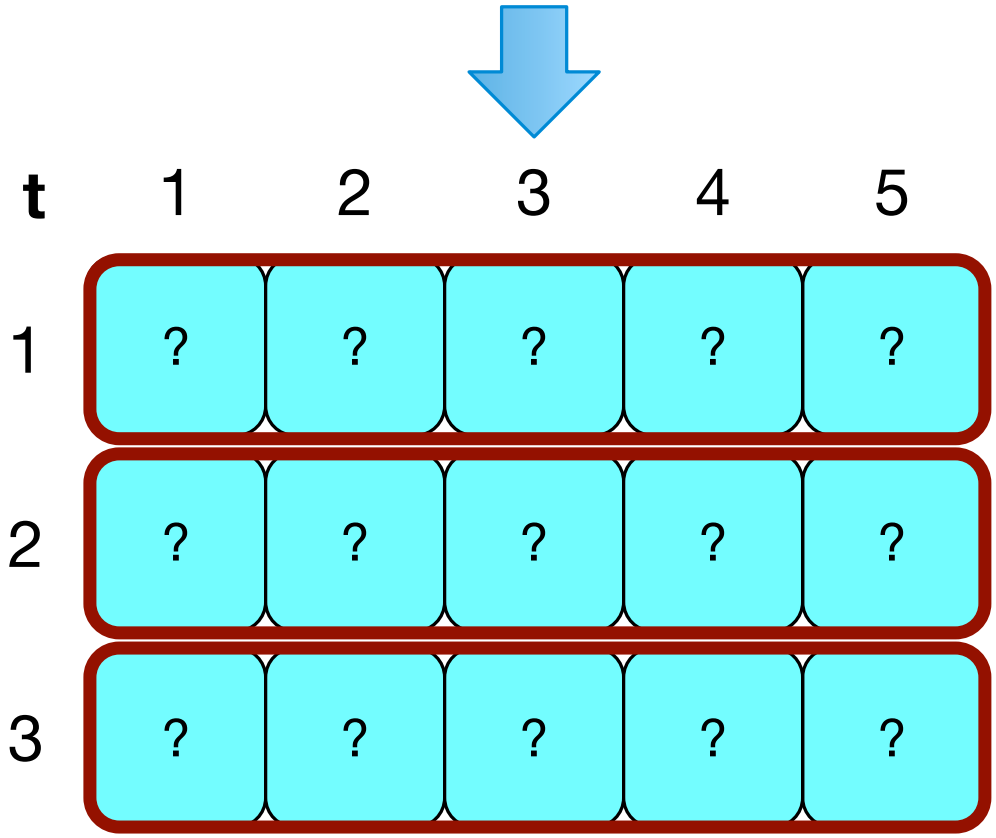
IUT de Valence
Département Informatique

v1.0, 5 novembre 2025

IUT Valence
Université Grenoble Alpes

Tableaux 2D (et +) : *Pseudo-code* (rappel)

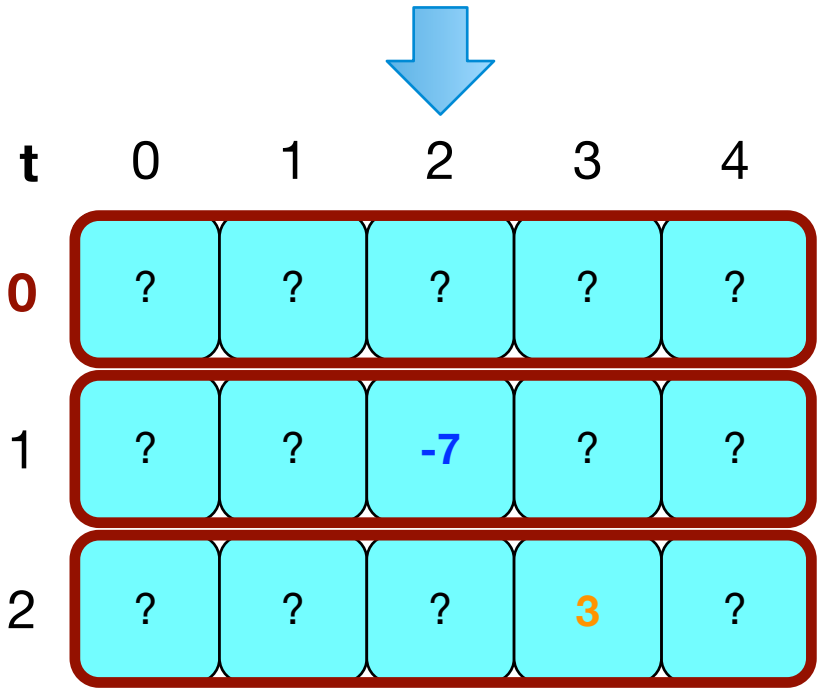
VARIABLE t : tableau 2D d'entiers [3,5]



IUT Valence
Université Grenoble Alpes

Tableaux 2D (et +) en C : déclaration et manipulation

```
int t[3][5];
```



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

IUT Valence Université Grenoble Alpes

Tableaux 2D (et +) en C : initialisation

- Initialisation après déclaration
  - case par case, avec ou sans double boucle `for` (sur chaque dimension)
- Initialisation pendant la déclaration
  - Liste de liste entre accolades de toutes les valeurs, dans l'ordre

Code C

```c
int t[3][4] = {
                {-7, 5, 4, 0},
                {-12, 4, 57, 25},
                {-1, 1, 1, -1}
              };
```

IUT Valence Université Grenoble Alpes

Passage de tableau 2D/3D en paramètre

- Paramètre de type tableau 2D/3D/... dans la signature d'une fonction
  → **indication obligatoire de la taille** pour chacune des dimensions autre que la 1ère

Code C

```c
int f(int t[][3]) {
  ...
}

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

- N.B. : passer des tableaux en paramètre sans devoir indiquer la taille des dimensions reste possible avec des pointeurs (cf. plus tard)

Exercice 1 : spécification

Enoncé du problème

On veut calculer le pourcentage de pixels blancs d'une image noir et blanc.

Spécification du problème

- **Donnée d'entrée** : t, tableau 2D de booléens
  - Les dimensions sont des constantes globales `HAUTEUR` et `LARGEUR`
  - L'intersection ligne/colonne représente un pixel (`VRAI` pour blanc)
- **Donnée de sortie** : p, réel (pourcentage de blanc)
- **Pré-conditions** : le tableau `t` a exactement `HAUTEUR` lignes et `LARGEUR` colonnes.
- **Post-condition** : `p` est égal au pourcentage de pixels blanc de `t`.

Signature de la fonction

- **pour_blanc** (t : tableau 2D de booléens) : réel

Valence
Université Grenoble Alpes

Exercice 1 : *pseudo code*

```
CONSTANTE HAUTEUR : entier (4)
CONSTANTE LARGEUR : entier (4)

FONCTION pour_blanc (t : tableau 2D de booléens) : réel
 VARIABLE somme   : entier
 VARIABLE ligne   : entier
 VARIABLE colonne : entier

 somme ← 0
 POUR ligne de 1 A HAUTEUR PAR PAS DE 1
  POUR colonne de 1 A LARGEUR PAR PAS DE 1
   SI t[ligne][colonne] = VRAI ALORS
    somme ← somme + 1
   FIN SI
  FIN POUR
 FIN POUR
 RETOURNER (somme / (HAUTEUR * LARGEUR)) * 100
FIN FONCTION
```

IUT Valence
Université Grenoble Alpes

Exercice 1 : traduction en C

Code C

```c
#include <stdio.h>
#include <stdbool.h>
#define HAUTEUR 4
#define LARGEUR 4

float pour_blanc(bool t[HAUTEUR][LARGEUR]) {
  float somme = 0.0; // float pour division réelle

  for (int lig = 0; lig < HAUTEUR; lig = lig+1) {
    for (int col = 0; col < LARGEUR; col = col+1) {
      if (t[lig][col]) {
        somme = somme + 1;
       }
    }
  }
  return (somme / (HAUTEUR * LARGEUR)) * 100;
}
```

Exercice 1 : traduction en C

Code C

```
 ...

int main() {

  bool t[HAUTEUR][LARGEUR] =
  { {true, true, true, true},
    {false, false, false, false},
    {true, false, false, true},
    {false, true, false, false} };

  printf("%f\n", pour_blanc(t));
  return 0;
}
```

Exercice 1 : traduction en C

Exemple de jeu d'essai

```
t = {
     {false,false,false,false}, {false,false,false,false},
     {false,false,false,false}, {false,false,false,false}
    }  -> 0.000000

t = {
     {true,true,true,true}, {true,true,true,true},
     {true,true,true,true}, {true,true,true,true}
    } -> 100.000000

t = {
     {true,true,true,true}, {false,false,true,true},
     {true,true,false,false}, {false,false,false,false},
    } -> 50.000000
```

Exercice 2

Enoncé du problème

On veut produire l'image en miroir horizontal d'une image noir et blanc.

Spécification du problème

- **Donnée d'entrée** : *t*, tableau 2D de booléens
  - Les dimensions sont des constantes globales `HAUTEUR` et `LARGEUR`
- **Donnée de sortie** : *m*, tableau 2D de booléens (image miroir)
- **Post-condition** : `m` est le tableau représentant l'image miroir horizontal de l'image représentée par `t`.

Signature de la fonction

- **miroir_h** (t : tableau 2D de booléens) : tableau 2D de booléens

IUT Valence
Université Grenoble Alpes

Exercice 2

```
FONCTION miroir_h (t : tableau 2D de booléens) :
                    tableau 2D de booléens)

 VARIABLE ligne   : entier
 VARIABLE colonne : entier

 POUR ligne de 1 A HAUTEUR PAR PAS DE 1

  POUR colonne de 1 A LARGEUR PAR PAS DE 1

   m[ligne][LARGEUR+1-colonne] = t[ligne][colonne]

  FIN POUR

 FIN POUR

FIN FONCTION
```

Exercice 2 : traduction en C

Code C

```c
#include <stdio.h>
#include <stdbool.h>
#define HAUTEUR 4
#define LARGEUR 4

void miroir_h(bool tab[HAUTEUR][LARGEUR],
              bool m[HAUTEUR][LARGEUR]) {

  for (int l = 0; l < HAUTEUR; l = l + 1) {
    for (int c = 0; c < LARGEUR ; c = c + 1) {
      m[l][LARGEUR - 1 - c] = tab[l][c];
    }
  }
} ...suite après
```

Valence
Université Grenoble Alpes

Exercice 2 : traduction en C

Code C

```
 ...

int main() {

  bool tab[HAUTEUR][LARGEUR] =
  { {true, true, false, false},
    {false, false, false, false},
    {false, false, false, false},
    {false, false, true, true} };

  bool miroir[HAUTEUR][LARGEUR];
  miroir_h(tab, miroir);

  affichage_tab2D(miroir);
  return 0;
}
```

Exercice 2 : traduction en C

Exemple de jeu d'essai

```
t = { {false,false,false,false}, {false,false,false,false},
      {false,false,false,false}, {false,false,false,false} }
->
m = { {false,false,false,false}, {false,false,false,false},
      {false,false,false,false}, {false,false,false,false} }

t = { {true,true,false,false}, {false,false,false,false},
      {false,false,true,true}, {false,false,false,false} }
->
m = { {false,false,true,true}, {false,false,false,false},
      {true,true,false,false}, {false,false,false,false} }
```

IUT Valence Université Grenoble Alpes

Exercice 3

Enoncé du problème

On représente une image RGB (4x4x4) par un tableau 3D d'entiers naturels (entre 0 et 255 inclus). La première dimension représente la composante de couleur (RGB), et chacun des tableaux 2D représente l'intensité de chaque pixel de l'image pour cette composante. On veut produire une image équivalente en noir et blanc. On décide qu'un pixel sera noir si la moyenne des 3 composantes pour ce pixel est strictement inférieure à 128.

Spécification du problème

- **Donnée d'entrée** : *t*, tableau 3D d'entiers (4x4x4) (l'image RGB)
- **Donnée de sortie** : *nb*, tableau 2D de booléens (image noir et blanc)
- **Pré-condition** : les valeurs de `t` sont comprises entre 0 et 255 inclus
- **Post-condition** : `nb` est le tableau représentant la binarisation (noir/blanc) de l'image représentée par `t`.

Signature de la fonction

- **noir_blanc** (t : tableau 3D d'entiers, nb : tableau 2D de booléens) :

Exercice 3

```
FONCTION noir_blanc (t : tableau 3D d entiers,
                     nb : tableau 2D de booleens)
                     : (aucun)

 CONSTANTE TAILLE : entier (4)
 CONSTANTE SEUIL  : entier (128)
 VARIABLE ligne   : entier
 VARIABLE colonne : entier
 VARIABLE somme     : réel
```

... suite après

IUT Valence Université Grenoble Alpes

Exercice 3

```
...

POUR ligne de 1 A TAILLE PAR PAS DE 1
 POUR colonne de 1 A TAILLE PAR PAS DE 1

   somme ← 0
   POUR couleur de 1 A 3 PAR PAS DE 1
     somme ← somme + (t[couleur][ligne][colonne])
   FIN POUR

    nb[ligne][colonne] = (somme / 3) >= SEUIL

 FIN POUR
FIN POUR

FIN FONCTION
```