

# Tableaux à deux dimensions

Algorithmes et structures de données, 2025-2026

---

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA  
2025-10-14

En partie inspiré des supports de cours de P. Albuquerque

## Rappel / devoirs : Crible d'Ératosthène

- But :
  - Générer tous les nombres premiers plus petits qu'un entier  $N$ ,
  - en n'utilisant qu'un tableau de booléens,
  - et que des multiplications.
- Exercice : Écrire l'algorithme en C.

# Crible d'Ératosthène : solution

```
#include <stdio.h>
#include <stdbool.h>
#define SIZE 51
int main() {
    bool tab[SIZE];
    for (int i=0;i<SIZE;i++) {
        tab[i] = true;
    }
    for (int i = 2; i < SIZE; i++) {
        if (tab[i]) {
            printf("%d ", i);
            int j = i;
            while (j < SIZE) {
                j += i;
                tab[j] = false;
            }
        }
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

# Réusinage de code (refactoring)

Le réusinage est ?

# Réusinage de code (refactoring)

## Le réusinage est ?

- Le processus de restructuration d'un programme :
  - en modifiant son design,
  - en modifiant sa structure,
  - en modifiant ses algorithmes,
  - mais en **conservant ses fonctionnalités**.

# Réusinage de code (refactoring)

## Le réusinage est ?

- Le processus de restructuration d'un programme :
  - en modifiant son design,
  - en modifiant sa structure,
  - en modifiant ses algorithmes,
  - mais en **conservant ses fonctionnalités**.

## Avantages ?

# Réusinage de code (refactoring)

## Le réusinage est ?

- Le processus de restructuration d'un programme :
  - en modifiant son design,
  - en modifiant sa structure,
  - en modifiant ses algorithmes,
  - mais en **conservant ses fonctionnalités**.

## Avantages ?

- Amélioration de la lisibilité ;
- Amélioration de la maintenabilité ;
- Réduction de la complexité.

# Réusinage de code (refactoring)

## Le réusinage est ?

- Le processus de restructuration d'un programme :
  - en modifiant son design,
  - en modifiant sa structure,
  - en modifiant ses algorithmes,
  - mais en **conservant ses fonctionnalités**.

## Avantages ?

- Amélioration de la lisibilité ;
- Amélioration de la maintenabilité ;
- Réduction de la complexité.

**“Make it work, make it nice, make it fast”, Kent Beck.**



# Réusinage de code (refactoring)

## Le réusinage est ?

- Le processus de restructuration d'un programme :
  - en modifiant son design,
  - en modifiant sa structure,
  - en modifiant ses algorithmes,
  - mais en **conservant ses fonctionnalités**.

## Avantages ?

- Amélioration de la lisibilité ;
- Amélioration de la maintenabilité ;
- Réduction de la complexité.

“Make it work, make it nice, make it fast”, Kent Beck.

## Exercice :

- Réusiner le code se trouvant sur [Cyberlearn](#).

Les tableaux à deux dimensions

## Tableau à deux dimensions (1/4)

**Mais qu'est-ce donc ?**

# Tableau à deux dimensions (1/4)

**Mais qu'est-ce donc ?**

- Un tableau où chaque cellule est un tableau.

**Quels cas d'utilisation ?**

# Tableau à deux dimensions (1/4)

## Mais qu'est-ce donc ?

- Un tableau où chaque cellule est un tableau.

## Quels cas d'utilisation ?

- Tableau à double entrée ;
- Image ;
- Écran (pixels) ;
- Matrice (mathématique) ;

## Tableau à deux dimensions (2/4)

Exemple : tableau à 3 lignes et 4 colonnes d'entiers

indices	0	1	2	3
0	7	4	7	3
1	2	2	9	2
2	4	8	8	9

### Syntaxe

```
int tab[3][4]; // déclaration d'un tableau 3 par 4
tab[2][1]; // accès case: ligne 2, colonne 1
tab[2][1] = 14; // assignation de 14 à la position 2, 1
```

## Tableau à deux dimensions (3/4)

### Exercice :

Déclarer et initialiser aléatoirement un tableau 50x100 avec des valeurs 0 à 255

# Tableau à deux dimensions (3/4)

## Exercice :

Déclarer et initialiser aléatoirement un tableau 50x100 avec des valeurs 0 à 255

```
#define NX 50
#define NY 100
int tab[NX][NY];
for (int i = 0; i < NX; ++i) {
    for (int j = 0; j < NY; ++j) {
        tab[i][j] = rand() % 256; // 256 niveaux de gris
    }
}
```

## Exercice : afficher le tableau



# Tableau à deux dimensions (3/4)

## Exercice :

Déclarer et initialiser aléatoirement un tableau 50x100 avec des valeurs 0 à 255

```
#define NX 50
#define NY 100
int tab[NX][NY];
for (int i = 0; i < NX; ++i) {
    for (int j = 0; j < NY; ++j) {
        tab[i][j] = rand() % 256; // 256 niveaux de gris
    }
}
```

## Exercice : afficher le tableau

```
for (int i = 0; i < NX; ++i) {
    for (int j = 0; j < NY; ++j) {
        printf("%d ", tab[i][j]);
    }
    printf("\n");
}
```

# Tableau à deux dimensions (4/4)

## Attention

- Les éléments ne sont **jamais** initialisés.
- Les bornes ne sont **jamais** vérifiées.

```
int tab[3][2] = { {1, 2}, {3, 4}, {5, 6} };  
printf("%d\n", tab[2][1]); // affiche?  
printf("%d\n", tab[10][9]); // affiche?  
printf("%d\n", tab[3][1]); // affiche?
```

La couverture de la reine

# La couverture de la reine

- Aux échecs, la reine peut se déplacer horizontalement et verticalement.
- Pour un échiquier 5x6, elle *couvre* les cases comme ci-dessous.

---

	0	1	2	3	4	5
0	*		*		*	
1		*	*	*		
2	*	*	R	*	*	*
3		*	*	*		
4	*		*		*	

---

## Exercice

- En utilisant les structures de contrôle, les tableaux à deux dimensions, et des `char` uniquement.
- Implémenter un programme qui, à partir des coordonnées de la reine, affiche un tableau comme ci-dessus pour un échiquier 8x8.

## Poster le résultat sur `element`