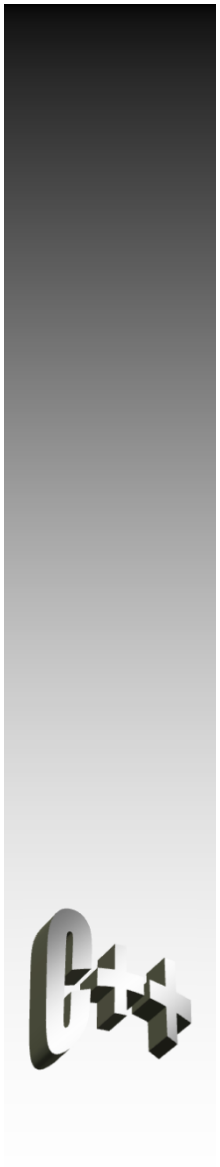


Chapitre 4
Tableaux

Auteur : Marco Lavoie

Adaptation : Sébastien Bois

Langage C++

25908 IFM

Objectifs

- Présenter les tableaux
  - Déclaration
  - Exploitation
- Tableaux comme arguments dans l'invocation de fonctions
- Techniques de manipulation
  - Tri
  - Recherche binaire
- Tableaux multidimensionnels

Aperçu

- Déclaration de tableaux
- Exemples d'utilisation de tableaux
  - Introduction des manipulateurs de flux
- Passer des tableaux à des fonctions
- Trier des tableaux
- Recherche dans les tableaux
- Tableaux à indices multiples

Déclaration d'un tableau

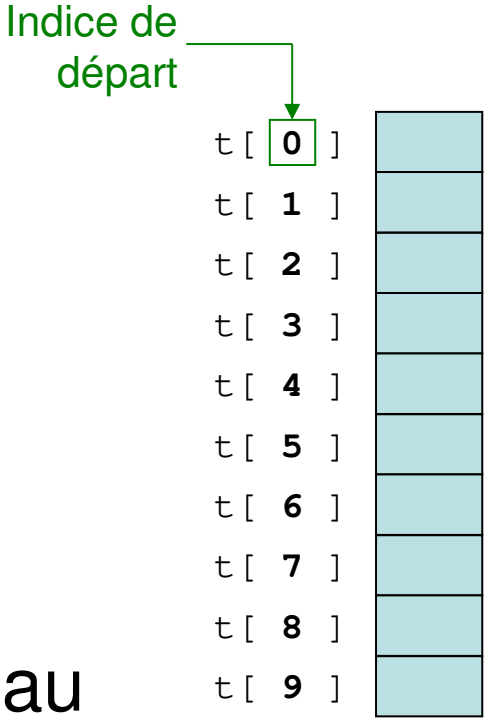
- Tableau unidimensionnel
- Syntaxe

```
type identificateur[ taille ];
```

- Exemple

```
int t[ 10 ];
```

  - Indice de départ est toujours 0
  - Donc indices valides d'un tableau de taille *n* : 0 à *n*-1



Déclaration d'un tableau (suite)

- Tableaux statiques
  - Notez que contrairement au *Java*, il n'est pas nécessaire d'explicitement créer le tableau avec l'opérateur **new**

```
int t[ 10 ] = new int[ 10 ];
```

    - Cette fonctionnalité est une des raisons qui fait du C++ un langage efficace, mais aussi complexe
- L'opérateur `new` sera utiliser plus tard pour créer des tableaux dynamiques

Initialisation d'un tableau

- Tableau non initialisé
  - Par défaut, un tableau a un contenu indéterminé s'il n'est pas initialisé

```
// Contenu de tableau non initialisé
int main() {
  const int TAILLE = 5;
  int i, n[ TAILLE ];

  cout << "Élément" << setw( 14 )
       << "Valeur" << endl;

  for ( i = 0; i < TAILLE; i++ )
    cout << setw( 7 ) << i
         << setw( 14 ) << n[ i ]
         << endl;

  return 0;
}
```

```
C:\>tab.exe
Élément        Valeur
      0    -858993460
      1    -858993460
      2    -858993460
      3    -858993460
      4    -858993460
C:\>
```

Initialisation d'un tableau (suite)

- Initialisation à l'exécution

```
// Contenu de tableau non initialisé
int main() {
  const int TAILLE = 5;
  int i, n[ TAILLE ];

  // Boucle d'initialisation
  for ( i = 0; i < TAILLE; i++ )
    n[ i ] = 0;

  cout << "Élément" << setw( 14 )
       << "Valeur" << endl;

  for ( i = 0; i < TAILLE; i++ )
    cout << setw( 7 ) << i
         << setw( 14 ) << n[ i ]
         << endl;

  return 0;
}
```

```
C:\>tab.exe
Élément         Valeur
      0              0
      1              0
      2              0
      3              0
      4              0
C:\>
```

Initialisation d'un tableau (suite)

- Initialisation à la déclaration
  - Avec l'opérateur d'affectation et une *liste d'initialiseurs*

```
// Contenu de tableau non initialisé
int main() {
  const int TAILLE = 5;
  int i, n[ TAILLE ] = { 10, 20, 30, 40, 50 };

  cout << "Élément" << setw( 14 )
       << "Valeur" << endl;

  for ( i = 0; i < TAILLE; i++ )
    cout << setw( 7 ) << i
         << setw( 14 ) << n[ i ]
         << endl;

  return 0;
}
```

```
C:\>tab.exe
Élément         Valeur
      0             10
      1             20
      2             30
      3             40
      4             50
C:\>
```

Initialisation d'un tableau (suite)

- Initialisation à la déclaration (suite)
  - S'il manque des initialiseurs, le reste du tableau est initialisé à 0

```
// Contenu de tableau non initialisé
int main() {
  const int TAILLE = 5;
  int i, n[ TAILLE ] = { 0 };

  cout << "Élément" << setw( 14 )
       << "Valeur" << endl;

  for ( i = 0; i < TAILLE; i++ )
    cout << setw( 7 ) << i
         << setw( 14 ) << n[ i ]
         << endl;

  return 0;
}
```

```
C:\>tab.exe
Élément         Valeur
      0              0
      1              0
      2              0
      3              0
      4              0
C:\>
```

  - Erreur de syntaxe s'il y a trop d'initialiseurs

Initialisation d'un tableau (suite)

- Initialisation à la déclaration (suite)
  - On peut omettre la taille du tableau s'il est initialisé à la déclaration

```
// Contenu de tableau non initialisé
int main() {
  int i, n[] = { 10, 20, 30, 40 };  // de taille 4

  cout << "Élément" << setw( 14 )
       << "Valeur" << endl;

  for ( i = 0; i < 4; i++ )
    cout << setw( 7 ) << i
         << setw( 14 ) << n[ i ]
         << endl;

  return 0;
}
```

Utilisation d'un tableau

- Les éléments du tableau peuvent être exploités comme une variable

```
// Contenu de tableau non initialisé
int main() {
  const int TAILLE = 5;
  int i, n[ TAILLE ] = { 10 };  // n = { 10, 0, 0, 0, 0 }

  for ( i = 0; i < TAILLE; i++ )
    n[ i ] += i;                // n = { 10, 1, 2, 3, 4 }

  for ( i = 0; i < TAILLE; i++ )
    n[ i ]++;                   // n = { 11, 2, 3, 4, 5 }

  for ( i = 0; i < TAILLE; i++ )
    cout << &n[ i ];            // affiche l'adresse de chaque élément

  return 0;
}
```

Priorité des opérateurs

- Incluant tous les opérateurs vus à date, en ordre décroissant de priorité

Opérateurs	Associativité
`() []`	De gauche à droite
`static_cast<`*`type`*`>() ++ --` *(versions suffixe)*	**De droite à gauche**
`++ -- + -` *(versions préfixe)*	De gauche à droite
`!`	De gauche à droite
`* / %`	De gauche à droite
`+ -`	De gauche à droite
`<< >>`	De gauche à droite
`< <= > >=`	De gauche à droite
`== !=`	De gauche à droite
`&&`	De gauche à droite
`||`	De gauche à droite
`?:`	**De droite à gauche**
`= += -+ *= /= %=`	**De droite à gauche**

Manipulateurs de flux

- Accessibles dans l'espace de nom `std` via `#include <iomanip>`
- Permettent de formater les flux `cin` et (surtout) `cout`
  - Nous verrons plus tard les mapipulateurs en détails (pour lecture et écriture formatées dans les fichiers)
  - Nous nous concentrons uniquement ceux permettant de formater `cout`

Manipulateurs de flux (suite)

- Manipulateurs couramment exploités avec `cout`

Manipulateur	Description
`setw( n )`	Largeur de champ de sortie Exemple : `cout << setw( 10 );`
`setprecision( n )`	Nombre de chiffres affichés après la décimale pour les flottants Exemple : `cout << setprecision( 2 );`
`setiosflags( flags )`	Activation de différentes options de formatage, tels que
`ios::fixed`	force l'affichage des flottants en notation à virgule fixe
`ios::scientific`	force l'affichage des flottants en notation scientifique (ex: 1.4e-17)
`ios::showpoint`	force l'affichage de la décimale des flottants contenant une valeur entière
`ios::left`	justification à gauche dans un champ de sortie trop large
`ios::right`	justification à droite dans un champ de sortie trop large
`ios::hex`	affichage des entiers en base hexadécimale
`ios::dec`	affichage des entiers en base décimale
	Exemple : `cout << setiosflags( ios::fixed | ios::showpoint );`

OU logique

Tableaux de caractères

- Une chaîne peut être stockée dans un tableau de caractères

```
char s[] = "bonjour";
```

  - La taille du tableau `s` est 8, soit
    - Les 7 caractères de **`bonjour`**
    - Un délimiteur de fin de chaîne appelé *caractère nul* et représenté par **`'\0'`**
  - Alternative à l'initialisation ci-dessus :

```
char s[] = { 'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0' };
```

Caractère nul '\0'

- Le délimiteur de fin de chaîne est essentiel
  - Pour l'affichage

```
char s[] = "bonjour";
std::cout << s;   // Affiche jusqu'au '\0'
```

    - Équivalent à

```
for ( int i = 0; s[ i ] != '\0'; i++ )
  std::cout << s[ i ];
```

  - Pour la lecture

```
char s[10];
std::cin >> s;   // Ajoute '\0' à la fin de s
```

    - Équivalent à

```
int  i = 0;
char c;
while ( cin.get( c ) != '\n' )
  s[ i++ ] = c;
s[ i ] = '\0';
```

Source fréquente d'erreurs

- Causes des erreurs de syntaxe ou à l'execution
  - Tableau de caractères trop petit

```
char s[ 7 ] = "bonjour";
```

    - Pas de place pour le `'\0'`
  - Oublier d'ajouter le `'\0'`

```
// Mettre xyz dans s
char s[ 5 ];
s[ 0 ] = 'x';
s[ 1 ] = 'y';
s[ 2 ] = 'z';
std::cout << s;  // Affiche xyz suivi de caractères divers
```

Tableaux en arguments

- Passer un tableau en paramètre à une fonction

```
// Prototypes
void randomiserTableau( int [], int );
void afficherTableau( int [], int );

// Tableau en argument
int main() {
  int tab[5];

  randomiserTableau( tab, 5 );
  afficherTableau( tab, 5 );

  return 0;
}
```

```
// Initialiser contenu de t aléatoirement
void randomiserTableau( int t[], int n ) {
  for ( int i = 0; i < n; i++ )
    t[ i ] = rand();
}

// Afficher le contenu de t
void afficherTableau( int t[], int n ) {
  std::cout << t [ 0 ];
  for ( int i = 1; i < n; i++ )
    std::cout << ", " << t[ i ];
}
```

  - Un tableau est toujours passé en référence
    - Même si le paramètre n'est pas déclaré avec &

Tableaux en arguments

- Pourquoi un tableau est-il toujours passé en référence ?
  - Considérations de performance : s'il était passé par valeur, le contenu du tableau en argument devrait être copié dans le tableau en paramètre $\rightarrow$ $$$ CPU
  - En C++ il est impossible de passer en tableau en paramètre par valeur

Exercice #4.1

- Convertir une chaîne en majuscules
  - Complétez le programme suivant en concevant la fonction **`majuscules() :`**
  - N'oubliez pas les conventions d'écriture
  - Soumettez votre projet selon les indications de l'instructeur

```
int main() {
  const int MAXLEN = 256;
  char chaine[ MAXLEN ] = "";

  std::cout << "Chaîne ? ";
  std::cin >> chaine;

  // Convertir en majuscules
  majuscules( chaine );

  // Afficher le résultat
  std::cout << "Résultat: "
            << chaine << std::endl;

  return 0;
}
```

Paramètres `const`

- Si la fonction ne modifie pas le contenu du tableau reçu en paramètre, ce dernier peut être déclaré constant (`const`)

```
// Prototypes
void randomiserTableau( int [], int );
void afficherTableau( const int [], int );

// Tableau en argument
int main() {
  int tab[5];

  randomiserTableau( tab, 5 );
  afficherTableau( tab, 5 );

  return 0;
}
```

```
// Initialiser contenu de t aléatoirement
void randomiserTableau( int t[], int n ) {
  for ( int i = 0; i < n; i++ )
    t[ i ] = rand();
}

// Afficher le contenu de t
void afficherTableau( const int t[], int n ){
  std::cout << t [ 0 ];
  for ( int i = 0; i < n; i++ )
    std::cout << ", " << t[ i ];
}
```

- `void randomiserTableau(const int t[], int n)` ne compilerait pas car la fonction modifie le contenu de `t[]`

Paramètres `const` (suite)

- Si le tableau en argument est déclaré `const`, le paramètre correspondant doit obligatoirement être déclaré `const`
  - Afin que le compilateur puisse assurer que la fonction ne modifie par l'argument

```
// Prototypes
void fonc( int [], int );

// Tableau en argument
int main() {
  const int tab[] = { 10, 20, 30 };

  fonc( tab, 3 );

  return 0;
}
```

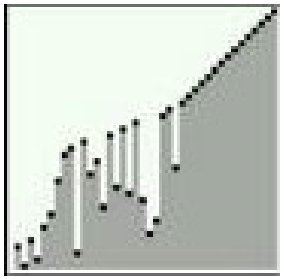
Erreur de compilation : *`fonc` pourrait modifier le contenu de `tab[]`*

```
// Prototypes
void fonc( const int [], int );
```

✓

Trier un tableau

- Il existe plusieurs algorithmes de tri, dont certains sont plus performants
  - Nous étudions le plus simple : **tri par bulles**
    - Analogie aux bulles d'une boisson gaseuze qui remontent à la surface
    - L'algorithme remonte graduellement les plus petites valeurs au haut du tableau
    - À la fin du processus, le tableau est trié en ordre ascendant
    - Code source simple, mais *un des moins performants*

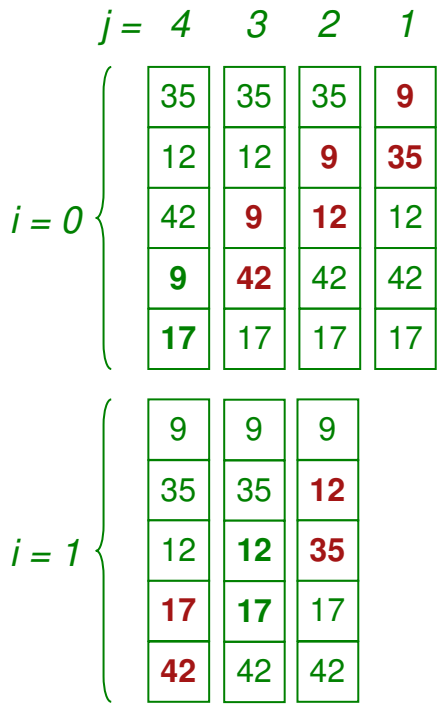


Tri par bulles

- Stratégie : passer plusieurs fois à travers le tableau, de la fin vers le début, en remontant à chaque fois la plus petite valeur rencontrée

```
// Tri par bulles
void triParBulles( int tab[], int taille ) {
  for ( int i = 0; i < taille; i++ )
    for ( int j = taille - 1; j > i; j-- )
      if ( tab[ j ] < tab[ j - 1 ] )
        interchanger( tab[ j ], tab[ j - 1 ] );
}

// Interchange le contenu des deux références
void interchanger( int &a, int &b ) {
  int temp = a;
  a = b;
  b = temp;
}
```



Recherche binaire

- Trouver la position d'une valeur dans le contenu d'un tableau préalablement trié
- Stratégie : comme chercher dans un bottin téléphonique
  - Considérer la valeur au milieu afin de déterminer dans quelle moitié se situe la valeur recherchée
  - Appliquer cette stratégie successivement dans la moitié contenant la valeur

Recherche binaire : code source

- Cette fonction assume que le tableau où chercher est déjà trié

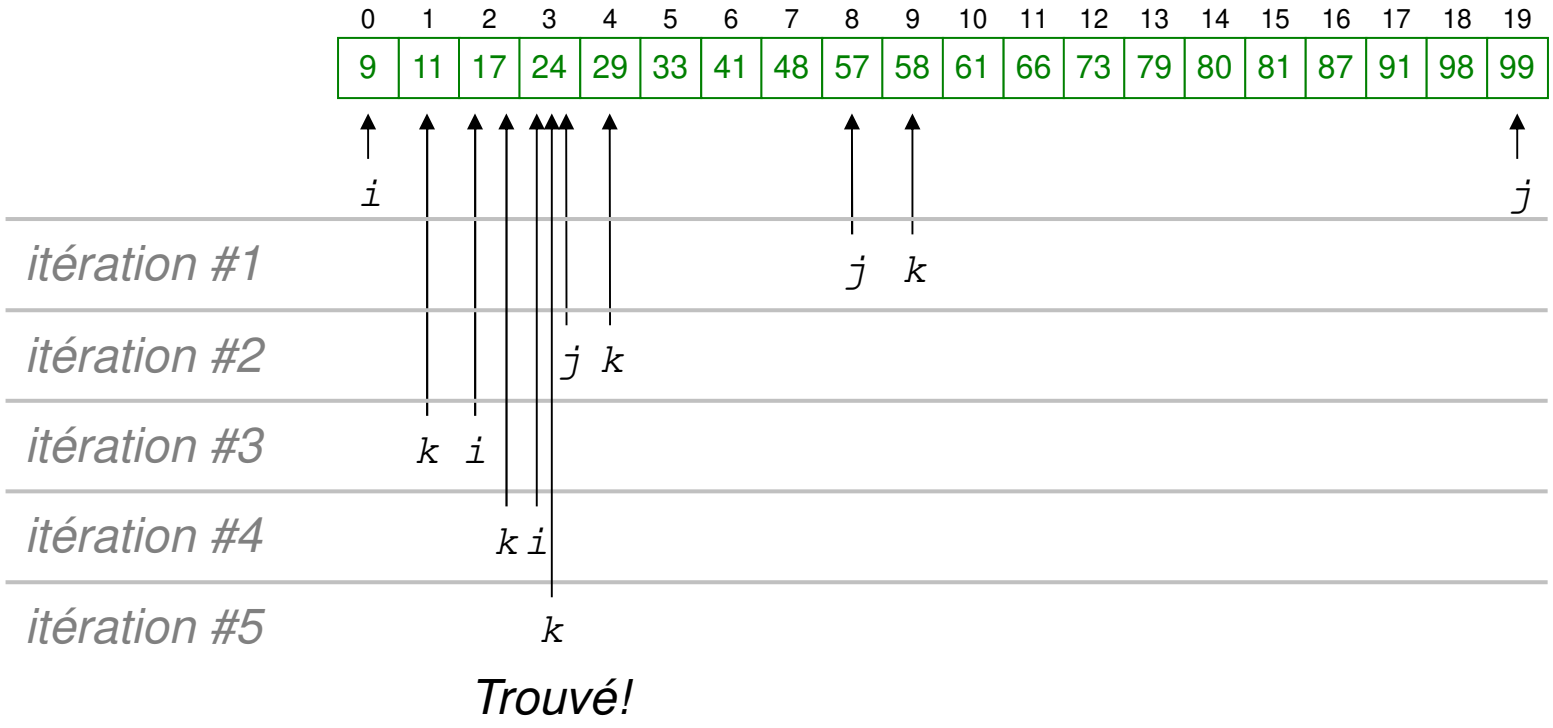
```
// Recherche binaire de valeur dans tableau tab trié
int rechercheBinaire( const int tab[], int taille, int valeur ) {
  int i = 0, j = taille - 1;

  // Tant que valeur non trouvée ou qu'on détermine que
  // valeur n'est pas dans tab (i > j)
  while ( i <= j ) {
    int k = ( i + j ) / 2;          // indice de valeur mitoyenne

    if ( valeur == tab[ k ] )
      return k;                     // valeur trouvée
    else if ( valeur < tab[ k ] )
      j = k - 1;                    // valeur dans première moitié
    else
      i = k + 1;                    // valeur dans dernière moitié
  }
  return -1;  // valeur pas de tab
}
```

Recherche binaire : exemple

- Rechercher 24 (`valeur = 24`)



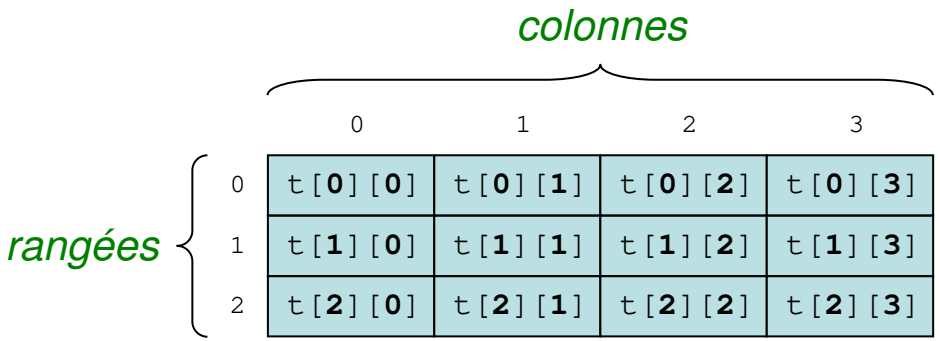
Tableaux à indices multiples

- En C++, un tableau peut posséder jusqu'à 12 dimensions (i.e. indices)
  - Mais on utilise rarement plus de deux dimensions (trop compliqué à déboguer)
- Syntaxe (pour deux dimensions)

```
type identificateur[ rangées ][ colonnes ];
```

- Exemple

```
int t[ 3 ][ 4 ];
```



Initialisation d'un tableau

- Même façon que pour un tableau unidimensionnel
  - Le tableau est initialisé par rangées

```
int t[ 2 ][ 3 ] = { 1, 2, 3, 4, 5, 6 };
```

1	2	3
4	5	6

  - Pour faciliter la lecture

```
int t[ 2 ][ 3 ] = { 1, 2, 3,
                    4, 5, 6 };
```

  - Le compilateur permet aussi d'isoler chaque rangée entre accolades

```
int t[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
```

Ne pas oublier les virgules

Initialisation d'un tableau (suite)

- Attention aux formulations suivantes

```
int t[ 2 ][ 3 ] = { 1, 2, 3, 4 };
```

1	2	3
4	**0**	**0**

```
int t[ 2 ][ 3 ] = { 0 };
```

0	**0**	**0**
0	**0**	**0**

```
int t[ 2 ][ 3 ] = { { 1, 2 }, { 3, 4, 5 } };
```

1	2	**0**
3	4	5

```
int t[ 2 ][ 3 ] = { { 1 }, { 2 } };
```

1	**0**	**0**
2	**0**	**0**

```
int t[ 2 ][ 3 ] = { { 1, 2, 3 } };
```

1	2	3
0	**0**	**0**

Tableaux en paramètre

- Rappel : tableau à une dimension
  - Pas obligé de spécifier la taille du tableau

```
// Prototypes
void fonction( int [], int );

// Code de fonction
void fonction( int t[], int n ) {
  ...
}
```

- Tableau multidimensionnel
  - Seule la première dimension peut être omise

```
// Prototypes
void fonction( int [][10], int, int );

// Code de fonction
void fonction( int t[][10], int m, int n ) {
  ...
}
```

Erreurs de programmation

- Confondre le 7e élément du tableau avec l'élément à l'indice 7
  - Le premier élément commençant à 0, le 7e élément est à l'indice 6
- Oublier de réserver de l'espace pour le caractère nul (`'\0'`) à la fin d'un tableau de caractères contenant une chaîne
- Oublier que les tableaux sont passés en paramètre par référence
  - La fonction peut modifier le contenu du tableau en argument
- Référencer un élément de tableau multidimensionnel par `[x,y]` plutôt que par `[x][y]`
  - `x,y` signifie évalue `x` puis `y`, donc réfère à la rangée `y`

Bonnes pratiques de programmation

- Utiliser une constante pour définir la taille d'un tableau

```
const int NB_ETUDIANTS = 100;
int notes[NB_ETUDIANTS];
```

- Si possible, initialiser le contenu d'un tableau lors de sa déclaration plutôt que via une boucle `for`
- Toujours s'assurer que la valeur d'indice utilisée pour accéder à un tableau de taille *n* est entre 0 et *n*-1 inclusivement

Devoir #3

- Voici un programme principal faisant appel à une fonction à définir
  - Complétez le code source du projet en définissant la fonction manquante
- N'oubliez pas les conventions d'écriture
- Respectez l'échéance imposée par l'instructeur
- Soumettez votre projet selon les indications de l'instructeur

```
int main() {
  const int N = 10;
  int notes[ N ] = { 0 };
  double moyenne,     // moyenne de notes[]
         ecart;       // écart-type de notes[]
  int    mediane;     // valeur médiane de notes[]

  // Lire les notes
  for ( int i = 0; i < N; i++ ) {
    std::cout << "Note #" << i + 1 << "? ";
    std::cin >> notes[ i ];
  }

  // Calculer les statistiques
  statistiques( notes, N, moyenne, ecart,
                mediane );

  // Afficher les résultats
  std::cout << "Moyenne    = " << moyenne
            << "Ecart-type = " << ecart
            << "Médiane    = " << mediane
            << std::endl;

  return 0;
}
```

Pour la semaine prochaine

- Vous devez relire le contenu de la présentation du chapitre 4
  - Il y aura un quiz sur ce contenu au prochain cours
    - À livres et ordinateurs fermés
  - Profitez-en pour réviser le contenu des chapitres précédents