

# Information, Calcul et Communication (partie programmation) : Cours de programmation (C++) Tableaux dynamiques

Jamila Sam

Laboratoire d'Intelligence Artificielle  
Faculté I&C

# Vidéos (1 à 6), transparents et quiz (vector)

[www.coursera.org/learn/initiation-programmation-cpp/](http://www.coursera.org/learn/initiation-programmation-cpp/)

## Semaine 5

# Objectifs du cours d'aujourd'hui

- ▶ Rappels sur les tableaux dynamiques
  - ▶ Concepts centraux
  - ▶ Initialisation
  - ▶ Accès et parcours
  - ▶ Fonctions spécifiques
- ▶ Compléments sur les types de base
- ▶ Etude de cas

# Les vector

|                                                          |     | taille initiale connue <i>a priori</i> ? |                                      |
|----------------------------------------------------------|-----|------------------------------------------|--------------------------------------|
|                                                          |     | non                                      | oui                                  |
| taille pouvant varier lors de l'utilisation du tableau ? | oui | vector                                   | (vector)                             |
|                                                          | non | (vector)                                 | array (C++11)<br>tableaux « à la C » |

**Tableau dynamique** : *collection* de données de *même type*, c'est-à-dire dont le nombre peut changer au cours du déroulement du programme,

En C++ : type `vector`

Nécessite : `#include <vector>`

(En toute rigueur, `vector` n'est pas un type, mais un « *template* ».

Nous verrons les « templates » tard au second semestre).



# Initialisation d'un tableau dynamique

En C++11, il y a cinq façons d'initialiser un tableau dynamique :

- ▶ vide

```
vector<double> tab;
```

- ▶ avec un ensemble de valeurs initiales

```
vector<double> tab({ 2.0, 3.5, 2.6, 3.8, 22.2 });
```

- ▶ avec une taille initiale donnée et tous les éléments « nuls »

```
vector<double> tab(5);
```

- ▶ avec une taille initiale donnée et tous les éléments à une même valeur donnée

```
vector<double> tab(5, 1.0);
```

- ▶ avec une copie d'un autre tableau

```
vector<int> tab(tab2);
```

**Note :** depuis C++17, il n'est pas nécessaire de préciser le type des éléments si celui-ci peut être déduit du contexte

```
vector tab(5, 1.0);
```

# Accès direct aux éléments d'un tableau

Le  $i+1^{\text{ème}}$  élément d'un tableau `tab` est désigné par  
`tab[i]`



**Attention !** Les indices correspondant aux éléments d'un tableau de taille `taille` **varient entre 0 et `taille-1`**

Le 1<sup>er</sup> élément d'un tableau `tab` précédemment déclaré est donc `tab[0]` et son 10<sup>e</sup> élément est `tab[9]`



**Attention !!** Il n'y a **pas de contrôle de débordement !!**

`tab[10000]`

Il est impératif que l'élément que vous désigniez **existe** effectivement !

Sinon risque de **Segmentation Fault** !

Exemple (à ne **pas** suivre !) d'erreur classique :

```
vector<double> v;  
v[0] = 5.4; // NON !!
```

→ `v[0]` n'existe pas encore !

# Accès aux éléments d'un tableau (2)

Très souvent, on voudra accéder aux éléments d'un tableau en effectuant une **itération** sur ce tableau.

Il existe en fait au moins *trois* façons d'itérer sur un tableau :

- ▶  avec les itérations sur ensemble de valeurs

```
for (auto element : tableau)
for (auto& element : tableau)
```

- ▶ [C, C++98] avec une itération **for** « classique » :

```
for (size_t i(0); i < tableau.size(); ++i)
```

- ▶  [C++98] avec des itérateurs (2<sup>e</sup> semestre)

Lequel choisir ?

- ▶ à chaque fois que c'est possible : le premier
- ▶ sinon : le deuxième.

# Fonctions spécifiques

Quelques fonctions disponibles pour un tableau dynamique

`tableau` de type `vector<type>` :

`tableau.size()` : renvoie la taille de `tableau` (type de retour : `size_t`)

`tableau.front()` : renvoie une référence au 1<sup>er</sup> élément  
`tableau.front()` est donc équivalent à `tableau[0]`

`tableau.back()` : renvoie une référence au dernier élément.  
`tableau.back()` est donc équivalent à `tableau[tableau.size() - 1]`

`tableau.empty()` : détermine si `tableau` est vide ou non  
(`bool`).

`tableau.clear()` : supprime tous les éléments de `tableau` (et le transforme donc en un tableau vide). Pas de (type de) retour.

`tableau.pop_back()` : supprime le dernier élément de `tableau`. Pas de (type de) retour.

`tableau.push_back(valeur)` : ajoute un nouvel élément de  
valeur `valeur` à la fin de `tableau`. Pas de (type de) retour.

# Les tableaux multidimensionnels

tableau à plusieurs dimensions : **tableau de tableaux...**

Le type de base d'un tableau peut être n'importe quel type, y compris composé. En particulier, le type de base d'un tableau peut être lui même un tableau.

Exemple :

```
vector<vector<int>> tab(5, vector<int>(6));
```

correspond à la déclaration d'un tableau de 5 tableaux de 6 entiers

`tab[i]` est donc un « `vector<int>` », c'est-à-dire un tableau dynamique d'entiers (qui, au départ, en contient 6)

`tab[i][j]` sera alors le  $(j+1)$ -ième élément de ce tableau.

# Approfondissement

## ► Complément sur les types de base

# Types élémentaires « avancés »

En plus des types composés, signalons qu'il existe aussi d'autres types élémentaires, dérivés des types élémentaires présentés.

Trois **modificateurs** peuvent être utilisés :

- ▶ pour les `int` et les `double`, on peut demander d'avoir une *plus grande précision* de représentation à l'aide du modificateur `long`, et même `long long` (C++11).

Exemple : `long int nb_etoiles;`

- ▶ pour les `int`, on peut aussi demander d'avoir une *moins grande précision* de représentation à l'aide du modificateur `short`.

Exemple : `short int nb_cantons;`

- ▶ pour les `int` (et les `char`), on peut demander de travailler avec des données *non signées*, à l'aide du modificateur `unsigned`.

Exemple : `unsigned int nb_cacahouetes;`

On peut bien sûr combiner :

`unsigned long int nb_etoiles;`

`unsigned short int nb_cantons;`

# Types élémentaires « avancés »

En C++, **la taille des types n'est pas spécifiée** dans la norme.

Seules indications :

- ▶ le plus petit type est `char`
- ▶ les inégalités suivantes sont toujours vérifiées sur les tailles mémoires :

`char ≤ short int ≤ int ≤ long int`

`double ≤ long double`

Cependant, les tailles généralement utilisées sont

8 bits pour les `char`

16 bits pour les `short int`

32 bits pour les `long int`

En , existe(ro)nt également les types entiers de tailles définies : `int8_t`, `uint8_t`, ..., `int64_t`, `uint64_t`

# Types élémentaires « avancés »

Ces choix typiques conduisent aux bornes suivantes (cf `numeric_limits` dans la bibliothèque `<limits>`, par exemple `numeric_limits<int>::min()`) :

| type                            | min.        | max.       |
|---------------------------------|-------------|------------|
| <code>char</code>               | -128        | 127        |
| <code>unsigned char</code>      | 0           | 255        |
| <code>short int</code>          | -32768      | 32767      |
| <code>unsigned short int</code> | 0           | 65535      |
| <code>long int</code>           | -2147483648 | 2147483647 |
| <code>unsigned long int</code>  | 0           | 4294967295 |

| type                     | min. (valeur absolue) | max.          | précision   |
|--------------------------|-----------------------|---------------|-------------|
| <code>double</code>      | 2.22507e-308          | 1.79769e+308  | 2.22045e-16 |
| <code>long double</code> | 3.3621e-4932          | 1.18973e+4932 | 1.0842e-19  |

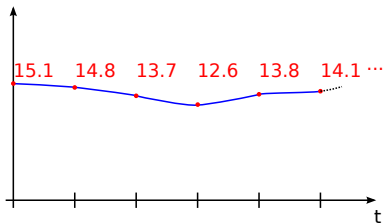
Note : « *précision* » correspond au plus petit nombre  $x$  tel que  $1 + x \neq 1$ .

# Etude de cas

## ► Analyse de données (températures)

# Etude de cas analyse de données

On souhaite écrire un programme qui fasse des statistiques sur un ensemble de relevés de températures :



(moyenne, températures extrêmes etc.)

# Pour préparer le prochain cours

- ▶ Vidéos et quiz du MOOC semaine 5 (suite) :
  - ▶ Tableaux de taille fixe : array [16 :04]
- ▶ Vidéos et quiz du MOOC semaine 6 :
  - ▶ string : introduction [10 :09]
  - ▶ string : traitements [12 :37]