Objectifs

Rappels et compléments

Types élémentaires avancés

Etude de cas

Information, Calcul, Communication (partie programmation) :

Tableaux dynamiques

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam

EPFL

Objectifs

Rappels et compléments

Types élémentaires avancés

Etude de cas

Rappel du calendrier

		MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11
1	12.09.24	--	-1	prise en main	Bienvenue/Introduction
2	19.09.24	1. variables	0	variables / expressions	variables / expressions
3	26.09.24	2. if	0	if – switch	if – switch
4	03.10.24	3. for/while	0	for / while	for / while
5	10.10.24	4. fonctions	0	fonctions (1)	fonctions (1)
6	17.10.24		1	fonctions (2)	fonctions (2)
-	24.10.24				
7	31.10.24	5. tableaux (vector)	1	vector	vector
8	07.11.24	6. string + struct	1	array / string	array / string
9	14.11.24		2	structures	structures
10	21.11.24	7. pointeurs	2	pointeurs	pointeurs
11	28.11.24		-	entrées/sorties	entrées/sorties
12	05.12.24		-	erreurs / exceptions	erreurs / exceptions
13	12.12.24		-	révisions	théorie : sécurité
14	19.12.24	8. étude de cas	-	révisions	Révisions

(ne sont pas sur le MOOC)

©EPFL 2024
Jean-Cédric Chappelier & Jamila Sam

EPFL

Objectifs

Rappels et compléments

Types élémentaires avancés

Etude de cas

Objectifs du cours d'aujourd'hui

- Rappels sur les tableaux dynamiques :
  - concept(s)
  - initialisations
  - parcours, `for(auto x: tab)`
  - « fonctions propres » : `size()`, `pop_back`, `push_back()`
- Complément sur les types de base
- Etude de cas

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Les vector

		taille initiale connue *a priori* ?	
		non	oui
taille pouvant varier lors de l'utilisation du tableau ?	oui	`vector`	(`vector`)
	non	(`vector`)	`array` (C++11) tableaux « à la C »

Tableau dynamique : *collection* de données de *même type*, 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.)

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

C++11 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<double> tab(tab2);
```

Note C++17 : 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);
```

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

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$^{\text{er}}$ élément d'un tableau `tab` précédemment déclaré est donc `tab[0]` et son 10$^{\text{e}}$ élément est `tab[9]`

Attention ! Il n'y a **pas de contrôle de débordement !!** `tab[10000]`
Il est impératif que l'élément que vous désignez **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 !

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

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 (2e semestre)

Lequel choisir ?

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

©EPFL 2024

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 1er 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.

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Les tableaux dynamiques

```
#include <vector>
```

Déclaration : `vector<type> identificateur;`
Déclaration/Initialisation :

```
vector<type> identificateur({ ... });
vector<type> identificateur = { ... };
vector<type> identificateur(taille);
vector<type> identificateur(taille, valeur);
```

Accès au (`i`+1)-ème élément (quand il existe !) : `tab[i]`

Fonctions spécifiques :
`tab.size()` : renvoie la taille (type `size_t`)
`tab.empty()` : détermine s'il est vide ou non (type `bool`)
`tab.clear()` : supprime tous les éléments
`tab.pop_back()` : supprime le dernier élément
`tab.push_back(valeur)` : ajoute un nouvel élément à la fin

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;
```

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Objectifs
Rappels et compléments
Types élémentaires avancés
Etude de cas

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` $\leq$ `short int` $\leq$ `int` $\leq$ `long int`
  `double` $\leq$ `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 C++11, existent également les types entiers de tailles définies :
`int8_t`, `uint8_t`, ..., `int64_t`, `uint64_t`

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Types élémentaires « avancés »

exemples de valeurs possibles
(définis par `numeric_limits` dans la bibliothèque `<limits>` ; par exemple `numeric_limits<int>::min()`) :

type	min.	max.
`char`	–128	127
`unsigned char`	0	255
`short int`	–32768	32767
`unsigned short int`	0	65535
`long int`	–2147483648	2147483647
`unsigned long int`	0	4294967295

type	min. (valeur absolue)	max.	précision
`double`	2.22507e–308	1.79769e+308	2.22045e–16
`long double`	3.3621e–4932	1.18973e+4932	1.0842e–19

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

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Objectifs
Rappels et compléments
Types élémentaires avancés
Etude de cas

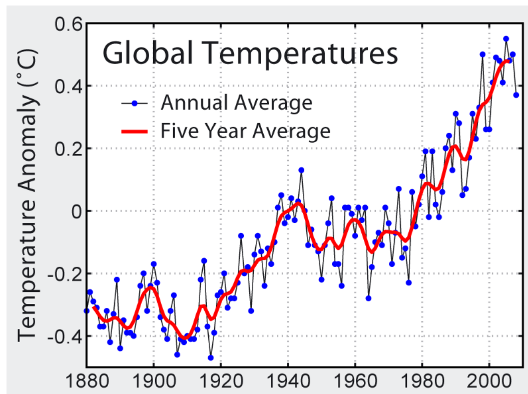
Etudes de cas

- « exercice 0 » de cette semaine : moyenne de classe
- moyenne mobile d'un signal échantilloné
  (par exemple sur des températures journalières)

©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Intérêt ?

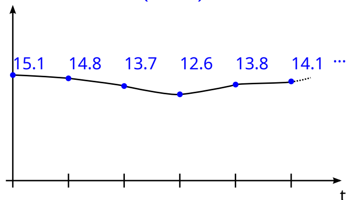
La moyenne mobile est un **filtre passe-bas**
très souvent utilisé pour présenter des résultats, pour les lisser :



©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam

Moyenne mobile sur un signal échantilloné

Signal échantillonné : suite de valeurs $X(nT_e)$:

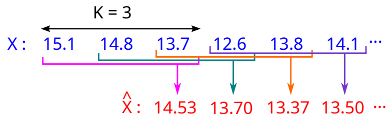


$\longrightarrow$ `vector<double>`

Moyenne mobile : moyenne sur les K valeurs précédentes (K : un nombre entier) :

$$\widehat{X}(nT_e) = \frac{1}{K} \sum_{m=n-K+1}^{n} X(mT_e)$$

Exemple :



©EPFL 2024
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Objectifs
Rappels et compléments
Types élémentaires avancés
Etude de cas

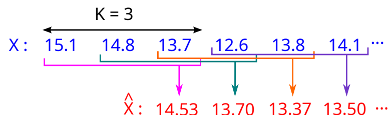
Moyenne mobile sur un signal échantilloné

$$\widehat{X}(n\,T_e) = \frac{1}{K} \sum_{m=n-K+1}^{n} X(m\,T_e)$$

Algorithme ?

DEUX boucles :

- une sur n : calcul de *plusieurs* valeur de $\widehat{X}$
- une sur m : chaque calcul de valeur de $\widehat{X}$ est une somme (boucle)



©EPFL 2024

Jean-Cédric Chappelier
& Jamila Sam
EPFL