

Information, Calcul et Communication (partie programmation) : Cours de programmation (C++) Tableaux et chaînes de caractères

Jamila Sam

Laboratoire d'Intelligence Artificielle
Faculté I&C

Vidéos, transparents et quiz (array,string)

www.coursera.org/learn/initiation-programmation-cpp/

- 👉 Semaine 5 (vidéo 7)
- 👉 Semaine 6 (vidéos 1 et 2)

Objectifs du cours d'aujourd'hui

- ▶ Rappels sur les tableaux de taille fixe
- ▶ Rappels sur les chaînes de caractères
- ▶ Accès et parcours
- ▶ Etude de cas

Etude de cas

► Ancienne série notée

Les array

		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) tableaux « à la C »

Nécessite : C++ 11 et

```
#include <array>
```

Inconvénients des tableaux de taille fixe à la C

Les tableaux de taille fixe à la C :

- ▶ sont toujours passés par référence
- ▶ n'ont pas connaissance de leur propre taille
- ▶ ne peuvent pas être manipulés globalement (pas de « = »)
- ▶ ne peuvent pas être retournés par une fonction
- ▶ ont une syntaxe d'initialisation particulière

👉 **AUCUN** avantage !

Mais je pense qu'ils resteront malgré tout assez répandus (inertie)... : – (

Pour ceux que cela intéresse : voir l'annexe (site du MOOC)



Déclaration d'un tableau de taille fixe

Une variable correspondant à un tableau de taille fixe se déclare de la façon suivante :

```
array<type, taille> identificateur;
```

où *identificateur* est le nom du tableau, *type* correspond au type des éléments du tableau et *taille* est le nombre d'éléments que contient le tableau.

Ce nombre doit être **connu à l'avance** (→ sinon *vector*)



Initialisation d'un tableau de taille fixe

Comme pour les variables de type élémentaire, un tableau de taille fixe peut être initialisé directement lors de sa déclaration :

```
array<type, taille> identificateur({val1, ... , valt});
```

ou

```
array<type, taille> identificateur = {val1, ... , valt};
```

Exemple :

```
constexpr int taille(5);

/* pas encore supporté par tous les *
 * compilateurs :-(
 */
array<int, taille> ages (
    { 20, 35, 26, 38, 22 } );

// alternative :
array<int, taille> ages = {
    20, 35, 26, 38, 22
};
```

Âge
20
35
26
38
22

Un `array` non initialisé contient « n'importe quoi ».

Pour résumer

Tableaux dynamiques

```
#include <vector>
vector<double> tab;
vector<double> tab2(5);
```

```
tab[i][j]
```

```
tab.size()
```

```
for(auto element : tab)
```

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

```
tab.push_back(x);
```

```
tab.pop_back();
```

```
vector<vector<int>> tableau(
    { { 0, 1, 2, 3, 42 },
      { 4, 5, 6 },
      { 7, 8 },
      { 9, 0, 1 } }
);
```

Tableaux statiques

```
#include <array>
```

```
array<double, 5> tab;
```

```
—
```

```
—
```

```
array<array<int, 3>, 4> matrice =
{
    { 0, 1, 2 },
    { 3, 4, 5 },
    { 6, 7, 8 },
    { 9, 0, 1 }
};
```



Les tableaux de taille fixe



```
#include <array>
```

Déclaration : `array<type, taille> identificateur;`

Déclaration/Initialisation :

```
array<type, taille> identificateur =  
    {val1, ... , valtaille};
```

Accès aux éléments : `tab[i]` `i` entre **0** et **taille-1**

Fonctions spécifiques :

`size_t tab.size()` : renvoie la taille

Tableau multidimensionnel :

```
array<array<type, nb_colonnes>, nb_lignes>  
identificateur;  
tab[i][j] = ...;
```

La classe string

```
"Bonjour tout le monde !"
```

Les chaînes de caractères C++ sont définies par le type **string**. (En toute rigueur, ce n'est pas un type comme les types élémentaires mais une classe.)

Pour utiliser des chaînes de caractères, il faut tout d'abord **importer les définitions** :

```
#include <string>
```

La déclaration d'une chaîne de caractères se fait alors avec :

```
string identificateur;
```

Exemple :

```
#include <string>
// declaration
string un_nom;
// declaration avec initialisation
string maxime("Why use Windows when there are doors?");
```

La concaténation : exemple

Constitution du nom complet à partir du nom de famille et du prénom :

```
string nom;  
string prenom;  
string famille;  
  
...  
nom = famille + ' ' + prenom;
```

Ajout d'un 's' final au pluriel :

```
string reponse("solution");  
//...  
if (n > 1) {  
    reponse = reponse + 's';  
}
```

Fonctions spécifiques aux chaînes

Certaines fonctions *propres aux string* sont définies.
Elle s'utilisent avec la syntaxe suivante :

```
nom_de_chaine.nom_de_fonction(arg1, arg2, ...);
```

Les fonctions suivantes sont définies (où *chaine* est une variable de type *string*) :

chaine.size() : renvoie la taille (i.e. le nombre de caractères) de *chaine*.

chaine.insert(position, chaine2) : insère, à partir de la position (indice) *position* dans la chaîne *chaine*, la *string* *chaine2*

Exemple :

```
string exemple("abcd"); // exemple vaut "abcd"  
exemple.insert(1, "xx"); // exemple vaut "axxbcd"
```

construit la chaîne "axxbcd".

Fonctions spécifiques aux chaînes

`chaine.replace(position, n, chaine2)` : remplace les `n` caractères d'indice `position`, `position+1`, ..., `position+n-1` de `chaine` par la `string` `chaine2`.

Exemple :

```
string exemple("abcd");  
exemple.replace(1, 2, "1234");  
construit la chaîne "a1234d" (dans exemple).
```

Remarque : la fonction `replace()` peut également servir à supprimer des caractères dans une chaîne.

Exemple :

```
string exemple("abcd");  
exemple.replace(1, 2, "");  
  
exemple vaut "ad".
```

Fonctions spécifiques aux chaînes

`chaine.find(souschaine)` : renvoie l'indice dans `chaine` du 1er caractère de l'occurrence *la plus à gauche* de la `string` `souschaine`.

Exemple :

```
string exemple("baabbaab");  
exemple.find("ab") renvoie 2.
```

`chaine.rfind(souschaine)` : renvoie l'indice dans `chaine` du 1er caractère de l'occurrence *la plus à droite* de la `string` `souschaine`.

Exemple :

```
string exemple("baabbaab");  
exemple.rfind("ab") renvoie 6.
```

Dans les cas où les fonctions `find()` et `rfind()` ne peuvent s'appliquer, elles renvoient la valeur prédéfinie `string::npos`

Exemple :

```
if (exemple.find("xy") != string::npos) ...
```

Fonctions spécifiques aux chaînes

`chaine.substr(depart, longueur)` : renvoie la chaîne de `chaine`, de longueur `longueur` et commençant à la position `depart`.

Exemple :

```
string exemple("Salut à tous !");  
exemple.substr(8, 4) renvoie la string "tous".
```




Complément : conversion vers et depuis string

Convertir vers une `string` : `to_string()`

Exemple :

```
string s("Ma valeur : ");  
int val(42);  
...  
s += to_string(val);
```

Convertir depuis une `string` : `stoX()`

avec `X = i` (pour `int`), `l` (`long int`), `ul` (`unsigned long int`), `ll` (`long long int`),

`ull` (`unsigned long long int`), `d` (`double`) ou `ld` (`long double`)

Exemple :

```
double val(3.14);  
string texte("12.345");  
...  
val += stod(texte);
```



Valeurs littérales de type `string`



En toute rigueur, la valeur littérale `"xyz"` n'est pas de type `string`

(🐍 elle est de type `const char*`)

La conversion se fait souvent de façon totalement **transparente**.

Mais si jamais il est nécessaire de vraiment spécifier, **C++14** a introduit le suffix `s` pour préciser.

Son utilisation nécessite :

```
using namespace std::string_literals;
```

Exemple : `throw "Un message"s;`



C++17 : string_view (1/2)



C++17 introduit une généralisation des `const string` : les `string_view`.

A préférer donc ! (lorsque c'est **vraiment** une `const string`)

Exemple :

```
#include <string_view>
```

```
...
```

```
void genereLettre(bool masculin,  
                 string_view destinataire, string_view sujet,  
                 string_view politesse,   string_view auteur);
```

S'utilise comme des `const string` :
`vue.size()`, `vue[i]`, `vue.substr()`, ...



C++17 : string_view (2/2)



Ont même deux « modificateurs » (qui modifient la « vue », pas la chaîne elle-même!) : `remove_prefix(size_t n)` et `remove_suffix(size_t n)`

Exemple :

```
string s("Un exemple simple !");

string_view vue(s);
cout << vue << endl;

vue.remove_prefix(3);
vue.remove_suffix(2);
cout << vue << endl;

cout << s << endl;
```

affiche :

```
Un exemple simple !
exemple simple
Un exemple simple !
```



Les chaînes de caractères



```
#include <string>
```

déclaration/initialisation : `string identificateur("valeur");`

Affectation : `chaine1 = chaine2;`
`chaine1 = "valeur";`
`chaine1 = 'c';`

Concaténation : `chaine1 = chaine2 + chaine3;`
`chaine1 = chaine2 + "valeur";`
`chaine1 = chaine2 + 'c';`

Accès au (i+1)-ème caractère : `chaine[i];`

Fonctions spécifiques :

taille : `chaine.size()`
insertion : `chaine.insert(position, chaine2)`
remplacement : `chaine.replace(position, longueur, chaine2)`
suppression : `chaine.replace(position, longueur, "")`
sous-chaîne : `chaine.substr(position, longueur)`
recherche : `chaine.find(souschaine)`
`chaine.rfind(souschaine)`

valeur "pas trouvé" d'une recherche : `string::npos`

Pour préparer le prochain cours

- ▶ Vidéos et quiz du MOOC semaine 6 :
 - ▶ Typedef : alias de types [07 :30]
 - ▶ Structures [24 :18]
- ▶ Le prochain cours :
 - ▶ Résumé et quelques approfondissements