Information, Calcul et Communication

Composante Pratique: Programmation C++

MOOC sem5-6 : vector / array / tableau “à la C” / string

Tableau la C, **vector** ou **array** ? (suite)

Le type **string** pour les chaînes de caractères

Chaîne «à la C»

tableau à la C, vector ou array ? (suite)

		Taille connue à la **compilation** ? non	Taille connue à la **compilation** ? oui
Taille pouvant varier à **l'exécution** ?	oui	vector	(vector)
	non	(vector)	**tableau «à la C» array**

```
#include <array>
```

Comparaison entre «tableau à la C» et array:

- même stockage ligne par ligne en mémoire
- également des valeurs indéfinies si non-initialisé
- **array connaît sa taille** avec la méthode **size()**
- **array** peut être transmis par valeur / *pas possible* pour un tableau à la C

Comparaison tableau dynamique et statique

Tableau dynamique

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

Tableau statique

```
#include <array>
array<double,5> tab;        // taille 5
```

```
tab[i]
tab.size()
for(auto  elem : tab)
for(auto& elem : tab)
```

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

```
vector<vector<int>> tab2(
    { {1,2,3,4},
      {5,6},
      {7,8,9,10,11,12} }
);  // deux paires d'accolades
```

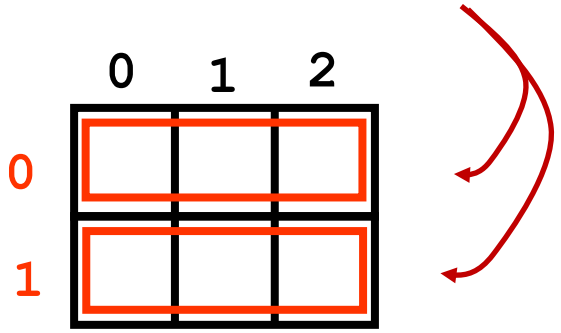
```
                    col li
array<array<int,4>,3> tab2 = {
    1, 2, 3, 4,
    5, 6, 7, 8,
    9,10,11,12
};                  // une seule paire d'accolade
```

```
li col
tab2[i][j]
```

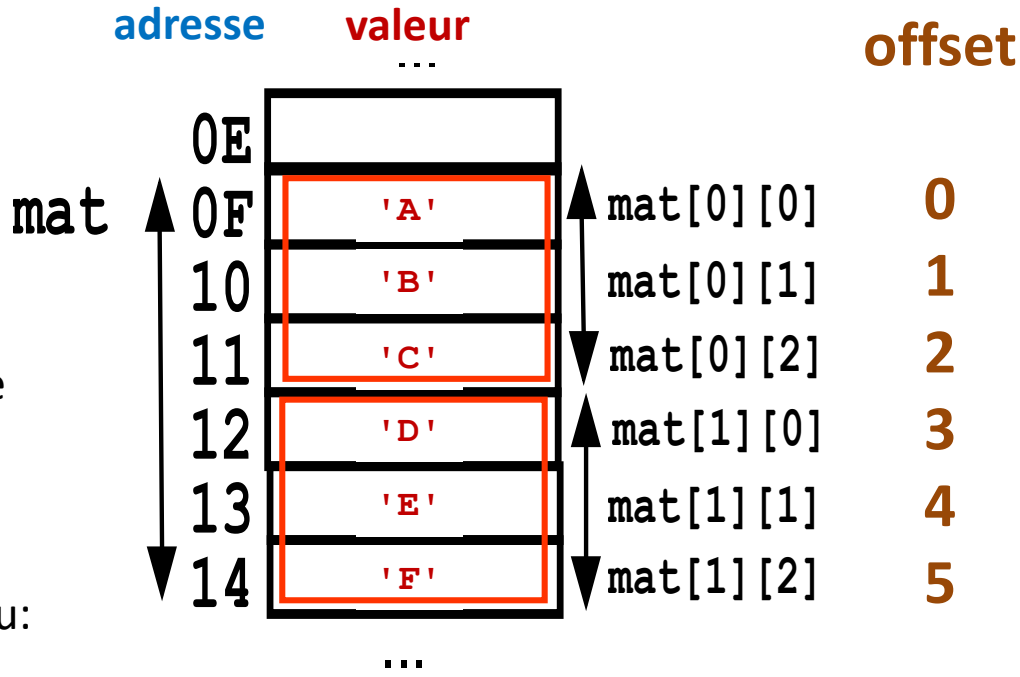
array à deux indices en mémoire

Un **array à 2 indices** est rangé **ligne par ligne** en mémoire (comme un tableau-à-la-C)

```
                    col  li
array<array<char,3>,2> mat = {'A','B','C','D','E','F'}  // une seule paire d'accolade
```



li col



Définition: l'élément **mat[i][j]** du tableau est situé en mémoire à un emplacement qui est décalé d'un **offset** par rapport à l'adresse du premier élément :

L'**offset** de `[0][0]` est zéro car c'est le premier élément du tableau:

```
offset de [i][j] = i * TAILLE_LIGNE  + j
```

En cas d'erreur d'indice: c'est l'emplacement mémoire défini par **l'offset** qui est accédé/modifié

SpeakUp : choose one answer about this piece of code

```
…
array<array<char,3>,2> mat = {'A','B','C','D','E','F'};

cout << mat[0][1] << mat[0][0] << mat[0][3] << endl;
…
```

A. Il ne compile pas à cause de **mat[0][3]**
B. Il s'exécute et affiche : **BAC**
C. Il s'exécute et affiche : **DAD**
D. Il s'exécute et affiche : **BAD**
E. Il s'exécute et affiche : **BED**
F. Il produit un comportement indéterminé à l'exécution

Fonctions/méthodes spécifiques à string

```
#include <string>
```

Le type **`string`** s'utilise pour une **variable** mémorisant une **chaîne de caractères**

```
string ecole("EPFL"); // la variable ecole est initialisée avec
                      // la constante littérale "EPFL"

string chaine1;       // cette chaine de caractères est vide
```

La constante littérale `""` représente la chaîne vide

Le type **`string`** permet de changer dynamiquement la chaîne de caractère, y compris sa taille, à l'aide de l'opérateur de concaténation + et d'un riche ensemble de fonctions (méthodes)

```
string id, nom, prenom;
…
id = nom + ' ' + prenom ;    // on peut aussi concaténer un char
```

Fonctions/méthodes spécifiques à string

#include <string> offre un ensemble de fonctions spécifiques au type **string**

Syntaxe de l'appel d'une méthode: **nom_chaine.nom_methode(parametres)**

chaine.size() : renvoie le nombre de caractères de **chaine** (type : **size_t**)

chaine.insert(pos, ch2) : insère **ch2** dans **chaine** à partir de l'indice **pos**

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

chaine.replace(pos, n, ch2) : remplace **n** caractères de **chaine** à partir de l'indice **pos** et jusqu'à l'indice **pos+n-1** avec tout le contenu de **ch2**.

```
string exemple("abcd");
exemple.replace(1, 2,"1234"); // exemple vaut ensuite "a1234d"
exemple.replace(1, 2,"");     // exemple vaut ensuite "a34d"
```

Fonctions/méthodes spécifiques à string (2)

chaine.find(ch2) : renvoie le premier indice de **chaine** où on trouve le premier caractère de **ch2**

```
string exemple("baabbaab");
size_t index(exemple.find("ab")); // index vaut ensuite 2
```

chaine.rfind(ch2) : renvoie le dernier indice de **chaine** où on trouve le premier caractère de **ch2**

```
string exemple("baabbaab");
size_t index(exemple.rfind("ab")); // index vaut ensuite 6
```

find et **rfind** renvoient la valeur prédéfinie **string::npos** si elles ne trouvent rien.

Lecture et conversion vers string(3)

Rappel: la lecture d'une variable de type **char** avec **cin** filtre les séparateurs (espace, tabulation, passage à la ligne). On peut effectuer une véritable lecture **caractère** par **caractère**, comme pour **`cesar.cc`** (série Topic6), comme suit :

```
char c;
cin.get(c);
```

Par défaut la lecture d'une variable de type **string** avec **cin** s'effectue **mot** par **mot**.
Si on désire lire une ligne entière il faut utiliser la fonction **`getline`** :

```
string chaine;
cin >> chaine;           // lit un seul mot
getline(cin, chaine);    // lit toute la ligne jusqu'à Enter
getline(cin >> ws, chaine); // idem avec filtrage du Enter précédent
```

La fonction **`to_string`(param)** renvoie la chaîne de caractère correspondant à la valeur numérique **param**:

```
string s("la valeur est :");
int val(42);
s += to_string(val); // s vaut ensuite "la valeur est :42"
```

"Constantes littérales de type chaîne de caractères" => chaîne «à la C»

Le type chaîne/string n'existant pas en C, une chaîne de caractères «à la C» est construite à l'aide d'un **tableau de char** «à la C» qui DOIT SE TERMINER avec le caractère `'\0'` qui signale la fin de la chaîne de caractère (son motif binaire est **nul**).

Le tableau de char doit explicitement prévoir l'espace pour ce caractère spécial, même pour une chaîne vide.

Toutes les **constantes littérales de type chaîne de caractères,** telle que `"ABC"` , sont des **constantes** de type chaînes «à la C» : elles se terminent avec `'\0'`.

Exemple: déclaration d'une chaine «à la C» avec initialisation avec une constante littérale:

```
char chaine_c[4] = "ABC";

int i(0);
while (chaine_c[i])
    cout << chaine_c[i++];
```