Les tableaux de taille fixe


```
#include <array>
```

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

`array<type, taille> identificateur = {val`$_1$`, ... , val`$_{taille}$`};`

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

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

Tableau multidimentionnel :

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

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

Les tableaux à la C

déclaration : *type identificateur*[*taille*];
déclaration/initialisation :

type identificateur[*taille*] = { val_1, ... , val_{taille} };

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

Le passage *type1* f(*type2* tab[]); d'un tableau tab à une fonction f se fait automatiquement **par référence**.
Pour éviter les effets de bord :

type1 f(const *type2* tab[]);

tableau multidimentionnel :
type identificateur[*taille1*][*taille2*];
tab[i][j];