Tableaux (introduction)

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les instructions suivantes :

- Lecture : `cin >> variable ;`
- Écriture : `cout << expression ;`
- Affectation : `variable = expression`
- Instruction conditionnelle : `if`
- Instructions itératives : `while`, `do ... while`, `for`
- Fonctions

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les instructions suivantes :

- Lecture : `cin >> variable ;`
- Écriture : `cout << expression ;`
- Affectation : `variable = expression`
- Instruction conditionnelle : `if`
- Instructions itératives : `while`, `do ... while`, `for`
- Fonctions

Pourquoi aller plus loin ?

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les instructions suivantes :

- Lecture : `cin >> variable;`
- Écriture : `cout << expression;`
- Affectation : `variable = expression`
- Instruction conditionnelle : `if`
- Instructions itératives : `while`, `do ... while`, `for`
- Fonctions

Pourquoi aller plus loin ?

Passage à l'échelle !

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les instructions suivantes :

- Lecture : `cin >> variable;`
- Écriture : `cout << expression;`
- Affectation : `variable = expression`
- Instruction conditionnelle : `if`
- Instructions itératives : `while`, `do ... while`, `for`
- Fonctions

Pourquoi aller plus loin ?

Passage à l'échelle !

Manipulation de collections de données

A. Motivation

Exemple (Fil conducteur)

Implantation d'un mini annuaire

B. Les tableaux

À retenir

- *Un tableau est une valeur composite formée de plusieurs valeurs du même type*
- *Une valeur (ou élément) d'un tableau t est désignée par son indice i dans le tableau ; on la note $t[i]$.*
- *En C++ : cet indice est un entier* **entre** 0 **et** $\ell - 1$,
  où ℓ est le nombre d'éléments du tableau

B. Les tableaux

À retenir

- *Un tableau est une valeur composite formée de plusieurs valeurs du même type*
- *Une valeur (ou élément) d'un tableau t est désignée par son indice i dans le tableau ; on la note t[i].*
- *En C++ : cet indice est un entier* **entre** 0 **et** $\ell - 1$,
  où ℓ est le nombre d'éléments du tableau

Exemple

- Voici un tableaux de huit entiers :

1	4	1	5	9	2	6	5

- Avec cet exemple, t[0] vaut 1, t[1] vaut 4, t[2] vaut 1, ...
- Noter que l'ordre et les répétitions sont importantes !

Les tableaux en C++

Exemple

tableaux.cpp

```
vector<int> t;
t = vector<int>(8);
t[0] = 1;
t[1] = 4;
t[2] = 1;
t[3] = 5;
t[4] = 9;
t[5] = 2;
t[6] = 6;
t[7] = 5;

cout << t[5] + t[7] << endl;
```

Les tableaux en C++

Exemple

tableaux.cpp

```
    vector<int> t;
    t = vector<int>(8);
    t[0] = 1;
    t[1] = 4;
    t[2] = 1;
    t[3] = 5;
    t[4] = 9;
    t[5] = 2;
    t[6] = 6;
    t[7] = 5;

    cout << t[5] + t[7] << endl;
```

Avec dans l'entête :

tableaux.cpp

```
#include <vector>
```

C. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

```
vector<int> t;
```

C. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

```
vector<int> t;
```

- Pour un tableau de nombres réels : `vector<double>`, etc.

C. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

```
vector<int> t;
```

- Pour un tableau de nombres réels : `vector<double>`, etc.
- ♣ `vector` est un *template*

C. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

```
vector<int> t;
```

- Pour un tableau de nombres réels : `vector<double>`, etc.
- ♣ `vector` est un *template*

Allocation d'un tableau de huit entiers

tableaux.cpp

```
t = vector<int>(8);
```

C. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

```
vector<int> t;
```

- Pour un tableau de nombres réels : `vector<double>`, etc.
- ♣ `vector` est un *template*

Allocation d'un tableau de huit entiers

tableaux.cpp

```
t = vector<int>(8);
```

Initialisation du tableau

tableaux.cpp

```
t[0] = 1;
t[1] = 4;
t[2] = 1;
```

Les trois étapes de la construction d'un tableau

À retenir

- *Une variable de type tableau se construit en* **trois étapes** *:*

Les trois étapes de la construction d'un tableau

À retenir

- *Une variable de type tableau se construit en* **trois étapes** *:*
  1. *Déclaration*
  2. *Allocation*
     Sans elle : **faute de segmentation** *(au mieux !)*
  3. *Initialisation*
     Sans elle : même problème qu'avec les variables usuelles

Les trois étapes de la construction d'un tableau

À retenir

- *Une variable de type tableau se construit en* **trois étapes** *:*
  1. *Déclaration*
  2. *Allocation*
     Sans elle : **faute de segmentation** *(au mieux !)*
  3. *Initialisation*
     Sans elle : même problème qu'avec les variables usuelles

Raccourci

Déclaration, allocation et initialisation en un coup :

```
vector<int> t = { 1, 4, 1, 5, 9, 2, 6, 5 };
```

Les trois étapes de la construction d'un tableau

À retenir

- *Une variable de type tableau se construit en* **trois étapes** *:*
  1. *Déclaration*
  2. *Allocation*
     Sans elle : **faute de segmentation** *(au mieux !)*
  3. *Initialisation*
     Sans elle : même problème qu'avec les variables usuelles

Raccourci

Déclaration, allocation et initialisation en un coup :

```
vector<int> t = { 1, 4, 1, 5, 9, 2, 6, 5 };
```

Introduit par la norme C++ de 2011 !

D. Utilisation des tableaux

Syntaxe

`t[i]` s'utilise comme une variable usuelle :

```
// Exemple d'accès en lecture
x = t[2] + 3*t[5];
y = sin(t[3]*3.14);

// Exemple d'accès en écriture
t[4] = 2 + 3*x;
cin >> t[5];
```

D. Utilisation des tableaux

Syntaxe

`t[i]` s'utilise comme une variable usuelle :

```
// Exemple d'accès en lecture
x = t[2] + 3*t[5];
y = sin(t[3]*3.14);

// Exemple d'accès en écriture
t[4] = 2 + 3*x;
cin >> t[5];
```

Attention !

- En C++ **les indices ne sont pas vérifiés** !
- Le comportement de `t[i]` n'est pas spécifié en cas de débordement
- Source no 1 des trous de sécurité ! ! !

D. Utilisation des tableaux

Syntaxe

`t[i]` s'utilise comme une variable usuelle :

```
// Exemple d'accès en lecture
x = t[2] + 3*t[5];
y = sin(t[3]*3.14);

// Exemple d'accès en écriture
t[4] = 2 + 3*x;
cin >> t[5];
```

Attention !

- En C++ **les indices ne sont pas vérifiés** !
- Le comportement de `t[i]` n'est pas spécifié en cas de débordement
- Source no 1 des trous de sécurité ! ! !
- Accès avec vérifications : `t.at(i)` au lieu de `t[i]`

Quelques autres opérations sur les tableaux

```
t.size();       // Taille du tableau
t.push_back(3); // Ajout d'un élément à la fin
```

Quelques autres opérations sur les tableaux

```
t.size();       // Taille du tableau
t.push_back(3); // Ajout d'un élément à la fin
```

Fonctions et tableaux

tableau-affiche.cpp

```
void affiche(vector<int> tableau) {
    for ( int i = 0; i < tableau.size(); i++ ) {
        cout << tableau[i] << " ";
    }
    cout << endl;
}

int main() {
    vector<int> t = { 1, 2, 3, 4 };
    affiche(t);
}
```

E. Retour sur les fonctions

Voir Cours 3 :

- Variables locales, variables globales
- Fonction main
- Procédures
- Fonctions récursives

Résumé

- Motivation : manipulation de collections de données
  Par exemple un annuaire

Résumé

- Motivation : manipulation de collections de données
  Par exemple un annuaire
- **Tableau** : valeur composite formée de plusieurs valeurs du même type

Résumé

- Motivation : manipulation de collections de données
  Par exemple un annuaire
- **Tableau** : valeur composite formée de plusieurs valeurs du même type
- Construction en trois étapes :
  - **Déclaration** : `vector<int> t;`
  - **Allocation** : `t = vector<int>(3);`
  - **Initialisation** : `t[0] = 3;t[1] = 0;...`

Résumé

- Motivation : manipulation de collections de données
  Par exemple un annuaire
- **Tableau** : valeur composite formée de plusieurs valeurs du même type
- Construction en trois étapes :
  - **Déclaration** : `vector<int> t;`
  - **Allocation** : `t = vector<int>(3);`
  - **Initialisation** : `t[0] = 3;t[1] = 0;...`
- Utilisation : `t[i] = t[i]+1`, `t.size()`, `t.push(3)`

Résumé

- Motivation : manipulation de collections de données
  Par exemple un annuaire
- **Tableau** : valeur composite formée de plusieurs valeurs du même type
- Construction en trois étapes :
  - **Déclaration** : `vector<int> t;`
  - **Allocation** : `t = vector<int>(3);`
  - **Initialisation** : `t[0] = 3;t[1] = 0;...`
- Utilisation : `t[i] = t[i]+1`, `t.size()`, `t.push(3)`

La semaine prochaine

- Représentation en mémoire des tableaux
- Sémantique de l'affectation
- Passage par valeurs
- Autres collections homogènes

Résumé

- Motivation : manipulation de collections de données
  Par exemple un annuaire
- **Tableau** : valeur composite formée de plusieurs valeurs du même type
- Construction en trois étapes :
  - **Déclaration** : `vector<int> t;`
  - **Allocation** : `t = vector<int>(3);`
  - **Initialisation** : `t[0] = 3;t[1] = 0;...`
- Utilisation : `t[i] = t[i]+1`, `t.size()`, `t.push(3)`

La semaine prochaine

- Représentation en mémoire des tableaux
- Sémantique de l'affectation
- Passage par valeurs
- Autres collections homogènes
- Représentation en mémoire et types de base