

Tableaux (introduction)

A. Les tableaux	6
Motivation	
Les tableaux	
Construction des tableaux	
Utilisation des tableaux	
B. Retour sur les fonctions	26
Programmes compilés et fonction main	
Portée des variables : variables locales et globales	
Fonctions récursive	
C. Résumé	45

Résumé des épisodes précédents

Pour le moment nous avons vu :

- ▶ Expressions : `3 * (4+5)` `1 < x and x < 5 or y == 3`
- ▶ Variables, types, affectation : `variable = expression`

Résumé des épisodes précédents

Pour le moment nous avons vu :

- ▶ Expressions : `3 * (4+5)` `1 < x and x < 5 or y == 3`
- ▶ Variables, types, affectation : `variable = expression`
- ▶ Instruction conditionnelle : `if`
- ▶ Instructions itératives : `while, do ... while, for`

Résumé des épisodes précédents

Pour le moment nous avons vu :

- ▶ Expressions : `3 * (4+5)` `1 < x and x < 5 or y == 3`
- ▶ Variables, types, affectation : `variable = expression`
- ▶ Instruction conditionnelle : `if`
- ▶ Instructions itératives : `while`, `do ... while`, `for`
- ▶ Fonctions, procédures

Résumé des épisodes précédents

Pour le moment nous avons vu :

- ▶ Expressions : `3 * (4+5)` `1 < x and x < 5 or y == 3`
- ▶ Variables, types, affectation : `variable = expression`
- ▶ Instruction conditionnelle : `if`
- ▶ Instructions itératives : `while`, `do ... while`, `for`
- ▶ Fonctions, procédures

Pourquoi aller plus loin ?

Résumé des épisodes précédents

Pour le moment nous avons vu :

- ▶ Expressions : `3 * (4+5)` `1 < x and x < 5 or y == 3`
- ▶ Variables, types, affectation : `variable = expression`
- ▶ Instruction conditionnelle : `if`
- ▶ Instructions itératives : `while`, `do ... while`, `for`
- ▶ Fonctions, procédures

Pourquoi aller plus loin ?

Passage à l'échelle !

Résumé des épisodes précédents

Pour le moment nous avons vu :

- ▶ Expressions : `3 * (4+5)` `1 < x and x < 5 or y == 3`
- ▶ Variables, types, affectation : `variable = expression`
- ▶ Instruction conditionnelle : `if`
- ▶ Instructions itératives : `while`, `do ... while`, `for`
- ▶ Fonctions, procédures

Pourquoi aller plus loin ?

Passage à l'échelle !

Manipulation de collections de données

A. 1. Motivation

Exemple (Fil conducteur)

Implantation d'un mini annuaire

A. 2. 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

A. 2. 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>
```

A.3. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

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

A.3. Construction des tableaux

Déclaration d'un tableau d'entiers

tableaux.cpp

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

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

A. 3. 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*

A. 3. 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);
```


A. 3. 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

A. 4. Utilisation des tableaux

Syntaxe et sémantique

`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;
```

A. 4. Utilisation des tableaux

Syntaxe et sémantique

`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;
```

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!!!

A. 4. Utilisation des tableaux

Syntaxe et sémantique

`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;
```

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;  
}
```

[tableau-affiche.cpp](#)

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

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;  
}
```

tableau-affiche.cpp

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

- ▶ Un tableau est une valeur comme les autres
- ▶ Il peut être passé en paramètre à ou renvoyé par une fonction

B. Retour sur les fonctions

- ▶ Programmes compilés et fonction `main`
- ▶ Variables locales, variables globales
- ▶ Fonctions récursives

Programmes compilés : un exemple

Jusqu'ici nous avons exécuté notre code C++ dans Jupyter.

Programmes compilés : un exemple

Jusqu'ici nous avons exécuté notre code C++ dans Jupyter.

On peut aussi écrire des programmes indépendants :

max.cpp

```
#include <iostream>
using namespace std;

float max(float a, float b) {
    if ( a >= b ) {
        return a;
    } else {
        return b;
    }
}

int main() {
    cout << max(1.5, 3.0) << endl;
    cout << max(5.2, 2.0) << endl;
    cout << max(2.3, 2.3) << endl;
    return 0;
}
```

Programmes compilés : la fonction main

Exemple

bonjour.cpp

```
int main() {  
    cout << "Bonjour!" << endl;  
    return 0;  
}
```

Programmes compilés : la fonction main

Exemple

bonjour.cpp

```
int main() {  
    cout << "Bonjour!" << endl;  
    return 0;  
}
```

À retenir

- ▶ *Un programme compilé peut être composé de plusieurs fonctions*

Programmes compilés : la fonction main

Exemple

bonjour.cpp

```
int main() {  
    cout << "Bonjour!" << endl;  
    return 0;  
}
```

À retenir

- ▶ *Un programme compilé peut être composé de plusieurs fonctions*
- ▶ *Une des fonctions doit s'appeler `main` (**fonction principale**)*
- ▶ *Au lancement du programme, la fonction `main` est exécutée*

Programmes compilés : la fonction main

Exemple

bonjour.cpp

```
int main() {  
    cout << "Bonjour !" << endl;  
    return 0;  
}
```

À retenir

- ▶ *Un programme compilé peut être composé de plusieurs fonctions*
- ▶ *Une des fonctions doit s'appeler `main` (**fonction principale**)*
- ▶ *Au lancement du programme, la fonction `main` est exécutée*
- ▶ *Cette fonction doit renvoyer une valeur entière*
- ▶ *Convention :*
 - ▶ *0 si l'exécution du programme s'est déroulée normalement*
 - ▶ *Un entier différent de 0 en cas d'erreur*
Cet entier indique quel genre d'erreur s'est produite

La fonction main : paramètres ♣

bonjour-nom.cpp

```
int main(int argv, char ** args) {
    string nom1, nom2;
    nom1 = args[1];
    nom2 = args[2];

    cout << "Bonjour " << nom1 << " !" << endl;
    cout << "Bonjour " << nom2 << " !" << endl;
    return 0;
}
```

La fonction main : paramètres ♣

bonjour-nom.cpp

```
int main(int argv, char ** args) {  
    string nom1, nom2;  
    nom1 = args[1];  
    nom2 = args[2];  
  
    cout << "Bonjour " << nom1 << " !" << endl;  
    cout << "Bonjour " << nom2 << " !" << endl;  
    return 0;  
}
```

À l'exécution :

```
> bonjour-nom Jean Paul  
Bonjour Jean!  
Bonjour Paul!
```

La fonction main : paramètres ♣

bonjour-nom.cpp

```
int main(int argv, char ** args) {
    string nom1, nom2;
    nom1 = args[1];
    nom2 = args[2];

    cout << "Bonjour " << nom1 << " !" << endl;
    cout << "Bonjour " << nom2 << " !" << endl;
    return 0;
}
```

À l'exécution :

```
> bonjour-nom Jean Paul
Bonjour Jean !
Bonjour Paul !
```

Note

Le `char **` est un résidu des chaînes de caractères en C

Vous verrez au second semestre ce que font les trois premières lignes

Portée des variables : un exemple

Exécutons pas à pas le programme suivant :

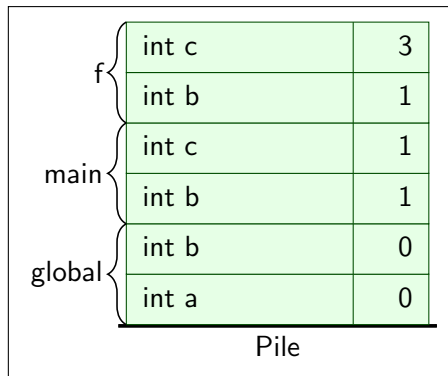
`variables-locales-globales.cpp`

```
int a = 0, b = 0;    // variables globales

int f(int b) {      // paramètre formel (donc local à f)
    int c = 3;      // variable locale à f
    return a + b + c;
}

int main() {
    int b = 1, c = 1; // variables locales à main
    a + b + c;        // b et c: locales à main, a: globale
    {
        long a = 2, c = 2;
        a + b + c;    // a et c: locales au bloc, b: locale à main
    }
    a + b + c;        // b et c: locales à main, a: globale
    cout << f(b) << endl;
}
```

Portée des variables : un exemple (2)



Portée des variables

Contexte lexical

- ▶ Une variable est visible depuis sa déclaration jusqu'à la fin du bloc où elle est déclarée

Portée des variables

Contexte lexical

- ▶ Une variable est visible depuis sa déclaration jusqu'à la fin du bloc où elle est déclarée
- ▶ Elle peut masquer des variables issues des contextes englobants

Portée des variables

Contexte lexical

- ▶ Une variable est visible depuis sa déclaration jusqu'à la fin du bloc où elle est déclarée
- ▶ Elle peut masquer des variables issues des contextes englobants
- ▶ *Variable locale* : définie dans le bloc d'une fonction
- ▶ *Variable globale* : définie ailleurs (entête du programme)
- ▶ *Paramètre formel* : se comporte comme une variable locale

Portée des variables

Contexte lexical

- ▶ Une variable est visible depuis sa déclaration jusqu'à la fin du bloc où elle est déclarée
- ▶ Elle peut masquer des variables issues des contextes englobants
- ▶ *Variable locale* : définie dans le bloc d'une fonction
- ▶ *Variable globale* : définie ailleurs (entête du programme)
- ▶ *Paramètre formel* : se comporte comme une variable locale

À retenir

- ▶ *Une variable locale à une fonction n'existe que le temps d'exécution de la fonction*
- ▶ *La valeur de cette variable d'un appel à la fonction est perdue lors du retour au programme appelant et ne peut être récupérée lors d'un appel ultérieur*

Variables globales

- ▶ Accessible à l'intérieur de toutes les fonctions

Variables globales

- ▶ Accessible à l'intérieur de toutes les fonctions

Attention !

- ▶ On peut modifier la valeur d'une variable globale
Ceci est fortement déconseillée (*effet de bord*)
- ▶ Une variable locale masque une variable globale du même nom
Ceci est fortement déconseillée (ambiguïté à la lecture rapide)
- ▶ On évitera ces pratiques dans le cadre de ce cours

Fonctions récursives ♣

Exercice

On considère la fonction :

[fonction-factorielle-recursive.cpp](#)

```
int factorielle(int n) {  
    if ( n == 0 ) {  
        return 1;  
    } else {  
        return n * factorielle(n-1);  
    }  
}
```

Exécuter pas-à-pas l'exécution de `factorielle(3)`

C. Résumé

Tableaux

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

C. Résumé

Tableaux

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

C. Résumé

Tableaux

- ▶ 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;...`

C. Résumé

Tableaux

- ▶ 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_back(3)`

C. Résumé

Tableaux

- ▶ 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_back(3)`

Retour sur les fonctions

- ▶ Programmes indépendants, fonction main
Pour le moment : savoir les comprendre (exercices WIMS)
Après le partiel : mise en pratique, compilation, ...

C. Résumé

Tableaux

- ▶ 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_back(3)`

Retour sur les fonctions

- ▶ Programmes indépendants, fonction main
Pour le moment : savoir les comprendre (exercices WIMS)
Après le partiel : mise en pratique, compilation, ...
- ▶ Variables locales et globales

C. Résumé

Tableaux

- ▶ 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_back(3)`

Retour sur les fonctions

- ▶ Programmes indépendants, fonction main
Pour le moment : savoir les comprendre (exercices WIMS)
Après le partiel : mise en pratique, compilation, ...
- ▶ Variables locales et globales
- ▶ Fonctions récursives