Introduction à la programmation
Travaux pratiques: séance 6
INFO0201-1

X. Baumans
(xavier.baumans@ulg.ac.be)

Programme de la séance

- Tableaux statiques (à 1 et 2 dimensions)
  $\rightarrow$ stockage d'une série d'éléments d'un même type

Programme de la séance

- Tableaux statiques (à 1 et 2 dimensions)
  $\rightarrow$ stockage d'une série d'éléments d'un même type

- Nombres aléatoires
  $\rightarrow$ génération de nombres pseudo-aléatoires

Les tableaux statiques

Lorsque nous devons utiliser une suite ou série de données, il peut être plus commode d'utiliser un tableau. Celui-ci permet de centraliser les données sous un même nom de variable et l'accès aux différents éléments est réalisé par l'utilisation d'un indice.

Un tableau permet de mémoriser une suite ou série de valeurs d'un même type. La taille d'un tableau statique est fixée pour toute l'exécution du programme.

Exemples :

Les tableaux statiques

Lorsque nous devons utiliser une suite ou série de données, il peut être plus commode d'utiliser un tableau. Celui-ci permet de centraliser les données sous un même nom de variable et l'accès aux différents éléments est réalisé par l'utilisation d'un indice.

Un tableau permet de mémoriser une suite ou série de valeurs d'un même type. La taille d'un tableau statique est fixée pour toute l'exécution du programme.

Exemples :

- 1D : un vecteur, une liste, un ensemble de données, ...

Les tableaux statiques

Lorsque nous devons utiliser une suite ou série de données, il peut être plus commode d'utiliser un tableau. Celui-ci permet de centraliser les données sous un même nom de variable et l'accès aux différents éléments est réalisé par l'utilisation d'un indice.

Un tableau permet de mémoriser une suite ou série de valeurs d'un même type. La taille d'un tableau statique est fixée pour toute l'exécution du programme.

Exemples :

- 1D : un vecteur, une liste, un ensemble de données, ...
- 2D : une matrice, ...

Les tableaux statiques : déclaration

Comment déclarer un tableau 1D :

type nom_tableau[taille_tableau] ;

Un tableau 2D :

type nom_tableau[taille_1][taille_2] ;

Exemple :

```
1 int v[3]; // tableau de 3 éléments entiers
2 double tableau_2D[10][20]; // tableau de 10x20 double
```

Les tableaux statiques : déclaration

Comment déclarer un tableau 1D :

type nom_tableau[taille_tableau] ;

Un tableau 2D :

type nom_tableau[taille_1][taille_2] ;

Exemple :

```
1 int v[3]; // tableau de 3 éléments entiers
2 double tableau_2D[10][20]; // tableau de 10x20 double
```

- **Type** de variable : **int**, **float**, **double**, **char**, ...

Les tableaux statiques : déclaration

Comment déclarer un tableau 1D :

type nom_tableau[taille_tableau] ;

Un tableau 2D :

type nom_tableau[taille_1][taille_2] ;

Exemple :

```
1 int v[3]; // tableau de 3 éléments entiers
2 double tableau_2D[10][20]; // tableau de 10x20 double
```

- **Type** de variable : **int**, **float**, **double**, **char**, ...
- **Nom** du tableau : le plus **représentatif** et **concis** possible

Les tableaux statiques : déclaration

Comment déclarer un tableau 1D :

type nom_tableau[taille_tableau] ;

Un tableau 2D :

type nom_tableau[taille_1][taille_2] ;

Exemple :

```
1 int v[3]; // tableau de 3 éléments entiers
2 double tableau_2D[10][20]; // tableau de 10x20 double
```

- **Type** de variable : **int**, **float**, **double**, **char**, ...
- **Nom** du tableau : le plus **représentatif** et **concis** possible
- **[...]** : autant de paires de crochets que de dimensions

Les tableaux statiques : déclaration

Comment déclarer un tableau 1D :

type nom_tableau[taille_tableau] ;

Un tableau 2D :

type nom_tableau[taille_1][taille_2] ;

Exemple :

```
1 int v[3]; // tableau de 3 éléments entiers
2 double tableau_2D[10][20]; // tableau de 10x20 double
```

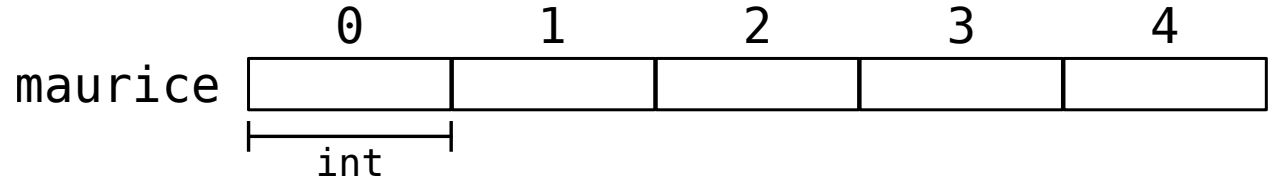
- **Type** de variable : **int**, **float**, **double**, **char**, ...
- **Nom** du tableau : le plus **représentatif** et **concis** possible
- **[...]** : autant de paires de crochets que de dimensions
- **Taille** : entre chaque crochet préciser la taille de la dimension correspondante du tableau

Les tableaux statiques : déclaration et initialisation

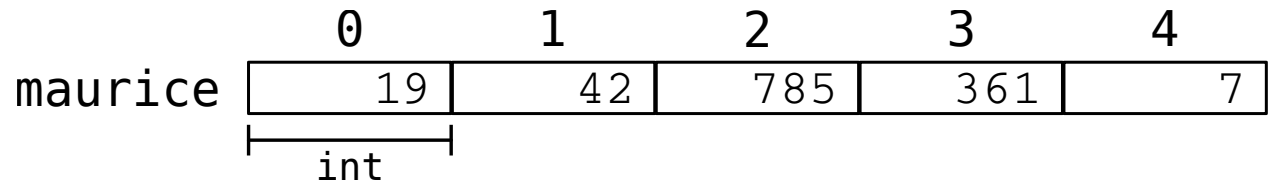
Déclaration d'un tableau = **réservation** de la mémoire

Les éléments (cases) d'un tableau sont contigus dans la mémoire

```
1 int maurice[5]; // déclaration d'un tableau de 5 int
```




```
1 // déclaration et initialisation d'un tableau de 5 int
2 int maurice[5] = {19, 42, 785, 361, 7};
```

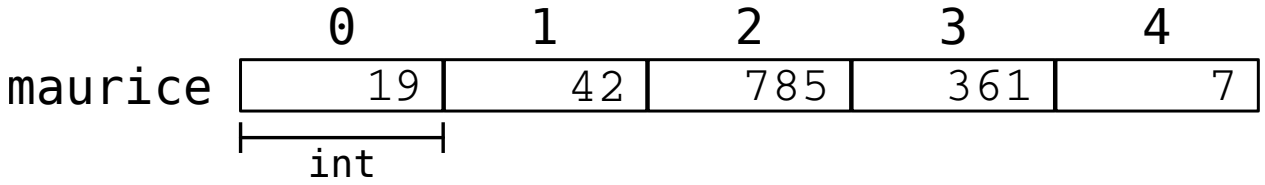


Si on veut initialiser tout le tableau à zéro :

```
1 int peter[5] = {0}; // équivalent à {0, 0, 0, 0, 0}
```

Attention au volume de mémoire !

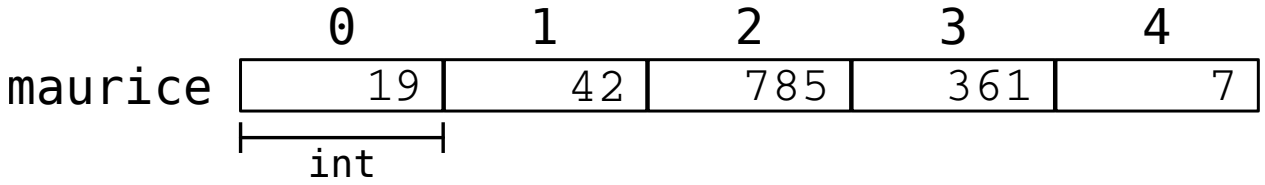
Les tableaux statiques : accès aux élements



Les éléments du tableau sont numérotés par des **indices** de 0 à $N - 1$, où N est la taille du tableau.

Ils représentent l'***offset*** (décalage) de l'élément à partir du début du tableau : c'est pour cela que le premier élément possède l'indice 0.

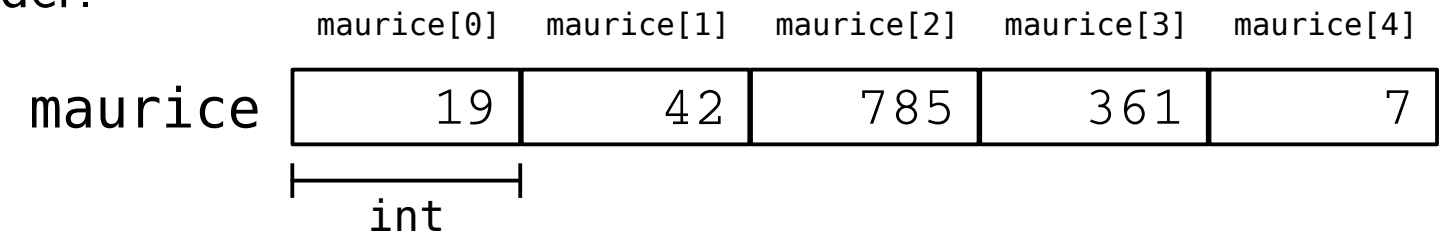
Les tableaux statiques : accès aux élements



Les éléments du tableau sont numérotés par des **indices** de 0 à $N-1$, où N est la taille du tableau.

Ils représentent l'***offset*** (décalage) de l'élément à partir du début du tableau : c'est pour cela que le premier élément possède l'indice 0.

L'accès aux élements du tableau s'effectue au moyen de crochets [...] entre lesquels on place l'indice de l'élément auquel on veut accéder.



L'indice doit toujours être un **entier** !
Il peut être une valeur **constante** ou la valeur d'une **variable**.

Les tableaux statiques : accès aux élements

Attention à la numérotation des indices !

```
1 int v[3] = {4, 6, 8};
2 cout << "Element 1 = " << v[0] << endl; // affiche 4
3 cout << "Element 2 = " << v[1] << endl; // affiche 6
4 cout << "Element 3 = " << v[2] << endl; // affiche 8
5 v[1] = 12;
6 cout << "Element 2 = " << v[1] << endl; // affiche 12
```

Tout comme pour les tableaux 1D, chaque dimension d'un tableau à D dimensions est numérotée de 0 à $N_D - 1$ où N_D est la taille de la D-ième dimension du tableau.

```
1 int v[3][4] = { {1, 2, 3,  4},
2                 {5, 6, 7,  8},
3                 {8, 9, 10, 11} };
4 cout << v[1][2] << endl; // affiche 7
5 cout << v[2][0] << endl; // affiche 8
```

Les tableaux statiques : accès aux élements

- Erreur d'accès à un tableau
  L'utilisation d'un indice non compris entre 0 et $N - 1$ ne sera pas détectée par le compilateur (C/C++).
  Le programme tentera donc d'accéder à une partie de la mémoire qui n'a pas été réservée pour le tableau. Dans ce cas :

Les tableaux statiques : accès aux élements

- Erreur d'accès à un tableau
  L'utilisation d'un indice non compris entre 0 et $N-1$ ne sera pas détectée par le compilateur (C/C++).
  Le programme tentera donc d'accéder à une partie de la mémoire qui n'a pas été réservée pour le tableau. Dans ce cas :
  - Le programme peut s'arrêter avec une erreur du type "segmentation fault" (accès à une zone mémoire interdite par le système)

Les tableaux statiques : accès aux élements

- Erreur d'accès à un tableau
  L'utilisation d'un indice non compris entre 0 et $N-1$ ne sera pas détectée par le compilateur (C/C++).
  Le programme tentera donc d'accéder à une partie de la mémoire qui n'a pas été réservée pour le tableau. Dans ce cas :
  - Le programme peut s'arrêter avec une erreur du type "segmentation fault" (accès à une zone mémoire interdite par le système)
  - Parfois l'exécution du programme ne s'arrêtera pas. On accède alors à de la mémoire non allouée, qui peut correspondre à d'autres variables et être modifiée pendant l'exécution. Seules des erreurs de valeur et donc de fonctionnement du programme apparaitront dans ce cas : ce sont les plus difficiles à détecter.

Exemple en pratique

La plupart du temps, on utilise des boucles pour initialiser, traiter et afficher des tableaux. Dans ce cas, une variable entière est utilisée pour accéder aux éléments du tableaux.

```
1  int main()
2  {
3     int v[1000] = {0};
4     // initialiser avec des multiples de 2
5     for(int i = 0; i < 1000; i++)
6     {
7        v[i] = i*2;
8     }
9     return 0;
10 }
```

Nombres aléatoires

Il est fréquent en programmation d'avoir besoin de générer des **nombres aléatoires**. Pour ce faire, il existe une fonction

rand() ;

Cette fonction retourne un entier positif pseudo-aléatoire compris dans l'intervalle [0 :RAND_MAX].

Pour l'utiliser, il faut inclure la bibliothèque **cstdlib** :

```
1 #include <cstdlib>
```

Nombres aléatoires : manipulations

- Pour définir les limites de l'intervalle des nombres générés
  $\rightarrow$ Utilisation de la fonction modulo

```
1 int a = rand()%100; // nombre entre 0 et 99
```

Nombres aléatoires : manipulations

- Pour définir les limites de l'intervalle des nombres générés
  $\rightarrow$ Utilisation de la fonction modulo

```
1 int a = rand()%100; // nombre entre 0 et 99
```

- Obtenir un nombre réel
  $\rightarrow$ Convertir en réel et diviser

```
1 // nombre entre 0 et 1
2 double a = (double)rand()/RAND_MAX;
3
4 // nombre entre 10 et 20
5 double b = 10. + 10.*(double)rand()/RAND_MAX;
```

Nombres aléatoires : initialisation du générateur

Le générateur produira toujours la même suite de nombres (pseudo-aléatoires). Pour obtenir des nombres aléatoires qui ne soient pas identiques d'une exécution à l'autre, on utilise la fonction **srand(x)** qui permet de commencer la série à un endroit déterminé.

Cette valeur de départ x est appelée ***seed*** (graine) et doit être différente à chaque exécution.
On utilise généralement pour cette valeur le nombre de secondes écoulées depuis le 1^{er} Janvier 1970 (qui est donc différent à chaque exécution). Ce nombre est obtenu par la fonction **time(NULL)**.

```
1 // initialisation:
2 srand(time(NULL));
3 // nombre aléatoire différent à chaque exécution:
4 int a = rand();
```

La fonction **time** nécessite d'inclure la libraire <**ctime**>

Exercices

1. Traitement des cotes d'un examen :
   - Déclarer un tableau pour 10 cotes ;
   - Demander à l'utilisateur de saisir les valeurs des cotes ;
   - Afficher le tableau des cotes ;
   - Afficher la meilleure et la moins bonne cote ;
   - Calculer la moyenne et l'afficher.
2. Recherche dans un tableau
   - Créer un tableau de 50 éléments de type **double** ;
   - Le remplir de nombres aléatoires ;
   - Afficher le tableau ;
   - Déterminer l'indice du plus petit élément du tableau ;
   - Afficher la valeur de cet élément et l'indice correspondant.

Exercices

3. Tri de tableau :
   Modifier l'exercice précédent pour trier les éléments du tableau du plus petit au plus grand.
   - Afficher le tableau non trié ;
   - Parcourir le tableau à la recherche du plus petit élément ;
   - Permuter cet élément avec le premier du tableau ;
   - Chercher le plus petit élément parmi ceux restants ;
   - Permuter cet élément avec le second du tableau ;
   - Poursuivre jusqu'à ce que le tableau soit trié ;
   - Afficher le tableau trié ;
4. Calcul de déterminant :
   - Demander à l'utilisateur d'encoder une matrice 3x3 ;
   - Afficher sa matrice dans la console ;
   - Calculer son déterminant et l'afficher dans la console ;

Exercices

5. Petit modèle météorologique
   - Écrire un petit programme qui va afficher les prévisions météo sur plusieurs jours (nombre au choix). Pour ce faire, procéder dans le cadre suivant :
     - Considérer un modèle binaire (seulement deux types de temps, par exemple : beau temps et pluie) ;
     - Se baser sur l'affirmation suivante donnée jadis par grand-père : "Demain verra paraître le même temps qu'aujourd'hui dans 60% des cas !" ;
     - S'aider évidemment des nombres aléatoires pour modéliser cette probabilité ;
     - A la fin des prévisions, afficher le nombre total de jours de beau temps, idem pour le nombre total de jours de pluie.