Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Leçon # 4 : Les tableaux

Alessandro Bravar Alessandro.Bravar@unige.ch
tél.: 96210 bureau: EP 206

assistants

Johanna Gramling Johanna.Gramling@unige.ch
tél.: 96368 bureau: EP 202A

Mark Rayner Mark.Rayner@unige.ch
tél.: 96263 bureau: EP 219

http://dpnc.unige.ch/~bravar/C++2015/L4
pour les notes du cours, les exemples, les corrigés, …

Plan du jour #4

Corrigé du contrôle continu #1
http://dpnc.unige.ch/~bravar/C++2015/CC1

Les tableaux

texte Micheloud et Rieder
chap. 9 (15 et app. B)

Déclaration et initialisation d'un tableau

Accès aux éléments du tableau

Tableaux multidimensionnels

Tableaux et fonctions

Structures des données

Lecture / écriture depuis un fichier

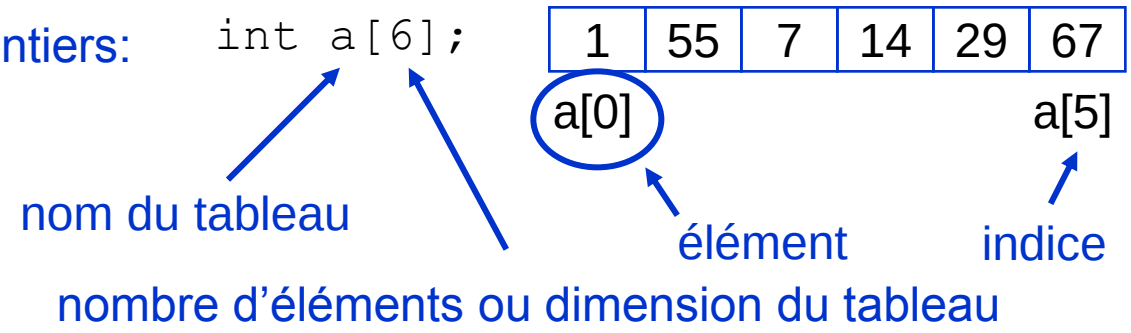
Les Tableaux (arrays)

Le tableau est une séquence d'objets du même type stockés consécutivement en mémoire, auxquels sont attribués des valeurs indépendantes. Ces objets sont appelés éléments du tableau. Chaque élément est accessible individuellement avec un indice.
Les tableaux unidimensionnels peuvent être comparés au *vecteurs* algébriques.
La syntaxe pour déclarer un tableau est la suivante :

```
type nom_tableau[taille];
```

où `type` représente le type (`int`, `double`, `char`, …) des éléments du tableau et `taille` le nombre d'éléments dont il se compose. La taille d'un tableau est un entier constant qui ne peut pas changer pendant l'exécution du programme.

tableau unidimensionnel d'entiers:



Le premier élément du tableau de taille `N` est l'élément `0`, le dernier l'élément `N-1` !
Si le nom d'un tableau est `a`, `a[0]` est le premier élément.

Les éléments sont numérotés de façon continue en commençant par 0.
Si le nombre d'éléments est `n`, le dernier élément est `a[n-1]` et non pas `a[n]`

Cette numérotation est qualifiée d'indexation basée sur zéro. Faîtes-y attention !

Déclaration et initialisation

Il y a des méthodes différentes pour déclarer et initialiser les tableaux :

```
int tab[5];
tab[0] = 10;
tab[1] = 20;
```

ou

```
int tab[5] = {10, 20, 30, 40, 50};
```

Ces instructions déclarent `tab` comme un tableau de cinq entiers et affectent la valeur 10 à `tab[0]`, la valeur 20 à `tab[1]`, etc.

Si vous omettez la taille du tableau comme ci-dessous

```
int tab[ ] = {10, 20, 30, 40, 50};
```

le compilateur compte les éléments de la liste et détermine la dimension du tableau.
Pour connaître la taille du tableau, on peut utiliser la fonction `sizeof` :

```
int taille = sizeof(tab)/sizeof(tab[0]);
```

Il est interdit d'initialiser plus d'éléments que n'en contient le tableau :

```
int tab[5] = {10, 20, 30, 40, 50, 60};
```

Cette instruction générera une erreur de compilation.

Par contre,

```
int tab[5] = {10, 20};
```

initialisera à zéro les éléments 2 à 4 du tableau.

Déclaration et initialisation (2)

Pour initialiser tous les éléments à zéro on peut simplement écrire

```
int tab[5] = {0};
```

La taille du tableau peut être un variable entière, mais elle doit être de type `const int` !

```
const int SIZE = 6;
int tab[SIZE];
```

Vous pouvez initialiser des tableaux, mais vous ne pouvez pas les affecter un à l'autre :

```
int tab1[5] = {10, 20, 30, 40, 50};
int tab2[5];
tab2 = tab1;
```

interdit !

Pour affecter le tableau `tab1` au tableau `tab2`, il faut écrire une boucle pour affecter élément après élément (`n` est le nombre d' éléments de chaque tableau):

```
for(i=0; i<=n-1; i++)
  tab2[i] = tab1[i];
```

Pour afficher un tableau il faut le parcourir avec une boucle (élément après élément) !

```
for(i=0; i<=n-1; i++)
  cout << a[i] << endl;
```

En effet, `a` est l'adresse mémoire du premier élément du tableau :
p.ex. `cout << a` imprimera cette adresse et pas tous les éléments du tableau.

Exemple

Dans cet exemple, on définit un tableau de 6 éléments ;
les valeurs sont entrées par le clavier et sont affichées en ordre inverse.

```
#include <iostream>

using namespace std;

int main() {
  const int SIZE = 6;
  int a[SIZE];

  cout << "Entrez " << SIZE << " nombres entiers :\n";
  for (int i=0; i<SIZE; i++) {
    cout << i+1 << ": ";
    cin >> a[i]; }

  cout << "Maintenant ils sont dans l'ordre inverse :\n";
  for (int i=SIZE-1; i>=0; i--)
    cout << a[i] << endl;

  return 0;
}
```

```
int SIZE = 6;
int a[SIZE];
```

l'omission de **const** générera un erreur de compilation : SIZE n'est pas un entier constant

attention au dernier élément du tableau !

voir Ex_tab.cpp

Exemple 2

Dans cet exemple la dimension du tableau est entrée par le clavier et donc elle n'est pas connue avant l'exécution du programme.
Selon les normes ANSI C++ ce procédé est interdit : la dimension d'un tableau doit être impérativement connue avant le lancement du programme (elle ne peut pas changer pendant l'exécution) affin que le programme puisse réserver un espace mémoire suffisant pour enregistrer le tableau. Ce procédé est tolérée par la majorité des compilateurs inclus le notre, mais nous ne l'accepterons non plus !
Plus tard dans le cours nous verrons comment réserver de l'espace mémoire pendant l'exécution du programme avec le pointeurs (allocation dynamique de la mémoire).

voir Ex_tab2.cpp

```
#include <iostream>

using namespace std;

int main() {
  int dim;
  cout << "Quelle est la dimension du tableau ? ";
  cin >> dim;
  int a[dim];

  cout << "Entrez " << dim << " nombres entiers :\n";
  for (int i=0; i<dim; i++) {
    cout << i+1 << ": ";
    cin >> a[i]; }

  cout << "Maintenant ils sont dans l'ordre inverse :\n";
```

Interdit par le norme C++ et dans ce cours !

Opérations avec les tableaux

On peut effectuer toutes les opérations arithmétiques, logiques, etc. aussi avec les tableaux à condition de traiter le tableau élément par élément :

multiplication par une constante : `c * a[i]`

addition de deux tableaux : `c[i] = a[i] + b[i]`

multiplication de deux tableaux : `c[i] = a[i] * b[i]`

logique : `if(a[i] > b[j]) ...`

P. ex. pour additionner les éléments de deux tableaux, il faut parcourir les deux tableaux avec un indice ; on ne peut pas additionner les deux tableaux directement, i.e. écrire `c = a + b` comme pour les variable.

premier indice du tableau = 0 !
dernier indice du tableau = 9 !

```
int a[10], b[10];
int c1[10], c2[10], c3[10];
for (int i=0; i<10; i++) {
  c1[i] = c * a[i];
  c2[i] = a[i] + b[i];
  c3[i] = a[i] * b[i];
}
```

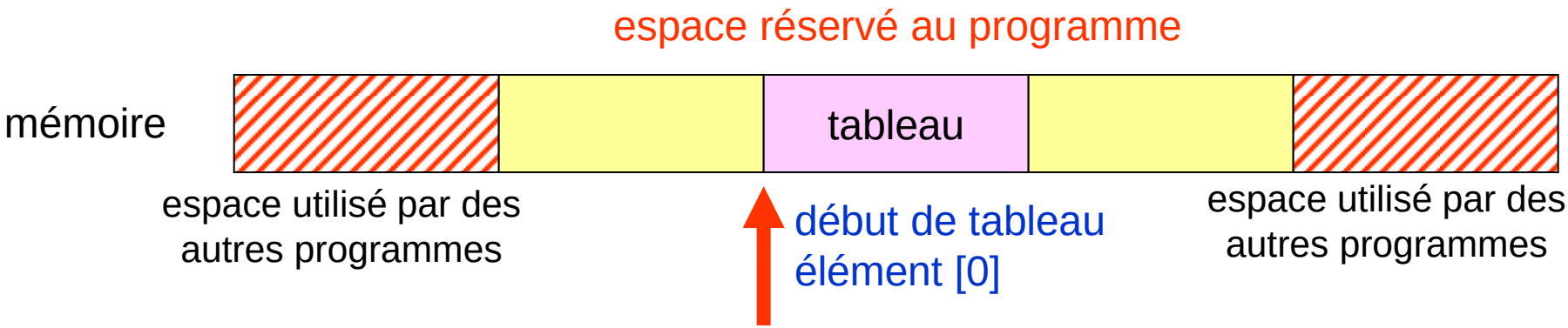
voir Prod_scal.cpp et Prod_vect.cpp

Il en est de même si l'on veut imprimer un tableau : il faut imprimer chaque élément du tableau individuellement, p.ex. avec une boucle.

P.ex. `cout << a;` imprimera l'adresse mémoire du premier élément du tableau !

Attention !

Il n'y a pas de contrôle de débordement du tableau !
Si le programme essaie d'accéder à un élément du tableau avec un indice inférieur à 0 ou supérieur à la taille déclarée, il ne générera pas d'erreurs d'exécution.
Les concepteurs de C++ ont décidé de laisser le programmeur vérifier ses données.



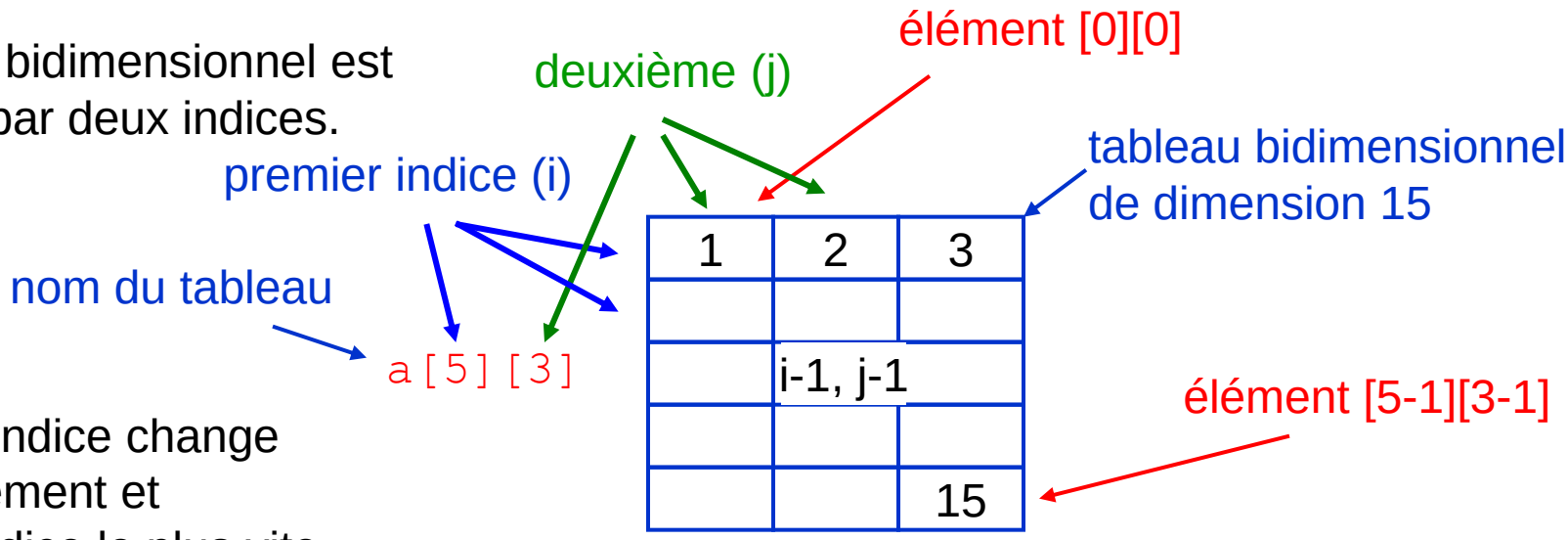
Si vous essayez d'accéder à des éléments dont l'indice n'est pas dans l'intervalle défini, vous obtiendrez des résultats imprévisibles. Si la référence se trouve dans les limites de l'espace réservé au programme, la référence est valide : vous lirez des données qui n'ont aucun sens ou vous écrirez des données par-dessus le programme.
Si la référence est située hors de l'espace réservé au programme, vous essaierez d'accéder à de la mémoire non allouée au programme et vous obtiendrez une *erreur de segmentation* (segmentation fault) et le programme s'arrêtera.

Pour vérifier la dimension en octets d'un tableau, utilisez l'opérateur `sizeof()`.
P.ex. le tableau `int a[5][3]` est composé par 15 éléments de type entier ; `sizeof(a)` est égal à 60 octets, parce que la dimension d'un entier est de 4 octets.

Tableaux multidimensionnels

Un tableau peut avoir comme éléments des tableaux du même type.
Dans ce cas, il s'agit d'un tableau multidimensionnel.
P.ex. une matrice est un tableau bidimensionnel : les éléments de la matrice sont des tableaux unidimensionnels dont les éléments sont des tableaux unidimensionnels.

Un tableau bidimensionnel est déterminé par deux indices.



Le premier indice change le plus lentement et le dernier indice le plus vite.
Le premier indice fait référence à la ligne, le deuxième à la colonne.

Déclaration et initialisation d'un tableau d'entiers :

```
int a[5][3];
a[0][0] = 1;
....
a[4][2] = 15;
```

ou

```
int a[5][3]=
{{1,2,3},
  ....
 {13,14,15}};
```

`int a[5][3] = {0};` tous les éléments sont initialisés à zéro.

Opérations avec les tableaux multidimensionnels

Les opérations mathématiques avec les tableaux multidimensionnels sont similaires à celles avec les tableaux unidimensionnels et suivent les même règles.
Pour parcourir tous les éléments d'un tableau bidimensionnel, on utilise 2 boucles imbriquées, une pour chaque indice.

addition de deux tableaux bidimensionnels :

```
int a[10][5], b[10][5], c[10][5];
for (int i=0; i<10; i++)
  for (int j=0; j<5; j++)
    c[i][j] = a[i][j] + b[i][j];
```

multiplication matricielle ligne par colonne :
$(i \times j) \times (j \times k) \rightarrow (i \times k)$

```
const int I=3, J=5, K=6;
double matA[I][J];
double matB[J][K];
double matC[I][K] = {0};
for (int i=0; i<I; i++)
  for (int k=0; k<K; k++)
    for (int j=0; j<J; j++)
      matC[i][k] +=
        matA[i][j] * matB[j][k];
```

à noter i et I, j et J, k et K !
il s'agit de 2 variables différentes

voir ProdMatrices.cpp

On pourrait redéfinir, i.e. surcharger les opérateurs arithmétiques, logiques, etc. comme on a fait avec les fonctions pour pouvoir écrire p.ex. `C = A * B;`

dans ce cas, il faudra déclarer les tableaux comme de classes.

Passage d'un tableau à une fonction

Pour passer un tableau à une fonction, la liste des paramètres doit indiquer qu'il s'agit d'un tableau (on utilise les crochets `[ ]`) et son type sans préciser sa dimension.

La fonction ne connaît pas la dimension du tableau, donc il faut passer séparément la dimension du tableau avec une autre variable. Attention au débordement !

déclaration de la fonction :

indique à la fonction qu'il s'agit d'un tableau de type double, mais pas sa dimension

```
double somme(double x[], int n);
```

la dimension du tableau est passée séparément

programme principal :

```
int main() {
  double x[5] = {1, 2, 3, 4, 5};
  double y = somme(x,5);
  cout << "somme = " << y << endl;
  return 0; }
```

définition de la fonction :

(cette fonction peut additionner les éléments d'un tableau de n'importe quelle longueur) :

```
double somme(double x[], int n) {
  double somme=0.;
  for (int i=0; i<n; i++) somme = somme + x[i];
  return somme; }
```

voir Prod_scalF.cpp et Prod_vectF.cpp

Tableaux et fonctions

En effet, on passe à la fonction l'adresse mémoire du premier élément du tableau et la taille du tableau (i.e. le nombre des variables du tableau).
P. ex.

```
double z[5];
cout << z;
```

affichera l'adresse mémoire du premier élément du tableau et non pas les éléments du tableau. C'est cet adresse mémoire qu'on transmet à la fonction.
(nous étudierons l'adresse mémoire et les pointeurs plus tard)

Le tableau passé à la fonction est donc en mode lecture/écriture, c.à.d. que toutes les modifications apporté au tableau dans la fonction sont définitives.
Pour empêcher tout modification on peut passer un tableau constant à la fonction, i.e.

```
double somme(const double x[], int n);
```

Le tableau `x` ne peut pas être modifié par la fonction.

La fonction ne renvoie qu'une seule valeur (plus en avant on verra qu'on peut renvoyer l'adresse mémoire du tableau). Par contre, on peut passer à la fonction un tableau *vide*, qui sera rempli par la fonction. Ces données seront aussi disponible dans le programme qui a appelé cette fonction.

voir Prod_vectF.cpp

Passage d'un tableau multidimensionnel

Pour passer un tableau multidimensionnel à une fonction, la liste des paramètres doit contenir le type du tableau comme pour le tableaux unidimensionnels et toutes les dimensions sauf la première doivent être précisées, afin que le compilateur puisse calculer l'emplacement de chaque élément du tableau. Bien que les dimensions sont spécifiées (sauf la première), la fonction ne les connaît pas. Les dimensions du tableau peuvent être passées par d'autres variables. P.ex. un tableau bidimensionnel `a[3][5]` est stocké sous la forme d'un tableau unidimensionnel composé de 3 tableaux de 5 éléments.

indique à la fonction qu'il s'agit d'un tableau bidimensionnel (tableau unidimensionnel composé de 5 tableau)

déclaration de la fonction :

```
void multParConst(double a[][5], int n, int m, double c);
```

les dimension du tableau sont passée séparément

programme principal :

```
int main() {
  double a[3][5];
  multParConst(a, 3, 5, 7.);
  . . . }
```

définition de la fonction :

```
void multParConst(double a[][5], int n, int m, double c) {
  for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
      a[i][j] = c * a[i][j];
}
```

*Structures

Les structures permettent de grouper des données des données de type différentes dans la même structure (au contraire des éléments d'un tableaux, qui doivent être du même type). Tous les types de données peuvent être utilisés dans une structure. Une structure est un type de données dérivé (nouveau type de données) dont les éléments sont d'un autre type.

type *simple* : `int`, `float`, `bool`, `char`, ...

type *dérivé* : les tableaux, les pointeurs, …

Pour utiliser une donnée de type structure, il faut d'abord «construire» la structure, introduite par le mot clé **`struct`** avec un identificateur (nom) :

```
struct indentificateur {
   type1 membre1;
   type2 membre2;
   ...
   ...
   typen membren;
};
```

```
struct element {
   char symbol[3];
   int Z;
   double A;
   double densite;
   bool gaz;
};
element plomb;
```

La déclaration se termine avec un `;` . Puis il faut déclarer une variable de ce type. Dans le C++ les structures sont remplacées par les classes (`class`).

L'accès aux membres d'une structure se fait au moyen de l'opérateur de sélection de membre direct . (point) :

```
element cuivre;
cuivre.densite = 8.96;
double x = cuivre.A;
```

déclaration de la variable cuivre
affectation d'une composante de cuivre
utilisation d'une composante de cuivre

Les membres d'une variable de type structure peuvent être initialisés pendant la déclaration de la variable :

```
element cuivre = {"Cu", 29, 63.55, 8.96, false};
```

ou plus tard dans le programme :

```
element cuivre;
cuivre.densite = 8.96;
...
cin << cuivre.densite;
cout >> cuivre.densite;
```

voir InfoElements.cpp

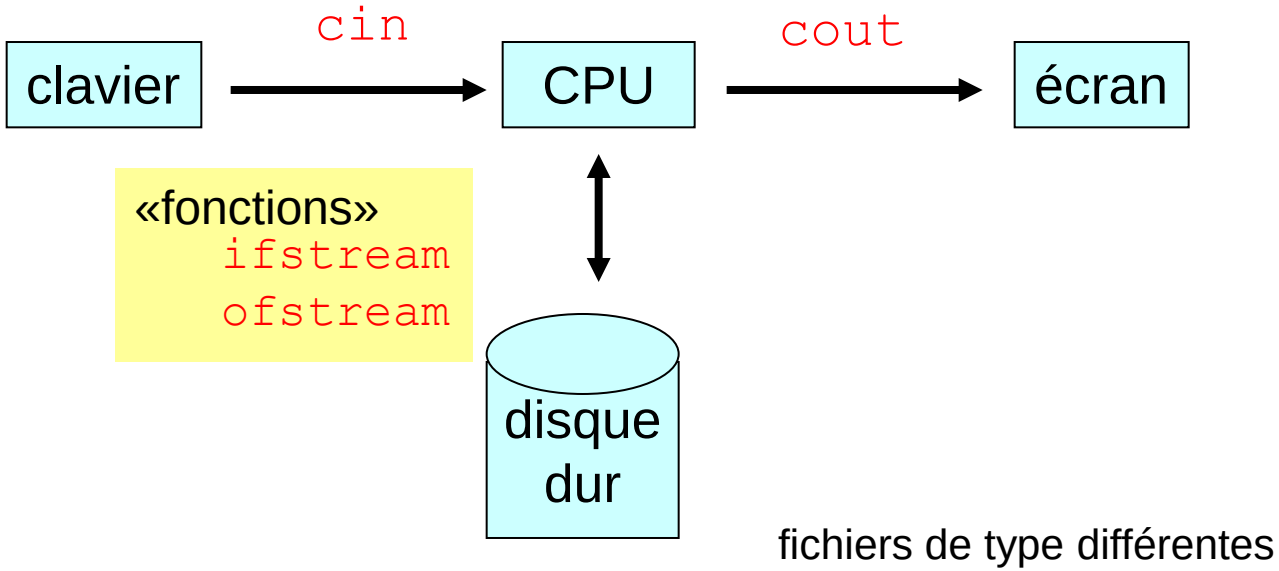
Les structures peuvent être imbriquées :

```
struct date {int jour, mois, an;};
struct joueur {
  char nom[80];
  date dateNaissance;
  float poids;
  ...
};
```

Entrée / Sortie

Jusqu' ici nous avons utilisé les fonctions `cin` et `cout` pour entrer et sortir les données avec l'opérateur d'extraction `>>` et l'opérateur d'insertion `<<` .

On peut aussi utiliser les fonctions du langage C `scanf` et `printf`.



fichiers de type différentes

Entrée / Sortie

Nous pouvons aussi écrire des données dans un fichier ou lire des données à partir d'un fichier.

On utilise les fonctions `ofstream` (sortie) et `ifstream` (entrée) définies dans le fichier en-tête `fstream` de la bibliothèque de C++.

```
#include <fstream>
...
ifstream fin("inputfile.dat");
fin >> ...

ofstream fout("outputfile.dat");
fout << ...
```

«fonctions»
`ifstream`
`ofstream`

`ofstream` définit le flux de sortie `fout`
et ouvre un fichier avec le nom `outputfile.dat`
(i.e. crée la fonction `fout` et la met en relation avec le fichier `outputfile.dat`)

`ifstream` définit le flux d'entrée `fin`
et ouvre un fichier avec le nom `inputfile.dat`
(i.e. crée la fonction `fin` et la met en relation avec le fichier `inputfile.dat`)

Les noms fin, fout, et inputfile.dat, outputfile.dat sont choisis par le programmeur.
`fin` et `fout` se comportent de la même façon que `cin` et `cout`.

Exemple de lecture depuis un fichier

```
#include<iostream>
#include<fstream>

using namespace std;

int main() {
 ifstream fin("tableau1.dat");  //ouverture du fichier tableau1.dat
 if (!fin) {
   cout << "Erreur: le fichier n'existe pas! STOP \n";
   return 1;
  }

  const int SIZE = 100;
  int a[SIZE];      //declaration de tableau de dimension SIZE =
  int i=0;
  while (1) {    //boucle infinie
    fin >> a[i];
    if (!fin) { cout << "Les donnees sont finies !\n";   break; }
    if (i>=100) { cout << "Il y a trop de donnees !\n";   break; }
    i++;
  }
  cout << "Il y a " << i << " elements \n";
  cout << "Maintenant ils sont dans l'ordre inverse :\n";
  for (int j=i; j>0; j--)
    cout << a[j-1] << endl;

  return 0; }
```

d'abord il faut créer le fichier `tableau1.dat`

p. ex. avec Bloc-notes

ouverture du fichier d'entrée `tableau1.dat`

et définition du flux d'entrée `fin`

voir Inoutfile1.cpp

Lire depuis un fichier

1. Créez un fichier de données avec Bloc-notes ou WordPad (dans Accessoires) et enregistrez-le sans formatage (`fichier.dat`)

2. Dans votre programme C++ ouvrez le fichier (même nom)

```
ifstream fin(fichier.dat);
```

`ifstream` est une fonction défini dans le fichier en-tête `fstream` et associe le fichier `fichier.dat` à la fonction `fin` (`#include <fstream>`)

`fin` est un nom arbitraire associé au fichier, nom que vous pouvez choisir librement.

2a. Vérifiez que le fichier a été ouvert correctement : vérifiez l'état de l'*objet* `fin` , i.e. la valeur renvoyée par la fonction `fin` (la valeur renvoyé est de type booléenne) : si l'opération (ouverture de fichier) a réussie, la valeur renvoyée sera `true` (i.e. $\neq 0$!) si l'opération a échoué, la valeur renvoyé sera `false` (i.e. = 0 ; `!fin == true`)

```
if (!fin) cout << "erreur";
```

3. Lisez les données

```
fin >> a;
```

3a. Vérifiez si vous êtes arrivés à la fin du fichier (les données sont finies) ou s'il y a des erreurs de lecture (comme pour 2a)

```
if (!fin) cout << "fin des données";
```

Pour chaque opération (ouverture du fichier, lecture des données) la valeur renvoyé par `fin` représente l'état de l'opération.

*Mise en page

Essayez:

```
cout << 3 << 45;
```

sur l'écran (ou fichier) il apparaîtra `345` !, il n'y a pas d'espaces vides

On à disposition plusieurs manipulateurs pour contrôler le champ, la précision ou le format de l'opération de sortie. Ils sont définis dans le fichier en-tête `iomanip` (`#include <iomanip>`).

Pour fixer la largeur du champ (i.e. ajouter des espaces vides) utilisez `setw(n)` :

```
cout << setw(6) << 3 << setw(8) << 45;
```

affichera `_____3_______45`

6 espaces 8 espaces

Par défaut, les nombres `float` ou `double` sont sortis avec 6 chiffres significatifs. Pour contrôler le nombre de chiffres significatifs, utilisez `setprecision(n)` ;

```
cout << setprecision(10) << M_PI;
```

affichera `3.141592654` au lieu de `3.14159`.

On peut aussi contrôler la mise en page avec les caractères de contrôle, comme `\n` (nouvelle ligne) ou `\t` (tab), etc.

Exemple d'écriture dans un fichier

```
#include <iostream>
#include <cmath>
#include <fstream>     //entree / sortie d'un fichier
#include <iomanip>     //manipulateur d'entree / sortie

using namespace std;

int main() {
  //ecriture de i, j, x, y dans outputfile.dat
  ofstream fout("outputfile.dat");
  int i = 3, j = 4;
  fout << setw(10) << i << setw (10) << j << endl;
  double x = 3.5, y = M_PI;
  fout << setw(15) << setprecision(6) << x;
  fout << setw(15) << setprecision(4) << y;
  fout << setw(15) << setprecision(8) << y << endl;

  //lecture des donnes depuis le meme fichier
  ifstream fin("outputfile.dat");
  int i1, j1;
  double x1, y1, y2;
  fin >> i1 >> j1 >> x1 >> y1 >> y2;

  //affichage des donnes sur l'ecran
  cout << i1 << "\t" << x1 << "\t" << y1 << "\t" << y2 << endl;

  return 0; }
```

ouverture du fichier `outputfile.dat`

et définition du flux de sortie `fout`

`setw(n)`

fixe à n la largeur du champ du nombre

`setprecision(n)`

fixe la précision du nombre à n chiffres significatifs

ouverture du fichier d'entrée (même nom) `outputfile.dat`

et définition du flux d'entrée `fin`

voir Inoutfile2.cpp

Écrire dans un fichier

(très similaire à la lecture depuis un fichier)

1. Dans votre programme C++ ouvrez (créez) le fichier `fichier.dat` avec

```
ofstream fout(fichier.dat);
```

`ofstream` est une fonction défini dans le fichier en-tête `fstream` et associe le fichier `fichier.dat` à la fonction `fout` (`#include <fstream>`)
`fout` est un nom arbitraire associé au fichier `fichier.dat`,
que vous pouvez choisir librement

1a. Vérifiez si le fichier a été ouvert correctement : vérifiez l'état de l'*objet* `fout`

```
if (!fout) cout << "erreur";
```

2. Écrivez les données dans le fichier `fichier.dat`

```
fout << a;
```

2a. Vérifiez s'il y a eu des erreurs d'écriture : vérifiez l'état du *objet* `fout`

```
if (!fout) cout << "erreur de sortie";
```

Le fichier sera écrit dans le même dossier où se trouve le programme exécutable (i.e. le fichier généré par le compilateur). Pour écrire le fichier dans un autre endroit, il faut modifier le *chemin*,
sur Windows p. ex. `"X:\\Mes Documents\\xyz\\fichier.dat"` .

*Exemple de mise en page

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <iomanip>

using namespace std;

int main() {
  ofstream fout("outputfile3.dat");
  if(!fout) {
    cout << "Erreur: le fichier n'a pas ete ouvert! STOP\n";
    system("PAUSE");
    return 1; }

  int i=3, j=45;
  fout << i << j << endl;
  fout << setw(10) << i << setw(10) << j << endl;
  double y = M_PI;
  fout << y;
  fout << setprecision(4) << y;
  fout << setprecision(10) << y << endl;
  fout << setw(15) << y;
  fout << setw(15) << setprecision(4) << y;
  fout << setw(15) << setprecision(10) << y << endl;

  return 0; }
```

`setw`, `setprecision`, etc. sont définies dans `iomanip`

ouverture du fichier `outputfile.dat`

et définition du flux de sortie `fout`

`setw(n)`

fixe la largeur de champ du nombre à n

`setprecision(n)`

fixe la précision du nombre à n chiffres significatifs

voir Inoutfile3.cpp

Résumé

Ce qu'il faut retenir / savoir faire à la fin de cette leçon :

Notion de tableau, dimension d'un tableau

Déclaration et initialisation d'un tableau unidimensionnel

Déclaration et initialisation d'un tableau multidimensionnel

Opérations avec les tableaux

Passage des tableaux aux fonctions

Ecriture dans un fichier / Lecture depuis un fichier

```
ofstream
ifstream
```

Mise en page

Exercices – série 4

Questions

1. Combien de types différents peuvent avoir les éléments d'un tableau ?

2. De quel type doit être un indice de tableau, et dans quel intervalle doit-il se situer ?

3. Quelles seront les valeurs des éléments d'un tableau si sa déclaration n'inclut pas d'initialisation, et si la déclaration inclut un nombre inférieur d'éléments du tableau ?

4. Que se passe-t-il si l'initialisation d'un tableau contient davantage de valeurs que la taille du tableau ?

5. Lorsqu'un tableau multidimensionnel est passé à une fonction, pourquoi C++ exige-t-il que toutes les dimensions soient spécifiées sauf la première ?

6. Citez le premier et le dernier élément de `tableau[25]`.

7. Qu'est ce que donne `a[-5]` ?

8. Cherchez l'erreur dans ces fragments de code:

```
int tab[5][4];
for(int i=0; i<4; i++)
   for(int j=0; j<5; j++)
      tab[i][j] = i+j;
```

```
int tab[5][4];
for(int i=0; i<=5; i++)
   for(int j=0; j<=5; j++)
      tab[i][j] = 0;
```

Exercices

1. Initialisez un tableau (page 6).

2. Calculez le produit scalaire et vectoriel de deux *vecteurs* (page 7).

3. Rangez un tableau de 10 (puis n) éléments en ordre ascendant (i.e. `a[i+1] > a[i]`)

4. Passez des tableaux unidimensionnels aux fonctions.

5. Imprimez la transposée d'une matrice.

6. Ecrivez un programme pour effectuer une rotation d'un vecteur en 3 dimensions (produit de deux matrices, page 11) :
   - lisez le vecteur
   - lisez les 3 angles de rotation
   - calculez la matrice de rotation et affichez-la
   - effectuez la rotation

7. Calculez le produit de deux matrices (page 11).

8. Calculez le déterminant d'une matrice, essayez avec 2×2, puis 3×3, 5×5. Généralisez le programme à des matrices $n \times n$; essayez aussi de calculer le déterminant par récursivité.

9. Ecrivez une fonction pour additionner, multiplier, … deux matrices.

10. Ecrivez et étudiez les exemples d'entrée / sortie (p. 18, 20 et 23)

11. Reprenez l'exercice 3. Rangez un tableau de 100 (puis n) éléments en ordre descendant. Créez un fichier avec 100 entiers, lisez les données, rangez les données, écrivez le résultat dans un autre fichier.

12. Reprenez le problème 1 de la 2ème leçon : à chaque rebond, écrivez dans un fichier la vitesse juste après le rebond et l'hauteur maximale atteignable après le rebond.
La masse de la balle est 100 g.

13. Essayez d'écrire de programmes qui utilisent des structures (voir p. 15 et 16).