





Fichiers, flux, exceptions

A. Introduction	3
B. Fichiers	12
C. Digression : traitement des erreurs et exceptions	47

Comment vous sentez-vous en ce début de cours ?

	Curieux, heureux, joyeux, ...
	Grr, Rostugnududju, ...
	Inquiet
	Flemme

A. Bilan du partiel

- ▶ Sujet et corrigé en ligne
- ▶ Correction : mardi après-midi, vendredi après-midi
- ▶ Notes : à la rentrée, ou plus tôt si moyen sur e-campus
- ▶ Minimum : 0, Maximum : ?, Moyenne : ?, écart type : ?
- ▶ < 10 ? revoir votre méthode de travail

A. Bilan du partiel

- ▶ Sujet et corrigé en ligne
- ▶ Correction : mardi après-midi, vendredi après-midi
- ▶ Notes : à la rentrée, ou plus tôt si moyen sur e-campus
- ▶ Minimum : 0, Maximum : ?, Moyenne : ?, écart type : ?
- ▶ < 10 ? revoir votre méthode de travail

- ▶ Consultation des copies : Jeudi 7 novembre, 12h30-13h45, amphi H2

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections

Pourquoi aller plus loin ?

Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections

Pourquoi aller plus loin ?

Passage à l'échelle !

Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections

Pourquoi aller plus loin ?

Passage à l'échelle !

Données persistantes

Étude de cas : afficher un annuaire

annuaire.ipynb

```
In [1]: #include <iostream>
#include <vector>
using namespace std;
```

```
In [2]: void afficheAnnuaire(vector<string> noms, vector<string> telephones)
    for ( int i = 0; i < noms.size(); i++ )
        cout << noms[i] << " : " << telephones[i] << endl;
    }
```

```
In [3]: vector<string> noms =
    { "Jean-Claude", "Alban", "Tibo", "Célestin" };
vector<string> telephones =
    { "0645235432", "0734534534", "+1150343234", "0634534534"};
```

```
In [4]: afficheAnnuaire(noms, telephones)
```

```
Jean-Claude: 0645235432
Alban: 0734534534
Tibo: +1150343234
Célestin: 0634534534
```

Étude de cas : afficher un annuaire

annuaire.ipynb

```
In [1]: #include <iostream>
#include <vector>
using namespace std;

In [2]: void afficheAnnuaire(vector<string> noms, vector<string> telephones)
    for ( int i = 0; i < noms.size(); i++ )
        cout << noms[i] << ": " << telephones[i] << endl;
    }

In [3]: vector<string> noms =
    { "Jean-Claude", "Alban", "Tibo", "Célestin" };
vector<string> telephones =
    { "0645235432", "0734534534", "+1150343234", "0634534534"};

In [4]: afficheAnnuaire(noms, telephones)

Jean-Claude: 0645235432
Alban: 0734534534
Tibo: +1150343234
Célestin: 0634534534
```

Problèmes

- ▶ Séparation programme / données
- ▶ persistance des données

Afficher un annuaire : ce que l'on voudrait

► Un fichier :

[annuaire.txt](#)

```
Jean-Claude 0645235432
Alban       0734534534
Tibo        +1150343234
Célestin    0634534534
```

Afficher un annuaire : ce que l'on voudrait

- ▶ Un fichier :

[annuaire.txt](#)

```
Jean-Claude 0645235432
Alban       0734534534
Tibo        +1150343234
Célestin    0634534534
```

- ▶ Un programme :

Afficher un annuaire : ce que l'on voudrait

- ▶ Un fichier :

[annuaire.txt](#)

```
Jean-Claude 0645235432  
Alban      0734534534  
Tibo       +1150343234  
Célestin   0634534534
```

- ▶ Un programme :

Comment l'écrire ?

Qu'est-ce qu'un fichier

Définition

Un **fichier informatique** est, au sens commun, une collection d'informations numériques réunies sous un même **nom**, enregistrées sur un support de stockage tel qu'un disque dur, un CD-ROM ou une bande magnétique, et manipulées comme une unité.

Qu'est-ce qu'un fichier

Définition

Un **fichier informatique** est, au sens commun, une collection d'informations numériques réunies sous un même **nom**, enregistrées sur un support de stockage tel qu'un disque dur, un CD-ROM ou une bande magnétique, et manipulées comme une unité.

Techniquement

Un fichier est une **information numérique** constituée d'une **séquence d'octets**, c'est-à-dire d'une séquence de nombres, permettant des usages divers.

Qu'est-ce qu'un fichier

Définition

Un **fichier informatique** est, au sens commun, une collection d'informations numériques réunies sous un même **nom**, enregistrées sur un support de stockage tel qu'un disque dur, un CD-ROM ou une bande magnétique, et manipulées comme une unité.

Techniquement

Un fichier est une **information numérique** constituée d'une **séquence d'octets**, c'est-à-dire d'une séquence de nombres, permettant des usages divers.

Comme la mémoire, mais en persistant !

Qu'est-ce qu'un fichier

Définition

Un **fichier informatique** est, au sens commun, une collection d'informations numériques réunies sous un même **nom**, enregistrées sur un support de stockage tel qu'un disque dur, un CD-ROM ou une bande magnétique, et manipulées comme une unité.

Techniquement

Un fichier est une **information numérique** constituée d'une **séquence d'octets**, c'est-à-dire d'une séquence de nombres, permettant des usages divers.

Comme la mémoire, mais en persistant !

Format du fichier : comment l'information est-elle encodée ?

Écriture dans un fichier

fichier-ecriture.ipynb

```
In [1]: #include <fstream>
        using namespace std;

In [2]: ofstream fichier;           // Déclaration

In [3]: fichier.open("bla.txt");    // Ouverture

In [4]: fichier << "Noel " << 42 << endl; // Écriture;

In [5]: fichier.close();           // Fermeture
```

Écriture dans un fichier

fichier-ecriture.ipynb

```
In [1]: #include <fstream>
        using namespace std;

In [2]: ofstream fichier;           // Déclaration

In [3]: fichier.open("bla.txt");    // Ouverture

In [4]: fichier << "Noel " << 42 << endl; // Écriture;

In [5]: fichier.close();           // Fermeture
```

Quatre étapes :

1. Déclaration
2. Ouverture du fichier
3. Écriture
4. Fermeture du fichier

Lecture depuis un fichier

fichier-lecture.ipynb

```
In [1]: #include <fstream>
        using namespace std;
```

```
In [2]: ifstream fichier;           // Déclaration
```

```
In [3]: fichier.open("bla.txt");    // Ouverture du fichier
```

```
In [4]: string s;
        fichier >> s;               // Lecture
        s
```

```
Out [4]: "Noel"
         type: std::__cxx11::basic_string<char, std::char_traits<char>, std::
```

```
In [5]: int i;
        fichier >> i;
        i
```

```
Out [5]: 42
         type: int
```

```
In [6]: fichier.close();           // Fermeture du fichier
```

Lecture depuis un fichier

fichier-lecture.ipynb

```
In [1]: #include <fstream>
        using namespace std;
```

```
In [2]: ifstream fichier;           // Déclaration
```

```
In [3]: fichier.open("bla.txt");    // Ouverture du fichier
```

```
In [4]: string s;
        fichier >> s;               // Lecture
        s
```

```
Out[4]: "Noel"
        type: std::__cxx11::basic_string<char, std::char_traits<char>, std::
```

```
In [5]: int i;
        fichier >> i;
        i
```

```
Out[5]: 42
        type: int
```

```
In [6]: fichier.close();           // Fermeture du fichier
```

Quatre étapes :

1. Déclaration
2. Ouverture du fichier
3. Lecture
4. Fermeture du fichier

Exemple : Afficher un annuaire contenu dans un fichier

annuaire-fichier.ipynb

```
In [1]: #include <iostream>
        #include <fstream>
        using namespace std;

In [2]: ifstream annuaire;
        annuaire.open("annuaire.txt");

In [3]: string nom;
        string tel;

In [4]: for (int i=0; i <4 ; i++ ) {
        annuaire >> nom;
        annuaire >> tel;
        cout << nom << ": " << tel << endl;
        }
```

```
Jean-Claude: 0645235432
Alban: 0734534534
Tibo: +1150343234
Célestin: 0634534534
```

Exemple : Afficher un annuaire contenu dans un fichier

annuaire-fichier.ipynb

```
In [1]: #include <iostream>
#include <fstream>
using namespace std;

In [2]: ifstream annuaire;
annuaire.open("annuaire.txt");

In [3]: string nom;
string tel;

In [4]: for (int i=0; i <4 ; i++ ) {
annuaire >> nom;
annuaire >> tel;
cout << nom << ": " << tel << endl;
}
```

```
Jean-Claude: 0645235432
Alban: 0734534534
Tibo: +1150343234
Célestin: 0634534534
```

Problème

Comment savoir le nombre de lignes ?

État d'un fichier

Une variable de type fichier peut être dans un **bon état** :

- ▶ « jusqu'ici tout va bien »

État d'un fichier

Une variable de type fichier peut être dans un **bon état** :

- ▶ « jusqu'ici tout va bien »

ou un **mauvais état** :

- ▶ Fichier non trouvé à l'ouverture, problème de permissions
- ▶ Lecture ou écriture incorrecte
- ▶ Fin du fichier atteinte
- ▶ Plus de place disque

État d'un fichier

Une variable de type fichier peut être dans un **bon état** :

- ▶ « jusqu'ici tout va bien »

ou un **mauvais état** :

- ▶ Fichier non trouvé à l'ouverture, problème de permissions
- ▶ Lecture ou écriture incorrecte
- ▶ Fin du fichier atteinte
- ▶ Plus de place disque

Syntaxe

```
if ( fichier ) { ...  
if ( fichier >> i ) { ...
```

Sémantique

- ▶ Le fichier est-il en bon état ?
- ▶ la lecture s'est elle bien passée ?

État d'un fichier

Une variable de type fichier peut être dans un **bon état** :

- ▶ « jusqu'ici tout va bien »

ou un **mauvais état** :

- ▶ Fichier non trouvé à l'ouverture, problème de permissions
- ▶ Lecture ou écriture incorrecte
- ▶ Fin du fichier atteinte
- ▶ Plus de place disque

Syntaxe

```
if ( fichier ) { ...  
if ( fichier >> i ) { ...
```

Sémantique

- ▶ Le fichier est-il en bon état ?
- ▶ la lecture s'est elle bien passée ?

Remarque : si un fichier n'est pas en bon état, on peut en savoir plus

Exemple : Afficher un annuaire contenu dans un fichier

annuaire-fichier-while.ipynb

```
In [1]: #include <iostream>
#include <fstream>
using namespace std;
```

```
In [2]: ifstream annuaire;
annuaire.open("annuaire.txt");
```

```
In [3]: string nom;
string tel;
```

```
In [4]: while ( annuaire >> nom and annuaire >> tel ) {
    cout << nom << ": " << tel << endl;
}
```

Jean-Claude: 0645235432

Alban: 0734534534

Tibo: +1150343234

Célestin: 0634534534

Bonne pratique : vérifier l'état d'un fichier

fichier-ouverture-etat.ipynb

```
In [1]: #include <iostream>
#include <fstream>
using namespace std;
```

```
In [2]: ifstream fichier;
fichier.open("annuaire.txt"); // Un fichier existant
```

```
In [3]: if ( not fichier ) {
    cout << "Erreur à l'ouverture" << endl;
}
```

```
In [4]: fichier.close();
```

```
In [5]: fichier.open("oups.txt"); // Un fichier non existant
```

```
In [6]: if ( not fichier ) {
    cout << "Erreur à l'ouverture" << endl;
}
```

```
Erreur à l'ouverture
```

Bonne pratique : vérifier l'état d'un fichier

fichier-ouverture-etat.ipynb

```
In [1]: #include <iostream>
#include <fstream>
using namespace std;
```

```
In [2]: ifstream fichier;
fichier.open("annuaire.txt"); // Un fichier existant
```

```
In [3]: if ( not fichier ) {
        cout << "Erreur à l'ouverture" << endl;
    }
```

```
In [4]: fichier.close();
```

```
In [5]: fichier.open("oups.txt"); // Un fichier non existant
```

```
In [6]: if ( not fichier ) {
        cout << "Erreur à l'ouverture" << endl;
    }
```

```
Erreur à l'ouverture
```

Remarque

Pour mieux signaler l'erreur, on peut utiliser une **exception**. Voir plus loin.

Lecture depuis le clavier

cin.cpp

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

int main () {
    string nom;
    cout << "Comment t'appelles-tu?" << endl;
    cin >> nom;
    cout << "Bonjour " << nom << " !" << endl;
}
```

Lecture depuis le clavier

cin.cpp

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

int main () {
    string nom;
    cout << "Comment t'appelles-tu?" << endl;
    cin >> nom;
    cout << "Bonjour " << nom << " !" << endl;
}
```

Syntaxe

Lecture d'une variable :

```
cin >> variable;
```


Lecture depuis le clavier

cin.cpp

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

int main () {
    string nom;
    cout << "Comment t'appelles-tu?" << endl;
    cin >> nom;
    cout << "Bonjour " << nom << " !" << endl;
}
```

Syntaxe

Lecture d'une variable :

```
cin >> variable;
```

Sémantique

- ▶ Lit une valeur du type approprié sur le clavier
- ▶ Affecte cette valeur à la variable

Lecture depuis une chaîne de caractères

istringstream.ipynb

```
In [1]: #include <sstream>
        using namespace std;
```

```
In [2]: string s = "1 2 4 8 16";
```

```
In [3]: istringstream flux(s);
```

```
In [4]: int i, j;
```

```
In [5]: flux >> i;
```

```
In [6]: i
```

```
Out[6]: 1
        type: int
```

```
In [7]: flux >> j;
```

```
In [8]: j
```

```
Out[8]: 2
        type: int
```

```
In [9]: i + j
```

```
Out[9]: 3
        type: int
```

```
In [10]: string bla;
```

Lecture depuis une chaîne de caractère

Syntaxe

Lecture d'une variable :

```
istringstream flux(s);  
flux >> variable1;  
flux >> variable2;  
...
```

Lecture depuis une chaîne de caractère

Syntaxe

Lecture d'une variable :

```
istringstream flux(s);  
flux >> variable1;  
flux >> variable2;  
...
```

Sémantique

Lit les variables successivement depuis la chaîne de caractères `s`

Écriture dans une chaîne de caractères

ostringstream.ipynb

```
In [1]: #include <iostream>
#include <sstream>
using namespace std;
```

```
In [2]: ostream flux;
flux << 3.53 << " coucou " << 1 << endl;
flux << 42 << endl;
```

```
In [3]: string s = flux.str();
```

```
In [4]: s
```

```
Out[4]: "3.53 coucou 1
42
"
type: std::__cxx11::basic_string<char, std::char_traits<char>, std::
```

```
In [ ]:
```

Notion de flux

Remarque

- ▶ On a utilisé la même syntaxe pour écrire à l'écran ou dans un fichier :
`xxx << expression`
- ▶ On a utilisé la même syntaxe pour lire au clavier ou depuis un fichier :
`xxx >> variable`

Notion de flux

Remarque

- ▶ On a utilisé la même syntaxe pour écrire à l'écran ou dans un fichier :
`xxx << expression`
- ▶ On a utilisé la même syntaxe pour lire au clavier ou depuis un fichier :
`xxx >> variable`

Définition (Flux de données)

- ▶ Un *flux sortant de données* est un dispositif où l'on peut écrire des données **l'une après l'autre**
- ▶ Un *flux entrant de données* est un dispositif où l'on peut lire des données **l'une après l'autre**

Exemples de flux sortants de données

- ▶ cout : **sortie standard** du programme
Typiquement : écran
 - ♣ Avec tampon

Exemples de flux sortants de données

- ▶ cout : **sortie standard** du programme
Typiquement : écran
 - ♣ Avec tampon
- ▶ cerr : **sortie d'erreur** du programme
 - ♣ Sans tampon

Exemples de flux sortants de données

- ▶ cout : **sortie standard** du programme
Typiquement : écran
 - ♣ Avec tampon
- ▶ cerr : **sortie d'erreur** du programme
 - ♣ Sans tampon
- ▶ fichiers (ofstream)

Exemples de flux sortants de données

- ▶ cout : **sortie standard** du programme
Typiquement : écran
 - ♣ Avec tampon
- ▶ cerr : **sortie d'erreur** du programme
 - ♣ Sans tampon
- ▶ fichiers (ofstream)
- ▶ chaînes de caractères (ostringstream)

Exemples de flux sortants de données

- ▶ cout : **sortie standard** du programme
Typiquement : écran
 - ♣ Avec tampon
- ▶ cerr : **sortie d'erreur** du programme
 - ♣ Sans tampon
- ▶ fichiers (ofstream)
- ▶ chaînes de caractères (ostringstream)
- ▶ connexion avec un autre programme ...

Exemples de flux entrants de données

- ▶ cin : **entrée standard** du programme
Typiquement : clavier

Exemples de flux entrants de données

- ▶ `cin` : **entrée standard** du programme
Typiquement : clavier
- ▶ fichiers (`ifstream`)

Exemples de flux entrants de données

- ▶ `cin` : **entrée standard** du programme
Typiquement : clavier
- ▶ fichiers (`ifstream`)
- ▶ chaînes de caractères (`istringstream`)

Exemples de flux entrants de données

- ▶ `cin` : **entrée standard** du programme
Typiquement : clavier
- ▶ fichiers (`ifstream`)
- ▶ chaînes de caractères (`istringstream`)
- ▶ connexion avec un autre programme ...

D. Digression : traitement des erreurs et exceptions

Exemple (gestion d'entrées invalides)

exception.cpp

```
int factorielle(int n) {
    if ( n < 0 ) throw "Argument invalide: n doit être positif";
    if ( n == 0 ) return 1;
    return n * factorielle(n-1);
}

int main() {
    cout << factorielle(-1) << endl;
}
```

Signaler une exception

Syntaxe

```
throw e;
```

Sémantique

- ▶ Une situation exceptionnelle que je ne sais pas gérer s'est produite
- ▶ Je m'arrête immédiatement et je préviens mon boss c'est-à-dire la fonction appelante
- ▶ On dit qu'on **signale une exception**
- ▶ La situation est décrite par `e`
`e` est un objet quelconque ; par exemple une **exception standard**
- ▶ Si mon boss ne sait pas gérer, il prévient son boss
- ▶ ...
- ▶ Si personne ne sait gérer, **le programme s'arrête**

Quelques exceptions standard

- ▶ exception
- ▶ `invalid_argument`, `out_of_range`, `length_error`
- ▶ `logic_error`, `bad_alloc`, `system_error`

[exception-standard.cpp](#)

```
#include <stdexcept>

int factorielle(int n) {
    if ( n < 0 ) throw invalid_argument("n doit être positif");
    if ( n == 0 ) return 1;
    return n * factorielle(n-1);
}

int main() {
    cout << factorielle(-1) << endl;
}
```

Exemple de gestion d'exception

exception-gestion.cpp

```
int factorielle(int n) {
    if ( n < 0 ) throw invalid_argument("n doit être positif");
    if ( n == 0 ) return 1;
    return n * factorielle(n-1);
}

int main() {
    int n;
    cin >> n;
    try {
        cout << factorielle(n) << endl;
    } catch (invalid_argument & e) {
        cout << "Valeur de n invalide" << endl;
    }
}
```

♣ Gestion des exceptions

Syntaxe

```
try {  
    bloc d'instructions;  
} catch (type & e) {  
    bloc d'instructions;  
}
```

Sémantique

- ▶ Exécute le premier bloc d'instructions
- ▶ Si une exception de type `type` est levée, ce n'est pas grave, je sais gérer :
 - ▶ L'exécution du premier bloc d'instruction s'interrompt
 - ▶ Le deuxième bloc d'instruction est exécuté

Résumé

Fichiers

- ▶ Comment lire et écrire dans des fichiers en C++
- ▶ Un concept uniforme pour lire et écrire : les **flux**
 - ▶ Entrée et sortie standard d'un programme : `cin`, `cout`
 - ▶ Fichiers : `ifstream`, `ofstream`
 - ▶ Chaînes de caractères : `istringstream`, `ostringstream`

Résumé

Fichiers

- ▶ Comment lire et écrire dans des fichiers en C++
- ▶ Un concept uniforme pour lire et écrire : les **flux**
 - ▶ Entrée et sortie standard d'un programme : `cin`, `cout`
 - ▶ Fichiers : `ifstream`, `ofstream`
 - ▶ Chaînes de caractères : `istringstream`, `ostringstream`

Exceptions

- ▶ Comment signaler une erreur dans un programme
- ▶ Comment traiter de telles erreurs

Résumé

Fichiers

- ▶ Comment lire et écrire dans des fichiers en C++
- ▶ Un concept uniforme pour lire et écrire : les **flux**
 - ▶ Entrée et sortie standard d'un programme : `cin`, `cout`
 - ▶ Fichiers : `ifstream`, `ofstream`
 - ▶ Chaînes de caractères : `istringstream`, `ostringstream`

Exceptions

- ▶ Comment signaler une erreur dans un programme
- ▶ Comment traiter de telles erreurs