

Fichiers, flux, exceptions

A. Introduction	1
B. Fichiers	6
C. Digression : traitement des erreurs et exceptions	30

A. Projets de fin de semestre

Thèmes

- ▶ Traitement de Données Libres
- ▶ Jeu Isola

En pratique

- ▶ Volume horaire : une vingtaine d'heures
- ▶ Travail en binôme recommandé
- ▶ Commencer tôt !
- ▶ Voir la [page web](#) pour les détails

Calendrier

1. Semaine 9 : Travail personnel
2. Semaine 10 et 11 : Travail personnel + TP
3. Semaine 12 : Travail personnel
Salle usuelle à disposition pendant les horaires de TP
4. Semaine 13 : Soutenance en TP

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- ▶ Lecture, écriture
- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions
- ▶ Variables, tableaux, valeurs composites
- ▶ Tests, débogage
- ▶ Programmation modulaire

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- ▶ Lecture, écriture
- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions
- ▶ Variables, tableaux, valeurs composites
- ▶ Tests, débogage
- ▶ Programmation modulaire

Pourquoi aller plus loin ?

Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Lecture, écriture
- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions
- ▶ Variables, tableaux, valeurs composites
- ▶ Tests, débogage
- ▶ Programmation modulaire

Pourquoi aller plus loin ?

Passage à l'échelle !

Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Lecture, écriture
- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions
- ▶ Variables, tableaux, valeurs composites
- ▶ Tests, débogage
- ▶ Programmation modulaire

Pourquoi aller plus loin ?

Passage à l'échelle !

Données persistantes

B. Introduction

Étude de cas : un annuaire

Problèmes

- ▶ Séparation programme / données
- ▶ persistance des données

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 !

Écriture dans un fichier

fichier-ecriture.cpp

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

int main() {
    ofstream fichier;           // 1. Déclaration
    fichier.open("bla.txt");   // 2. Ouverture
    fichier << "Noel" << " " << 42 << endl; // 3. Écriture
    fichier.close();           // 4. Fermeture
}
```

Écriture dans un fichier

fichier-ecriture.cpp

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

int main() {
    ofstream fichier;           // 1. Déclaration
    fichier.open("bla.txt");    // 2. Ouverture
    fichier << "Noel" << " " << 42 << endl; // 3. Écriture
    fichier.close();           // 4. Fermeture
}
```

Quatre étapes :

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

Lecture depuis un fichier

fichier-lecture.cpp

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

int main() {
    ifstream fichier;                // 1. Déclaration
    fichier.open("bla.txt");         // 2. Ouverture
    string nom;
    int i;
    fichier >> nom >> i;            // 3. Lecture
    cout << "i: " << i+1 << " nom: " << nom << endl;
    fichier.close();                // 4. Fermeture
}
```

Lecture depuis un fichier

fichier-lecture.cpp

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

int main() {
    ifstream fichier;                // 1. Déclaration
    fichier.open("bla.txt");          // 2. Ouverture
    string nom;
    int i;
    fichier >> nom >> i;              // 3. Lecture
    cout << "i: " << i+1 << " nom: " << nom << endl;
    fichier.close();                 // 4. Fermeture
}
```

Quatre étapes :

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

Exemple : recherche dans un annuaire

Code :

- ▶ `annuaire-2015-amphi1.cpp`
- ▶ `annuaire-2015-amphi2.cpp`

Notion de tampon

Métaphore de la boîte aux lettres

- ▶ Aller poster une lettre, cela prend du temps
- ▶ Donc on attend souvent d'avoir accumulé plusieurs lettres pour les envoyer toutes d'un coup

Notion de tampon

Métaphore de la boîte aux lettres

- ▶ Aller poster une lettre, cela prend du temps
- ▶ Donc on attend souvent d'avoir accumulé plusieurs lettres pour les envoyer toutes d'un coup

Stratégie d'écriture

- ▶ De même, écrire un fichier octet par octet, ce n'est pas efficace
- ▶ On accumule les informations à écrire dans une **mémoire tampon**
- ▶ Lorsqu'il y en a assez on les écrit toutes d'un coup (**flush** : tirer la chasse d'eau)

Notion de tampon : de l'importance de fermer les fichiers

Définition (Mémoire Tampon)

Une zone de mémoire utilisée pour stocker temporairement des données, notamment entre deux processus ou deux pièces d'équipement ne fonctionnant pas à la même vitesse

Notion de tampon : de l'importance de fermer les fichiers

Définition (Mémoire Tampon)

Une zone de mémoire utilisée pour stocker temporairement des données, notamment entre deux processus ou deux pièces d'équipement ne fonctionnant pas à la même vitesse

Attention !

Si on ne ferme pas un fichier, **les données dans le tampon à la fin de l'exécution du programme sont perdues !**

Notion de tampon : de l'importance de fermer les fichiers

Définition (Mémoire Tampon)

Une zone de mémoire utilisée pour stocker temporairement des données, notamment entre deux processus ou deux pièces d'équipement ne fonctionnant pas à la même vitesse

Attention !

Si on ne ferme pas un fichier, **les données dans le tampon à la fin de l'exécution du programme sont perdues !**

Autre instance du même phénomène

Perte de données si on retire une clef USB trop vite !

« vous pouvez maintenant retirer le périphérique en toute sécurité »

Notion de flux

Remarque

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

Notion de flux

Remarque

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

Définition (Flux de données)

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

Exemples

Flux entrant de données

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

Exemples

Flux entrant de données

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

Flux sortant de données

- ▶ cout : **sortie standard** du programme
Avec tampon
- ▶ cerr : **sortie d'erreur** du programme
Sans tampon
- ▶ fichiers (ofstream)
- ▶ chaînes de caractères (ostream) : cf. `ostream.cpp`
- ▶ connexion avec un autre programme ...

État d'un fichier

Sémantique

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

Sémantique

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

Dans un contexte booléen un fichier est évalué à **true** s'il est en bon état :

```
if ( fichier ) ...
```


État d'un fichier

Sémantique

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

Dans un contexte booléen un fichier est évalué à `true` s'il est en bon état :

```
if ( fichier ) ...
```

Remarque

Si un fichier n'est pas en bon état, il y a moyen d'en savoir plus

Notion de format de fichier

Notion de format de fichier

Exemple (trois façons de représenter les mêmes informations)

[annuaire.txt](#)

```
Nicolas 0677903279  
Albert 0323452345  
Sylvie 2342342332
```

[annuaire.csv](#)

```
Nicolas;0677903279  
Albert;0323452345  
Sylvie;2342342332
```

[annuaire.xml](#)

```
<annuaire>  
<fiche><nom>Nicolas</nom><prenom>0677903279</prenom></fiche>  
<fiche><nom>Albert</nom><prenom>0323452345</prenom></fiche>  
<fiche><nom>Sylvie</nom><prenom>2342342332</prenom></fiche>  
</annuaire>
```

Notion de format de fichier

Format d'un fichier

- ▶ Le **format d'un fichier** spécifie comment les informations y sont représentées

Notion de format de fichier

Format d'un fichier

- ▶ Le **format d'un fichier** spécifie comment les informations y sont représentées
- ▶ Analogue du **type d'une variable** qui spécifie comment les données sont représentées en mémoire

Notion de format de fichier

Format d'un fichier

- ▶ Le **format d'un fichier** spécifie comment les informations y sont représentées
- ▶ Analogue du **type d'une variable** qui spécifie comment les données sont représentées en mémoire
- ▶ Le format spécifie la syntaxe et la sémantique

Notion de format de fichier

Format d'un fichier

- ▶ Le **format d'un fichier** spécifie comment les informations y sont représentées
- ▶ Analogue du **type d'une variable** qui spécifie comment les données sont représentées en mémoire
- ▶ Le format spécifie la syntaxe et la sémantique

Types de format

- ▶ Formats texte / binaires
- ▶ Formats structurés (xml)
- ▶ Formats ouverts / fermés (ou privés)

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`
- ▶ Tampons
- ▶ Formats de fichier

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`
- ▶ Tampons
- ▶ Formats de fichier

Exceptions

- ▶ Comment signaler une erreur dans un programme
- ▶ Comment traiter de tels 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`
- ▶ Tampons
- ▶ Formats de fichier

Exceptions

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

Votre dernier cours de programmation impérative au S1