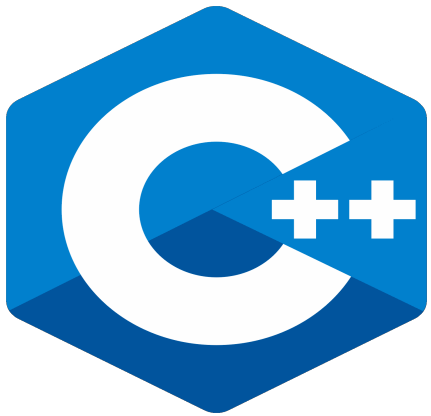
C++ : fichiers

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

C++

Fichiers ?

- Un des premiers supports de stockage
- Dans ce cours, on va détailler
  - les fichiers de code source
  - les fichiers de données

C++

Fichiers de code, pourquoi ?

- Permettant de mieux structurer le code (les fonctions)
- Deux types de fichier en **C++**
  - `.h` : pour déclarer le prototype de nos fonctions
  - `.cpp` : pour implémenter les prototypes déclarés dans le fichier `.h`

C++

Pour la suite, créons les deux fichiers suivants

- `calcul.cpp`
- `calcul.h`

C++

Commençons par inclure `calcul.h` dans `calcul.cpp`

```
#include "calcul.h"
```

C++

Commençons par inclure `calcul.h` dans `calcul.cpp`

```
#include "calcul.h"
```

Pour les fichiers d'entête définis par le développeur, on utilise les " " à la place de < >

C++

Ajoutons le code suivant dans `calcul.h`

```
#ifndef CALCUL_H
#define CALCUL_H

// votre code ici

#endif
```

C++

Ajoutons le code suivant dans `calcul.h`

```
#ifndef CALCUL_H
#define CALCUL_H

// votre code ici

#endif
```

Explication

- Le code ajouté dans `calcul.h` permet d'éviter les inclusions multiples de ce fichier.
- Les prototypes doivent être déclarés au même endroit que le commentaire `votre code ici`.
- Certains IDE/extensions utilisent `#pragma once` pour faire la même chose : solution moins verbeuse, plus performante mais non encore standardisée (pas reconnu par tous les compilateurs).

C++

Déplaçons maintenant la fonction `somme` dans `calcul.cpp`

```
#include "calcul.h"

int somme(int a, int b)
{
    return a + b;
}
```

C++

Déplaçons maintenant la fonction `somme` dans `calcul.cpp`

```
#include "calcul.h"

int somme(int a, int b)
{
    return a + b;
}
```

Déclarons son prototype dans `calcul.h`

```
#ifndef CALCUL_H
#define CALCUL_H

int somme(int a, int b);

#endif
```

C++

Nouveau contenu de la fonction principale

```
#include <iostream>
#include "calcul.h"

using namespace std;

int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}
```

C++

Nouveau contenu de la fonction principale

```
#include <iostream>
#include "calcul.h"

using namespace std;

int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}
```

En lançant le projet, on obtient `undefined reference to 'somme(int, int)'`

C++

Problème

Le fichier `calcul.cpp` n'a pas été compilé.

C++

Problème

Le fichier `calcul.cpp` n'a pas été compilé.

Deux solutions

- Soit utiliser la commande `g++ main.cpp -o main calcul.cpp`
- Soit utiliser la commande `g++ *.cpp -o main`
- Soit configurer **VSC** afin qu'il compile, chaque fois, tous les fichiers `.cpp`

C++

Configurons **VSC**

- Aller dans `Terminal > Configurer les tâches`
- Choisir un fichier selon le compilateur utilisé
- Dans `args`, remplacer `${file}` par `"${workspaceFolder}\\*.cpp"`
- Redémarrer **VSC**
- Compiler avec `Ctrl` + `Shift` + `b`

C++

Exemple avec un type avancé (string, vector...) : code de `calcul.cpp`

```
#include "calcul.h"

int somme(int a, int b)
{
    return a + b;
}

int nbrVoyelle(std::string str)
{
    int nbr = 0;
    for (char c : str)
    {
        if (c == 'a' || c == 'o' || c == 'e' || c == 'u' || c == 'i' ||
            c == 'y')
        {
            nbr++;
        }
    }
    return nbr;
}
```

C++

N'oublions pas de déclarer le prototype de la fonction `nbrVoyelle` **dans** `calcul.h`

```
#ifndef CALCUL_H
#define CALCUL_H

int somme (int a, int b);
int nbrVoyelle (std::string str);

#endif
```

C++

`string` **est un type qu'il faut inclure, sinon erreur**

```
#ifndef CALCUL_H
#define CALCUL_H

#include <string>

int somme (int a, int b);
int nbrVoyelle (std::string str);

#endif
```

C++

Deux questions

1. Pourquoi inclure `string` mais pas `int`, `float`... ?
2. Pourquoi ne pas inclure `string` dans `main.cpp` aussi ?

C++

Deux questions

1. Pourquoi inclure `string` mais pas `int`, `float`... ?
2. Pourquoi ne pas inclure `string` dans `main.cpp` aussi ?

Réponses

Q1 : `int`, `float`... sont des types définis dans le langage **C** mais `string` est un type **C++**

Q2 : inclure `iostream` $\Rightarrow$ inclure `string` (pour **g++** mais pas tous les autres, il vaut mieux l'inclure si on risque de changer de compilateur)

C++

Contenu de la fonction principale

```
#include <iostream>
#include "calcul.h"

using namespace std;

int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    cout << "Le nombre de voyelle dans bonjour est " << nbrVoyelle("
      bonjour") << endl;
    return 0;
}
```

C++

Fichiers de données, pourquoi ?

- Permettant de lire et écrire des données utilisées dans un programme
- Plusieurs étapes pour l'utilisation d'un fichier
  - inclusion de la librairie `fstream` (pour file stream)
  - déclaration d'un fichier en lecture ou en écriture
  - Ouverture
  - utilisation
  - fermeture

C++

Commençons par inclure la librairie `fstream`

```
#include <fstream>
```

C++

Commençons par inclure la librairie `fstream`

```
#include <fstream>
```


fstream contient trois classes

- **ofstream** : permet la création et l'écriture dans un fichier
- **ifstream** : permet la lecture d'un fichier
- **fstream** : permet la création, la lecture et l'écriture (combinaison de **ofstream** et **ifstream**)

C++

Pour déclarer un fichier d'écriture (`output file stream`)

```
ofstream file;
```

C++

Pour déclarer un fichier d'écriture (`output file stream`)

```
ofstream file;
```

Pour ouvrir un fichier

```
file.open("fichier.txt");
```

C++

Pour déclarer un fichier d'écriture (output file stream)

```
ofstream file;
```

Pour ouvrir un fichier

```
file.open("fichier.txt");
```


Explication

- `fichier.txt` est le nom de notre fichier sur le disque.
- Il se trouve dans le même dossier que nos fichiers sources, s'il n'existe pas il sera créé et s'il existe il sera écrasé.
- Il est possible d'indiquer un chemin différent : par exemple `file.open("C:/Users/elmou/fichier.txt");`

Les deux instructions précédentes pourront être remplacées par la suivante

```
ofstream file("fichier.txt");
```

Les deux instructions précédentes pourront être remplacées par la suivante

```
ofstream file("fichier.txt");
```

Pour créer un fichier anonyme (sans l'ouvrir)

```
ofstream {"fichier.txt"};
```

C++

Avant d'utiliser le fichier, il faut s'assurer qu'il a été ouvert avec succès

```
if(file)
{
    // Ouverture réussie, on peut utiliser le fichier
}
else
{
    cout << "Problème d'ouverture de fichier" << endl;
}
```

C++

Avant d'utiliser le fichier, il faut s'assurer qu'il a été ouvert avec succès

```
if(file)
{
    // Ouverture réussie, on peut utiliser le fichier
}
else
{
    cout << "Problème d'ouverture de fichier" << endl;
}
```


Lancez le projet et vérifiez la création du fichier `fichier.txt`, si un message d'erreur ne s'affiche pas.

C++

Pour écrire dans le fichier

```
if (file)
{
    file << "bonjour" << endl;
    file << 'a' << endl;
    file << 85 << endl;
}
else
{
    cout << "Problème d'ouverture de fichier" << endl;
}
```

C++

Pour écrire dans le fichier

```
if (file)
{
    file << "bonjour" << endl;
    file << 'a' << endl;
    file << 85 << endl;
}
else
{
    cout << "Problème d'ouverture de fichier" << endl;
}
```

À chaque exécution, le contenu précédent est écrasé.

Pour écrire à la fin du fichier, on ajoute le paramètre `app` (pour append)

```
file.open("fichier.txt", ios::app);
```

Pour écrire à la fin du fichier, on ajoute le paramètre `app` (pour append)

```
file.open("fichier.txt", ios::app);
```

Ou si nous utilisons le raccourci

```
ofstream file("fichier.txt", ios::app);
```

Pour écrire à la fin du fichier, on ajoute le paramètre `app` (pour append)

```
file.open("fichier.txt", ios::app);
```

Ou si nous utilisons le raccourci

```
ofstream file("fichier.txt", ios::app);
```


Relancez l'exécution du projet plusieurs fois et vérifiez que le contenu s'ajoute chaque fois à la suite.

Une bonne pratique consiste à fermer le fichier, après utilisation, afin de libérer l'espace mémoire occupé

```
if (file)
{
    file << "bonjour" << endl;
    file << 'a' << endl;
    file << 85 << endl;
    file.close();
}
else
{
    cout << "Problème d'ouverture de fichier" << endl;
}
```

C++

Pour déclarer un fichier de lecture (input file stream)

```
ifstream file;
```

C++

Pour déclarer un fichier de lecture (`input file stream`)

```
ifstream file;
```

Pour ouvrir un fichier

```
file.open("fichier.txt");
```

Pour déclarer un fichier de lecture (`input file stream`)

```
ifstream file;
```

Pour ouvrir un fichier

```
file.open("fichier.txt");
```


Un raccourci pour les deux instructions précédentes

```
ifstream file("fichier.txt");
```

Trois fonctions/opérateurs pour lire le contenu

- `get()` permet de lire un caractère
- `getline()` permet de lire une ligne
- `>>` permet de lire un mot

C++

Pour lire la première ligne du fichier

```
if(file)
{
    string line;
    getline(file, line);
    cout << line << endl;
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

C++

Pour lire la première ligne du fichier

```
if(file)
{
    string line;
    getline(file, line);
    cout << line << endl;
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

`getline()` : place la ligne suivante dans `line` et retourne `true`. S'il n'y a plus de ligne à lire, elle retourne `false`.

Exercice

Écrivez un code **C++** qui permet de lire tout le contenu de `fichier.txt` ligne par ligne et l'afficher dans le terminal.

Solution

```
if(file)
{
    string line;
    while(getline(file, line))
    {
        cout << line << endl;
    }
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

C++

Pour lire le premier caractère

```
if(file)
{
    char c;
    file.get(c)
    cout << c << endl;
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

C++

Pour lire le premier caractère

```
if(file)
{
    char c;
    file.get(c)
    cout << c << endl;
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

`get()` : place le caractère suivant dans `line` et retourne `true`. S'il n'y a plus de caractère à lire, elle retourne `false`.

C++

Exercice

Écrivez un code **C++** qui permet de lire tout le contenu de `fichier.txt` caractère par caractère et l'afficher dans le terminal.

Solution

```
if(file)
{
    char c;
    while(file.get(c))
    {
        cout << c << endl;
    }
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

C++

Pour lire le premier mot

```
if(file)
{
    string word;
    file >> word;
    cout << word << endl;
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

C++

Pour lire le premier mot

```
if(file)
{
    string word;
    file >> word;
    cout << word << endl;
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

Les séparateurs considérés sont l'espace et le retour à la ligne

C++

Exercice

Écrivez un code **C++** qui permet de lire tout le contenu de `fichier.txt` mot par mot et l'afficher dans le terminal.

Solution

```
if(file)
{
    string word;
    while (file >> word)
    {
        cout << word << endl;
    }
}
else
{
    cout << "Problème d'ouverture de file" << endl;
}
```

Remarque

Il est aussi possible de lire un nombre entier ou réel.

C++

Remarque

Il est aussi possible de lire un nombre entier ou réel.

Exemple

```
double x;
file >> x;
```

C++

Étant donné le fichier `phrases.txt` ayant le contenu suivant

```
Une première phrase.
Et voici une deuxième.
Et encore une troisième. Ciao.
```

C++

Étant donné le fichier `phrases.txt` ayant le contenu suivant

```
Une première phrase.
Et voici une deuxième.
Et encore une troisième. Ciao.
```

Exercice 1

Écrivez un code **C++** qui permet de lire les données du fichier `phrases.txt` et d'afficher le nombre total de mots et de lignes et de phrases.

Étant donné le fichier `notes.txt` ayant le contenu suivant

```
wick 17 13 15
dalton 20 9 13
maggio 16 12 20
baggio 8 7  9
```

Étant donné le fichier `notes.txt` ayant le contenu suivant

```
wick 17 13 15
dalton 20 9 13
maggio 16 12 20
baggio 8 7  9
```

Exercice 2

Écrivez un code **C++** qui permet de lire les données du fichier `notes.txt`, calculer la moyenne de chaque personne et l'écrire avec le nom dans `moyennes`.

C++

filesystem

- Librairie contenant des fonctions permettant des opérations OS sur les fichiers
  - renommer
  - copier
  - déplacer
  - ...

Commençons par inclure la librairie `filesystem`

```
#include <filesystem>
```

C++

Commençons par inclure la librairie `filesystem`

```
#include <filesystem>
```

Pour copier un fichier

```
filesystem::copy("fichier.txt", "file.txt");
```

C++

Commençons par inclure la librairie `filesystem`

```
#include <filesystem>
```

Pour copier un fichier

```
filesystem::copy("fichier.txt", "file.txt");
```

Pour renommer un fichier

```
filesystem::rename("fichier.txt", "f.txt");
```

Remarque

Les fonctions précédentes sont aussi disponibles dans la librairie **cstdio** du langage **C** qui est utilisable en **C++**.

C++

Pour créer un répertoire (dossier)

```
filesystem::create_directory("r1");
```

C++

Pour créer un répertoire (dossier)

```
filesystem::create_directory("r1");
```

Pour vérifier l'existence d'un fichier ou répertoire

```
cout << boolalpha << filesystem::exists("f.txt") << endl;
// affiche true
```

C++

Pour créer un répertoire (dossier)

```
filesystem::create_directory("r1");
```

Pour vérifier l'existence d'un fichier ou répertoire

```
cout << boolalpha << filesystem::exists("f.txt") << endl;
// affiche true
```

Pour récupérer le chemin relatif ou absolu d'un fichier ou répertoire

```
cout << boolalpha << filesystem::relative("f.txt") << endl;
// affiche "f.txt"

cout << boolalpha << filesystem::absolute("f.txt") << endl;
// affiche "C:\\Users\\...\\f.txt"
```

C++

Pour lister le contenu d'un répertoire [C++17]

```
for (auto entry : filesystem::directory_iterator("r1"))
{
    cout << entry.path() << endl;
}
```

C++

Pour tester s'il s'agit d'un dossier ou un fichier

```
cout << boolalpha << filesystem::is_directory("r1") << endl;
// affiche true

cout << boolalpha << filesystem::is_directory("f.txt") << endl;
// affiche false
```

C++

Pour tester s'il s'agit d'un dossier ou un fichier

```
cout << boolalpha << filesystem::is_directory("r1") << endl;
// affiche true

cout << boolalpha << filesystem::is_directory("f.txt") << endl;
// affiche false
```


Pour vérifier si le dossier est vide

```
cout << boolalpha << filesystem::is_empty("r1") << endl;
// affiche true
```

C++

Pour supprimer un fichier

```
filesystem::remove("f.txt");
```

C++

Pour supprimer un fichier

```
filesystem::remove("f.txt");
```

C++

Pour supprimer un fichier

```
filesystem::remove("f.txt");
```


Remarque

`filesystem::remove()` permet aussi de supprimer les dossiers (vides).

C++

Exercice (lire tout l'énoncé avant de répondre aux questions)

Écrire un programme **C++** qui permet de

1. créer deux répertoires `d1` et `d2`
2. créer trois fichiers `f1.txt`, `f2.txt` et `f3.txt` dans `d1`
3. déplacer tous les fichiers de `d1` dans `d2`
4. ajouter le préfixe `test_` au nom de chaque fichier de `d2`
5. lister le nouveau contenu de `d2`
6. supprimer `d1`, `d2` et son contenu
7. demander l'accord de l'utilisateur avant chaque opération