Ce chapitre est encore en cours de rédaction, voire à l'état d'ébauche. N'hésitez pas a faire des remarques ou poser des questions sur le forum de Zeste de Savoir ou de OpenClassroom.

Les fichiers

Représentation binaire et texte

Permettre la lecture et l'écriture de fichiers. 2 types de fichiers : binaire et texte. Déjà vu les différentes représentation d'un nombre, en binaire :

main.cpp

```
#include <iostream>
#include <bitset>

int main() {
    int const i { 123456 };
    std::cout << "représentation numérique: " << i << std::
endl;
    std::cout << "représentation binaire: " << std::bitset<
sizeof(i)*8>(i) << std::endl;
    std::cout << sizeof(123) << std::endl;
    std::cout << sizeof("123") << std::endl;
    std::cout << sizeof(123456) << std::endl;
    std::cout << sizeof("123456") << std::endl;
    std::cout << sizeof(123456789) << std::endl;
    std::cout << sizeof("123456789") << std::endl;
}
```

affiche :

```
représentation numérique: 123456
représentation binaire: 00000000000000011110001001000000
4
```

```
4
4
7
4
10
```

En mode texte, chaque chiffre prend 1 octet (+ un octet de fin de chaîne), en binaire, toujours la même taille pour un int. Donc fichier texte permet d'être lu avec un simple éditeur, mais plus gros en taille.

Création et ouverture de fichier

Utilisation de stream, comme pour `cout` et `cin`, avec `«` pour écrire et `»` pour lire.

Utiliser fstream pour lecture et/ou écriture (“file stream”) ou classes plus spécialisées ofstream (“output file stream”) et ifstream (“input file stream”).

On peut simplement ouvrir un fichier en créant un ifstream :

```
#include <fstream>

int main() {
    std::ifstream text_file("in.txt"); // mode texte
    std::ifstream binary_file("in.bin", std::ios::binary);
// mode binaire
}
```

Le fichier est automatiquement enregistré et fermée lorsque les variable text_file et binary_file sont détruite (donc à la fin du bloc). En utilisant la portée des variables, on peut donc décider quand les fichiers sont ouvert et fermé :

```
#include <fstream>

int main() {
    cout << "pas encore ouvert" << endl;
    {
```

```
        cout << "ouverture du fichier" << endl;
        std::ifstream file("in.txt");
    } // fermeture du fichier
    cout << "le fichier est fermé" << endl;
}
```

De la même façon, pour ouvrir un fichier en écriture :

```
#include <fstream>

int main() {
    std::ofstream file("out.txt");
}
```

Si le fichier n'existe pas, il est créé.

Le premier paramètre est le nom du fichier. Attention, le chemin par défaut est celui de l'application, pas celui du fichier .cpp (s'ils ne sont pas au même endroit). Possibilité de mettre chemin relatif ou absolu :

```
std::ofstream file("out.txt");
std::ofstream file("../../out.txt");
std::ofstream file("C:/myapp/out.txt");
```

Remarque : normalement, sous Windows, le séparateur de chemin est la barre oblique inversée `\`. Vous pouvez l'utiliser, mais n'oublier pas que ce caractère correspond au caractère d'échappement en C++, il faut dont écrire `\\` ou utiliser une chaîne brute :

```
std::ofstream file("C:\\myapp\\out.txt");  // double barre
oblique inversée
std::ofstream file(R"(C:\myapp\out.txt)"); // chaîne brute
```

Le second paramètre permet de spécifier les options d'ouverture :

- `std::ios::binary` : ouverture en mode binaire ;
- `std::ios::in` : en lecture (pour fstream) ;
- `std::ios::out` : en écriture (pour fstream) ;
- `std::ios::app` (*append* = “ajouter”) : ajoute à la fin du fichier ;

- `std::ios::trunc` (*truncate* = “tronquer”) : supprimer le contenu ;
- `std::ios::ate` (*at the end* = “à la fin”) : ajoute à la fin lors de l'ouverture.

Validation a la compilation et typage fort
Supposons que vous recevez un `std::fstream` quelconque, sans rien savoir sur ce flux. Pour pouvoir travailler dessus, vous devrez tester lors de l'execution si le fichier est ouvert en ecriture et/ou lecture pour travailler dessus.

```
std::fstream f { ... };
assert(f.is_open());
assert(f.is_readable());
f << "hello world";
```

En toute rigueur, vous devrez faire cela dans tout les codes qui utilise `std::fstream`, mais c'est lourd.

Possible d'utiliser le typage fort du C++ pour simplifier le code. Au lieu d'utiliser un type generique comme `std::fstream`, utiliser un type plus specifique, qui apporte des garanties sur ce que l'on peut faire ou non avec le flux. Par exemple `std::ofstream` garantie que l'on peut ecrire dessus (s'il est valide). Pas besoin de tester.

```
std::ofstream f { ... };
assert(f.is_open());
f << "hello world";
```

La validation est faite a la compilation = plus performant, plus sur et code plus simple.

Et cette approche est interessante en C++, a faire des que possible. Par exemple, meme avec `ofstream`, vous devez tester si le flux est valide (ouvert). Au lieu d'ajouter un test systematique, creer une classe `valide_ofstream` :

```
std::fstream f { ... };
assert(f.is_open());
```

```
assert(f.is_readable());
f << "hello world";

// par

valide_ofstream f { ... };
f << "hello world";
```

Ecriture et lecture

En utilisant les opérateurs de flux « pour l'écriture et » pour la lecture. (attention a ce que le fichier soit ouvert avec le “bon” mode).

Par exemple, pour écrire une liste de valeur, séparé par un espace :

```
file << i << ' ' << j << ' ' << k << endl;
```

Remarque : avec mode texte, les chiffres sont écrit les uns à la suite des autres. Si on écrit :

```
file << 1;
file << 2;
file << 3;
```

cela produit dans le fichier “123”. Si on essaie de lire, on lira qu'un seul nombre (123). Donc penser à ajouter des séparateurs.

Pour la lecture :

```
file >> i;
```

Problème de séparation des caractères. De lecture d'une ligne

lecture complet dans un string ?

Caractère spéciaux : tabulation \t retour à la ligne \n

Problème lecture windows/linux : \n et \r\n

ouverture et fermeture manuelle ? (open, is_open, close)

Se positionner dans un fichier ? (eof, fpos, etc.)

Gestion des erreurs, exceptions

Pratiquer

- fichier csv (comma separated values) : tableau avec séparation par `,`
- fichier texte excel : tabulation + retour à la ligne
- génération xml
- génération json
- lecture et écriture image ?

Chapitre précédent	Sommaire principal	Chapitre suivant