Programmation C++

Gestion des Entrées / Sorties

ING2-GSI

CY Tech

2023-2024

Généralité sur les flots

- Les entrées et sorties en C++ se font toujours par l'intermédiaire de **flots** (ou **flux**) :
  - cin : flot d'entrée standard
  - cout : flot de sortie standard
  - cerr : flot standard pour les erreurs
- Les opérateurs :
  - << : écriture sur un flot de sortie
  - >> : écriture sur un flot d'entrée

Ex1 : Entrée/Sortie avec les classes

Fraction.hpp

```
#include <iostream>

class Fraction {
private :
    int numerateur = 0;
    int denominateur = 1;
public :
    // ...
    friend ostream& operator<<(ostream& out,
        const Fraction & f);
    friend istream& operator>>(istream& in, Fraction & f);
};
```

Ex1 : Opérateur de sortie

Fraction.cpp

```
ostream& operator<<(ostream& out, const Fraction & f) {
    out << f.numerateur << "/" << f.denominateur << endl;
    return out;
}
```

```
Fraction f(5,6);
cout << f;
// afficher 5/6
```

Ex1 : Opérateur d'entrée

```
Fraction f;
cout << "Entrez une fraction :" ;
cin >> f;          // format n/d
```

Ex1 : Opérateur d'entrée

- Vérifier le format (n/d) et gérer les erreurs :

Fraction.cpp

```
istream& operator>>(istream& in, Fraction & f) {
    char c;
    if ( !(in >> f.numerateur >> c >> f.denominateur)
        || (c!='/')) {
        in.setstate(ios::failbit);
    }
    return in;
}
```

Entrées / Sorties avec des fichiers

- **Fichier physique** : collection d'octets sauvegardées sur un support physique.
- **Fichier logique** : variable liée au fichier physique, utilisée dans le programme (*flux* ou *flot*).

- Avant toute manipulation, un fichier doit être **ouvert**.
- A la fin des traitements du fichier, il doit être **fermé**.

Entrées / Sorties avec des fichiers

#include <fstream>

Dans <fstream>, la librairie standard fournit :

- ofstream : pour écrire sur les fichiers
- ifstream : pour lire à partir de fichiers
- fstream : lire et écrire à partir de/vers des fichiers

Ouverture et Fermeture d'un fichier

- Ouvrir en mode lecture (constructeur ifstream)

```
ifstream nomFlux("cheminFichier" [,mode])
```

- Ouvrir en mode ´ecriture (constructeur ofstream)

```
ofstream nomFlux("cheminFichier" [,mode])
```

- Fermer un flux

```
nomFlux.close();
```

Mode d'ouverture

Les différents modes d'ouverture sont :

- ios::in permet la lecture (input) [1]
- ios::out permet l'écriture (output) [2]
- ios::app ajoute à la fin du fichier (append)
- ios::ate met le curseur à la fin du fichier (at end)
- ios::binary ouvre un fichier binaire (binary)
- ios::trunc vide le fichier à l'ouverture (truncate)

1. Mode par défaut de ifstream
2. Mode par défaut de ofstream

Ecriture dans un fichier

- L'écriture dans un fichier s'effectue comme l'affichage à l'écran avec l'opérateur <<
- Toute variable ou constante de type simple (bool, char, int, float, double, string, ...) peut être écrite

- Syntaxe :

```
flux << variable;
```

Ex2 : Ecriture dans un fichier

Test.cpp

```
Fraction f1(1,2);
Fraction f2(2,3);
ofstream ofs("Fractions.txt", ios::out);
ofs << f1;
ofs << f2;
ofs.close();
```

```
$> cat Fractions.txt
1/2
2/3
```

Lecture dans un fichier

- La lecture dans un fichier s'effectue avec l'op´erateur >>
- Toute variable ou constante de type simple (bool, char, int, float, double, string, ...) peut être lue

- Syntaxe :

```
flux >> variable;
```

- La lecture s'arrˆete au premier espace ou au premier caract`ere qui ne peut pas faire partie de la repr´esentation ASCII du type lu

Lecture dans un fichier

- La fonction qui permet de lire un fichier texte ligne par ligne : **getline()**
- Syntaxe :

```
bool getline(ifstream & is, string & line)
```

true : il reste encore des donn´ees
false : c'est la fin du fichier

Lecture dans un fichier

- Il est conseill´e avant toute op´eration sur le flux de tester les indicateurs d'état en appelant une de ces méthodes :

  bad() renvoie true si une op´eration de lecture ou d'´ecriture ´echoue (exemple : ´ecrire dans un fichier qui n'est pas ouvert en écriture)

  fail() retourne true dans les mˆemes cas que bad(), mais aussi dans le cas où une erreur de format se produit (exemple : lecture/´ecriture d'un caract`ere au lieu d'un entier)

  eof() renvoie true si un fichier ouvert en lecture a atteint la fin

  good() c'est le drapeau d'´etat le plus g´en´erique. Il retourne false lorsque les méthodes précédentes retourneraient true.

Ex3 : Lecture dans un fichier

```
ifstream ifs("Fractions.txt");
if (ifs) {
    Fraction f;
    while (!ifs.eof()) {
        ifs >> f;
        if (!ifs.fail()) {
            cout <<f;
        }
    }
    ifs.close();
}
else {
    cerr << "Impossible d'ouvrir le fichier" << endl;
}
```

Position dans le fichier

- Pour connaître la position courante :
  - tellg() : s'il est ouvert avec ifstream
  - tellp() : s'il est ouvert avec ofstream
- Ces deux m´ethodes renvoient la position courante dans le fichier (le numéro de l'octet courant depuis le début du fichier)

Position dans le fichier

- Pour se déplacer à une position précise dans le fichier :
  seekg(pos,mode) : s'il est ouvert avec ifstream
  seekp(pos,mode) : s'il est ouvert avec ofstream
- **pos** : le numéro d'octet où se positionner
- **mode** : le mode de d´eplacement :
  - ios::beg octet indiqué depuis le début du fichier
  - ios::cur octet indiqu´e depuis la position courante dans le fichier
  - ios::end octet indiqué depuis la fin du fichier

Exceptions

Exceptions : pourquoi ?

```
int division(int a, int b) {
    return (a/b);
}
```

- Si b = 0 : `Floating point exception (core dumped)`
- Mauvaise solution 1 : retourner une valeur d'erreur ERROR VALUE par défaut. Mais laquelle ?
- Mauvaise solution 2 : afficher un message d'erreur au lieu de faire le calcul. Mais quelle valeur retourner ? Et comment faire avec un GUI ?
- Moyenne solution 3 : changer le prototype de la méthode
  - `bool division(int a, int b, int& res);`
  - pas pratique d'utilisation
- Moyenne solution 4 : quitter le programme
  - `exit(-1);`

Exceptions : comment ?

- Lancer une exception : retourner une erreur sous la forme d'une valeur (message, code, objet exception)

```
throw 0;
throw string("Erreur de calcul");
throw MonException(...);

#include <exception>
 class MonException : public exception {
  virtual const char* what() const throw()
  {
     return "Oups ! MonException ... ";
  }
}
```

Exceptions : comment ?

- Attraper une exception : définir un bloc qui encadre l'instruction directement (ou indirectement). S'il y a un problème, on l'*attrape*.

```
try
{   // code generant une exception
}
catch (int code)
{   cerr << "Exception " << code << endl;
}
catch (exception& e)
{   cout << e.what() << endl;
}
catch (...)
{   cout << "Attape toutes les exceptions";
}
```

Exceptions

- Il existe plusieurs classes d'exceptions déjà définies ▸ Documentation CPlusPlus
- Exceptions de logique :
  - `domain_error` : Erreur de domaine mathématique
  - `invalid_argument` : Argument invalide passé à une fonction
  - `length_error` : Taille invalide
  - `out_of_range` : Erreur d'indice de tableau
  - `logic_error` : Autre problème de logique
- Exceptions d'exécution :
  - `range_error` : Erreur de domaine
  - `overflow_error` : Erreur d'overflow
  - `runtime_error` : Autre type d'erreur