Chapitre 7 – Contrôle de flux en C++

1. Introduction au contrôle de flux

En C++, le contrôle de flux (Input/Output – I/O) désigne l'ensemble des mécanismes permettant :

- de **lire** des données (entrée / input)
- d'**écrire** des données (sortie / output)
- de **gérer des fichiers**
- de **contrôler l'état** d'un flux (erreurs, fin de fichier, etc.)

Le C++ utilise principalement la bibliothèque standard <iostream> pour les flux console, et <fstream> pour les fichiers.

2. Flux standards : cin, cout, cerr

2.1. Sortie standard : cout

Déclarée dans <iostream> dans l'espace de nom std.

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

int main() {
    cout << "Hello, world !" << endl;
    return 0;
}
```

- << est l'opérateur d'insertion (output).
- endl vide le buffer et passe à la ligne.

2.2. Entrée standard : cin

Utilisé pour lire des valeurs :

```
int x;
cin >> x;
```

L'opérateur >> est l'opérateur d'extraction (input).

Lecture d'une ligne complète :

```
string nom;
getline(cin, nom);
```

2.3. Flux d'erreur : cerr, clog

- cerr : erreurs, non-bufferisé
- clog : informations de log, bufferisé

cerr << "Erreur : division par zéro" << endl;

3. État d'un flux

Les flux possèdent des indicateurs d'état essentiels pour détecter des erreurs.

3.1. Fonctions d'état principales

Méthode	Signification
cin.good()	Tout va bien
cin.fail()	Lecture échouée (ex : lettre à la place d'un nombre)
cin.eof()	Fin de fichier atteinte
cin.bad()	Erreur grave (ex : corruption)

Exemple :

```
int x;
cin >> x;
if (cin.fail()) {
    cerr << "Erreur de saisie !" << endl;
    cin.clear();    // Réinitialiser l'état
    cin.ignore(1000, '\n'); // Ignorer l'entrée erronée
}
```

4. Surcharge des opérateurs de flux

Les classes personnalisées peuvent définir comment elles s'affichent ou se lisent.

4.1. Surcharge de operator<< (affichage)

```
class Point {
    int x, y;
public:
    Point(int x, int y) : x(x), y(y) {}
    friend ostream& operator<<(ostream& os, const Point& p) {
        os << "(" << p.x << ", " << p.y << ")";
        return os;
    }
};
```

Utilisation :

```
Point p(3,4);
cout << p;   // affiche : (3,4)
```

4.2. Surcharge de operator>> (lecture)

```
friend istream& operator>>(istream& is, Point& p) {
    is >> p.x >> p.y;
    return is;
}
```

Pour manipuler les fichiers, la bibliothèque <fstream> fournit trois types de flux :

Classe	Rôle
ifstream	lecture

ofstream	écriture
fstream	lecture + écriture

6. Ouverture de fichier

6.1. Écriture dans un fichier

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

int main() {
    ofstream fichier("donnees.txt");
    if (!fichier) {
        cerr << "Erreur d'ouverture !" << endl;
        return 1;
    }
    fichier << "Bonjour C++ !" << endl;
}
```

6.2. Lecture d'un fichier

```cpp
ifstream fichier("donnees.txt");
string ligne;

while (getline(fichier, ligne)) {
    cout << ligne << endl;
}
```

7. Modes d'ouverture

Exemples :

```cpp
ofstream("data.txt", ios::app);    // ajout (append)
ofstream("data.txt", ios::binary); // binaire
ofstream("data.txt", ios::trunc); // écrase le fichier
```

Principaux flags :

Mode	Signification
ios::in	lecture
ios::out	écriture
ios::app	ajouter à la fin
ios::binary	binaire
ios::trunc	écrase le fichier
ios::ate	position en fin à l'ouverture

8. Lecture / écriture binaire

Utilisé pour stocker des structures ou objets.

```
struct Data {
    int a;
    double b;
};

Data d{5, 3.14};

ofstream f("bin.dat", ios::binary);
f.write((char*)&d, sizeof(Data));
```

Lecture:

```
Data d2;
ifstream f("bin.dat", ios::binary);
f.read((char*)&d2, sizeof(Data));
```

9. Flux et buffers

Les flux C++ utilisent un buffer interne. Le vidage du buffer (flush) se fait :

- Automatiquement avec endl
- En fin de programme
- Manuellement avec :cout << flush;