

Ce document reprend la théorie vue sur les flux d'E/S ainsi que les E/S dans les fichiers un peu dans le détail (à défaut de prise de notes correcte ;)

Pour le réaliser, j'ai parcouru mes notes prises au cours et je me suis inspiré d'un site internet dont le plan Tutoriel C++ suivait à quelques petites choses près le plan de Madame Buseyne.

Voici l'adresse du site :

<http://membres.lycos.fr/dancel/cplusplus/cours/flux40.html>

Après lecture de ce document, n'oubliez pas de vous ruer sur les exercices vus au cours...vous devriez les résoudre facilement ;)

BOURVIL (Delplanque Yoann)
2° INFO

Les flux d'entrée/sortie

PLAN

- Généralités sur les flux
- Les classes utilisées pour les flux
- La classe ostream
- La classe istream
- La classe iostream
- Formatage de l'information
- Les manipulateurs
- Contrôle de l'état d'un flux
- Entrées et sorties dans un fichier
- La classe istrstream
- La classe ostrstream

Généralités sur les flux

Les entrées et sorties se font par l'intermédiaire de flux

Un flux est un canal:

- Recevant l'information (sortie)
- Fournissant l'information (entrée)

Il existe 3 flux:

- cout: sortie standard
- cin: entrée standard
- cerr: sortie standard d'erreur
- clog: sortie standard d'erreur avec tampon

Les classes utilisées pour les flux

Les classes déclarées dans iostream.h permettent la manipulation de périphériques standards

Les classes déclarées dans fstream.h permettent la manipulation des fichiers disques

Les classes déclarées dans strstream.h permettent de simuler des opérations d'entrée/sortie avec des tampons en mémoire centrale.

La classe OSTREAM

Classe dérivée de ios pour les flux de sortie

Fournit des sorties formatées ou non

Surdéfinition de l'opérateur << (ostream& operator<<(expression))

Méthodes de ostream:

- **ostream & put(char c);** : insère un caractère dans le flot

Exemple :

```
cout.put ('\n');
```

- **ostream & write(const char *, int n);** : insère *n* caractères dans le flot

Exemple :

```
cout.write("Bonjour", 7);
```

- **streampos tellp();** : retourne la position courante dans le flot
- **ostream & seekp(streampos n);** : se positionne à *n* octet(s) par rapport au début du flux. Les positions dans un flot commencent à 0 et le type *streampos* correspond à la position d'un caractère dans le fichier.
- **ostream & seekp(streamoff dep, seek_dir dir);** : se positionne à *dep* octet(s) par rapport :
 - au début du flot : *dir = beg*
 - à la position courante : *dir = cur*
 - à la fin du flot : *dir = end* (et *dep* est négatif!)

Exemple :

```
streampos old_pos = fout.tellp(); // mémorise la position
fout.seekp(0, end); // se positionne à la fin du flux
cout << "Taille du fichier : " << fout.tellp() << " octet(s)\n";
fout.seekp(old_pos, beg); // se repositionne comme au départ
```

- **ostream & flush();** : vide les tampons du flux

La classe ISTREAM

Classe dérivée de ios pour le flux en entrée

Fournit des entrées formatées ou non

Surdéfinition de l'opérateur >> (*istream*& operator>>(expression))

Méthodes de ISTREAM

- lecture d'un caractère :
 - **int get();** : retourne la valeur du caractère lu (ou EOF si la fin du fichier est atteinte)
 - **istream & get(char &c);** : extrait le premier caractère du flux (même si c'est un espace) et le place dans *c*.
 - **int peek();** : lecture non destructrice du caractère suivant dans le flux. Retourne le code du caractère ou EOF si la fin du fichier est atteinte.
- lecture d'une chaîne de caractères :
 - **istream & get(char *ch, int n, char delim='\n');** : extrait *n - 1* caractères du flux et les place à l'adresse *ch*. La lecture s'arrête au délimiteur qui est par défaut le '\n' ou la fin de fichier.

Le délimiteur ('\n' par défaut) n'est pas extrait du flux.
 - **istream & getline(char *ch, int n, char delim='\n');** : comme la méthode précédente sauf que le délimiteur est extrait du flux mais n'est pas recopié dans le tampon.
- **istream & read(char *ch, int n);** : extrait un bloc d' au plus *n* octets du flux et les place à l'adresse *ch*. Le nombre d'octets effectivement lus peut être obtenu par la méthode **gcount()**.
- **int gcount();** : retourne le nombre de caractères non formatés extraits lors de la dernière lecture
- **streampos tellg();** : retourne la position courante dans le flot
- **istream & seekg(streampos n);** : se positionne à *n* octet(s) par rapport au début du flux. Les positions dans un flot commencent à 0 et le type *streampos* correspond à la position d'un caractère dans le fichier.
- **istream & seekg(streamoff dep, seek_dir dir);** : se positionne à *dep* octet(s) par rapport :
 - au début du flot : *dir = beg*
 - à la position courante : *dir = cur*
 - à la fin du flot : *dir = end* (et *dep* est négatif!)
- **istream & flush();** : vide les tampons du flux

La classe IOSTREAM

- Elle est utilisée lorsqu'on souhaite réaliser à la fois des lectures et des écritures
- Elle hérite simplement de ostream et de istream

Formatage de l'information

L'indicateur de format du flot, est un entier long défini (en protected) dans la classe *ios*, dont les classes stream héritent.

Extrait de *ios* :

```
class ios {
public :
// ...
enum {
    skipws,      // ignore les espaces en entrée
    left,        // justifie les sorties à gauche
    right,       // justifie les sorties à droite
    internal,    // remplissage après le signe ou la base
    dec,         // conversion en décimal
    oct,         // conversion en octal
    hex,         // conversion en hexadécimal
    showbase,    // affiche l'indicateur de la base
    showpoint,   // affiche le point décimal avec les réels
    uppercase,  // affiche en majuscules les nombres hexadécimaux
    showpos,     // affiche le signe + devant les nombres positifs
    scientific,  // notation 1.234000E2 avec les réels
    fixed,       // notation 123.4 avec les réels
    unitbuf,     // vide les flots après une insertion
    stdio        // permet d'utiliser stdout et cout
};
//...
protected :
    long x_flags; // indicateur de format
//...
};
```

Ces valeurs peuvent se combiner avec l'opérateur | comme dans l'exemple :

```
ios::showbase | ios::showpoint | ios::showpos
```

Des constantes sont aussi définies dans la classe *ios* pour accéder à un groupe d'indicateurs :

- **static const long basefield;** : permet de choisir la base (*dec*, *oct* ou *hex*)
- **static const long adjustfield;** permet de choisir son alignement (*left*, *right* ou *internal*)
- **static const long floatfield;** permet de choisir sa notation pour les réels (*scientific* ou *fixed*)

Les méthodes suivantes (définies dans la classe *ios*) permettent de lire ou de modifier la valeur des indicateurs de format :

- **long flags();** : retourne la valeur de l'indicateur de format
- **long flags(long f);** : modifie l'ensemble des indicateurs en concordance avec la valeur de *f*. Elle retourne l'ancienne valeur de l'indicateur.
- **long setf(long setbits, long field);** : remet à zéro les bits correspondants à *field* (*basefield*, *adjustfield* ou *floatfield*) et positionne ceux désignés par *setbits*. Elle retourne l'ancienne valeur de l'indicateur de format.

Exemples :

```
cout << setf(ios::dec, ios::basefield) << i;
cout << setf(ios::left, ios::adjustfield) << hex << 0xFF;
// affiche : 000xFF
cout << setf(ios::internal, ios::adjustfield) << hex << 0xFF;
//affiche : 0x00FF
cout << setf(ios::scientific, ios::floatfield) << f;
long old = cout.setf(ios::left, ios::adjustfield);
cout << data;
cout.setf(old, ios::adjustfield);
cout.setf(0L, ios::basefield); // état par défaut de basefield
```

- **long setf(long f);** : positionne l'indicateur de format. Elle retourne l'ancienne valeur de l'indicateur de format.

```
cout.setf( ios::skipws );
cout.setf( ios::dec | ios::right );
```

- **long unsetf(long);** : efface les indicateurs précisés.

Méthodes de la classe *ios* manipulant le format des informations :

- **int width(int);** : positionne la largeur du champ de sortie.
- **int width();** : retourne la largeur du champ de sortie.
- **char fill(char);** : positionne le caractère de remplissage
- **char fill();** : retourne la valeur du caractère de remplissage
- **int precision(int);** : positionne le nombre de caractères (non compris le point décimal) qu'occupe un réel.
- **int precision();** : retourne la précision (voir ci-dessus).

Exemples :

```
cout << "Largeur par défaut du champ de sortie : ";
cout << cout.width() << endl;
// affiche: 0
cout.width(10);
cout.fill('*');
cout << '|' << 1234 << "|\n";
// affiche: |*****1234|
cout.setprecision(6);
cout << 12.345678 << endl;
// affiche: 12.3457 (noter l'arrondi à l'affichage)
```

Les manipulateurs

Ils permettent d'écrire un code plus compact et plus lisible qu'avec les méthodes de formatage. Le fichier **iomnip.h** définit un certain nombre de manipulateurs et offre la possibilité de créer ses propres manipulateurs.

Manipulateurs prédéfinis:

```
dec, hex, oct, endl, ends, flush, ws, setbase(int b), setfill(int c), setprecision(int p),  
setw(int l), setiosflags(long n), resetiosflags(long b)
```

L'usage des méthodes de formatage de l'information rend les instructions lourdes à écrire et un peu trop longues. Les manipulateurs permettent d'écrire un code plus compact et plus lisible.

Ainsi, au lieu d'écrire :

```
cout.width(10);  
cout.fill('*');  
cout.setf(ios::hex, ios::basefield);  
cout << 123 ;  
cout.flush();
```

on préférera écrire l'instruction équivalente :

```
cout << setw(10) << setfill('*') << hex << 123 << flush;
```

dans laquelle, *setw* et *setfill* sont des manipulateurs avec un argument alors que *hex* et *flush* sont des manipulateurs sans argument. De plus nous pourrions écrire nos propres manipulateurs.

Les classes *ios*, *istream* et *ostream* implémentent les manipulateurs prédéfinis. Le fichier d'entête **iomnip.h** définit un certain nombre de ces manipulateurs et offre la possibilité de créer ses propres manipulateurs.

- **dec** : la prochaine E/S utilise la base décimale
- **hex** : la prochaine E/S utilise la base hexadécimale
- **oct** : la prochaine E/S utilise la base octale
- **endl** : écrit un '\n' puis vide le flot
- **ends** : écrit un '\0' puis vide le flot
- **flush** : vide le flot
- **ws** : saute les espaces (sur un flot en entrée uniquement)
- **setbase(int b)** : positionne la base *b* pour la prochaine sortie. *n* vaut 0 pour le décimal, 8 pour l'octal et 16 pour l'hexadécimal.
- **setfill(int c)** : positionne le caractère de remplissage *c* pour la prochaine E/S
- **setprecision(int p)** : positionne la précision à *p* chiffres pour la prochaine E/S
- **setw(int l)** : positionne la largeur à *n* caractères.
- **setiosflags(long n)** : active les bits de l'indicateur de format spécifiés par l'entier *n*. On l'utilise comme la méthode *flags()*.
- **resetiosflags(long b)** : désactive les bits de l'indicateur de format.

Contrôle de l'état d'un flux

La classe `ios` est une classe de base virtuelle, elle est utilisée pour tester l'état d'un flux ou pour contrôler le formatage des informations

Méthodes de la classe de base `ios` :

- **`int good()`**; : retourne une valeur différente de zéro si la dernière opération d'entrée/sortie s'est effectuée avec succès et une valeur nulle en cas d'échec.
- **`int fail()`**; : fait l'inverse de la méthode précédente
- **`int eof()`**; : retourne une valeur différente de zéro si la fin de fichier est atteinte et une valeur nulle dans le cas contraire.
- **`int bad()`**; : retourne une valeur différente de zéro si vous avez tenté une opération interdite et une valeur nulle dans le cas contraire.
- **`int rdstate()`**; : retourne la valeur de la variable d'état du flux. Retourne une valeur nulle si tout va bien.
- **`void clear()`**; : remet à zéro l'indicateur d'erreur du flux. C'est une opération obligatoirement à faire après qu'une erreur se soit produite sur un flux.
- Surdéfinition des opérateurs `()` et `!` :
tester un flot en le considérant comme une valeur logique (vrai ou faux)

Exemples :

```
if ( f1 ) ... // équivalent à : if ( f1.good() )
if ( !f1 ) ... // équivalent à : if ( !f1.good() )
if ( ! (cin >> x) ) ...
```

Entrées et sorties dans un fichier

Plan

- Associer un flux d'E/S à un fichier
- Déclaration des fichiers
- Ouverture des fichiers
- Lecture et écriture des fichiers
- Fermeture des fichiers
- Vérification des erreurs d'opérations sur les fichiers
- Autres méthodes d'entrée/sortie

Associer un flux d'E/S à un fichier

- 3 classes permettent de manipuler des fichiers sur un disque (elles sont définies dans `fstream.h`)
- `ifstream` permet l'accès à un flux en lecture uniquement
- `ofstream` permet l'accès du flux en écriture seulement
- `fstream` permet l'accès du flux en lecture/écriture

Déclaration des fichiers

Fichier d'entrée :

```
ifstream entree;
```

Fichier de sortie:

```
ofstream sortie;
```

Fichier d'entrée/sortie:

```
fstream entre_sortie;
```

Ouverture de fichiers

Utiliser le constructeur de `ofstream`, `ifstream` ou `fstream` avec le nom du fichier en paramètre

```
ifstream(char* nom, int mode) ;
```

```
exemple :   ifstream i ("test.txt",ios ::in) ;      entrée
            ofstream o ("test.txt",ios ::out) :     sortie
            fstream f ("test.txt",ios::in|ios::out); entrée/sortie
```

Utiliser la fonction `open(char *nom,int mode)`

```
Exemple :   ifstream i ;
            i.open(« test.txt »,ios::in);
```

Modes d'ouverture des fichiers:

```
app      // ajout des données en fin de fichier.
ate     // positionnement à la fin du fichier.
in      // ouverture en lecture (par défaut pour ifstream).
Out     // ouverture en écriture (par défaut pour ofstream).
Binary  // ouverture en mode binaire (par défaut en mode texte).
trunc   // détruit le fichier s'il existe et le recrée
          // (par défaut si out est spécifié sans que ate ou app ne
          // soit activé).
Nocreate // si le fichier n'existe pas, l'ouverture échoue.
noreplace // si le fichier existe, l'ouverture échoue,
           // sauf si ate ou app sont activés.
```

!!! Pour les classes *ifstream* et *ofstream* le mode par défaut est respectivement **ios::in** et **ios::out**. !!!

Exemple :

```
#include <fstream.h>

ifstream f1;
f1.open("essai1.tmp"); // ouverture en lecture du fichier
ofstream f2;
f2.open("essai2.tmp"); // ouverture en écriture du fichier
fstream f3;
f3.open("essai3.tmp", ios::in | ios::out); // ouverture en lecture/écriture
```

On peut aussi appeler les constructeurs des différentes classes et combiner les 2 opérations de définition du flot et d'ouverture.

```
#include <fstream.h>

ifstream f1("essai1.tmp"); // ouverture en lecture du fichier
ofstream f2("essai2.tmp"); // ouverture en écriture du fichier
fstream f3("essai3.tmp", ios::in | ios::out); // ouverture en
lecture/écriture
```

Lecture et écriture des fichiers

Les opérations d'entrée se font comme du clavier à l'aide de l'opérateur >>
o<<< test >><<endl ;

Les opérations de sortie se font comme à l'écran à l'aide de l'opérateur <<
i>>c ;

Avec des méthodes:

- En lecture: `char get()`, `read(char* , int)`, ...
- En écriture: `put(char)`, `write(char* , int)`, ...

Fermeture des fichiers

Pour fermer un fichier, il faut utiliser la méthode `close()` ;

Exemples :

```
i.close() ;  
o.close() ;  
f.close() ;
```

Vérifications des erreurs d'opération sur les fichiers

Des méthodes permettent de vérifier les états d'un flux d'entrée/sortie:

- `good()`, `eof()`, `fail()`, `bad()`

Des constantes existent et permettent de décrire les différents états dans lequel un flux d'entrée/sortie peut se trouver:

- `goodbit`
- `eofbit`
- `failbit`
- `badbit`

Autres méthodes d'entrée/sortie

(voir `OSTREAM` et `ISTREAM`)

- `ifstream & seekg(streampos n)`;
- `ifstream & seekg(streampos n, seek_dir dir)`;
- `streampos tellg()`;
- `ofstream & seekp(streampos n)`;
- `ofstream & seekp(streamoff dep, seek_dir dir)`;
- `streampos tellp()`;
- ...

La classe `OSTRSTREAM`

Classe dérivée de `ostream` permettant l'écriture dans un tampon mémoire (à la manière de `sprintf`)

La classe `ISTRSTREAM`

Classe dérivée de `istream` permettant la lecture dans un tampon mémoire (à la manière de la fonction `sscanf`)