LES FLUX

Généralités sur les flux

Un *flux* (ou *stream*) est une abstraction logicielle représentant un flot de données entre :

- une source produisant de l'information
- une cible consommant cette information.

Il peut être représenté comme un buffer et des mécanismes associés à celui−ci et il prend en charge, quand le flux est créé, l'acheminement de ces données.

Comme pour le langage C, les instructions entrées/sorties ne font pas partie des instructions du langage. Elles sont dans une librairie standardisée qui implémente les flots à partir de classes.

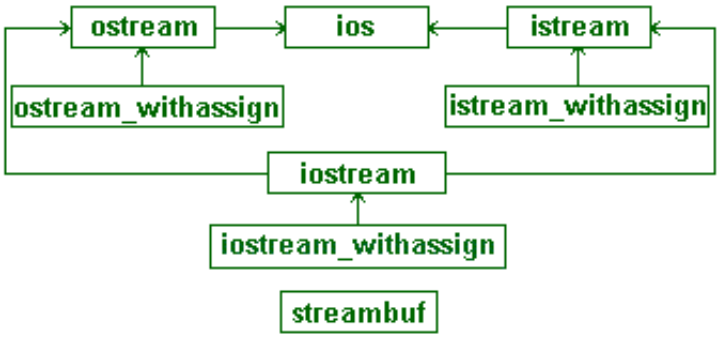
Par défaut, chaque programme C++ peut utiliser 3 flots :

- *cout* qui correspond à la sortie standard
- *cin* qui correspond à l'entrée standard
- *cerr* qui correspond à la sortie standard d'erreur.

Pour utiliser d'autres flots, vous devez donc créer et attacher ces flots à des fichiers (fichiers normaux ou fichiers spéciaux) ou à des tableaux de caractères.

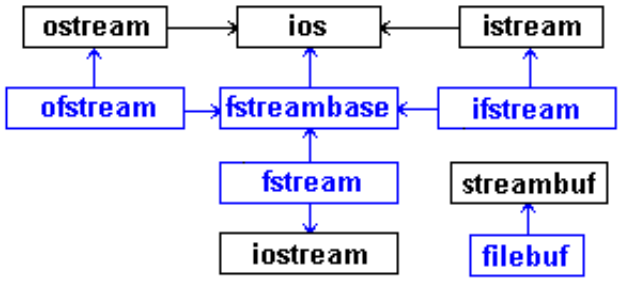
Flots et classes

- Classes déclarées dans ***iostream.h*** et permettant la manipulation des périphériques standards :



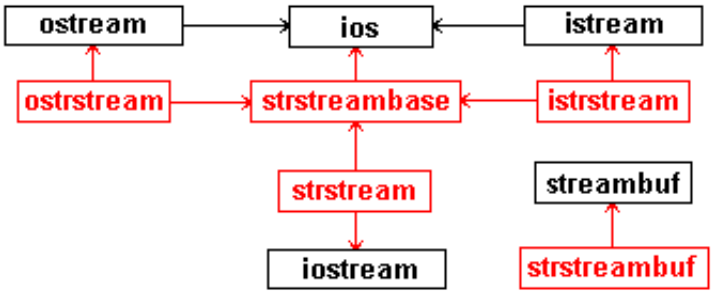
  - **ios** : classe de base des entrées/sorties par flot. Elle contient un objet de la classe *streambuf* pour la gestion des tampons d'entrées/sorties.
  - **istream** : classe dérivée de *ios* pour les flots en entrée.
  - **ostream** : classe dérivée de *ios* pour les flots en sortie.
  - **iostream** : classe dérivée de *istream* et de *ostream* pour les flots bidirectionnels.
  - **istream_withassign**, **ostream_withassign** et **iostream_withassign** : classes dérivées respectivement de *istream*, *ostream* et *iostream* et qui ajoute l'opérateur d'affectation. Les flots standards *cin*, *cout* et *cerr* sont des instances de ces classes.

- Classes déclarées dans ***fstream.h*** permettant la manipulation des fichiers disques :



- **fstreambase** : classe de base pour les classes dérivées *ifstream*, *ofstream* et *fstream*. Elle même est dérivée de *ios* et contient un objet de la classe *filebuf* (dérivée de *streambuf*).
- **ifstream** : classe permettant d'effectuer des entrées à partir des fichiers.
- **ofstream** : classe permettant d'effectuer des sorties sur des fichiers.
- **fstream** : classe permettant d'effectuer des entrées/sorties à partir des fichiers.

- Classes déclarées dans ***strstream.h*** permettant de simuler des opérations d'entrées/sorties avec des tampons en mémoire centrale. Elles opèrent de la même façon que les fonctions du langage C *sprintf()* et *sscanf()* :



- **strtreambase** : classe de base pour les classes suivantes. Elle contient un objet de la classe *strstreambuf* (dérivée de *streambuf*).
- **istrstream** : classe dérivée de *strtreambase* et de *istream* permettant la lecture dans un tampon mémoire (à la manière de la fonction *sscanf*).
- **ostrstream** : classe dérivée de *strtreambase* et de *ostream* permettant l'écriture dans un tampon mémoire (à la manière de la fonction *sprintf*).
- **strstream** : classe dérivée de *istrstream* et de *iostream* permettant la lecture et l'écriture dans un tampon mémoire.

Le flot de sortie : *ostream*

- Il fournit des sorties formatées et non formatées (dans un *streambuf*)
- Surdéfinition de l'opérateur <<
- Il gère les types prédéfinis du langage C++
- On peut (doit) le surdéfinir pour ses propres types

Surdéfinition de l'opérateur d'injection dans un flot :

Exemple :

```
class Exemple {
    friend ostream &operator<<(ostream &os, const Exemple &ex);
  public :
    // ...
  private :
    char _nom[20];
    int  _valeur;
};

ostream &operator<<(ostream &os, const Exemple &ex) {
  return os << ex._nom << " " << ex._valeur;
}

void main() {
  Exemple e1;

  // ...
  cout << "Exemple = " << e1 << endl;
  // ...
}
```

La fonction **operator<<** doit retourner une référence sur l'*ostream* pour lequel il a été appelé afin de pouvoir le cascader dans une même instruction, comme dans la fonction *main()* ci−dessus.

Principales méthodes de *ostream*

En plus de l'opérateur d'injection (<<), la classe *ostream* contient les principales méthodes suivantes :

- **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

Le flot d'entrée : *istream*

- Il fournit des entrées formatées et non formatées (dans un *streambuf*)

- Surdéfinition de l'opérateur >>

- Il gère les types prédéfinis du langage C++

- On peut (doit) le surdéfinir pour ses propres types

- Par défaut >> ignore tous les espaces (voir *ios::skipws*)

Surdéfinition de l'opérateur d'extraction >> :

Exemple :

```
class Exemple {
    friend ostream &operator>>(ostream &os, Exemple &ex);
  public :
    // ...
  private :
    char _nom[20];
    int  _valeur;
};

ostream &operator>>(ostream &is, Exemple &ex) {
  return is >> ex._nom >> ex._valeur;
}

void main() {
  Exemple e1;

  // ...
  cout << "Entrez un nom et une valeur : ";
  cin >> e1;
  // ...
}
```

La fonction **operator>>** doit retourner une référence sur l'*istream* pour lequel il a été appelé afin de pouvoir le cascader dans une même instruction.

Principales méthodes de *istream*

En plus de l'opérateur d'extraction (>>), la classe *istream* contient les principales méthodes suivantes :

- 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

Contrôle de l'état d'un flux

La classe *ios* décrit les aspects communs des flots d'entrée et de sortie. C'est une classe de base virtuelle pour tous les objets flots. Vous n'aurez jamais à créer des objets de la classe *ios*. Vous utiliserez ses méthodes pour tester l'état d'un flot 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 de **()** et **!** :

  Exemples :

```
if ( f1 )  ...  // équivalent à : if ( f1.good() )

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

Associer un flot d'E/S à un fichier

Il est possible de créer un objet flot associé à un fichier autre que les fichiers d'entrées/sorties standards.

3 classes permettant de manipuler des fichiers sur disque sont définies dans ***fstream.h*** :

1. **ifstream** : (dérivée de *istream* et de *fstreambase*) permet l'accès du flux en lecture seulement

2. **ofstream** : (dérivée de *ostream* et de *fstreambase*) permet l'accès du flux en écriture seulement

3. **fstream** : (dérivée de *iostream* et de *fstreambase*) permet l'accès du flux en lecture/écriture.

Chacune de ces classes utilise un buffer de la classe *filebuf* pour synchroniser les opérations entre le buffer et le flot.

La classe *fstreambase* offre un lot de méthodes communes à ces classes (*open*, *close*, *attach* ...).

Ouverture du fichier et association avec le flux :

C'est la méthode *open()* qui permet d'ouvrir le fichier et d'associer un flux avec ce

dernier.

```
void open(const char *name, int mode,int prot=filebuf::openprot);
```

- *name* : nom du fichier à ouvrir
- *mode* : mode d'ouverture du fichier (*enum open_mode* de la classe *ios*) :

```
enum open_mode {
   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**.

- *prot* : il définit les droits d'accès au fichier (par défaut les permissions de lecture/écriture sont positionnés) dans le cas d'une ouverture avec création (sous UNIX).

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
```

Formatage de l'information

Chaque flot conserve en permanence un ensemble d'indicateurs spécifiant l'état de formatage. Ceci permet de donner un comportement par défaut au flot, contrairement aux fonctions *printf()* et *scanf()* du langage C, dans lesquelles on fournissait pour chaque opération d'entrée/sortie l'indicateur de format.

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.

  Exemples :

  ```
  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

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 préfèrera 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 pourrons écrire nos propres manipulateurs.

Comment cela marche−t il ?

Les manipulateurs les plus célèbres sont **endl** et **flush**. Vous les connaissez certainement. Voici comment ils sont définis :

```
ostream &flush(ostream &ps) { return os.flush(); }

ostream &endl(ostream &os) { return os << '\n' << flush; }
```

Pour pouvoir les utiliser dans une instruction comme :

```
cout << end;
```

la fonction *operator*<< est surchargée dans la classe *ostream* comme :

```
ostream &ostream::operator<<(ostream& (* f)(ostream &)) {
```

```
    (*f)(*this);
    return *this;
  }
```

Il est donc très simple de définir son propre manipulateur (sans paramètre) :

Exemple : création du manipulateur *tab* :

```
ostream &tab(ostream &os){ return os << '\t'; }
```

que l'on peut utiliser ainsi :

```
cout << 12 << tab << 34;
```

Manipulateurs prédéfinis :

Les classes *ios*, *istream* et *ostream* implémentent les manipulateurs prédéfinis. Le fichier d'entête **iomanip.h** défini 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.

Création d'un nouveau manipulateur (avec un paramètre) :

L'implémentation d'un nouveau manipulateur est implanté en 2 parties :

- le **manipulateur** : sa forme générale est (pour un ostream) :

```
ostream &nom_du_manipulateur(ostream &, type );
```

type est le type du paramètre du manipulateur.

Cette fonction ne peut pas être appelée directement par une instruction d'entrée/sortie. Elle sera appelée seulement par l'applicateur.

- l'**applicateur** : il appelle le manipulateur. C'est une fonction globale. Sa forme générale est :

```
xxxMANIP( type ) nom_du_manipulateur(type arg) {
  return xxxMANIP(type) (nom_du_manipulateur, arg);
}
```

avec *xxx* valant **O** pour les flots manipulant un ostream (ou ses dérivées), **I**, **S** et **IO** pour respectivement les flots manipulant un *istream*, *ios* et un *iostream*.

Exemple : un manipulateur pour afficher un entier en binaire :

```
#include <iomanip.h>
#include <limits.h>  // ULONG_MAX

ostream &bin(ostream &os, long val) {
  unsigned long masque = ~(ULONG_MAX >> 1);
  while ( masque ) {
    os << ((val & masque) ? '1' : '0');
    masque >>= 1;
  }
  return os;
}

OMANIP(long) bin(long val) {
  return OMANIP(long) (bin, val);
}

void main() {
  cout << "1997 en binaire = " << bin(1997) << endl;
}
```
