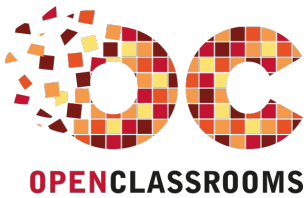
[C++] Les manipulateurs de flux

Par Matthieu Schaller (Nanoc)



www.openclassrooms.com

Dernière mise à jour le 4/07/2011

Sommaire

<< [C++] Les manipulateurs de flux

Par  Matthieu Schaller (Nanoc)

Mise à jour : 04/07/2011
Difficulté : Intermédiaire Durée d'étude : 2 heures

Bonjour tout le monde !

La première chose que l'on apprend à faire en programmation c'est à utiliser la console pour afficher du texte à l'intérieur. Puis petit à petit, on apprend aussi à y lire des informations et à écrire dans des fichiers.

Lors de toutes ces opérations, on aimerait souvent ajouter un peu d'esthétisme afin de rendre l'affichage plus joli ou mieux structuré.

Dans ce tutoriel, je vais vous apprendre à utiliser les manipulateurs de flux qui permettent facilement d'améliorer la lisibilité de vos sorties en console ou permettent par exemple de modifier le format de vos sorties dans les fichiers.

Pour comprendre ce tutoriel, il vaut mieux avoir lu la première partie du cours de M@teo21 sur le C++.

Sommaire du tutoriel :

Petit rappel sur les flux

En C++, il existe plusieurs manières pour un programme d'interagir avec les éléments extérieurs que sont, la console, les fichiers, les périphériques, etc. Un des moyens particulièrement puissant d'interaction est le **flux** ou **flot**. Il existe des flux dans 2 directions, du programme vers l'extérieur et évidemment dans le sens contraire, ce qui permet ainsi d'influencer le déroulement d'un programme.
La bibliothèque standard du C++ propose 3 types différents de flux ayant chacun 2 directions. Ces flux sont des instances de classes dont les noms sont présentés dans le tableau suivant :

Type de flux	Flux entrant	Flux sortant	Fichier à inclure
Entrée/Sortie standard (console)	std::istream	std::ostream	iostream
Flux sur les fichiers	std::ifstream	std::ofstream	fstream
Flux sur les chaînes de caractères	std::istringstream	std::ostringstream	sstream

Le dernier type de flux n'est pas fait pour interagir avec le monde extérieur mais plutôt pour utiliser la puissance des flux pour gérer plus facilement les chaînes de caractères.

Il est possible de créer autant de flux que l'on désire sur les chaînes de caractères ou sur les fichiers, en créant un objet des classes ci-dessus. Par contre il n'existe qu'un nombre restreint de flux vers la console disponibles de base. Ils sont au nombre de 4 :

Type de flux	Nom	Description
Entrant	std::cin	Flux standard d'entrée
Sortant	std::cout	Flux standard de sortie
Sortant	std::cerr	Flux standard pour les erreurs (non-bufferisé)
Sortant	std::clog	Flux standard pour les erreurs (bufferisé)

Les 2 derniers flux sont utilisés pour afficher des messages d'erreurs. **std::cerr** ne possède pas de buffer, cela veut dire qu'il affiche directement le contenu qui lui est passé sans attendre un appel à **std::flush**.

En réalité, l'ordinateur n'écrit pas directement dans la console ou dans un fichier lorsqu'on met quelque chose dans le flux. Il stocke ce qu'on lui passe dans une mémoire tampon et n'écrit réelement que lors d'un appel à **std::flush**, à **std::endl**, si la mémoire tampon est pleine ou si on est arrivé à la fin du programme. **std::cerr** ne possède pas de mémoire tampon, il écrit donc directement dans la console.

Tout cela est bien joli, mais comment les utilise-t-on ? C'est ce que je vais vous montrer tout de suite en basant mes exemples sur les flux standards **std::cin** et **std::cout**, puisque une fois le flux créé, l'utilisation est la même dans tous les cas. Comme vous l'aurez remarqué, on utilise tout le temps l'espace de nom **std**. A partir d'ici, je ne mettrais plus les **std::** devant les fonctions que j'utilise. Ceci afin d'alléger l'écriture du code.

Pour plus d'informations sur la manière d'utiliser les flux sur les fichiers, je vous invite à lire l'excellent tutoriel de Xav57 intitulé Lecture et écriture dans les fichiers en C++.

Les flux sortant

Si vous avez lu le tutoriel de M@teo21, vous savez certainement que l'on utilise principalement un flux via l'opérateur << qui permet d'envoyer différents types de données dans le flux.

Code : C++ - Utilisation d'un flux de sortie

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

cout << "Salut";

int entier=2;
cout << entier;

char lettre='b';
cout << lettre;

string phrase = "Bonjour";
cout << phrase;
//...
```

On peut donc via l'opérateur <<, envoyer des variables de nimporte quel type de base ainsi que les strings à un flux. Un des défauts de cette manière d'utiliser le flux est que les données sont affichées à la suite, sans aucune possibilité d'influencer la position des caractères ou le format des nombres affichés. Pour le faire, il faut utiliser les manipulateurs de flux que je vous présente dans la suite.

Si vous désirez forcer l'affichage dans la console, vous pouvez faire appel à **endl** ou **flush**. Le premier ajoutant en plus un retour à la ligne.

Code : C++ - endl et flush

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

int main()
{
   cout << "Bonjour !" << flush << "Comment allez-vous ?" << endl;
   cout << "Bien et vous ?" << endl;
   //...
}
```

produit l'affichage suivant :

Code : Console

```
Bonjour !Comment allez-vous ?
Bien et vous ?
```

Les flux entrant

Les flux entrant sont tout aussi simples à utiliser, il suffit de faire appel à l'opérateur >> pour lire les données et les stocker dans une variable.

Code : C++ - Utilisation d'un flux entrant

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

cout << "Entrez un nombre: ";
int nombre;
cin >> nombre;  //nombre prend la valeur entrée par l'utilisateur

//...
```

C'est donc extrêmement simple. Il y a juste un petit problème, si l'on essaye de lire une phrase complète contenant des espaces. En utilisant l'opérateur >> sur un flux entrant, la lecture s'arrête au premier "blanc". Pour lire une phrase, il faut alors passer par la fonction **getline()**.

Code : C++ - Utilisation de getline()

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

cout << "Entrez une phrase: ";
string phrase;

getline(cin,phrase);  //Copie toute la ligne de ce qui se trouve
dans le flux dans la
                      //chaîne phrase.

//...
```

Il est possible que la chose lue par le flux ne soit pas compatible avec la variable dans laquelle vous désirez la stocker. Vous aurez alors droit à une erreur et le comportement du flux sera alors indéfini. Pour plus de détails, je vous invite à

lire ce tutoriel de ooprog.
Il faut également faire attention si vous mélangez les utilisations de >> et **getline()**, le symbole de fin de fichier peut subsister dans le flux. Je vous invite à suivre ce lien pour y remédier.

Voilà, c'est tout pour ce petit rappel sur les flux. Il est temps d'entrer dans le vif du sujet. Dans le but de simplifier les exemples, je vais utiliser le flux de sortie standard `std::cout` , mais il est possible d'utiliser les manipulateurs sur **n'importe quel autre flux** (`std::ofstream` , `std::ostringstream` , `std::cerr` , ...) . C'est ce qui sera fait dans l'exercice final.

Les manipulateurs basiques

Syntaxe des manipulateurs

Un manipulateur de flux est une instruction permettant de modifier le comportement du flux dans certains cas, par exemple afficher ou non les nombres après la virgule ou réaliser un alignement en colonne. Il existe deux syntaxes équivalentes pour utiliser un manipulateur.

La première manière correspond à un appel d'une fonction :

nom_du_manipulateur(nom_du_flux);

La deuxième s'appuie sur la surcharge de l'opérateur << pour les flux :

nom_du_flux << nom_du_manipulateur;

Personellement je préfère la deuxième méthode, mais les deux sont tout à fait équivalentes. Bon c'est bien joli tout ça, mais je crois que vous attendez un exemple.

Mon premier manipulateur

Il arrive souvent que l'on ait à afficher des booléens dans la console pendant que l'on développe pour vérifier que tout s'est bien passé. Il serait plus agréable de pouvoir afficher **true** et **false**, plutôt que **1** et **0**. Pour cela, il existe le manipulateur **boolalpha**. Essayons pour voir :

Code : C++ - Utilisation de boolalpha

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

int main()
{
    cout << true << " et " << false << endl;  //Affichage "normal"
    cout << boolalpha;                        //On applique le
manipulateur
    cout << true << " et " << false << endl;  //Affichage "modifié"

    return 0;
}
```

On aurait aussi pu écrire boolalpha(cout); à la ligne 7. Le résultat aurait été le même.

Ce qui nous donne :

Code : Console - Manipulateur boolalpha

```
1 et 0
true et false
```

C'est simple non ? Et encore, celui là on aurait pu le faire nous-même à l'aide d'une macro. Vous n'avez pas encore vu la suite !

Et si j'en veux plus de cet affichage ?

Bonne question ! Il suffit d'utiliser le manipulateur opposé, **noboolalpha** qui est justement là pour ça.

Les manipulateurs sur les nombres entiers

Vous savez certainement qu'il existe différentes manières de représenter un même nombre. Tout dépend de la base choisie. Nous, les êtres humains, utilisons la base 10. L'ordinateur, lui, ne connait que la base 2. C'est pourquoi les informaticiens sont parfois amener à utiliser les bases 8 et 16, appelées respectivement octal et hexadécimal. Il pourrait donc être intéressant d'afficher un nombre dans une ou l'autre de ces bases.
Pour cela, il existe 3 manipulateurs :

Nom	Fonction
dec	Affiche les nombres en base 10.(Valeur par défaut)
oct	Affiche les nombres en base 8.
hex	Affiche les nombres en base 16.

Essayons donc pour voir :

Code : C++ - Les manipulateurs de changement de base

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

int main()
{
    cout << 33 << endl;         //Affichage "normal" en base 10
    cout << oct << 33 << endl; //Affichage en base 8
    cout << hex << 33 << endl; //Affichage en base 16
    cout << dec << 33 << endl; //Retour à la base 10

    return 0;
}
```

Ce qui donne comme prévu :

Code : Console - Dites 33 !

```
33
41
21
33
```

Les manipulateurs suivants, qui ont un effet sur les nombres réels ont aussi un effet sur les nombres entiers, ce qui n'est pas le cas des changements de base.

Les manipulateurs sur les nombres à virgule

A nouveau, il existe plusieurs manières d'afficher un nombre à virgule. On peut le faire en affichant tous les chiffres ou en utilisant la notation scientifique. Pour faire de même en C++, il faut utiliser les manipulateurs **scientific** et **fixed** pour forcer la représentation scientifique ou pour forcer la notation fixée.

Par défaut, le flux ne se trouve dans aucun des 2 cas. Si vous souhaitez remettre le flux dans son état original, il faut utiliser la fonction **unsetf(flag)** de la manière suivante: cout.unsetf(ios_base::scientific) ou cout.unsetf(ios_base::fixed), selon la modification que vous avez faites. Cette fonction n'est pas directement associée aux manipulateurs, mais c'est la seule manière de le faire.

Un petit exemple :

Code : C++ - Notation scientifique

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

int main()
{
    double nombre =  31223.565465;

    cout << nombre << endl;              //Affichage "normal"
    cout << fixed << nombre << endl;     //Affichage de tous les
chiffres
    cout << scientific << nombre << endl; //Affichage scientifique

    return 0;
}
```

Ce qui produit le résultat suivant :

Code : Console

```
31223.6
31223.565465
3.122357e+004
```

Il peut parfois être utile d'afficher le **.** qui sépare la partie entière et décimale même si le nombre ne possède pas de partie décimale. Cela se fait via le manipulateur **showpoint** et son opposé **noshowpoint**.

Il est également possible de forcer l'affichage du signe **+** pour les nombres positifs en utilisant le manipulateur **showpos** et son contraire **noshowpos**, pour revenir à l'état normal.

Si l'on souhaite modifier le nombre de chiffres après la virgule affichés, il faut passer par un nouveau manipulateur, **setprecision()**. Cependant, ce maniuplateur ne se trouve pas avec les autres. Pour une raison qui m'est inconnue, les créateurs de la bibliothèque standard du C++ ont décidé de mettre 6 manipulateurs dans un autre fichier, le fichier iomanip. Tous les manipulateurs dont le nom commence par **set...** sont définis dans ce fichier. C'est notre cas, il faut donc l'inclure.

L'utilisation de **setprecision()** est très simple, il faut lui passer en argument le nombre de chiffres que vous souhaitez afficher (au maximum 16).

Code : C++ - Le manipulateur setprecision()

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

int main()
{
```

```
        double pi = 3.141592653 ;

        cout << pi << endl;
        cout << setprecision(0) << pi << endl;
        cout << setprecision(10) << pi << endl;

        return 0;
    }
```

donnera :

Code : Console - Pi-Pi-Pi

```
3.14159
3
3.141592653
```

C'est tout pour les manipulateurs modifiant le comportement du flux lors de l'affichage de certains types. Passons à une partie plus intéressante où nous allons apprendre à formater la sortie.

Quelques manipulateurs plus avancés

Il est maintenant temps de voir quelques manipulateurs permettant de modifier l'affichage des données. Imaginons que nous ayons envie d'afficher un tableau. Il serait alors pratique d'afficher les éléments de ce tableau réelement en colonne quelque soit la longueur du contenu de la cellule.

Cela se fait en utilisant le manipulateur **setw(n)**. En utilisant ce manipulateur, on indique que la prochaine sortie se fera sur n caractères.

Ce manipulateur n'agit que sur la prochaine sortie. Cela veut uniquement sur ce qui est passé en argument au prochain << et pas pour toutes les fois suivantes.

Code : C++ - Le manipulateur setw()

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

int main()
{
cout << "Alignement normal" << endl;
cout << setw(10) << 1234 <<  endl;  //On aligne tout dans une "case"
de 10 caractères
cout << setw(10) << "un mot" << endl;    // de même

cout << "Sans le setw(), par contre on revient à la normale" <<
endl;
}
```

Ce qui donne le résultat suivant :

Code : Console

```
Alignement normal
      1234
    un mot
Sans le setw(), par contre on revient à la normale
```

Si la longueur du texte à mettre dans la "cellule" est plus grand que la longueur de cette dernière, le texte dépassera; il ne sera pas coupé.

Ceci peut donc être très pratique pour réaliser des tableaux de valeurs. On peut aussi décider d'aligner le texte à gauche de la "cellule" plutôt qu'à droite en utilisant le manipulateur **left** et son opposé **right** si on veut ré-aligner à droite.

Un petit exemple pour saisir toute la puissance de la chose :

Code : C++ - Création de tableaux

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

int main()
{
cout << "Un joli tableau" << endl << endl;
cout << setw(10) << left << "Pseudo: " << "|" << setw(8) << right <<
"Nanoc" << endl;
cout << setw(10) << left << "Langage: " << "|" << setw(8) << right
<< "C++" << endl;
}
```

Code : Console

```
Un joli tableau

Pseudo:    |   Nanoc
Langage:   |     C++
```

On a ainsi réussi à aligner du texte verticalement sans avoir besoin de compter manuellement des espaces ! Magique, non ?

Si le fait de mettre des espaces dans les "cellules" des tableaux ne vous plaît pas, vous pouvez changer le caractère de remplissage via le manipulateur **setfill(char)**, soit par exemple :

Code : C++ - Tableau revisité

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

int main()
{
cout << "Un joli tableau" << endl << endl;
cout << setfill('.');
cout << setw(10) << left << "Pseudo: " << "|" << setw(8) << right <<
"Nanoc" << endl;
cout << setw(10) << left << "Langage: " << "|" << setw(8) << right
<< "C++" << endl;
}
```

qui donne cette fois :

Code : Console

```
Un joli tableau

Pseudo: ..|...Nanoc
Langage: .|.....C++
```

J'en ai ainsi terminé avec la présentation des manipulateurs de flux les plus courants. Il en existe encore quelques autres qui sont plus rarement utilisés. Sachez également qu'**endl** et **flush** sont également à considérer comme des manipulateurs de flux, même si on ne vous l'a jamais dit avant .

Si cela vous intéresse d'avoir la liste complète, je vous invite à faire un tour par ici.

Exercice

Pour terminer, je vous propose un petit exercice pour que vous mettiez tout ceci en pratique.

Vous devez réaliser l'affichage suivant en utilisant un maximum de manipulateurs de flux :

Code : Console

```
+-----------------+-------+
| Julien          |  2.45 |
| Maurice         |  3.70 |
| Andre           |  4.00 |
| Luc             |  3.46 |
| Adrien          |  5.21 |
+-----------------+-------+
```

à partir du fichier "moyennes.txt" donné ici :

Code : Autre - moyennes.txt

```
Julien 2.45
Maurice 3.7
Andre 4
Luc 3.45667
Adrien 5.215
```

Les noms doivent être affichés dans une colonne de 15 caractères alignés à gauche et les chiffres dans une colonne de 5 caractères alignés à droite. Les moyennes doivent êtres affichées avec 2 chiffres après la virgule dans tous les cas.

Bonne chance à tous !

Vous avez terminé ? Vous êtes sûr ? Bon d'accord. Voici le corrigé, je vous laisse comparer et corriger si nécessaire.

Code : C++ - Solution de l'exercice

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

using namespace std;

int main()
{
    ifstream fichier("moyennes.txt");  //On crée un flux en lecture
```

```
sur le fichier

    //On affiche le cadre du tableau
    cout << "+" << setfill('-') << setw(18) << "+";  //On veut 15
caracteres + 2 pour pas
    cout << setfill('-') << setw(8) << "+" << endl;  //coller au
bord + 1 pour le '+'
                                                     //donc 18 au
total idem pour la deuxieme colonne
    cout << setfill(' ');

    string nom;       //Une chaine pour le nom de l'eleve
    double moyenne;   //Une variable pour sa moyenne

    cout << fixed << setprecision(2);  //On force l'affichage de
deux chiffres apres la virgule

    while(fichier) //Tant qu'on est pas à la fin du fichier
    {
        fichier >> nom >> moyenne;   //On recupere les valeurs

        if(fichier) //Si on a bien pu lire quelque chose
        {
            cout << "| " << left << setw(15) << nom;   //On affiche
le nom
            cout << " | " << right << setw(5) << moyenne << " |" <<
endl;  //et la moyenne
        }
    }

    //On affiche le bas du tableau
    cout << "+" << setfill('-') << setw(18) << "+";
    cout << setfill('-') << setw(8) << "+" << endl;

    return 0;
}
```

Comme je vous l'ai dit au début de ce tutoriel, il est tout à fait possible d'utiliser les manipulateurs sur d'autres flux comme par exemple sur les flux d'écriture dans des fichiers. Si on voulait écrire le tableau de l'exemple précédent dans un fichier plutôt que de vouloir l'afficher à l'écran, on pourrait écrire le code suivant:

Code : C++

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

using namespace std;

int main()
{
    ifstream fichier("moyennes.txt");  //On crée un flux en lecture
sur le fichier
    ofstream sortie("tableau.txt");    //On crée un flux en
écriture vers un fichier

    //On affiche le cadre du tableau
    sortie << "+" << setfill('-') << setw(18) << "+";  //On veut 15
caracteres + 2 pour pas
    sortie << setfill('-') << setw(8) << "+" << endl;  //coller au
bord + 1 pour le '+'
    //donc 18 au total idem pour la deuxieme colonne
    sortie << setfill(' ');

    string nom;       //Une chaine pour le nom de l'eleve
```

```
    double moyenne;   //Une variable pour sa moyenne

    sortie << fixed << setprecision(2);  //On force l'affichage de
deux chiffres apres la virgule

    while(fichier) //Tant qu'on est pas à la fin du fichier
    {
        fichier >> nom >> moyenne;   //On recupere les valeurs

        if(fichier) //Si on a bien pu lire quelque chose
        {
            sortie << "| " << left << setw(15) << nom;   //On écrit
le nom dans le fichier
            sortie << " | " << right << setw(5) << moyenne << " |"
<< endl;  //et la moyenne
        }
    }

    //On affiche le bas du tableau
    sortie << "+" << setfill('-') << setw(18) << "+";
    sortie << setfill('-') << setw(8) << "+" << endl;

    return 0;
}
```

Je crois qu'il n'y a rien de compliqué dans cet exemple. J'ai juste tout envoyé dans le flux `sortie` plutôt que dans le flux `std::cout` .
J'espère que ce document a pu vous être utile et vous invite à faire un tour sur le forum C++ pour faire plus ample connaissance et poser des questions si mes explications ne sont pas assez claires.

Mise à jour le: 16.10.2010