> **Ce cours est une mise à jour du cours C++ de OpenClassRoom pour le mettre à jour pour le C++11/14. Le cours d'origine est consultable sur la page suivante : Programmez avec le langage C++, par Mathieu Nebra et Matthieu Schaller. Ce cours est sous licence CC-BY-NC-SA.**

Retourner au sommaire principal

Qu'est-ce que la bibliothèque standard ?

Maintenant que vous êtes des champions de Qt, découvrir une bibliothèque de fonctions ne devrait pas vous faire peur. Vous verrez qu'utiliser les outils standard n'est pas toujours de tout repos mais cela peut rendre vos programmes diablement plus simples à écrire et plus efficaces. La bibliothèque standard du C++ est la bibliothèque officielle du langage, c'est-à-dire qu'elle est disponible partout où l'on peut utiliser le C++. Apprendre à l'utiliser vous permettra de travailler même sur les plates-formes les plus excentriques où d'autres outils, comme Qt par exemple, n'existent tout simplement pas.

Ce chapitre d'introduction à la bibliothèque standard (la SL (Standard Library) ne devrait pas vous poser de problèmes de compréhension. On va commencer en douceur par se cultiver un peu et revoir certains éléments dont vous avez déjà entendu parler. Nous allons au prochain chapitre nous plonger réellement dans la partie intéressante de la bibliothèque, la célèbre STL (Standard Template Library).

Un peu d'histoire

Dans l'introduction de ce cours, je vous ai déjà un petit peu parlé de l'histoire des langages de programmation en général, afin de situer le C++. Je vous propose d'entrer un peu plus dans les détails pour comprendre pourquoi un langage possède une bibliothèque standard.

La petite histoire raccourcie du C++

Prenons notre machine à remonter le temps et retournons à cette époque de l'informatique où le CD n'avait pas encore été inventé, où la souris n'existait pas, où les ordinateurs étaient moins puissants que les processeurs de votre lave-linge ou de votre four micro-ondes...

Nous sommes en 1979, Bjarne Stroustrup, un informaticien de chez AT&T, développe le C with classes à partir du C et en s'inspirant des langages plus modernes et innovants de l'époque comme le Simula. Il écrit lui-même le premier compilateur de son nouveau langage et l'utilise dans son travail. À l'époque, son langage n'était utilisé que par lui-même pour ses recherches personnelles. Il voulait en réalité améliorer le C en ajoutant les outils qui, selon lui, manquaient pour se simplifier la vie.

Petit à petit, des collègues et d'autres passionnés commencent à s'intéresser au C with classes. Et le besoin se fait sentir d'ajouter de nouvelles fonctionnalités. En 1983, les références, la surcharge d'opérateurs et les fonctions virtuelles sont ajoutées au langage qui s'appellera désormais C++. Le langage commence à ressembler un peu à ce que vous connaissez. En fait, quasiment tout ce que vous avez appris dans les parties I et II de ce cours est déjà présent dans le langage. Vous savez donc utiliser des notions qui ont de l'âge. Le C++ commence à intéresser les gens et Stroustrup finit par publier une version commerciale de son langage en 1985.

En 1989, la version 2.0 du C++ sort. Elle apporte en particulier l'héritage multiple, les classes abstraites et d'autres petites nouveautés pour les classes. Plus tard, les templates et les exceptions (deux notions que nous verrons dans la partie V de ce cours !) seront ajoutés au langage qui est quasiment équivalent à ce que l'on connaît aujourd'hui. En parallèle à cette évolution, une bibliothèque plus ou moins standard se crée, dans un premier temps pour remplacer les printf et autres choses peu pratiques du C par les cout, nettement plus faciles à utiliser. Cette partie de la SL, appelée iostream existe toujours et vous l'avez déjà utilisée.

Plus tard d'autres éléments seront ajoutés à la bibliothèque standard, comme par exemple la STL (Standard Template Library), reprise de travaux plus anciens d'Alexander Stepanov notamment et « traduits » de l'Ada vers le C++. Cet énorme ajout, qui va nous occuper dans la suite, est une avancée majeure pour le C++. C'est à ce moment-là que des choses très pratiques comme les string ou les vector apparaissent ! De nos jours, il est difficile d'imaginer un programme sans ces « briques » de base. Et je suis sûr que le reste de la STL va aussi vous plaire.

En 1998, un comité de normalisation se forme et décide de standardiser le langage au niveau international, c'est-à-dire que chaque implémentation du C++ devra fournir un certain nombre de fonctionnalités minimales. C'est dans ce cadre qu'est fixé le C++ actuel. C'est aussi à ce moment-là que la bibliothèque standard est figée et inscrite dans la norme. Cela veut dire que l'on devrait obligatoirement retrouver la bibliothèque standard avec n'importe quel compilateur. Et c'est cela qui fait sa force. Si vous avez un ordinateur avec un compilateur C++, il proposera forcément toutes les fonctions de la SL. Ce n'est pas forcément le cas d'autres bibliothèques qui n'existent que sous Windows ou que sous Linux, par exemple.

Enfin, en 2011, le C++ a subi une révision majeure qui lui a apporté de nombreuses fonctionnalités attendues depuis longtemps. Parmi ces changements, on citera l'ajout de nouveaux mots-clés, la simplification des templates et l'ajout de nombreux éléments avancés à la bibliothèque standard (notamment des fonctionnalités venant de la célèbre bibliothèque boost). Si cette nouvelle norme, qu'on appelle C++11, vous intéresse, je vous renvoie vers l'encyclopédie Wikipedia C++11 sur Wikipédia. Au niveau de ce cours, les différences sont minimes et c'est pour cela que les éléments de la nouvelle norme n'ont pas été abordés.

Pourquoi une bibliothèque standard ?

C'est une question que beaucoup de monde pose. Sans la SL, le C++ ne contient en réalité pratiquement rien. Essayez d'écrire un programme sans string, vector, cout ou même new (qui utilise en interne des parties

de la SL) ! C'est absolument impossible ou alors, cela reviendrait à écrire nous-mêmes ces briques pour les utiliser par la suite, c'est-à-dire écrire notre propre version de cout ou de vector. Je vous assure que c'est très très compliqué à faire. Au lieu de réinventer la roue, les programmeurs se sont donc mis d'accord sur un ensemble de fonctionnalités de base utilisées par un maximum de personnes et les ont mises à disposition de tout le monde.

Tous les langages proposent des bibliothèques standard avec des éléments à disposition des utilisateurs. Le langage Java, par exemple, propose même des outils pour créer des fenêtres alors que le C, lui, ne propose quasiment rien. Quand on utilise ce langage, il faut donc souvent fabriquer ses propres fonctions de base. Le C++ est un peu entre les deux puisque sa SL met à disposition des objets vector ou string mais ne propose aucun moyen de créer des fenêtres. C'est pour cela que nous avons appris à utiliser Qt. Il fallait utiliser quelque chose d'externe.

Bon ! Assez parlé. Voyons ce qu'elle a dans le ventre, cette bibliothèque standard.

Le contenu de la SL

La bibliothèque standard est grosso modo composée de trois grandes parties que nous allons explorer plus ou moins en détail dans ce cours. Comme vous allez le voir, il y a des morceaux que vous connaissez déjà bien (ou que vous devriez connaître).

> Cette classification est assez arbitraire et selon les sources que l'on consulte, on trouve jusqu'à 5 parties.

L'héritage du C

L'ensemble de la bibliothèque standard du C est présente dans la SL. Les 15 fichiers d'en-tête du C (norme de 1990) sont disponibles quasiment à l'identique en C++. De cette manière, les programmes écrits en C peuvent (presque tous) être réutilisés tels quels en C++. Je vous

présenterai ces vénérables éléments venus du C à la fin de ce chapitre.

Les flux

Dans une deuxième partie, on trouve tout ce qui a trait aux flux, c'est-à-dire l'ensemble des outils permettant de faire communiquer les programmes avec l'extérieur. Ce sont les classes permettant :

- d'afficher des messages dans la console ;
- d'écrire du texte dans la console ;
- ou encore d'effectuer des opérations sur les fichiers.

J'espère que cela vous rappelle quelque chose !

Cet aspect sera brièvement abordé dans la suite de cette partie mais, comme vous connaissez déjà bien ces choses, il ne sera pas nécessaire de détailler tout ce qu'on y trouve.

Nous avons déjà appris à utiliser cout, cin ainsi que les fstream pour communiquer avec des fichiers plus tôt dans ce cours. Je ne vais pas vous en apprendre beaucoup plus dans la suite. Mais nous allons quand même voir quelques fonctionnalités, comme la copie d'un fichier dans un tableau de mots. Vous verrez que certaines opérations fastidieuses peuvent être réalisées de manière très simple une fois que l'on connaît les bons concepts.

La STL

La Standard Template Library (STL) est certainement la partie la plus intéressante. On y trouve des conteneurs, tels que les vector, permettant de stocker des objets selon différents critères. On y trouve également quelques algorithmes standard comme la recherche d'éléments dans un conteneur ou le tri d'éléments. On y trouve des itérateurs, des foncteurs, des prédicats, des pointeurs intelligents et encore plein d'autres choses mystérieuses que nous allons découvrir en détail.

Vous vous en doutez peut-être, la suite de ce cours sera consacrée principalement à la description de la STL.

Le reste

Je vous avais dit qu'il y avait trois parties...

Mais comme souvent, les classifications ne sont pas réellement utilisables dans la réalité. Il y a donc quelques éléments de la SL qui sont inclassables, en particulier la classe string, qui est à la frontière entre la STL et les flux. De même, certains outils concernant la gestion fine de la mémoire, les paramètres régionaux ou encore les nombres complexes ne rentrent dans aucune des trois parties principales. Mais cela ne veut pas dire que je ne vous en parlerai pas.

Se documenter sur la SL

Dans le chapitre Apprendre à lire la documentation de Qt (chapitre 6) vous avez appris à vous servir d'une documentation pour trouver les classes et fonctions utiles... Et vous vous dites sûrement qu'il en va de même pour la bibliothèque standard. Je vais vous décevoir, mais ce n'est pas le cas. Il n'existe pas de « documentation officielle » pour la SL. Et c'est bien dommage.

En fait, ce n'est pas tout à fait vrai. La description très détaillée de chaque fonctionnalité se trouve dans la norme du langage. C'est un gros document de près de 800 pages, entièrement en anglais, absolument indigeste. Par exemple, on y trouve la description suivante pour les objets vector :

Citation : Norme C++, Paragraphe 23.2.4.1

> A vector is a kind of sequence that supports random access iterators. In addition, it supports (amortized) constant time insert and erase operations at the end; insert and erase in the

> middle take linear time. Storage management is handled automatically, though hints can be given to improve efficiency. The elements of a vector are stored contiguously, meaning that if v is a vector<T, Allocator> where T is some type other than bool, then it obeys the identity &v[n] == &v[0] + n for all 0 ⇐ n < v.size().

Même si vous parlez anglais couramment, vous n'avez certainement pas compris grand chose. Et pourtant, je vous ai choisi un passage pas trop obscur. Vous comprendrez donc que l'on ne peut pas travailler avec cela.

Des ressources sur le Web

Heureusement, il existe quelques sites web qui présentent le contenu de la SL. Mais manque de chance pour les anglophobes, toutes les références intéressantes sont en anglais.

Voici quatre sources que j'utilise régulièrement. Elles ne sont pas toutes complètes mais elles suffisent dans la plupart des cas. Je les ai classées de la plus simple à suivre à la plus compliquée.

- cppreference.com.
- cplusplus.com - Une documentation plus simple qui présente l'ensemble de la SL. Les explications ne sont pas toujours complètes mais elles sont accompagnées d'exemples simples qui permettent de bien comprendre l'intérêt de chaque fonction et classe.
- Apache C++ - Une bonne documentation de la SL en général. Les fonctions sont accompagnées d'exemples simples permettant de comprendre le fonctionnement de chacune d'elles. Elle est surtout intéressante pour sa partie sur les flux
- sgi - Une documentation très complète de la STL. La description n'est pas toujours aisée à lire pour un débutant et certains éléments présentés ne font pas partie de la STL standard mais seulement d'une version proposée par SGI. Présente uniquement

la STL.

Je vous conseille de naviguer un peu sur ces sites et de regarder celui qui vous plaît le plus.

L'autre solution est de me faire confiance et découvrir le tout via ce cours. Je ne pourrai bien sûr pas tout vous présenter mais on va faire le tour de l'essentiel.

L'héritage du C

Le C++ étant en quelque sorte un descendant du C, la totalité de la bibliothèque standard du C est disponible dans la SL. Il y a quelques outils qui sont toujours utilisés, d'autres qui ont été remplacés par des versions améliorées et finalement d'autres qui sont totalement obsolètes. J'espère ne pas vous décevoir en ne parlant que des éléments utiles. C'est déjà beaucoup !

> Si vous avez fait du C, vous devriez reconnaître les en-têtes dont je vais vous parler. La principale différence réside dans le nom du fichier. En C, on utilise math.h, alors qu'en C++, c'est cmath. Le « .h » a disparu et un « c » a été ajouté devant le nom.

Comme tout le reste de la SL, la partie héritée du C est séparée en différents fichiers d'en-tête plus ou moins cohérents.

L'en-tête cmath

Celui-là, vous le connaissez déjà. Vous l'avez découvert tout au début du cours. C'est dans ce fichier que sont définies toutes les fonctions mathématiques usuelles. Comme je suis sympa, voici un petit rappel pour ceux qui dorment au fond de la classe.

```
#include<iostream>
#include<cmath>
```

```
using namespace std;

int main()
{
    double a(4.3), b(5.2);
    cout << pow(a,b) << endl;    //Calcul de a^b
    cout << sqrt(a) << endl;     //Calcul de la racine
carrée de a
    cout << cos(b) << endl;      //Calcul du cosinus de b
    return 0;
}
```

Ah je vois que vous vous en souvenez encore. Parfait ! C'est donc le fichier à inclure lorsque vous avez des calculs mathématiques à effectuer. Je ne vais pas vous réécrire toute la liste des fonctions, vous les connaissez déjà. Pour vous habituer à la documentation, essayez donc de retrouver ces fonctions dans les différentes ressources que je vous ai indiquées.

Documentation de cmath

L'en-tête cctype

Ce fichier propose quelques fonctions pour connaître la nature d'un char. Quand on manipule du texte, on doit souvent répondre à des questions comme :

- Cette lettre est-elle une majuscule ou une minuscule ?
- Ce caractère est-il un espace ?
- Ce symbole est-il un chiffre ?

Les fonctions présentes dans l'en-tête cctype sont là pour cela. Pour tester si un char donné est un chiffre, par exemple, on utilisera la fonction isdigit(). Comme dans l'exemple suivant :

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

```
int main()
{
    cout << "Entrez un caractere : ";
    char symbole;
    cin >> symbole;

    if(isdigit(symbole))
        cout << "C'est un chiffre." << endl;
    else
        cout << "Ce n'est pas un chiffre." << endl;

    return 0;
}
```

Comme vous le voyez, c'est vraiment très simple à utiliser. Le tableau suivant présente les fonctions les plus utilisées de cet en-tête. Vous trouverez la liste complète dans votre documentation favorite.

Nom de la fonction	Description
isalpha()	Vérifie si le caractère est une lettre.
isdigit()	Vérifie si le caractère est un chiffre.
islower()	Vérifie si le caractère est une minuscule.
isupper()	Vérifie si le caractère est une majuscule.
isspace()	Vérifie si le caractère est un espace ou un retour à la ligne.

En plus de cela, il y a deux fonctions tolower() et toupper() qui convertissent une majuscule en minuscule et inversement. On peut ainsi aisément transformer un texte en majuscules :

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

int main()
{
    cout << "Entrez une phrase : " << endl;
```

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

    //On parcourt la chaîne pour la convertir en majuscules
    for(int i(0); i<phrase.size(); ++i)
    {
         phrase[i] = toupper(phrase[i]);
    }

    cout << "Votre phrase en majuscules est : " << phrase <<
endl;
    return 0;
}
```

À nouveau, rien de bien sorcier. Je vous laisse vous amuser un peu avec ces fonctions. Essayez par exemple de réaliser un programme qui remplace tous les espaces d'une string par le symbole #. Je suis sûr que c'est dans vos cordes.

L'en-tête ctime

Comme son nom l'indique, ce fichier d'en-tête contient plusieurs fonctions liées à la gestion du temps. La plupart sont assez bizarres à utiliser et donc peu utilisées. De toute façon, la plupart des autres bibliothèques, comme Qt, proposent des classes pour gérer les heures, les jours et les dates de manière plus aisée.

Personnellement, la seule fonction de ctime que j'utilise est la fonction time(). Elle renvoie le nombre de secondes qui se sont écoulées depuis le 1er janvier 1970. C'est ce qu'on appelle l'heure UNIX.

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

int main()
{
    int secondes = time(0);
```

```
    cout << "Il s'est ecoule " << secondes << " secondes
depuis le 01/01/1970." << endl;
    return 0;
}
```

> La fonction attend en argument un pointeur sur une variable dans laquelle stocker le résultat. Mais bizarrement, elle renvoie aussi ce résultat comme valeur de retour. L'argument est donc en quelque sorte inutile. On fournit généralement à la fonction un pointeur ne pointant sur rien, d'où le 0 passé en argument.

Ce qui donne :

```
Il s'est ecoule 1302471754 secondes depuis le 01/01/1970.
```

Ce qui fait beaucoup de secondes !

> Euh… À quoi cela sert-il ?

Il y a principalement trois raisons d'utiliser cette fonction :

- La première utilisation est bien sûr de calculer la date. Avec un petit peu d'arithmétique, on retrouve facilement la date et l'heure actuelle. Mais comme je vous l'ai dit, la plupart des bibliothèques proposent des outils plus simples pour cela.
- Deuxièmement, on peut l'utiliser pour calculer le temps que met le programme à s'exécuter. On appelle la fonction time() en début de programme puis une deuxième fois à la fin. Le temps passé dans le programme sera simplement la différence entre les deux valeurs obtenues !
- Le dernier cas d'utilisation de cette fonction, vous l'avez déjà vu ! On l'utilise pour générer des nombres aléatoires. Nous allons voir comment dans la suite.

L'en-tête cstdlib

Voici à nouveau une vieille connaissance. Souvenez-vous, dans le premier TP, je vous avais présenté un moyen de choisir un nombre au hasard. Il fallait utiliser les fonctions srand() et rand().

C'est certainement l'en-tête le plus utile en C. Il contient toutes les briques de base et je crois qu'il n'y a pas un seul programme de C qui n'inclue pas stdlib.h (l'équivalent « C » de ce fichier). Par contre, en C++, eh bien... il ne sert quasiment à rien. Mise à part la génération de nombres aléatoires, tout a été remplacé en C++ par de nouvelles fonctionnalités.

Revoyons quand même en vitesse comment générer des nombres aléatoires. La fonction rand() renvoie un nombre au hasard entre 0 et RAND_MAX (un très grand nombre, généralement plus grand que 109). Si l'on souhaite obtenir un nombre au hasard entre 0 et 10, on utilise l'opérateur modulo (%).

```
nb = rand() % 10; //nb prendra une valeur au hasard entre 0
et 9 compris.
```

Jusque là, rien de bien compliqué. Le problème est qu'un ordinateur ne sait pas générer un nombre au hasard. Tout ce qu'il sait faire c'est créer des suites de nombres qui ont l'air aléatoires. Il faut donc spécifier un début pour la séquence. Et c'est là qu'intervient la fonction srand() : elle permet de spécifier le premier terme de la suite.

> Il ne faut appeler qu'une seule et unique fois la fonction srand() par programme !

Le problème est que si l'on donne chaque fois le même premier terme à l'ordinateur, il génère à chaque fois la même séquence ! Il faut donc lui donner quelque chose de différent à chaque exécution du programme. Et qu'est-ce qui change à chaque fois que l'on exécute un programme ? La date et l'heure, bien sûr ! La solution est donc d'utiliser le résultat de la fonction time() comme premier terme de la série.

```
#include <iostream>
#include <ctime>
#include <cstdlib>
```

```cpp
using namespace std;

int main()
{
    srand(time(0));    //On initialise la suite de nombres
aléatoires

    for(int i(0); i<10; ++i)
        cout << rand() % 10 << endl;  //On génère des
nombres au hasard

    return 0;
}
```

Libre à vous ensuite d'utiliser ces nombres pour mélanger des lettres comme dans le TP ou pour créer un jeu de casino. Le principe de base est toujours le même.

Les autres en-têtes

Mis à part cassert dont nous parlerons plus tard, le reste des 15 en-têtes du C n'est que très rarement utilisé en C++. Je ne vous en dirai donc pas plus dans ce cours.

Bon, assez travaillé avec les reliques du C ! Pour l'instant, je ne vous ai pas présenté de grandes révolutions pour vos programmes comme je vous l'avais promis. Ne le dites pas trop fort si vous rencontrez des amateurs de C mais c'est parce qu'on n'a pas encore utilisé la puissance du C++. Attachez vos ceintures, la suite du voyage va secouer.

En résumé

- La bibliothèque standard du C++ propose de nombreuses briques de base pour simplifier l'écriture de nos programmes.
- On peut considérer qu'elle est découpée en trois parties : la STL,

les flux et tout ce qui a été repris du langage C.
- Parmi les éléments repris du C, on utilise principalement cctype pour analyser des lettres, ctime pour la mesure du temps et cstdlib pour générer des nombres aléatoires.

Retourner au sommaire principal

Cours, C++