



C++ - Module 09

STL

Summary: Ce document contient les exercices du Module 09 de la Piscine C++.

Version: 2.1

Contents

I	Introduction	2
II	Consignes générales	3
III	Règles spécifiques au module	6
IV	Exercice 00: Bitcoin Exchange	7
V	Exercice 01: Reverse Polish Notation	9
VI	Exercice 02: PmergeMe	11
VII	Rendu et évaluation	14

Chapter I

Introduction

C++ est un langage de programmation polyvalent créé par Bjarne Stroustrup en tant qu'extension du langage de programmation C, ou "C avec classes" (source: [Wikipedia](#)).

L'objectif de ces modules est de vous initier à ;a **Programmation orientée-objet**. Ce sera le point de départ de votre parcours en C++. De nombreux langages sont recommandés pour apprendre la POO. Nous avons choisi le C++ car il est dérivé de votre vieil ami, le C. Comme il s'agit d'un langage complexe, et afin de simplifier les choses, votre code sera conforme à la norme C++98.

Nous sommes conscients que le C++ moderne est très différent à bien des égards. Ainsi, si vous souhaitez devenir un développeur C++ compétent, il vous appartient d'aller plus loin après le Tronc Commun de 42!

Chapter II

Consignes générales

Compilation

- Compilez votre code avec `c++` et les flags `-Wall -Wextra -Werror`
- Votre code doit compiler si vous ajoutez le flag `-std=c++98`

Format et conventions de nommage

- Les dossiers des exercices seront nommés ainsi : `ex00`, `ex01`, ... , `exn`
- Nommez vos fichiers, vos classes, vos fonctions, vos fonctions membres et vos attributs comme spécifié dans les consignes.
- Rédigez vos noms de classe au format **UpperCamelCase**. Les fichiers contenant le code d'une classe porteront le nom de cette dernière. Par exemple : `NomDeClasse.hpp`/`NomDeClasse.h`, `NomDeClasse.cpp`, ou `NomDeClasse.hpp`. Ainsi, si un fichier d'en-tête contient la définition d'une classe "BrickWall", son nom sera `BrickWall.hpp`.
- Sauf si spécifié autrement, tous les messages doivent être terminés par un retour à la ligne et être affichés sur la sortie standard.
- *Ciao Norminette !* Aucune norme n'est imposée durant les modules C++. Vous pouvez suivre le style de votre choix. Mais ayez à l'esprit qu'un code que vos pairs ne peuvent comprendre est un code que vos pairs ne peuvent évaluer. Faites donc de votre mieux pour produire un code propre et lisible.

Ce qui est autorisé et ce qui ne l'est pas

Le langage C, c'est fini pour l'instant. Voici l'heure de se mettre au C++ ! Par conséquent :

- Vous pouvez avoir recours à quasi l'ensemble de la bibliothèque standard. Donc plutôt que de rester en terrain connu, essayez d'utiliser le plus possible les versions C++ des fonctions C dont vous avez l'habitude.
- Cependant, vous ne pouvez avoir recours à aucune autre bibliothèque externe. Ce qui signifie que C++11 (et dérivés) et l'ensemble **Boost** sont interdits. Aussi, certaines fonctions demeurent interdites. Utiliser les fonctions suivantes résultera en la note de 0 : `*printf()`, `*alloc()` et `free()`.
- Sauf si explicitement indiqué autrement, les mots-clés `using namespace <ns_name>` et `friend` sont interdits. Leur usage résultera en la note de -42.
- **Vous n'avez le droit à la STL que dans les Modules 08 et 09.** D'ici là, l'usage des **Containers** (vector/list/map/etc.) et des **Algorithmes** (tout ce qui requiert d'inclure `<algorithm>`) est interdit. Dans le cas contraire, vous obtiendrez la note de -42.

Quelques obligations côté conception

- Les fuites de mémoires existent aussi en C++. Quand vous allouez de la mémoire (en utilisant le mot-clé `new`), vous ne **devez pas avoir de memory leaks**.
- Du Module 02 au Module 09, vos classes devront se conformer à la forme **canonique, dite de Coplien, sauf si explicitement spécifié autrement**.
- Une fonction implémentée dans un fichier d'en-tête (hormis dans le cas de fonction template) équivaudra à la note de 0.
- Vous devez pouvoir utiliser vos fichiers d'en-tête séparément les uns des autres. C'est pourquoi ils devront inclure toutes les dépendances qui leur seront nécessaires. Cependant, vous devez éviter le problème de la double inclusion en les protégeant avec des **include guards**. Dans le cas contraire, votre note sera de 0.

Read me

- Si vous en avez le besoin, vous pouvez rendre des fichiers supplémentaires (par exemple pour séparer votre code en plus de fichiers). Vu que votre travail ne sera pas évalué par un programme, faites ce qui vous semble le mieux du moment que vous rendez les fichiers obligatoires.
- Les consignes d'un exercice peuvent avoir l'air simple mais les exemples contiennent parfois des indications supplémentaires qui ne sont pas explicitement demandées.
- Lisez entièrement chaque module avant de commencer ! Vraiment.
- Par Odin, par Thor ! Utilisez votre cervelle !!!



Vous aurez à implémenter un bon nombre de classes, ce qui pourrait s'avérer ardu... ou pas ! Il y a peut-être moyen de vous simplifier la vie grâce à votre éditeur de texte préféré.



Vous êtes assez libre quant à la manière de résoudre les exercices. Toutefois, respectez les consignes et ne vous en tenez pas au strict minimum, vous pourriez passer à côté de notions intéressantes. N'hésitez pas à lire un peu de théorie.

Chapter III

Règles spécifiques au module

Il est obligatoire d'utiliser les conteneurs standard pour effectuer chaque exercice dans ce module.

Une fois qu'un conteneur est utilisé, vous ne pouvez pas l'utiliser pour le reste du module.



Il est conseillé de lire le sujet dans son intégralité avant de faire les exercices.



Vous devez utiliser au moins un conteneur pour chaque exercice, à l'exception de l'exercice 02 qui nécessite l'utilisation de deux conteneurs.


Vous devez soumettre un **Makefile** pour chaque programme, qui compilera vos fichiers source pour produire la sortie requise avec les indicateurs **-Wall**, **-Wextra** and **-Werror**.

Vous devez utiliser C++, et votre Makefile ne doit pas recréer les liens (relink).

Votre **Makefile** doit contenir au moins les règles **\$(NAME)**, **all**, **clean**, **fclean** and **re**.

Chapter IV

Exercice 00: Bitcoin Exchange

	Exercice : 00
Bitcoin Exchange	
Turn-in directory : <i>ex00/</i>	
Files to turn in : <code>Makefile</code> , <code>main.cpp</code> , <code>BitcoinExchange.{cpp, hpp}</code>	
Forbidden functions : None	

Vous devez créer un programme qui affiche la valeur d'une certaine quantité de bitcoins à une certaine date.

Ce programme doit utiliser une base de données au format CSV qui représentera le prix du bitcoin au fil du temps. Cette base de données est fournie avec le sujet.

Le programme prendra en entrée une deuxième base de données stockant les différents prix / dates à évaluer.

Votre programme doit respecter ces règles:

- Le nom du programme est `btc`.
- Votre programme doit prendre un fichier en argument.
- Chaque ligne de ce fichier doit utiliser le format suivant: "date | valeur".
- Une date valide sera toujours au format "Année-Mois-Jour" (YYYY-MM-DD).
- Une valeur valide doit être soit un nombre à virgule flottante (float), ou un entier positif, compris entre 0 et 1000.



Vous devez utiliser au moins un conteneur dans votre code pour valider cet exercice. Vous devez gérer les erreurs possibles avec un message d'erreur approprié.

Voici un exemple d'un fichier input.txt

```
$> head input.txt
date | value
2011-01-03 | 3
2011-01-03 | 2
2011-01-03 | 1
2011-01-03 | 1.2
2011-01-09 | 1
2012-01-11 | -1
2001-42-42
2012-01-11 | 1
2012-01-11 | 2147483648
$>
```

Votre programme va utiliser la valeur dans votre fichier input.

Votre programme doit afficher sur la sortie standard le résultat de la valeur multipliée par le taux de change en fonction de la date indiquée dans votre base de données.



Si la date utilisée en entrée n'existe pas dans votre BD, alors vous devez utiliser la date la plus proche contenue dans votre BD. Soyez attentifs à utiliser la date précédente la plus proche, et non la suivante.

L'exemple suivant démontre le fonctionnement de votre programme.


```
$> ./btc
Error: could not open file.
$> ./btc input.txt
2011-01-03 => 3 = 0.9
2011-01-03 => 2 = 0.6
2011-01-03 => 1 = 0.3
2011-01-03 => 1.2 = 0.36
2011-01-09 => 1 = 0.32
Error: not a positive number.
Error: bad input => 2001-42-42
2012-01-11 => 1 = 7.1
Error: too large a number.
$>
```



Attention: les conteneurs que vous utilisez pour valider cet exercice ne pourront plus être utilisés pour le reste du module.

Chapter V

Exercice 01: Reverse Polish Notation

	Exercice : 01
RPN	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <code>Makefile</code> , <code>main.cpp</code> , <code>RPN.{cpp, hpp}</code>	
Forbidden functions : <code>None</code>	

Vous devez créer un programme avec ces contraintes:

- Le nom du programme est `RPN`.
- Votre programme doit prendre en argument une expression mathématique en notation polonaise inversée.
- Les chiffres utilisés dans cette opération et transmis en tant qu'arguments seront toujours inférieurs à 10. Le calcul lui-même ainsi que le résultat ne tiennent pas compte de cette règle.
- Votre programme doit traiter cette expression et afficher le résultat correct sur la sortie standard.
- Si une erreur se produit pendant l'exécution du programme, un message d'erreur doit être affiché sur la sortie d'erreur.
- Votre programme doit être capable de gérer les opérations avec ces opérateurs: `+`, `-`, `/`, `*`.



Vous devez utiliser au moins un conteneur dans votre code pour valider cet exercice.



Vous n'avez pas besoin de gérer les parenthèses ou les nombres décimaux.

Voici un exemple d'utilisation standard:


```
$> ./RPN "8 9 * 9 - 9 - 4 - 1 +"  
42  
$> ./RPN "7 7 * 7 -"  
42  
$> ./RPN "1 2 * 2 / 2 * 2 4 - +"  
0  
$> ./RPN "(1 + 1)"  
Error  
$>
```



Attention: Le(s) conteneur(s) utilisé(s) dans l'exercice précédent sont interdits ici. Le(s) conteneur(s) utilisé(s) dans cet exercice ne pourront être réutilisés dans un autre exercice du module.

Chapter VI

Exercice 02: PmergeMe

	Exercice : 02
PmergeMe	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <code>Makefile</code> , <code>main.cpp</code> , <code>PmergeMe.{cpp, hpp}</code>	
Forbidden functions : None	

Vous devez créer un programme avec ces contraintes:

- Le nom du programme est PmergeMe.
- Votre programme doit prendre une séquence d'entiers positif en argument.
- Votre programme doit utiliser l'algorithme de tri "merge-insert" pour trier la séquence de nombres.



Afin d'être clair, oui, vous devez utiliser l'algorithme Ford-Johnson.

- Si une erreur survient durant l'exécution du programme, un message d'erreur doit être affiché sur la sortie d'erreur.



Vous devez utiliser au moins deux conteneurs dans votre code afin de valider cet exercice. Votre programme doit pouvoir gérer au moins 3000 entiers différents.



Il est fortement recommandé d'implémenter votre algorithme pour chaque conteneurs utilisé et donc d'éviter d'utiliser une fonction générique.

Voici quelques directives supplémentaires sur l'information que vous devriez afficher sur la sortie standard:

- Sur la première ligne, vous devez afficher un texte explicite suivi de la séquence d'entiers positifs non triés.
- Sur la deuxième ligne, vous devez afficher un texte explicite suivi de la séquence d'entiers positifs triés.
- Sur la troisième ligne, vous devez afficher un texte explicite indiquant le temps utilisé par votre algorithme en spécifiant le premier conteneur utilisé pour trier la séquence d'entiers positifs.
- Sur la dernière ligne, vous devez afficher un texte explicite indiquant le temps utilisé par votre algorithme en spécifiant le deuxième conteneur utilisé pour trier la séquence d'entiers positifs.



Le format d'affichage du temps utilisé pour effectuer votre tri est libre, mais la précision choisie doit permettre de voir clairement la différence entre les deux conteneurs utilisés.

Voici un **exemple** d'utilisation standard:

```
$> ./PmergeMe 3 5 9 7 4
Before: 3 5 9 7 4
After: 3 4 5 7 9
Time to process a range of 5 elements with std::[...] : 0.00031 us
Time to process a range of 5 elements with std::[...] : 0.00014 us
$> ./PmergeMe `shuf -i 1-100000 -n 3000 | tr "\n" " "`
Before: 141 79 526 321 [...]
After: 79 141 321 526 [...]
Time to process a range of 3000 elements with std::[...] : 62.14389 us
Time to process a range of 3000 elements with std::[...] : 69.27212 us
$> ./PmergeMe "-1" "2"
Error
$> # For OSX USER:
$> ./PmergeMe `jot -r 3000 1 100000 | tr '\n' ' '`
[...]
$>
```



L'indication du temps est délibérément étrange dans cet exemple. Bien sûr, vous devez indiquer le temps utilisé pour effectuer toutes vos opérations, à la fois la partie de tri et la partie de gestion des données.



Attention: Les conteneurs utilisés dans les exercices précédents ne peuvent pas être utilisés ici.



La gestion des erreurs liées aux doublons est laissée à votre discrétion.

Chapter VII

Rendu et évaluation

Rendez votre travail dans un répertoire `Git` comme d'habitude. Seulement le travail présent à l'intérieur du repo sera évalué lors de la défense. N'hésitez pas à revérifier les noms des dossiers et fichiers pour vous assurer qu'ils soient corrects.



```
16D85ACC441674FBA2DF65190663F33F793984B142405F56715D5225FBAB6E3D6A4F
167020A16827E1B16612137E59ECD492E47AB764CB10B45D979615AC9FC74D521D9
20A778A5E
```