



C++ - Module 08

Template de container, itérateurs, algorithmes

Résumé: Ce document contient le sujet pour le module 08 des modules C++ de 42.

Table des matières

I	Règles Générales	2
II	Day-specific rules	4
III	Exercice 00 : Easy find	5
IV	Exercice 01 : Span	6
V	Exercice 02 : Abomination mutante	8
VI	Exercice 03 : Ouvrez votre esprit, mais par pitié ne l'explotez pas	10
VII	Exercice 04 : In Poland, expression evaluates you	12

Chapitre I

Règles Générales

- Toute fonction implémentée dans une header (sauf pour les templates) ou tout header non-protégé, signifie 0 à l'exercice.
- Tout output doit être affiché sur stdout et terminé par une newline, sauf si autre chose est précisé.
- Les noms de fichiers imposés doivent être suivis à la lettre, tout comme les noms de classe, les noms de fonction, et les noms de méthodes.
- Rappel : vous codez maintenant en C++, et plus en C. C'est pourquoi :
 - Les fonctions suivantes sont **INTERDITES**, et leur usage se soldera par un 0 : `*alloc`, `*printf` et `free`
 - Vous avez l'autorisation d'utiliser à peu près toute la librairie standard. CÉPENDANT, il serait intelligent d'essayer d'utiliser la version C++ de ce à quoi vous êtes habitués en C, plutôt que de vous reposer sur vos acquis. Et vous n'êtes pas autorisés à utiliser la STL jusqu'au moment où vous commencez à travailler dessus (module 08). Ça signifie pas de Vector/List/Map/etc... ou quoi que ce soit qui requiert une include `<algorithm>` jusque là.
- L'utilisation d'une fonction ou mécanique explicitement interdite sera sanctionnée par un 0
- Notez également que sauf si la consigne l'autorise, les mot-clés `using namespace` et `friend` sont interdits. Leur utilisation sera punie d'un 0.
- Les fichiers associés à une classe seront toujours nommés `ClassName.cpp` et `ClassName.hpp`, sauf si la consigne demande autre chose.
- Vous devez lire les exemples minutieusement. Ils peuvent contenir des prérequis qui ne sont pas précisés dans les consignes.
- Vous n'êtes pas autorisés à utiliser des librairies externes, incluant C++11, Boost, et tous les autres outils que votre ami super fort vous a recommandé.
- Vous allez surement devoir rendre beaucoup de fichiers de classe, ce qui peut paraître répétitif jusqu'à ce que vous appreniez à scripter ça dans votre éditeur de code préféré.

- Lisez complètement chaque exercice avant de le commencer.
- Le compilateur est `clang++`
- Votre code sera compilé avec les flags `-Wall -Wextra -Werror -std=c++98`
- Chaque include doit pouvoir être incluse indépendamment des autres includes. Un include doit donc inclure toutes ses dépendances.
- Il n'y a pas de norme à respecter en C++. Vous pouvez utiliser le style que vous préférez. Cependant, un code illisible est un code que l'on ne peut pas noter.
- Important : vous ne serez pas noté par un programme (sauf si précisé dans le sujet). Cela signifie que vous avez un degré de liberté dans votre méthode de résolution des exercices.
- Faites attention aux contraintes, et ne soyez pas fainéant, vous pourriez manquer beaucoup de ce que les exercices ont à offrir
- Ce n'est pas un problème si vous avez des fichiers additionnels. Vous pouvez choisir de séparer votre code dans plus de fichiers que ce qui est demandé, tant qu'il n'y a pas de moulinette.
- Même si un sujet est court, cela vaut la peine de passer un peu de temps dessus afin d'être sûr que vous comprenez bien ce qui est attendu de vous, et que vous l'avez bien fait de la meilleure manière possible.


Chapitre II

Day-specific rules

- Vous remarquerez que dans ce sujet en particulier, beaucoup de problèmes que l'on vous demande de résoudre peuvent être résolus sans les algorithmes ou containers de la STL. Cependant, les utiliser est votre objectif, et si vous ne faites pas l'effort d'utiliser containers et algorithmes, vous aurez une mauvaise note. Ne soyez pas fainéants.

Chapitre III

Exercice 00 : Easy find

	Exercice : 00
	Easy find
	Dossier de rendu : <i>ex00/</i>
	Fichiers à rendre : easyfind.hpp main.cpp
	Fonctions interdites : Aucune

Un exercice facile pour bien commencer...


Faites une fonction template nommée **easyfind**, sur le type **T**, qui prends un **T** et un **int**.

T est un container d'int. Trouvez la première occurrence du second paramètre dans le premier paramètre. Si vous ne le trouvez pas, gérez l'erreur en utilisant une exception ou une valeur de retour. Inspirez-vous des containers existants.

Rendez un **main** qui teste de manière approfondie votre code.

Chapitre IV

Exercice 01 : Span

	Exercice : 01
Span	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : Span.cpp Span.hpp main.cpp	
Fonctions interdites : Aucune	

Faites une classe **Span** qui peut stocker **N ints**. **N** sera un unsigned int, et sera donné à la construction en paramètre unique.

Cette classe aura pour fonction de stocker un unique nombre (**addNumber**) qui la remplira. Tenter d'ajouter un autre nombre alors que **N** sont déjà stockés est une erreur et renverra donc une exception.

Faites maintenant deux fonctions, **shortestSpan** et **longestSpan**, qui vont respectivement trouver le plus petit et le plus grand écart entre tous les nombres contenus par l'objet, et le renvoyer. Si il n'y a pas de nombre stocké, ou seulement un, il n'y a pas de span à trouver, et vous devez renvoyer une exception.

Vous trouverez en dessous un exemple (beaucoup trop court) de **main** et l'output attendu. Rendez votre propre **main** pour l'évaluation, qui sera, bien entendu, beaucoup plus fourni. Vous devez tester avec au moins 10000 nombres. Il serait optimal également de donner des nombres à l'aide d'un range d'itérateurs, ce qui éviterait de faire des milliers de calls à **addNumber** : implémentez cette fonction.

```
int main()
{
    Span sp = Span(5);


    sp.addNumber(5);
    sp.addNumber(3);
    sp.addNumber(17);
    sp.addNumber(9);
    sp.addNumber(11);

    std::cout << sp.shortestSpan() << std::endl;
    std::cout << sp.longestSpan() << std::endl;
}
```

```
$> ./ex01
2
14
$>
```


Chapitre V

Exercice 02 : Abomination mutante

	Exercice : 02
Abomination mutante	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : Mutantstack.cpp Mutantstack.hpp main.cpp	
Fonctions interdites : Aucune	

Maintenant que les amuse-bouches sont terminés, passons aux choses sérieuses.

Le container `std::stack` est SUPER COOL, mais c'est l'un des rares containers de la STL qui n'est pas itérable. Quel dommage. Pourquoi serions nous d'accord avec ça, alors qu'on peut tout saccager pour rajouter des trucs sympa comme ça ?

Vous devez implémenter cette faculté au container `std::stack`, afin de réparer cette énorme injustice.

Faites une classe `MutantStack`, qui ressemblera fortement au container `std::stack`, mais qui offrira également des itérateurs.

Voici un exemple de code, dont l'output devrait être le même que si l'on utilisait une `std::list`.

Vous devez bien entendu rendre votre `main`, plus étoffé que ce que nous vous proposons ici.

```
int main()
{
    MutantStack<int>    mstack;

    mstack.push(5);
    mstack.push(17);

    std::cout << mstack.top() << std::endl;

    mstack.pop();

    std::cout << mstack.size() << std::endl;


    mstack.push(3);
    mstack.push(5);
    mstack.push(737);
    //[...]
    mstack.push(0);

    MutantStack<int>::iterator it = mstack.begin();
    MutantStack<int>::iterator ite = mstack.end();

    ++it;
    --ite;
    while (it != ite)
    {
        std::cout << *it << std::endl;
        ++it;
    }
    std::stack<int> s(mstack);
    return 0;
}
```

Chapitre VI

Exercice 03 : Ouvrez votre esprit, mais par pitié ne l'explotez pas

	Exercice : 03
Ouvrez votre esprit, mais par pitié ne l'explotez pas	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : <code>main.cpp</code> + Ce que vous voulez	
Fonctions interdites : Aucune	



Cet exercice et les suivants ne rapportent pas de points, mais peuvent être importants. Vous pouvez les faire, ou non.

Brainfuck est un langage de programmation très cool. Contrairement à la croyance populaire, cela ne signifie pas vraiment "ouvre ton esprit", cependant. Plus comme "avoir des relations intimes avec votre cervelet".

Nous voudrions que vous fassiez un interprète de Brainfuck, mais cela impliquerait que vous écriviez souvent "fuck", et comme nous n'aimons pas les blasphèmes, nous préférierions que vous n'écriviez pas trop "fuck". Parce que, vous savez, "fuck" est considéré comme un peu vulgaire, il ne serait donc pas très professionnel d'écrire "fuck" dans votre code. De plus, si nous écrivons "fuck" dans un sujet, nous nous inclinerions à la forme d'humour la plus basse pour tenter de vous rendre un peu plus intéressé par ce que vous faites ici. Donc, non, nous ne l'appellerons pas réellement Brainfuck dans ce sujet, car cela signifierait écrire "fuck", ici même, dans cette phrase même. Et ce serait dommage. Je veux dire, imaginons écrire "fuck" plusieurs fois dans un projet juste pour choquer. Qui ferait ça ?

Donc, au lieu de coder un interprète Brainfuck, vous coderez un Mindopen interprète. Qu'est-ce que Mindopen, demandez-vous ? C'est une façon d'écrire Brainfuck sans jamais

écrire "fuck".

Tout d'abord, vous lirez sur Brainfuck (Google it). Ensuite, vous définirez votre langue Mindopen en suivant les instructions de Brainfuck et en leur affectant d'autres symboles.

Ensuite, vous écrirez un programme qui effectue les tâches suivantes :


- Ouvrez un fichier contenant du code Mindopen
- Lisez ce fichier et, pour chaque instruction déchiffrée, créez un objet dérivé d'Instruction qui représente l'instruction à exécuter et mettez-le en file d'attente dans ... une file d'attente d'instructions en mémoire.
- Fermez le fichier
- Exécutez chaque instruction en file d'attente

Si cela n'est pas évident, cela signifie que vous devez également créer un ensemble de classes d'instructions, une pour chaque instruction réelle du langage, qui ont toutes une méthode telle que `execute` ou quelque chose qui exécute l'instruction proprement dite. Vous aurez probablement aussi besoin d'une interface pour manipuler toutes ces instructions et les stocker toutes dans le même conteneur ...

Un `main` complet et détaillé sera attendu, ainsi que des fichiers de test qui seront de véritables programmes Mindopen à utiliser.

Chapitre VII

Exercice 04 : In Poland, expression evaluates you

	Exercice : 04
In Poland, expression evaluates you	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : <code>main.cpp</code> + <code>Whatever you need</code>	
Fonctions interdites : <code>Aucune</code>	

Dans ce dernier exercice, vous devez faire un programme qui prend une expression mathématique en argument. Dans cette expression, vous trouverez uniquement des parenthèses, des `int` (qui rentrent dans un `int`) et les opérateurs `+` `-` `/*`.

Vous devez d'abord tokeniser cette expression, i.e. la convertir en un set d'objet dérivés de `Token`, puis les convertir en postfix (aka notation polonaise inversée).

Une fois terminé, vous devez évaluer l'expression, en affichant toutes les étapes sur l'output standard. Par chaque étape, nous voulons dire l'input reçu, l'opération qui en résulte, et le résultat lui-même.

Bien entendu, vous devez gérer les erreurs de manière appropriée. Votre `main` doit être détaillé et facile à lire.

Voici un exemple d'output :

```
$> ./ex04 "3 + ((1 + 4) * 2) - 1"
Tokens: Num(3) Op(+) ParOpen ParOpen Num(1) Op(+) Num(4) ParClose Op(*) Num(2) ParClose Op(-) Num(1)
Postfix : Num(3) Num(1) Num(4) Op(+) Num(2) Op(*) Op(+) Num(1) Op(-)
I Num(3) | OP Push | ST 3]
I Num(1) | OP Push | ST 1 3]
I Num(4) | OP Push | ST 4 1 3]
I Op(+) | OP Add | ST 5 3]
I Num(2) | OP Push | ST 2 5 3]
I Op(*) | OP Multiply | ST 10 3]
I Op(+) | OP Add | ST 13]
I Num(1) | OP Push | ST 1 13]
I Op(-) | OP Subtract | ST 12]
Result : 12
```