



C++ - Module 08

Conteneurs, itérateurs et algorithmes modélisés

Résumé :

Ce document contient les exercices du module 08 des modules C++.

Version : 8

Contenu

I	Introduction	2
II	Règles générales	3
III	Règles spécifiques aux modules	5
IV	Exercice 00 : Recherche facile	6
V	Exercice 01 : Span	7
VI	Exercice 02 : abomination mutante	9
VII	Soumission et évaluation par les pairs	11

Chapitre I

Introduction

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

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

Nous sommes conscients que le C++ moderne est très différent à bien des égards. Si vous voulez devenir un développeur C++ compétent, il ne tient qu'à vous d'aller plus loin après le 42e tronc commun !

Chapitre II Règles générales

Compilation

- Compilez votre code avec c++ et les options -Wall -Wextra -Werror
- Votre code devrait toujours être compilé si vous ajoutez le drapeau -std=c++98

Conventions de formatage et de dénomination

- Les répertoires d'exercices seront nommés de la manière suivante : ex00, ex01, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme l'exigent les lignes directrices.
- Les noms des classes sont écrits en **majuscules**. Les fichiers contenant du code de classe seront toujours nommés en fonction du nom de la classe. Par exemple : Nomdeclasse.hpp/Nomdeclasse.h, Nomdeclasse.cpp, ou Nomdeclasse.hpp. Ainsi, si vous avez un fichier d'en-tête contenant la définition d'une classe "BrickWall" représentant un mur de briques, son nom sera BrickWall.hpp.
- Sauf indication contraire, tous les messages de sortie doivent être terminés par un caractère de nouvelle ligne et affichés sur la sortie standard.
- *Au revoir Norminette !* Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit qu'un code que vos pairs évaluateurs ne peuvent pas comprendre est un code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code propre et lisible.

Autorisé/interdit

Vous ne codez plus en C. Il est temps de passer au C++ ! C'est pourquoi :

- Vous êtes autorisé à utiliser presque tout ce qui se trouve dans la bibliothèque standard. Ainsi, au lieu de vous en tenir à ce que vous connaissez déjà, il serait judicieux d'utiliser autant que possible les versions C++ des fonctions C auxquelles vous êtes habitué.
- Cependant, vous ne pouvez pas utiliser d'autres bibliothèques externes. Cela signifie que les bibliothèques C++11 (et formes dérivées) et Boost sont interdites. Les fonctions suivantes sont également interdites : `*printf()`, `*alloc()` et `free()`. Si vous les utilisez, votre note sera de 0 et c'est tout.

- Notez que, sauf indication contraire, les espaces de noms d'utilisation <ns_name> et les mots-clés amis sont interdits. Dans le cas contraire, votre note sera de -42.
- **Vous n'êtes autorisé à utiliser le STL que dans les modules 08 et 09.** Cela signifie : pas de **conteneurs** (vector/list/map/etc.) et pas d'**algorithmes** (tout ce qui nécessite d'inclure l'en-tête <algorithm>) jusqu'à cette date. Sinon, votre note sera de -42.

Quelques exigences en matière de conception

- Les fuites de mémoire se produisent également en C++. Lorsque vous allouez de la mémoire (en utilisant la fonction new), vous devez éviter les **fuites de mémoire**.
- Du module 02 au module 09, vos cours doivent être conçus selon la **forme canonique orthodoxe, sauf indication contraire explicite**.
- Toute implémentation de fonction placée dans un fichier d'en-tête (à l'exception des modèles de fonction) signifie 0 pour l'exercice.
- Vous devez pouvoir utiliser chacun de vos en-têtes indépendamment des autres. Ils doivent donc inclure toutes les dépendances dont ils ont besoin. Cependant, vous devez éviter le problème de la double inclusion en ajoutant des **gardes d'inclusion**. Dans le cas contraire, votre note sera de 0.

Lisez-moi

- Vous pouvez ajouter des fichiers supplémentaires si nécessaire (par exemple, pour diviser votre code). Comme ces travaux ne sont pas vérifiés par un programme, vous êtes libre de le faire tant que vous rendez les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer des exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez entièrement chaque module avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilisez votre cerveau ! !!



Vous devrez implémenter un grand nombre de classes. Cela peut sembler fastidieux, à moins que vous ne soyez capable d'écrire des scripts dans votre éditeur de texte préféré.



Vous disposez d'une certaine liberté pour réaliser les exercices. Toutefois, respectez les règles obligatoires et ne soyez pas paresseux. Vous passeriez à côté d'un grand nombre d'informations utiles ! N'hésitez pas à vous documenter sur les concepts théoriques.

Chapitre III

Règles spécifiques aux modules

Vous remarquerez que, dans ce module, les exercices peuvent être résolus SANS les conteneurs standard et SANS les algorithmes standard.


Cependant, **leur utilisation est précisément l'objectif de ce module**. Vous êtes autorisé à utiliser la STL. Oui, vous pouvez utiliser les **conteneurs** (vector/list/map/etc.) et les **algorithmes** (définis dans l'en-tête <algorithm>). En outre, vous devriez les utiliser autant que possible. Faites donc de votre mieux pour les appliquer partout où c'est approprié.

Vous obtiendrez une très mauvaise note si vous ne le faites pas, même si votre code fonctionne comme prévu. Ne soyez pas paresseux.

Vous pouvez définir vos modèles dans les fichiers d'en-tête comme d'habitude. Ou, si vous le souhaitez, vous pouvez écrire les déclarations de vos modèles dans les fichiers d'en-tête et écrire leurs implémentations dans les fichiers .tpp. Dans tous les cas, les fichiers d'en-tête sont obligatoires, tandis que les fichiers .tpp sont facultatifs.

Chapitre IV

Exercice 00 : Recherche facile

	Exercice : 00
Trouver facilement	
Annuaire de retournement : <i>ex00/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>main.cpp</code> , <code>easyfind.{h, hpp}</code> et fichier optionnel : <code>easyfind.tpp</code>	
Fonctions interdites : Aucune	

Un premier exercice facile permet de partir du bon pied.

Écrire une fonction template `easyfind` qui accepte un type `T`. Elle prend deux paramètres.

Le premier est de type `T` et le second est un entier.

En supposant que `T` est un conteneur **d'entiers**, cette fonction doit trouver la première occurrence du deuxième paramètre dans le premier paramètre.

Si aucune occurrence n'est trouvée, vous pouvez soit lancer une exception, soit renvoyer une valeur d'erreur de votre choix. Si vous avez besoin d'inspiration, analysez le comportement des conteneurs standard.


Bien entendu, mettez en œuvre et présentez vos propres tests pour vous assurer que tout fonctionne comme prévu.



Vous n'avez pas à gérer les conteneurs associatifs.

Chapitre V

Exercice 01 : Span

	Exercice : 01
La portée	
Répertoire de retournement : <i>ex01/</i>	
Fichiers à rendre : <i>Makefile</i> , <i>main.cpp</i> , <i>Span.{h, hpp}</i> , <i>Span.cpp</i>	
Fonctions interdites : Aucune	

Développez une classe **Span** qui peut stocker un maximum de N entiers. N est une variable int non signée et sera le seul paramètre passé au constructeur.

Cette classe aura une fonction membre appelée `addNumber()` pour ajouter un seul nombre à la Span. Elle sera utilisée pour la remplir. Toute tentative d'ajout d'un nouvel élément s'il y a déjà N éléments stockés doit lever une exception.

Ensuite, implémentez deux fonctions membres : `shortestSpan()` et `longestSpan()`

Ils trouveront respectivement l'étendue la plus courte ou la plus longue (ou la distance, si vous préférez) entre tous les nombres stockés, et la renverront. S'il n'y a pas de nombres stockés, ou s'il n'y en a qu'un seul, aucun intervalle ne peut être trouvé. Une exception est alors levée.

Bien sûr, vous écrirez vos propres tests et ils seront bien plus approfondis que ceux présentés ci-dessous. Testez votre Span avec un minimum de 10 000 numéros. Plus serait encore mieux.

Exécution de ce code :

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

    sp.addNumber(6) ;
    sp.addNumber(3) ;
    sp.addNumber(17) ;
    sp.addNumber(9) ;
    sp.addNumber(11) ;

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

    retour 0 ;
}
```

Devrait sortir :

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


Enfin, il serait merveilleux de pouvoir remplir votre Span en utilisant une **gamme d'itérateurs**. Faire des milliers d'appels à `addNumber()` est tellement ennuyeux. Implémentez une fonction membre pour ajouter plusieurs nombres à votre Span en un seul appel.



Si vous n'avez pas la moindre idée, étudiez les conteneurs. Certaines fonctions membres prennent une série d'itérateurs afin d'ajouter une séquence d'éléments au conteneur.

Chapitre VI

Exercice 02 : abomination mutante

	Exercice : 02
Abomination mutante	
Annuaire de retournement : <i>ex02/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>main.cpp</code> , <code>MutantStack.{h, hpp}</code> et fichier optionnel : <code>MutantStack.tpp</code>	
Fonctions interdites : Aucune	

Maintenant, il est temps de passer à des choses plus sérieuses. Développons quelque chose de bizarre.

Le conteneur `std::stack` est très agréable. Malheureusement, c'est l'un des seuls conteneurs STL qui n'est PAS itérable. C'est dommage.

Mais pourquoi l'accepterions-nous ? Surtout si nous pouvons prendre la liberté de massacrer la pile d'origine pour créer des fonctionnalités manquantes.

Pour réparer cette injustice, il faut rendre le conteneur `std::stack` itérable.

Ecrivez une classe **MutantStack**. Elle sera implémentée en termes de `std::stack`.

Il offrira toutes les fonctions de ses membres, ainsi qu'une fonctionnalité supplémentaire : les **itérateurs**.

Bien entendu, vous écrirez et remettrez vos propres tests pour vous assurer que tout fonctionne comme prévu.

Vous trouverez ci-dessous un exemple de test.

```
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) ;
    retour 0 ;
}
```

Si vous l'exécutez une première fois avec votre MutantStack, et une seconde fois en remplaçant la MutantStack par, par exemple, une std::list, les deux sorties devraient être les mêmes. Bien sûr, lorsque vous testez un autre conteneur, mettez à jour le code ci-dessous avec les fonctions membres correspondantes (push() peut devenir push_back()).

Chapitre VII

Soumission et évaluation par les pairs

Rendez votre travail dans votre dépôt Git comme d'habitude. Seul le travail contenu dans votre dépôt sera évalué lors de la soutenance. N'hésitez pas à vérifier les noms de vos dossiers et fichiers pour vous assurer qu'ils sont corrects.



```
16D85ACC441674FBA2DF65195A38EE36793A89EB04594B15725A1128E7E97B0E7B47
111668BD6823E2F873124B7E59B5CE94B47AB764CF0AB316999C56E5989B4B4F00C
91B619C70263F
```