



# C++ - Module 08

Conteneurs, itérateurs et algorithmes basés sur des modèles

Résumé :

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

Version : 10.0

# Contenu

I	Introduction	2
II	Règles générales	3
III	Règles spécifiques au module	6
IV	Instructions AI	7
	Exercice 00 : Recherche facile	9
VI	Exercice 01 : Portée	10
VII	Exercice 02 : Abomination mutée	12
VIII	Soumission et évaluation par les pairs	14

# Chapitre I

## Introduction

C++ est un langage de programmation à usage général créé par Bjarne Stroustrup comme une extension du langage de programmation C, ou « C avec classes » (source : [Wikipédia](#)).

L'objectif de ces modules est de vous initier à la programmation orientée objet.

Ce sera le point de départ de votre apprentissage du C++. De nombreux langages sont recommandés pour apprendre la POO. Nous avons choisi le C++, car il est dérivé du C.

Parce qu'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 souhaitez devenir un développeur C++ compétent, il vous appartient de poursuivre vos études après le Common Core 42 !

## Chapitre II

### Règles générales

#### Compilation

- Compilez votre code avec `c++` et les indicateurs `-Wall -Wextra -Werror`
- Votre code devrait toujours être compilé si vous ajoutez l'indicateur `-std=c++98`

#### Conventions de formatage et de dénomination

- Les répertoires d'exercices seront nommés de cette façon : `ex00`, `ex01`, ... , `exn`
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les lignes directrices.
- Écrivez les noms de classe en majuscules (UpperCamelCase) . Les fichiers contenant du code de classe seront toujours nommés selon le nom de la classe. Par exemple : `ClassName.hpp/ClassName.h`, `ClassName.cpp` ou `ClassName.tpp`. 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, chaque message de sortie doit se terminer par un caractère de nouvelle ligne et être affiché sur la sortie standard.
- Adieu Norminette ! Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit que le code que vos pairs ne comprennent pas est du code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code clair et lisible.

#### Autorisé/Interdit

Vous ne codez plus en C. Passez au C++ ! Par conséquent :

- Vous pouvez utiliser presque tout ce qui est proposé dans la bibliothèque standard. Ainsi, plutôt que 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 utiliser aucune autre bibliothèque externe. Cela signifie que C++11 (et ses dérivés) et les bibliothèques Boost sont interdits. Les fonctions suivantes sont également interdites : `*printf()`, `*alloc()` et `free()`. Si vous les utilisez, votre note sera de 0, point final.

- Notez que, sauf indication contraire explicite, l'espace de noms d'utilisation `<ns_name>` et Les mots-clés amis sont interdits. Sinon, votre note sera de -42.
- Vous êtes autorisé à utiliser le STL uniquement dans les modules 08 et 09. Cela signifie : pas de conteneurs (vecteur/liste/carte, etc.) ni d'algorithmes (tout ce qui nécessite l'inclusion de l'en-tête `<algorithm>`) jusqu'à cette date. Sinon, votre note sera de -42.

#### Quelques exigences de conception

- Les fuites de mémoire se produisent également en C++. Lorsque vous allouez de la mémoire (en utilisant le nouveau (mot-clé), vous devez éviter les fuites de mémoire.
- Du module 02 au module 09, vos cours doivent être conçus dans le style orthodoxe Forme canonique, 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 nécessaires. Cependant, vous devez éviter le problème de double inclusion en ajoutant des gardes d'inclusion. Sinon, votre note sera de 0.

#### Lis-moi

- Vous pouvez ajouter des fichiers supplémentaires si nécessaire (par exemple, pour fractionner votre code). Ces devoirs n'étant pas vérifiés par un programme, n'hésitez pas à le faire, à condition de fournir les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez chaque module en entier avant de commencer ! Vraiment, foncez.
- Par Odin, par Thor ! Utilisez votre cerveau !!!



Concernant le Makefile pour les projets C++, les mêmes règles qu'en C s'appliquent (voir le chapitre Norme sur le Makefile).



Vous devrez implémenter de nombreuses classes. Cela peut paraître fastidieux, à moins que vous ne puissiez écrire des scripts dans votre éditeur de texte préféré.



Vous disposez d'une certaine liberté pour réaliser les exercices.

Cependant, suivez les règles obligatoires et ne soyez pas paresseux. Vous

manquez beaucoup d'informations utiles ! N'hésitez pas à lire

concepts théoriques.

# Chapitre III

## Règles spécifiques au module

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

Cependant, leur utilisation est précisément le but de ce module.

Vous devez utiliser la STL — en particulier les conteneurs (vecteur/liste/carte/etc.) et les algorithmes (définis dans l'en-tête `<algorithm>`) — chaque fois qu'ils sont appropriés.

De plus, vous devriez les utiliser autant que possible.

Faites donc de votre mieux pour les appliquer là où cela est approprié.

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

# Chapitre IV

## Instructions de l'IA

- Context

Ce projet est conçu pour vous aider à découvrir les éléments fondamentaux de votre formation en TIC.

Pour ancrer correctement les connaissances et les compétences clés, il est essentiel d'adopter une approche réfléchie de l'utilisation des outils et du support de l'IA.

Un véritable apprentissage fondamental nécessite un véritable effort intellectuel — par le biais de défis, de répétitions et d'échanges d'apprentissage entre pairs.

Pour un aperçu plus complet de notre position sur l'IA — en tant qu'outil d'apprentissage, en tant qu'élément du programme des TIC et en tant qu'attente sur le marché du travail — veuillez vous référer à la FAQ dédiée sur l'intranet.

- Message principal

Construisez des bases solides sans raccourcis.

Développez réellement vos compétences techniques et énergétiques.

Vivez un véritable apprentissage entre pairs, commencez à apprendre à apprendre et à résoudre de nouveaux problèmes.

Le parcours d'apprentissage est plus important que le résultat.

Apprenez-en davantage sur les risques associés à l'IA et développez des pratiques de contrôle efficaces et des contre-mesures pour éviter les pièges courants.

- Règles de l'apprenant :

- Vous devez appliquer le raisonnement aux tâches qui vous sont assignées, en particulier avant de vous tourner vers l'IA.



- Vous ne devez pas demander de réponses directes à l'IA.
- Vous devriez en apprendre davantage sur l'approche globale 42 de l'IA.

## • Résultats de la phase :

Au cours de cette phase fondamentale, vous obtiendrez les résultats suivants :

- Obtenez des bases techniques et de codage appropriées.
- Sachez pourquoi et comment l'IA peut être dangereuse durant cette phase.

## • Commentaires et exemple :

- Oui, nous savons que l'IA existe et qu'elle peut résoudre vos projets. Mais vous êtes ici pour apprendre, pas pour prouver que l'IA a appris. Ne perdez pas votre temps (ni le nôtre) à simplement démontrer que l'IA peut résoudre un problème donné.

Apprendre à 42 ans ne consiste pas à connaître la réponse, mais à développer la capacité d'en trouver une. L'IA vous donne directement la réponse, mais cela vous empêche de construire votre propre raisonnement. Et le raisonnement demande du temps, des efforts et implique des échecs. Le chemin vers le succès n'est pas censé être facile.

- Gardez à l'esprit que pendant les examens, l'IA n'est pas disponible : pas d'Internet, pas de smartphones, etc. Vous vous rendrez vite compte si vous vous êtes trop appuyé sur l'IA dans votre processus d'apprentissage.
- L'apprentissage entre pairs vous expose à des idées et des approches différentes, améliorant ainsi vos compétences interpersonnelles et votre capacité à penser différemment. C'est bien plus précieux que de simplement discuter avec un robot. Alors n'hésitez pas : discutez, posez des questions et apprenez ensemble !
- Oui, l'IA sera intégrée au programme, à la fois comme outil d'apprentissage et comme sujet à part entière. Vous aurez même la possibilité de créer votre propre logiciel d'IA. Pour en savoir plus sur notre approche crescendo, vous consulterez la documentation disponible sur l'intranet.

### Bonnes pratiques :


Je suis bloqué sur un nouveau concept. Je demande à quelqu'un à proximité comment il l'a abordé. On discute 10 minutes, et soudain, ça fait tilt. Je comprends.

### Mauvaise pratique :

J'utilise l'IA en secret, je copie du code qui semble correct. Lors de l'évaluation par les pairs, je ne peux rien expliquer. J'échoue. À l'examen, sans IA, je suis à nouveau bloqué. J'échoue.

## Chapitre V

### Exercice 00 : Recherche facile

	Exercice : 00
Facile à trouver	
Répertoire de remise : ex00/	
Fichiers à remettre : Makefile, main.cpp, easyfind.{h, hpp} et fichier optionnel : easyfind.tpp Fonctions interdites : Aucune	

Un premier exercice facile est le moyen de démarrer du bon pied.

Écrivez un modèle de fonction `easyfind` qui accepte un type `T`. Il 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 lever une exception, soit renvoyer une valeur d'erreur de votre choix. Besoin d'inspiration ? Analysez le comportement des conteneurs standards.


Bien sûr, implémentez et remettez vos propres tests pour vous assurer que tout fonctionne comme prévu.



Vous n'avez pas besoin de gérer des conteneurs associatifs.

# Chapitre VI

## Exercice 01 : Portée

	Exercice : 01
Portée	
Répertoire de remise : ex01/	
Fichiers à remettre : Makefile, main.cpp, Span.{h, hpp}, Span.cpp Fonctions interdites :	
Aucune	

Développez une classe Span pouvant stocker un maximum de N entiers. N est une variable de type int non signé et sera le seul paramètre passé au constructeur.

Cette classe possède une fonction membre appelée addNumber() pour ajouter un nombre unique à la plage. Elle sera utilisée pour la remplir. Toute tentative d'ajout d'un nouvel élément si N éléments sont déjà stockés génère une exception.

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

Ils détermineront respectivement l'intervalle le plus court ou le plus long (ou la distance, si vous préférez) entre tous les nombres stockés, et les renverront. S'il n'y a aucun nombre stocké, ou un seul, aucun intervalle ne peut être trouvé. Une exception sera alors générée.

Bien sûr, vous écrierez vos propres tests, et ils seront beaucoup plus complets que les ceux ci-dessous. Testez votre Span avec au moins 10 000 nombres. Plus serait encore mieux.

Exécution de ce code :

```
int main() {  
  
    Portée sp = Portée(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;  
  
    retourner 0;  
}
```

Devrait afficher :

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


Enfin et surtout, il serait formidable de remplir votre Span en utilisant une gamme d'itérateurs. Effectuer des milliers d'appels à `addNumber()` est vraiment fastidieux. Implémentez une fonction membre pour ajouter plusieurs nombres à votre Span en un seul appel.



Si vous n'avez pas d'idée, étudiez les conteneurs. Certaines fonctions membres utilisent une série d'itérateurs pour ajouter une séquence de éléments au conteneur.

## Chapitre VII

### Exercice 02 : Abomination mutée

	Exercice : 02
Abomination mutée	
Répertoire de remise : ex02/	
Fichiers à remettre : Makefile, main.cpp, MutantStack.{h, hpp} et fichier optionnel : MutantStack.tpp Fonctions interdites : Aucune	

Maintenant, il est temps de passer aux choses sérieuses. Développons quelque chose d'étrange.

Le conteneur `std::stack` est très pratique. Malheureusement, c'est l'un des seuls conteneurs STL qui ne soit pas itérable. C'est dommage.

Mais pourquoi accepterions-nous cela ? Surtout si nous pouvons nous permettre de massacrer pile d'origine pour créer les fonctionnalités manquantes.

Pour réparer cette injustice, vous devez rendre le conteneur `std::stack` itérable.

Écrivez une classe `MutantStack`. Elle sera implémentée selon un `std::stack`. Il offrira toutes ses fonctions membres, plus une fonctionnalité supplémentaire : les itérateurs.

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

Trouvez un exemple de test ci-dessous.

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

Si vous l'exécutez une première fois avec votre MutantStack, puis une seconde fois en remplaçant ce dernier par, par exemple, une std::list, les deux sorties devraient être identiques. Bien entendu, lors du test d'un autre conteneur, mettez à jour le code ci-dessous avec les fonctions membres correspondantes (push() peut devenir push\_back()).

## Chapitre VIII

# Soumission et évaluation par les pairs

Remettez votre devoir dans votre dépôt Git comme d'habitude. Seul le travail effectué dans vos dépôts 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.

Lors de l'évaluation, une brève modification du projet peut être occasionnellement demandée. Il peut s'agir d'un changement de comportement mineur, de quelques lignes de code à écrire ou à réécrire, ou d'une fonctionnalité facile à ajouter.

Bien que cette étape ne soit pas applicable à tous les projets, vous devez vous y préparer si elle est mentionnée dans les directives d'évaluation.

Cette étape vise à vérifier votre compréhension réelle d'une partie spécifique du projet.

La modification peut être effectuée dans n'importe quel environnement de développement de votre choix (par exemple, votre configuration habituelle) et devrait être réalisable en quelques minutes, à moins qu'un délai spécifique ne soit défini dans le cadre de l'évaluation.

On peut par exemple vous demander d'effectuer une petite mise à jour d'une fonction ou d'un script, de modifier un affichage ou d'ajuster une structure de données pour stocker de nouvelles informations, etc.

Les détails (portée, cible, etc.) seront précisés dans les lignes directrices d'évaluation et peuvent varier d'une évaluation à l'autre pour un même projet.



16D85ACC441674FBA2DF65195A38EE36793A89EB04594B15725A1128E7E97B0E7B47  
111668BD6823E2F873124B7E59B5CE94B47AB764CF0AB316999C56E5989B4B4F00C  
91B619C70263F