



ft_containers

Les containers c++ c'est facile!

Résumé: Les multiples conteneurs disponibles en C++ ont tous des usages très différents. Pour être sûr de les comprendre, réimplémentons-les!

Version: 4

Table des matières

I	Règles Générales	2
II	Introduction	4
III	Partie obligatoire	5
IV	Partie bonus	7

Chapitre I

Règles Générales

- Toute fonction déclarée 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 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.
 - L'objectif de ce projet est de vous faire recoder les containers de la STL. Il vous est évidemment interdit de les utiliser.
- 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 -42.
- 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 attentivement. 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 et ses dérivés, ni Boost.
- 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

Introduction

Dans ce projet, vous allez devoir ré-implémenter les différents containers de la C++ STL (standard template library).

Pour chaque container, vous devez rendre des fichiers de classe nommés correctement.

Vous utiliserez le namespace `ft`, et vos containers seront appelés à l'aide de `ft::<container>`.

Vous devez respecter la structure de votre container de reference. N'implementez que les éléments présents.

Nous vous rapellons que vous codez en C++98, donc toutes les nouvelles fonctions des containers **NE DOIVENT PAS** être implémentées. Vous devez par contre implémenter les anciennes (même deprecated).

Chapitre III

Partie obligatoire

- Implémentez les containers suivants, et rendez les fichiers `<container>.hpp` nécessaires.
- Vous devez également rendre un fichier `main.cpp` qui teste votre rendu en vue des évaluations. (aller plus loin que dans les exemples!)
- Vous devez produire un binaire avec uniquement vos conteneurs et un autre avec le même test avec les conteneurs STL.
- Comparer les sorties et le temps (vous pouvez être jusqu'à 20 fois plus lent).
- Les fonctions membres, les non-membres et les overloads sont attendues.
- respectez le nommage, faites attention aux détails.
- Vous devez utiliser `std::allocator`.
- Vous devez justifier votre structure de données interne pour chaque conteneur (utiliser un simple tableau pour une map n'est pas acceptable).
- Si le conteneur a un système d'itérateur, vous devez l'implémenter.
- `iterators_traits`, `reverse_iterator`, `enable_if`, `is_integral`, `equal/lexicographical_compare`, `std::pair`, `std::make_pair`, doivent être réimplémenté.
- Vous pouvez utiliser <https://www.cplusplus.com/> et <https://cppreference.com/> comme références.
- Vous ne pouvez pas implémenter plus de fonctions publiques que celles proposées dans les conteneurs standard. Tout le reste doit être privé ou protégé. Chaque fonction/variable publique doit être justifiée.
- Pour les overloads non-membres, le mot-clé `friend` est autorisé. Chaque utilisation de `friend` doit être justifiée et sera vérifiée lors de l'évaluation.

Vous devez rendre les containers suivants et leur fonctions associés :

- Vector
- Map

- Stack

Pour votre implémentation vectorielle, il n'est pas obligatoire de coder la spécialisation `vector<bool>`.

Votre pile utilisera votre classe vectorielle comme conteneur sous-jacent par défaut, il doit toujours être compatible avec d'autres conteneurs comme celui de STL.

Les conteneurs STL sont interdits.

Vous êtes autorisé à utiliser la bibliothèque STD.

Chapitre IV

Partie bonus

Si vous avez complètement fini la partie obligatoire, vous pouvez essayer de rendre la partie bonus. Réimplémentez les containers suivants :

- Set - Mais cette fois un arbre Noir - Rouge est obligatoire.