TP 2 : Créer une classe comme `std::vector<T>`

Prénom : _____ **Nom :** _____

1 Introduction

Tous les développements se feront dans le *namespace* `local`.
On cherche à écrire en C++11 une classe qui fonctionne comme `std::vector<T>` : http://en.cppreference.com/w/cpp/container/vector

1.1 Quelle est la solution, disponible dans la bibliothèque standard, la plus adaptée pour stocker les données ?

1.2 Pourquoi cette solution n'est pas adaptée au problème ? Et quelle est la solution si on règle le problème dans notre classe `vector<T>` ?

2 `copy_ptr<T>`

On cherche à écrire une classe qui reprend le fonctionnement de la solution (imparfaite dans notre cas) venant la bibliothèque standard et à ajouter les fonctionnalités qui seront nécessaires pour notre classe `vector<T>`.

Écrire cette classe (compléter le fichier `include/copy_ptr.hpp`) (sans la spécialisation) et décommenter les tests au fur et à mesure dans `tests/copy_ptr.cpp` (ne pas hésitez à faire d'autres tests).

2.1 Pourquoi la classe `copy_ptr<T>` n'est pas adaptée si `T` est un `T []` ? Corriger le problème en faisant une spécialisation. Quelles sont les fonctionnalités basiques des tableaux à rajouter dans notre spécialisation.

3 `vector<T>`

On peut maintenant facilement écrire notre classe `vector<T>`.

Écrire cette classe (compléter le fichier `include/vector.hpp`) et décommenter les tests au fur et à mesure dans `tests/vector.cpp`.

3.1 Que faut-il ajouter à notre classe pour qu'elle fonctionne avec les *for range loop* ou `std::sort` ?

3.2 Dans le fichier `tests/vector.cpp`, quel mécanisme utilise la fonction `test_vector` pour exécuter des tests différents si les éléments du vecteurs sont arithmétiques ?

Ajouter les fonctions membres `.push_back(T const &)` et `.emplace_back(Args const & ...)` (sans construire l'élément en place) (vous pouvez utiliser des références universelles). Tester les en complétant le test `tests/vector.cpp`.

3.3 Si on fait beaucoup d'ajout, les performances seront (très) mauvaises. Quelle est la solution utilisée dans `std::vector<T>` pour régler ce problème ? Implémenter cette solution et tester la.

3.4 Bonus : compléter la classe `vector<T>` pour rajouter les fonctions membres manquantes (`.front()`, `.back()`, `.empty()`, `.resize()`, `.insert()`, ...)

4 Conlusion & Place libre (remarques, impressions, fin de réponse...)

Remarque : La signature complète de `std::vector<T>` est `std::vector<T, allocator_t = std::allocator<T>`. Pour en savoir plus sur le concept d'allocateur : http://en.cppreference.com/w/cpp/concept/Allocator