UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

IFT 339

Laboratoire#2 : La classe vector

Hiver 2019

Le but de ce laboratoire est de pratiquer les classes, l'allocation dynamique, et aussi d'apprendre à lire et compléter le code écrit par quelqu'un d'autre. Vous devez compléter une class `vector` à peu près équivalente fonctionnellement à celle de la SL (sans inclure toutes les fonctions de la classe `vector` de la bibliothèque standard (SL)). On vous fournit également une classe `iterator`, qui permet d'utiliser des itérateurs de tableaux en plus des indices pour accéder à un élément d'un tableau. La représentation d'un itérateur est un pointeur vers l'élément concerné.

Ce laboratoire devra être complété avant le vendredi 15 février 2019 à 23h59. Vous devez remettre, sur `opus.dinf.usherbrooke.ca`, les fichiers générés au cours de ce laboratoire.

Description de la tâche à réaliser

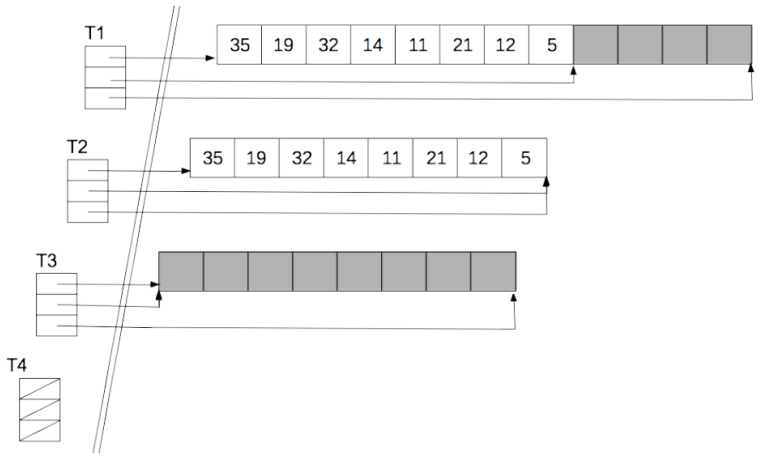
On vous fournit le code de base d'une classe générique `vector` séparé dans deux fichiers. Le premier, `vector.h`, contient les définitions des fonctions déjà codées, et appelle l'inclusion de l'autre, `vector2.h`, qui contient les entêtes des fonctions que vous devez coder. Le premier fichier appelle automatiquement le second, vous n'avez donc qu'un *include* de `vector.h` à faire dans un programme principal `main.cpp` qui utilise cette classe (vous devez écrire votre propre programme `main.cpp` pour tester votre code). La technique de représentation choisie est par pointeurs vers des éléments d'un tableau contigu alloué dynamiquement (c'est la représentation usuelle de la SL). Notez qu'avec des classes gabarits, tout le code est dans le fichier d'extension `.h`, il n'y a pas de fichier `.cpp`.

Les fonctions de la classe `vector` que vous devez coder sont les suivantes. Assurez-vous de bien traiter et tester TOUS les cas possibles.

- `constructeur de recopie`
- `resize` augmente ou diminue la dimension du vecteur et fait appel à `reserve` au besoin.
- `reserve` augmente au besoin la capacité du vector mais ne la diminue pas. En cas d'augmentation de capacité, la fonction `reserve` alloue le nouvel espace et fait les copies. En cas de diminution, la capacité courante est conservée, sans faire de réallocation. Seules les fonctions `clear` et `shrink_to_fit` réduisent effectivement la capacité.
- `push_back` augmente la dimension du vector de 1 et s'assure au besoin que les n prochains `push_back` seront O(1), n étant la dimension actuelle du vecteur.

— `pop_back` réduit la dimension du vector de 1.
— `operator[]` retourne une référence à un élément.
— `shrink_to_fit` réduit la capacité du vector à sa dimension. Si la capacité est plus grande que la dimension, elle alloue un nouvel espace et fait les copies.
— `clear` vide la mémoire dynamique et réduit la capacité à 0.

Voir les définitions déjà fournies dans le fichier `vector.h`. Il y a trois membres dans la classe `vector` : DEBUT, FIN_DIM et FIN_CAP. Ce sont des pointeurs à l'intérieur du tableau alloué dynamiquement. DEBUT est le début du tableau. FIN_DIM pointe immédiatement à la fin du dernier élément, et FIN_CAP pointe à la fin de l'espace dynamique réservé. La figure ci-dessous illustre bien la différence entre la fin et le dernier élément.



T1 est le cas général, où on dispose d'une capacité totale de 12 : la dimension est de 8, et il y a 4 éléments en réserve. T2 est un cas particulier où il n'y a pas de réserve. On voit que FIN_DIM et FIN_CAP pointent à la même adresse (c'est le résultat du `shrink_to_fit()` par exemple). T3 est un vector vide (son size() est à 0). Mais il a une réserve de 8 éléments pour des resize éventuels. T4 est un vector vide sans réserve (c'est le résultat du constructeur sans paramètre ou du `clear()`).

Inspirez-vous des fonctions déjà codées dans `vector.h` pour faire du code cohérent avec ce qui existe déjà. Vous devriez adopter un plan systématique de codage et de test. Vous devriez d'abord tout concevoir sur papier. Puis, coder les fonctions une par une dans un ordre qui vous permet de les tester de façon aussi indépendante que possible. Voici un tel ordre que vous pourriez adopter : 1. `clear`, 2. `operator[]`, 3. `le constructeur de recopie`, 4. `pop_back`, 5. `shrink_to_fit` (suite à une succession de `pop_back`), 6. `push_back` avec une réserve existante (suite à un `pop_back`), 7. `reserve` qui augmente la réserve, 8. `resize` qui réduit la dimension, qui l'augmente sans dépasser la réserve existante, et qui l'augmente au-delà de la réserve, 9. `push_back` sans réserve existante.

Il faut aussi être capable d'utiliser un `vector<vector<double> >`, ou d'autres structures du même genre, et non uniquement des tableaux de types primitifs. Utilisez la fonction `afficher` pour tester la conformité de votre code avec la spécification. Elle vous donne le plus d'information possible sur l'état de votre tableau, pour la mise au point. Elle affiche les 10 premiers éléments et les 10 derniers. Elle est même capable d'afficher un `vector<vector<double> >`.

Remise du travail

Soumettez votre code avant 23h59, le vendredi 15 février 2019, après vous être assuré qu'il fonctionne bien sur le serveur `grimpar.dinf.usherbrooke.ca` (vous pouvez vous connecter et transférer vos fichiers sur ce serveur par `ssh` et `scp` depuis un terminal).

Ne remettez qu'un seul fichier dont le nom doit être `vector2.h`. Ne remettez pas d'autre fichier ni surtout d'exécutable. Votre fichier sera recompilé. Il ne doit y avoir qu'une seule remise par équipe, à partir d'un seul CIP. Les noms des co-équipiers doivent être clairement indiqués en entête de chaque fichier soumis

Pour soumettre votre travail, connectez-vous, dans un navigateur, au serveur `opus.dinf.usherbrooke.ca` en utilisant l'un de vos CIP, puis choisissez le cours `IFT339` et le projet `labo2`. Chargez votre fichier `vector2.h` et soumettez-le.