UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

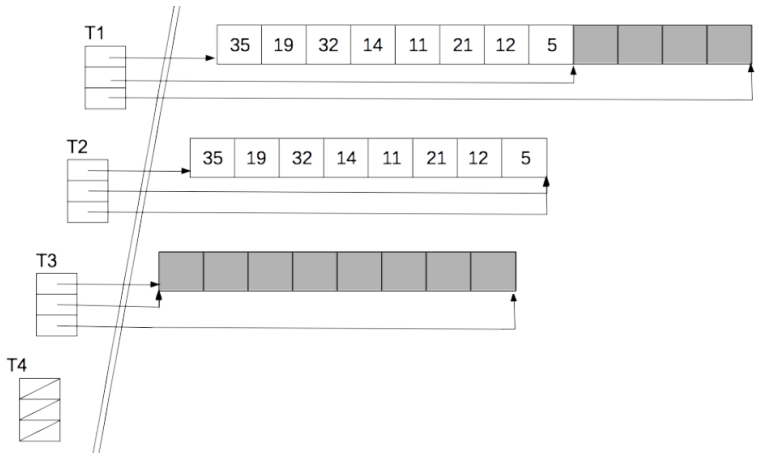
IFT 339

Laboratoire#2 : Types `vector` et `deque`

Le but de ce laboratoire est de pratiquer les classes, l'allocation dynamique, les pointeurs, ainsi que la lecture et la complétion d'un code écrit par quelqu'un d'autre. Vous devez compléter deux classes `vector` et `deque` à peu près équivalente fonctionnellement à celles de la bibliothèque standard SL (sans inclure toutes les fonctions des classes).

Ce devoir est à faire en équipe de deux obligatoirement. Il devra être complété avant le vendredi 13 février 2026 à 23h59. Vous devez remettre, sur `turnin.dinf.usherbrooke.ca`, les fichiers générés au cours de ce laboratoire.

Description de la tâche à réaliser
Classe `vector`



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). Notez qu'avec des classes génériques, il n'y a pas de fichier `.cpp` et tout le code est dans le fichier `.h`. 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). La

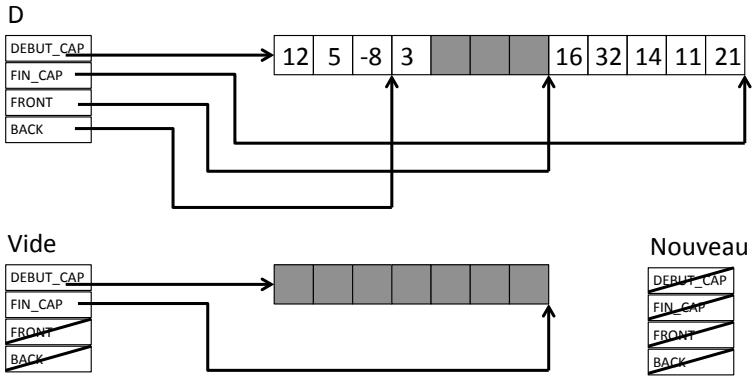
classe `vector` a trois attributs, DEBUT, FIN_DIM et FIN_CAP, qui sont des pointeurs. DEBUT pointe au début du tableau alloué dynamiquement. FIN_DIM pointe à la fin du dernier élément, et FIN_CAP pointe à la fin du tableau alloué dynamiquement. La figure ci-dessus illustre bien la différence entre la fin du dernier élément et la fin du tableau alloué dynamiquement.

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

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 avec paramètre`
- `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 fonction `reserve` ne fait rien. Notez l'utilisation de la fonction `reserve` dans les fonctions `resize` et dans `push_back`. Si la capacité du vector est atteinte, la fonction `push_back` fait appel à `reserve` pour que la capacité devienne le double de la dimension.
- `back x 2` retourne une référence au dernier élément.
- `front x 2` retourne une référence au premier élément.
- `operator[] x 2` retourne une référence à un élément.

Classe `deque`



On vous fournit le code de base d'une classe générique `deque`, séparé dans deux fichiers. Le premier, `deque.h`, contient les définitions des fonctions déjà codées, et appelle l'inclusion de l'autre, `deque2.h`, qui contient les entêtes des fonctions que vous devez coder. Le `deque` possède toutes les fonctionnalités du `vector`, et permet en plus d'ajouter des éléments au début. Il possède, comme le `vector`, une fonction `push_back` qui ajoute un élément à la fin, mais il possède aussi la fonction `push_front` qui en ajoute un au début. Ces deux fonctions doivent s'exécuter en temps O(1) amorti. Comme pour le `vector`, la technique de représentation choisie est par pointeurs vers des éléments d'un tableau contigu alloué dynamiquement, avec une zone commune pour les `push_back` et `push_front` (cette représentation est différente de celle la SL qui prévoit deux zones de mémoire

de réserve différentes pour effectuer les `push_front` et les `push_back`). La fonction `push_front` ajoute un élément avant le premier élément. Si le premier élément est au début du tableau alloué, l'ajout se fait juste avant la fin du tableau alloué. La fonction `push_back` ajoute un élément après le dernier élément. Si le dernier élément est juste avant la fin du tableau alloué, l'ajout se fait au début du tableau alloué.

La classe `deque` a quatre attributs : DEBUT_CAP, FIN_CAP, FRONT et BACK. Ce sont des pointeurs. DEBUT_CAP pointe au début du tableau alloué dynamiquement. FIN_CAP pointe à la fin du tableau alloué dynamiquement. FRONT pointe au début du premier élément du deque. BACK pointe au début du dernier élément du deque. La figure ci-dessus illustre bien la représentation. C'est ce que l'on appelle une implantation circulaire.

`D` est un cas général : on dispose d'une capacité totale de 12, la dimension est de 9, et la réserve est de 3. Le premier élément du deque est 16, et le dernier 3. La configuration actuelle du deque D pourrait être le résultat de `push_back` successifs des éléments 12, 5, -8 et 3, suivis de `push_front` successifs des éléments de valeurs 21, 11, 14, 32 et 16. `Vide` est un deque vide : son size() est à 0, mais il a une réserve de 7. Il pourrait être le résultat d'un `resize(0)`, ou d'un `pop_front` ou un `pop_back` sur un deque de dimension 1. `Nouveau` est un deque vide sans réserve : c'est le résultat du constructeur sans paramètre ou du `clear()`.

Inspirez-vous des opérateurs et classes déjà codés 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, codez les fonctions une par une dans un ordre qui vous permet de les tester de façon aussi indépendante que possible. Une fois votre programme complété, vous devez vous assurer qu'il compile et fonctionne bien sous Linux en le testant sur les machines du Département d'informatique..

Remise du travail

Pour soumettre votre travail, connectez-vous, dans un fureteur, au serveur `http://turnin.dinf.usherbrooke.ca` en utilisant votre CIP, puis choisissez le cours `IFT339` et le projet `TP2`. Chargez vos fichiers `vector2.h` et `deque2.h` et soumettez-les. Indiquez bien les noms des deux membres de l'équipe en commentaire dans ces fichiers. Ne faites qu'une seule soumission par équipe. Ne remettez pas d'autre fichier, ni d'exécutable. Vos fichiers de code seront intégrés à un programme de test contenant déjà les autres fichiers du programme. Vous n'avez donc pas à re-soumettre ces derniers.

Barême

- 25 points pour soumission réussie d'un programme qui compile sans erreur par une équipe de deux
- 10 points pour respect des normes de programmation (se référer au document sur les normes de programmation sur le site web du cours)
- 40 points pour le respect de la conception et des instructions fournies (spécifications des opérateurs et instructions de remise)
- 25 points la complétion correcte du code (5 points pour `vector2.h` et 20 points pour `deque2.`)