UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

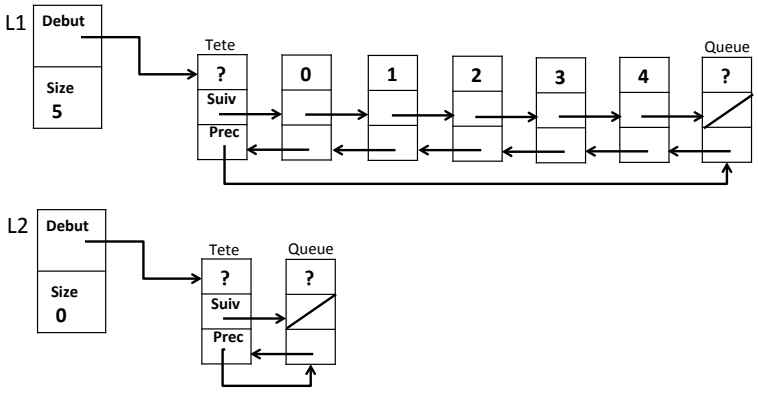
IFT 339

Laboratoire#3 : Type `list`

Le but de ce laboratoire est de continuer à pratiquer l'implantation de TAD fonctionnellement équivalents à ceux de la SL. Vous devez compléter une classe `list` à peu près équivalente fonctionnellement à celle de la bibliothèque standard (SL) de C++.

Ce devoir est à faire en équipe de deux obligatoirement. Il devra être complété avant le vendredi 14 mars 2025 à 23h59. Vous devez remettre, sur `turnin.dinf.usherbrooke.ca`, le fichier généré au cours de ce laboratoire.

Description de la tâche à réaliser



On vous fournit le code de base d'une classe générique `list` séparé dans deux fichiers. Le premier, `list.h`, contient les définitions des fonctions déjà codées, et appelle l'inclusion de l'autre, `list2.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 `list.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 `list` est une structure linéaire où l'insertion et l'élimination d'un élément peuvent se faire en temps constant. Mais la recherche par le contenu et la localisation d'un élément par son numéro prennent un temps O(n) car on n'a aucune façon de localiser la position d'un élément donné par un simple calcul comme pour les tableaux.

La technique de représentation est par cellules doublement chaînées, en utilisant une cellule de tête et une cellule de queue pour unifier toutes les positions dans la `list`. La `list` dispose de deux classes d'itérateurs `iterator` et `reverse_iterator`. La représentation d'un itérateur est un pointeur vers l'élément concerné. La position de début de `iterator` est la première cellule de la `list` et sa position de fin est la cellule de queue. Pour `reverse_iterator`, la position de début est la dernière cellule et la position de fin est la cellule de tête. La figure ci-dessus illustre la représentation. `L1` est une `list` générale de 5 éléments. `L2` est une `list` vide : c'est le résultat du constructeur sans paramètre ou du `clear()`.

Nous avons deux fonctions d'insertion dans la `list`, une avec un `iterator` et une autre avec un `reverse_iterator`. De même, il y a deux fonctions de suppression, une avec un `iterator` et une autre avec un `reverse_iterator`. Le travail de ces fonctions est codé dans deux fonctions de service privées d'insertion et de suppression qui reçoivent un pointeur de cellule plutôt qu'un itérateur en paramètre. Pour chacune des quatres fonctions publiques, la coquille publique récupère le pointeur de l'itérateur, appelle la fonction privée, puis s'occupe de retourner le bon résultat à l'usager.

Le contenu de la classe `list` que vous devez coder est le suivant. Assurez-vous de bien traiter et tester TOUS les cas possibles.

- les deux fonctions de service privées d'insertion et de suppression à partir d'un pointeur de cellule ;
- la classe `reverse_iterator` ;
- les opérateurs `rbegin()` et `rend()` ;
- les opérateurs d'insertion et de suppression à partir d'un `reverse_iterator` ;
- l'affectateur ;
- l'opérateur `resize` qui diminue la dimension de la `list` en supprimant des éléments à la fin, ou qui augmente la dimension en ajoutant des éléments à la fin. Dans ce cas, la valeur des éléments ajoutés est donnée par le second paramètre explicite de l'opérateur ;
- l'opérateur `splice` qui insère une `list L` donnée en paramètre dans la `list` courante à une position indiquée par un itérateur. La taille de la `list L` doît être remise à zéro sans supprimer les cellules qu'elle contenait ;
- l'opérateur `reverse` qui inverse l'ordre des éléments de la `list` ;
- l'opérateur `sort` qui trie les éléments de la liste en ordre croissant (ou non-décroissant si il y a des éléments répétés).

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, pour coder 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 le serveur `junco.dinf.usherbrooke.ca`.

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 `TP3`. Chargez votre fichier `list2.h` et soumettez-le. Indiquez bien les NOMS (pas les matricules) des deux membres de l'équipe en commentaire dans ce fichier. Ne faites qu'une seule soumission par équipe. Ne remettez pas d'autre fichier, ni d'exécutable. Ne modifiez pas le fichier `list.h`. Votre fichier `list2.h` sera intégré à un programme de test contenant déjà le fichier `list.h` et d'autres fichiers du programme. Vous n'avez donc pas à re-soumettre `list.h`.

Barême

- 25 points pour soumission réussie d'un programme qui compile sans erreur
- 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 (3 points pour la classe `reverse_iterator` et 2 points pour chacune des 11 fonctions à coder)