UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

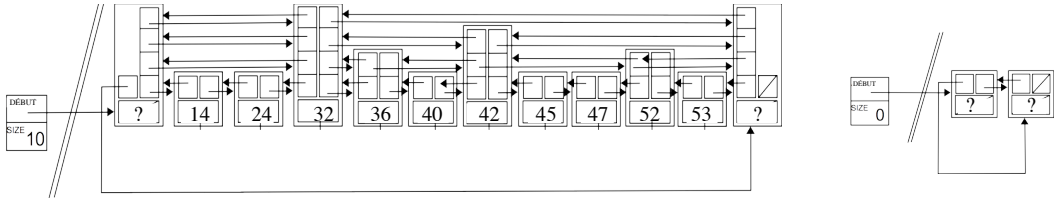
IFT 339

Laboratoire#4 : Type `set`

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 `set` 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 15 novembre 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 `set` séparé dans deux fichiers. Le premier, `set.h`, contient les définitions des fonctions déjà codées, et appelle l'inclusion de l'autre, `set2.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 `set.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 une liste à enjambement (une `skip_list`) par chaine de cellules. Une cellule contient trois éléments :

`CONTENU` : C'est un élément de type `TYPE`.

`PREC, SUIV` : Ce sont des tableaux de pointeurs de cellule. La dimension de ces tableaux est décidée aléatoirement au moment de l'insertion d'un élément. Les deux tableaux ont la même dimension dans une cellule donnée, sauf pour les cellules de tête et de queue.

Dans les illustrations ci-dessus, tous les pointeurs sont des `cellule*`. On voit ci-dessous l'illustration d'un ensemble de 10 éléments et d'un ensemble vide. Un itérateur est représenté par un pointeur de cellule. La définition de la classe `iterator` est entièrement fournie.

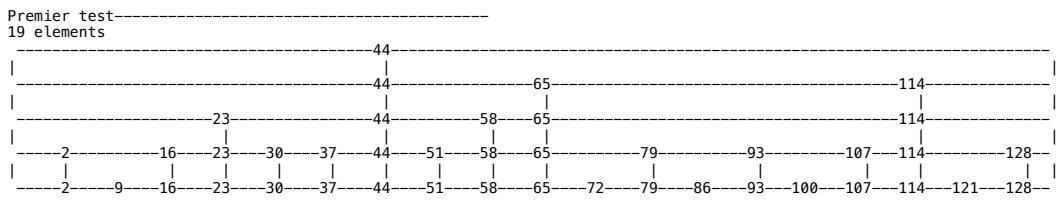
Il y a une fonction d'insertion privée et une fonction de suppression privée dans le set. Elles travaillent directement avec des pointeurs de cellule. Pour l'insertion d'une nouvelle cellule, la hauteur

est générée aléatoirement. Si la hauteur h générée est supérieure à la hauteur de la `skip_list`, la hauteur des cellules de début et fin est augmentée à h. Il y a deux fonctions d'insertion publiques et deux fonctions de suppression publiques qui se charge d'appeler les fonctions privées et ensuite de retourner le bon résultat à l'usager. La première fonction `insert` prend un élément, localise où on devrait l'insérer, puis appelle la fonction d'insertion privée. La seconde fonction `insert`, qui reçoit un `iterator` en paramètre doit vérifier que l'`iterator` reçu pointe au bon endroit, car l'usager peut l'appeler avec n'importe quelle valeur. Si l'`iterator` ne pointe pas à la bonne place, cette fonction appelle la première, sinon elle appelle la fonction privée pour faire le travail. La première fonction `erase` prend un élément, le localise, puis si l'élément est trouvé, elle appelle la fonction de suppression privée. La seconde fonction `erase` reçoit un `iterator` en paramètre, et appelle la fonction de suppression privée pour faire le travail. Notez que l'ordre des éléments est défini à partir de l'opérateur `<` des objets insérés dans l'ensemble. Aucun autre opérateur de comparaison n'est permis. Les fonctions `find`, `lower_bound` et `upper_bound` de localisation doivent avoir des complexités en temps logarithmiques.

Utilisez la fonction `afficher` pour tester la conformité de votre code avec la spécification. Elle permet de donner une image de la liste à enjambement. Elle ne fait un travail décent que pour les ensembles d'objets d'un type numérique inférieur à 1000 ou de courtes chaines de caractères. On peut lui fournir en paramètre une `string` qui permet d'identifier l'affichage. Par exemple, avec l'appel :

```
set<int> S;
//insérer des éléments dans l'ensemble
S.afficher("Premier test");
```

Vous pourriez avoir quelque chose comme :



Le contenu de la classe `set` 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 ;
- le constructeur par copie et le destructeur ;
- le constructeur par copie et le destructeur ;
- la fonction `clear` qui supprime tous les éléments du set ;
- les fonction de de localisation `find`, `lower_bound` et `upper_bound` ;
- la fonction `insert` qui reçoit un `iterator` en paramètre ;
- les deux fonctions `erase` publiques.

Inspirez-vous des opérateurs déjà codés pour faire du code cohérent avec ce qui existe déjà. Une fois votre programme complété, vous devez vous assurer qu'il compile et fonctionne bien sous Linux en le testant sur le serveur `drongo.dinf.usherbrooke.ca`.

Remise du travail

Pour soumettre votre travail, connectez-vous, dans un fureteur, au serveur `http://opus.dinf.usherbrooke.ca` en utilisant votre CIP, puis choisissez le cours `IFT339` et le projet `TP4`. Chargez votre fichier `set2.h` et soumettez-le. Indiquez bien les NOMS (pas les matricules) 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
- 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 chacune des fonctions privées d'insertion et suppression, 3 points pour la fonction `insert` qui reçoit un `iterator` en paramètre, et 2 points pour chacune des 8 autres fonctions à coder)