UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

IFT 339

Travail pratique#4 : Arbres, ensembles et dictionnaires

Ce travail a pour objectif de vous familiariser avec les arbres (partie 1) et les ensembles/dictionnaires (partie 2). Quelques modalités :

— Ce travail compte pour **7 %** de la session.

— La date de remise de ce travail est le **vendredi 22 novembre** jusqu'à 23h59.
Vous devez remettre sur `https://turnin.dinf.usherbrooke.ca/`.

Partie 1 : itération sur un arbre

Vous devez implémenter un itérateur qui parcourt un arbre en **postordre**. Nous fournissons le fichier noeud.h qui contient la classe Noeud, qui représente d'un arbre, ainsi qu'une classe iterator que vous devez compléter. À vous de déterminer la représentation de l'itérateur (variables, constructeur, autres fonctions, etc.).

L'ordre de visite postordre à partir d'un noeud x va comme suit : (1) on visite d'abord tous les enfants de x dans l'ordre (en répétant la procédure pour chaque enfant, ce qui aura pour effet de visiter tous les descendants) ; et (2) une fois que la visite des enfants est terminée, on visite x. Un parcours postordre sur l'arbre de la Figure 1 visite les noeuds dans cet ordre : 1-2-3-4-5-6-7-8-9-10-11-12-13. Voir la fonction afficher_postordre pour une implémentation récursive.

Il est important de noter que tout noeud v représente un arbre, celui qui est enraciné en v. On peut donc démarrer l'itération à partir de n'importe quel noeud pour **visiter seulement son sous-arbre**. Par exemple, si on démarre un parcours postordre à partir du noeud 12 de la Figure 1, on visitera les noeuds dans l'ordre 4-5-6-7-8-9-10-11-12. Dans ce cas, on ne visite par le noeud 13, ni le reste de l'arbre. Vous devez vous assurer que si on itère à partir d'un noeud x qui n'est pas la racine, on ne visite que les decendants de x et le parcours se termine en x.

Les fonctionnalités à implémenter sont les suivantes :

begin (dans la classe noeud). Retourne un itérateur pointant sur le premier noeud d'un parcours postordre.

end (dans la classe noeud). Retourne un itérateur pointant sur la fin du parcours. Il n'est pas attendu que monnoeud->end() pointe sur un élément particulier. Le comportement de l'instruction *monnoeud->end() n'est donc pas défini. Par contre, monnoeud->end()- - doit faire pointer l'itérateur sur le dernier élément du parcours (qui est en fait monnoeud).

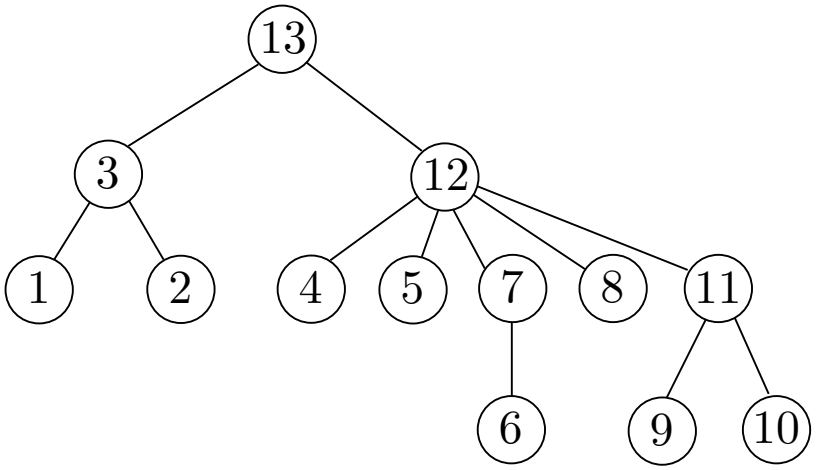

FIGURE 1 – Un arbre dans lequel les noeuds contiennent des entiers. Par un pur hasard, les noeuds sont étiquetés par leur position de visite dans un parcours postordre.

operator++. Fait passer l'itérateur au prochain noeud à visiter dans le parcours postordre. L'état interne de l'itérateur doit être modifié. Vous devez avoir l'incrémentation pré-fixe et l'incrémentation post-fixe. Le comportement de `monnoeud->end()++` n'est pas défini.

operator−−. Fait passer l'itérateur au noeud juste avant le noeud courant dans le parcours postordre. L'état interne de l'itérateur doit être modifié. Vous devez implémenter la version pré-fixe et post-fixe. Le comportement de monnoeud−>begin()−− n'est pas défini.

operator==. Vérifie que l'itérateur est égal au courant. Deux itérateur sont égaux s'ils ont été créés à partir du même noeud initial et s'ils pointent présentement sur le même noeud. Il faut aussi s'assurer que la comparaison avec `end()` fonctionne bien.

operator∗. Retourne la valeur du noeud pointé par l'itérateur.

Vous pourriez avoir besoin d'autres fonctions, selon votre représentation de l'itérateur. Il est notamment recommandé d'ajouter un constructeur à la classe iterator.

Notes sur l'évaluation. L'objectif pédagogique de ce travail est de vous familiariser avec la navigation à travers un arbre. Si votre solution passe à côté de cet objectif, vous serez pénalisé(e). Dans le doute, contactez-moi.

En particulier, un objet itérateur devrait prendre un espace $O(1)$. Vous ne pouvez donc pas simplement faire un parcours post-ordre récursif et stocker un vecteur correspondant à cet ordre dans votre itérateur. De plus, votre itération devrait supporter la suppression autant que possible. Si vous avez un itérateur et que le noeud qu'il pointe est supprimé, alors l'itérateur n'est plus valide. Par contre, si la suppression ne supprime pas le noeud de l'itérateur, ce dernier doit quand même fonctionner. Voir les cas de test dans `main.cpp`.

Partie 2 : recensement de la population

La partie 2 est indépendante de la partie 1. Il n'est donc pas attendu que votre code de la partie 1 serve pour la partie 2 (ceci est pour éviter que vos bugs en partie 1 se répercutent ici). Je recommande plutôt d'utiliser les classes `set` et/ou `map` de la librairie standard.

Vous devez recenser la population de chaque ville d'un pays quelconque. En entrée, vous recevez une liste de personnes, chacune représentée par : un identifiant, sa province, sa ville. Vous devez afficher le nombre de personnes par ville. Notez que deux provinces peuvent avoir une ville avec le même nom (par exemple, il y a 3 Windsor au Canada, en Nouvelle-Écosse/Ontario/Québec, et il y a 29 Washington aux USA). Ces villes sont différentes et ont chacune leur propre compte. Notez que même si les exemples utilisent le Canada, il n'y a pas de borne sur le nombre de provinces/villes.

Votre fonction doit afficher à l'écran le nombre de personnes par ville. L'ordre à la sortie n'est pas important. Si la liste donnée contient deux personnes avec le même identifiant, vous devez afficher qu'il y a une erreur et retourner.

Voici un exemple d'entrée, où une personne est représentée par un triplet (id, province, ville) :

(1, Québec, Sherbrooke)
(2, Ontario, Windsor)
(3, Québec, Sherbrooke)
(4, Québec, Windsor)
(5, Alberta, Calgary)
(6, Ontario, Windsor)
(7, Québec, Sherbrooke)

Sortie attendue :
Québec, Sherbrooke : 3
Ontario, Windsor : 2
Québec, Windsor : 1
Alberta, Calgary : 1

Autre exemple :

(1, Québec, Sherbrooke)
(2, Ontario, Windsor)
(3, Québec, Sherbrooke)
(4, Alberta, Calgary)
(3, Ontario, Windsor)
(5, Québec, Sherbrooke)

Sortie attendue :
ERREUR : il y a un identifiant répété.

Évaluation

Comme pour les autres devoirs, vous serez pénalisés pour les erreurs et imprécisions selon les critères suivants : Respect des spécifications (45 points) ; Qualité du code (45 points) ; Respect des normes de programmation (10 points).

Remise du travail

Vous devez remettre les fichiers noeud.h et main.cpp. Ne remettez pas d'autre fichier ni surtout d'exécutable. 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. Notez qu'il n'y a pas de script de correction automatique - votre code sera inspecté à la correction.