TP à rendre en fin de séance

1 Conditions de l'épreuve

Le travail est à faire en monome, sans interaction avec vos voisins, et sans explications de votre chargé de TP. Vous devez être placé dans la salle qui vous a été affectée dans Tomuss.

Vous aurez uniquement droit :
— à une page de votre navigateur ouverte sur Tomuss (AUCUNE AUTRE PAGE AUTORISEE),
— à un lecteur de pdf ouvert sur les slides du cours ainsi que sur les sujets de TP de l'UE (ces supports peuvent également être imprimés),
— à votre éditeur de code préféré ouvert sur un projet vide.

Cela signifie que vous n'aurez pas droit à internet ni à vos codes écrits précédemment (ni aux corrections fournies). Votre application de mail ou de tchat devra également être fermée, ainsi que votre téléphone portable.

Attention : Le chargé de TP pourra vérifier en cours de séance que vous respectez bien ces règles (liste des processus en train de tourner sur votre machine, fichiers précédemment ouverts dans votre éditeur : penser bien à tout purger au début de la séance). Votre écran doit rester visible tout le temps de la séance avec une luminosité suffisante.

A la fin de la séance vous devrez déposer une archive de votre travail sur Tomuss :
— Attention de ne mettre que les `.h` et `.cpp` ainsi qu'un `Makefile`.
— L'archive (`.zip` ou `.gz` ou `.tgz`) que vous déposez doit porter votre nom.
— **Attention, le dépôt ferme à 18h.**

Le sujet est proche des travaux faits pendant les séances de TP. Vous ne pourrez pas recourir aux `vector`, `deque`, `list`, `slist`, `set` et `map` de la STL, sauf si c'est spécifié dans le texte. Enfin, vous pourrez choisir de coder en utilisant ou non des éléments de la norme `C++11`, `C++14` ou `C++17`. Le but de ce travail est de tester votre capacité à aborder un problème en un temps limité et sans internet. Ce n'est pas inquiétant si vous ne terminez pas tout dans le temps imparti. Vous aurez ensuite du temps pour réfléchir au sujet pendant quelques semaines et vous reviendrez répondre à quelques questions supplémentaires sur le problème dans le cadre d'une interrogation sur papier.

2 Auprès de mon arbre

L'objet de ce TP consiste à mettre en œuvre le type abstrait `Set` en l'implémentant sous forme d'un Arbre Binaire de Recherche.

Arbre Binaire de Recherche

Un Arbre Binaire de Recherche correspond à une structure arborescente où les informations (éléments) sont rangées dans des `Nœud`s. Tout `Nœud` qui n'est pas une feuille possède au plus deux `Nœud`s fils, avec la propriété que l'élément contenu dans un `Nœud` est plus grand que tous les éléments de son sous-arbre gauche, et plus petit que tous les éléments de son sous-arbre droit.

Le modèle de classe Arbre Binaire de Recherche (`ABR`) que vous allez développer est *templaté* par le type `T` de ses éléments, pourvu que ce type `T` supporte les opérations de construction par défaut, de copie, d'affectation et de comparaison. Vous introduirez également le modèle de classe `Nœud` qui est indispensable pour permettre le stockage des éléments.

La structure de données `ABR` contiendra un champ contenant l'adresse du `Nœud` racine ainsi qu'un champ correspondant au nombre d'éléments dans l'arbre. La structure de données `Nœud`

contiendra un champ correspondant à une information de type `T`, ainsi que deux pointeurs sur `Noeud`s contenant l'adresse du fils gauche et du fils droit s'ils existent, `nullptr` sinon.

Le type `ABR` devra offrir les opérations adéquates pour être **initialisé par défaut**, permettre l'**insertion** ou la **recherche** d'un élément de type `T`, ainsi qu'une opération d'**affichage infixé** (ordre croissant des éléments). Vous devrez ajouter (ou bloquer) **toute autre fonctionnalité** qui vous paraîtra nécessaire pour une **gestion saine de l'espace mémoire** en gardant à l'esprit la "règle de 5". Pour rappel, l'insertion d'un élément se concrétise par la création d'un nouveau `Nœud` feuille, après avoir traversé une séquence de `Nœud`s en commençant à la racine. On compare l'élément avec l'élément contenu dans chacun des `Nœud`s traversés et on s'aiguille à gauche ou à droite suivant le résultat de cette comparaison.

Vous testerez vos `ABR` dans un programme principal contenant des instructions du type :

```
ABR<int> tree;
for(int i=0;i<10;i++)
         tree.insert(rand() % 100);  // #include<cstdlib>
std::cout << tree << std::endl;    // Affichage infixé
while(!tree.find(rand() % 100));
```

2.1 Qui grimpe ?

Définissez une classe template d'itérateur permettant d'accéder, en lecture uniquement, aux éléments d'un `ABR` par déréférencement, et de parcourir tous les éléments de cet `ABR` par incrémentation en suivant l'ordre infixé. Outre la possibilité d'incrémentation (version préfixée uniquement) et de déréférencement, vous munirez cet itérateur d'un test d'inégalité.

Moyennant l'ajout des fonctionnalités nécessaires dans `ABR`, ces itérateurs permettront de parcourir les éléments d'un `ABR<int>tree` avec des instructions du type :

```
ABR<int>::iterator it=tree.begin();
ABR<int>::iterator ite=tree.end();
for(;it!=ite;++it)
  std::cout << *it << std::endl;
```

Pour implémenter votre itérateur, vous pourrez recourir à une pile de la STL si vous le souhaitez. Donnez des indications sur la complexité de l'incrémentation d'un itérateur avec l'implémentation que vous aurez choisie.

2.2 De retour à l'arbre

Modifiez le code de la fonction membre *insert* d'un `ABR` afin que cette fonction retourne un iterateur permettant d'accéder à l´élément inséré. De la même manière, modifier la fonction membre *find* de manière à ce qu'elle renvoie un itérateur sur l'élément s'il est présent dans un `ABR tree`, `tree.end()` sinon.

3 Travail à faire

Votre travail sera décomposé en 3 étapes relatives aux 3 parties de l'énoncé. Merci de placer le code résultant de chacune des étapes dans des répertoires différents, avec un programme `main.cpp` spécifique à chaque étape. Chacun de ces programme principaux devra illustrer le bon fonctionnement des éléments mis en place. Il devra en particulier permettre l'instanciation et l'appel de l'ensemble des classes et fonctions template, et montrer que l'ensemble forme un tout cohérent et sans faille. Vous détaillerez ce que vous avez eu le temps de faire dans un README. Attention de bien vérifier que votre code correspond à une gestion saine de l'espace mémoire.