TP 8 : AVL

Dans le cadre de l'UE, vous êtes amenés à programmer en **C++** en utilisant les références pour réaliser vos passages de paramètres ainsi que les constructeurs, les destructeurs et les surcharges de l'opérateur d'affectation. Néanmoins, il sera important que vous sachiez toujours basculer dans un autre langage de programmation. Pour cela, il est important que vous établissiez votre raisonnement au niveau du pseudo-langage algorithmique, en utilisant les spécificités du langage utilisé seulement quand vous effectuez la mise en œuvre.

1 Etude du déséquilibre des Arbres Binaires de Recherche

Reprenez le module `Arbre Binaire de Recherche` que vous avez développé aux TPs précédents. Les nœuds de vos arbres contiennent des `Eléments` pour lesquels il est nécessaire de disposer d'une opération de comparaison (surcharge de l'opérateur <). Les `Eléments` sont décrits dans le module `Element` ainsi que les opérations associées. Vous opterez pour une version récursive de l'insertion d'un `Elément`.

Enrichissez à présent votre module `Arbre Binaire de Recherche` de la manière suivante :

- Ajout d'une information de `hauteur` dans chaque `Noeud` (les plus aguerris d'entre vous pourront directement remplacer cette information de hauteur par une donnée membre `diff` correspondant à la différence de hauteur entre le sous-arbre gauche et le sous-arbre droit du `Noeud`),
- Modification de la procédure d'insertion dans un Arbre Binaire de Recherche, de telle sorte que l'info de `hauteur` (ou de `diff`érence de hauteur) soit mise à jour dans les `Noeuds` (pour cela, la procédure interne récursive d'insertion dans un sous-arbre pourra par exemple retourner un booléen suivant que l'insertion de l'élément a incrémenté ou non la hauteur du sous-arbre).

Un exemple de programme utilisateur :

```
ABR tree;
for (int i=0;i<20;i++)
{
  tree.insere(rand()/100);
  tree.affiche()
}
```

2 AVL

Le but de ce TP est à présent de modifier votre module `Arbre Binaire de Recherche` pour en faire un module `AVL`. Il faudra pour cela que vous développiez les points suivants :

- Ajout des opérations de rotation vues en Cours : `void AVL::AVL_Rotation_Droite(Noeud *& pN)`,
  `void AVL::AVL_Rotation_Double_Droite(Noeud *& pN)`,
  `void AVL::AVL_Rotation_Gauche(Noeud *& pN)`,
  `void AVL::AVL_Rotation_Double_Gauche(Noeud * & pN)`.
- Suite à l'insertion d'un `Elément` dans un `AVL`, si déséquilibre il y a, une unique rotation doit être effectuée sur le `Noeud` le plus bas à subir le déséquilibre. Ce `Noeud` peut donc être identifié à la remontée récursive, lors de la mise à jour du champs `hauteur` ou `diff` de chaque `Noeud`.

Un exemple de programme utilisateur :

```
AVL tree;
for (int i=0;i<20;i++)
{
  tree.insere(rand()/100);
  tree.affiche()
}
AVL tree2(tree1);
tree2.insere(55);
tree.affiche();
tree2.affiche();
```