TD 06 : Arbres AVL

Consignes générales : N'oubliez pas pour ce TD comme pour les suivants de vous créer un répertoire consacré au TD et d'enregistrer vos codes dedans.

On rappelle que les commandes à taper dans le terminal pour compiler puis éxécuter votre programme C :
— Pour compiler : *gcc -o nom_executable nom_programme.c*
— Pour exécuter : *./nom_executable*

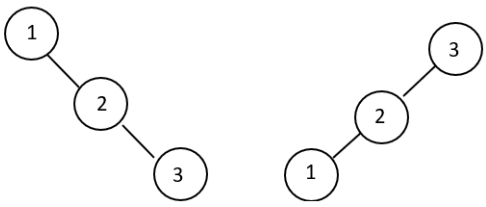
Exercice 1 (*Question de cours*)

1. Construire un AVL en insérant les éléments dans cet ordre ; 10, 3, 5, 15, 20, 12, 7, 9.
2. Construire un second AVL en insérant les éléments dans l'ordre inverse que précédement.
3. Effectuer un parcours infixe sur ces deux arbres. Que remarque-t-on ?
4. Supprimer l'élément 5 puis 12 du premier arbre. Redessiner l'arbre obtenu après chacune des suppressions

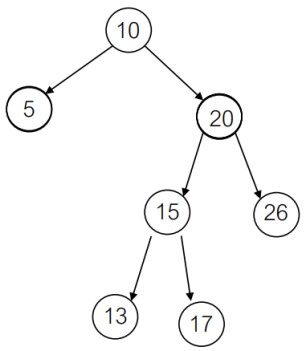
Exercice 2 (*Construction d'un AVL*)

Les versions "pseudo-code" des fonctions demandées sont dans le cours.

1. On rappelle que pour construire un AVL, chaque noeud de l'arbre doit être associé à un facteur d'équilibrage dont la valeur est :
   equilibre = hauteur sous arbre droit - hauteur sous arbre gauche
   Modifier la structure `Arbre` pour inclure ce nouveau champs.
2. Réécrire la fonction `creerArbre(x : Element)` pour inclure également ce nouveau champ et l'initialiser à la création d'un nouveau nœud de l'arbre.
3. Les opérations de rééquilibrage s'effectuent à l'aide de « rotations » des sous-arbres (cf cours). Écrire les fonctions `RotationGauche( A : Arbre)` permettant de faire la rotation du sous-arbre A avec son fils droit et `RotationDroite( A : Arbre)` permettant de faire la rotation du sous-arbre A avec son fils gauche.
4. Tester ces deux fonctions sur les deux arbres suivants (que vous aurez construit manuellement avec les fonctions `ajouterFilsDroit` et `ajouterFilsGauche` en prennant soin d'indiquer les bonnes valeurs d'élément ET d'équilibre pour chaque noeud) :



5. À partir des fonctions précédentes, écrire les fonctions permettant d'effectuer les doubles rotations : `DoubleRotationDroite(A : Arbre)` et `DoubleRotationGauche(A : Arbre)`.
6. Tester une de ces fonctions (celle la plus adaptée !) sur l' arbre suivant (que vous aurez construit manuellement avec les fonctions `ajouterFilsDroit` et `ajouterFilsGaucheen` prennant soin d'indiquer les bonnes valeurs d'élément ET d'équilibre pour chaque noeud) :



7. Écrire la fonction `equilibrerAVL( A : Arbre)` qui permet d'effectuer la bonne rotation de l'arbre en fonction du facteur d'équilibrage de A et de ses fils.
8. Écrire la fonction `insertionAVL( A : arbre , e : Element, h: pointeur sur entier)` qui insère dans l'arbre un nouveau nœud contenant l'élément e. L'insertion de l'élément est basée sur le même principe que l'insertion dans un ABR. Il faut cependant veiller à mettre à jour le facteur d'équilibrage de chaque nœud (dont l'évolution est gérée par le paramètre h) et à rééquilibrer l'arbre si besoin à l'aide de la fonction `equilibrerAVL`.
9. Écrire la fonction `suppAVL(A: Arbre, e: Element, h: pointeur sur entier)` permettant de supprimer un nœud contenant l'élément e de l'arbre A. Le raisonnement est le même que pour la question précédente.

Exercice 3 (*Test d'un AVL*)
Reprendre les questions de l'exercice 1 et construire les AVL demandés grâce aux fonctions écrites dans le TD. A chaque étape (ajout ou suppression), afficher l'arbre avec la fonction `affArbreGraphique` qui a été fournie.