Devoir surveillé ☐ Examen ☒ Session: principale ☒ de contrôle ☐

Matière : **Algorithmique**
Enseignant(s) : Riadh ROBBANA
Filière(s) : **MPI**
Barème : Exercice : 8 pts, Problème : 12 pts
Nombre de pages : Deux pages

Semestre: Second
Date: 20/05/2013
Durée: 1h 30 mn
Documents: autorisés ☐ non autorisés ☒

Exercice : (8 points)

On souhaite trier des entiers positifs contenus dans une pile, le plus petit sera en tête de pile. Pour cela, on ne s'autorise qu'une seule autre pile (autrement dit, on ne peut pas utiliser de tableaux) et un nombre constant de variables.

Question 1 : (6 points) Proposer une fonction en C qui n'utilise que les opérations suivantes sur les piles vus en cours.

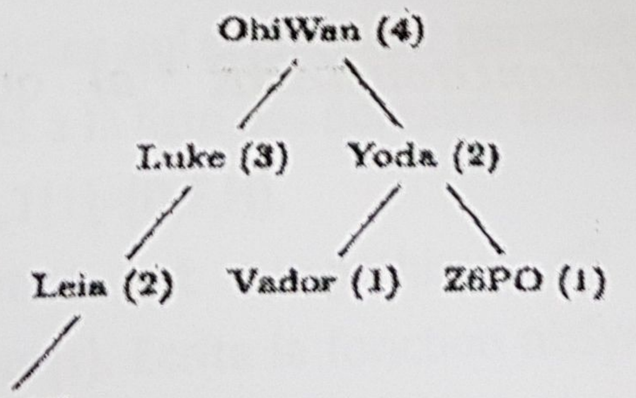
```
pile *creerpile() ;
char estvidepile(pile *p) ;
pile *empiler(pile *p, int donnee) ;
pile *depiler(pile *p) ;
int obtenir_tete (pile *p)
```

Question 2 : (2 points) Donner la complexité de votre fonction

Problème : Equilibrage d'un Arbre Binaire de Recherche (12 pts)

On considère un arbre binaire de recherche dans lequel chaque nœud possède un champ supplémentaire, qui est la hauteur de l'arbre dont il est racine. On note H(A) la hauteur d'un arbre A. La hauteur de l'arbre null est 0, celle d'une feuille est 1, et plus généralement celle de **(r,fg,fd)** est **1+max(H(fg),H(fd))**. Ce type d'enrichissement des données est très fréquent. Dans le contexte qui nous intéresse, c'est le premier pas vers les algorithmes d'équilibrage d'arbres.

Nous avons indiqué sur l'exemple qui suit la hauteur pour chaque nœud (donc la hauteur du sous-arbre de racine le nœud en question) :

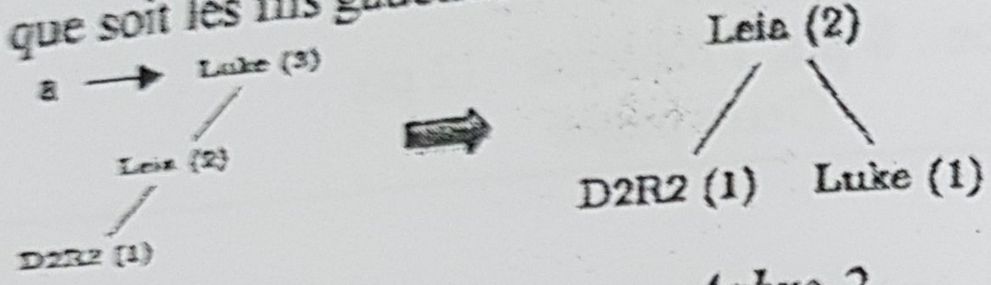


Question 1 (2pts): Proposez une structure de données qui permet de représenter cet arbre.
Question 2 (2pts): Ecrire une fonction qui retourne la hauteur de l'arbre.
Question 3 (3pts): Ecrire une fonction *ABR * ajouter (ABR * a, char * nom)* qui insère un nouveau nœud pour la chaîne *nom* dans l'arbre *a*. Cette fonction devra bien sûr respecter l'ordre ABR, mais également mettre à jour les nœuds sur le parcours, de sorte que la hauteur de tous les nœuds de l'arbre soient à jour après l'appel. On rappelle que si *s* et *t* sont deux chaines de caractères, l'instruction *strcmp(s, t)* retourne 0 si les deux chaînes sont égales, un nombre

strictement négatif si s est avant t dans l'ordre lexicographique et un nombre strictement positif sinon.

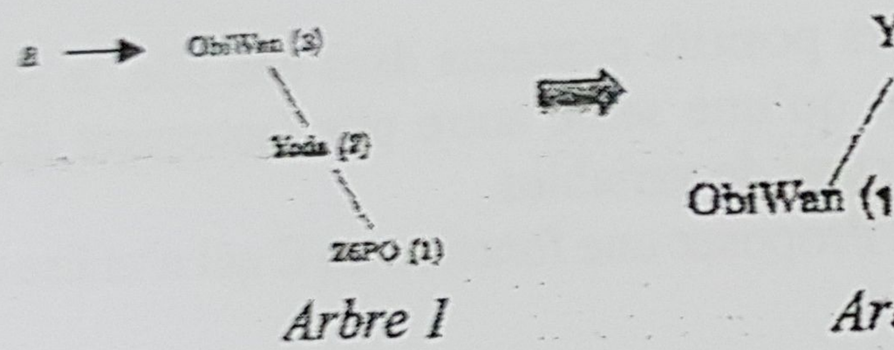
Question 4 (2pts): Nous voulons maintenant tester sur notre arbre la propriété suivante : Un arbre binaire de recherche est dite un arbre AVL (Adelson-Velskii et Landis) Si, pour n'importe lequel de ses nœuds, la différence de hauteur entre ses deux fils diffère d'au plus un.
Ecrivez une fonction qui permet de tester si un arbre donné est AVL ou non.

Question 5 (2pts): Nous procédons au rééquilibrage de notre arbre. Nous donnons ici un exemple de rééquilibrage dans un cas particulier. On reprend l'exemple donné au début de l'exercice: Nous supposons que soit les fils gauches sont nuls soit les fils droits sont nuls.



Arbre 1 *Arbre 2*

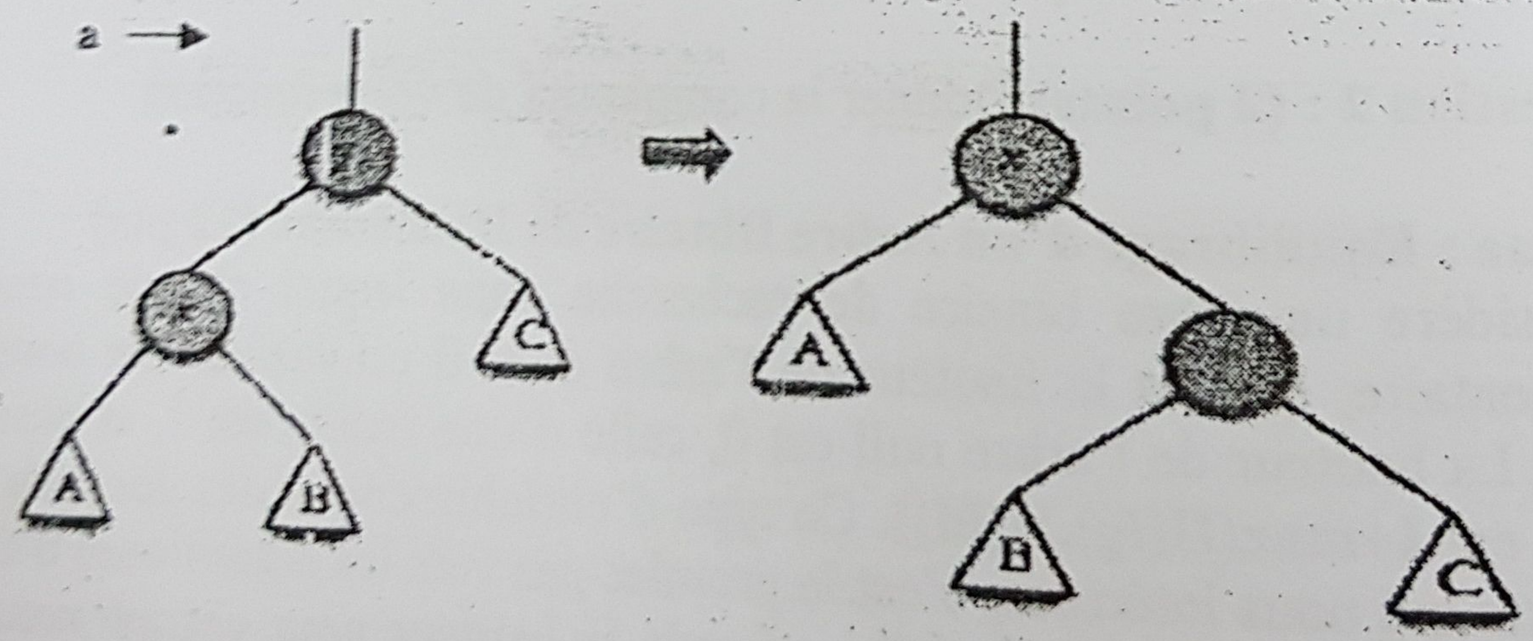
Ou encore



Arbre 1 *Arbre 2*

Ecrivez une fonction *ABR* RotationSimple(ABR * a)* qui permet d'obtenir les arbres suivants sans créer de nouveau nœud. Il faut distinguer les deux cas particulier.

Question,6 (1pts): Nous traitons maintenant le rééquilibrage pour un cas plus général. Pour cela nous devons faire des rotations. Nous donnons ici un exemple de rotation vers la droite.



Arbre 1 Arbre 2

Ecrire la fonction *ABR* RotationDroite(ABR * a)* qui permet d'obtenir l'arbre suivant sans créer de nouveau nœud.