Cours de programmation séquentielle

Arbre binaire de recherche

1 Préambule

Cette série n'est pas notée. Comme pour le premier semestre, vous pouvez obtenir un bonus de `0.5` sur la note de l'examen en rendant 2/3 des exercices non-notés du semestre. Vous devez mettre le lien de votre repo `git` dans le wiki correspondant sur `Cyberlearn`. **Ne nous rajoutez pas comme reporter!** Vous pouvez à choix laisser le repo public ou alors le mettre en interne. Pour la partie de compréhension écrivez vos réponses dans un fichier `ANSWERS.md` et placez le dans votre repo `git`.

2 Buts

Dans ce travail vous verrez les concepts suivants:

- Créer et utiliser une librairie d'arbre binaire de recherche d'entiers.
- Utilisation des pointeurs.
- Utilisation de fonctions récursives.

Vous avez vu la théorie sur les arbres binaires de recherche avec le Professeur Albuquerque.

3 Énoncé

Dans ce travail pratique utilisez la technique de développement pilotée par des tests pour écrire une librairie d'arbre binaire de recherche.

Chaque noeud d'un arbre binaire de recherche d'entiers est est une structure du type

```
typedef struct _node_t {
    int val;
    struct _node_t *left;
    struct _node_t *right;
} node_t;
```

Pour manipuler cette structure vous devez implémenter un certain nombre de fonctions **uniquement de façon récursive**.

```
// Fonctions de création, destruction et affichage
node_t *bst_create(); // création d'un arbre vide (retourne NULL)
void bst_destroy(node_t *tree); // détruit l'arbre et vide la mémoire
```

```c
void bst_print(node_t *tree); // affiche l'arbre (voir plus bas)

// insertion de val dans l'arbre et retourne l'arbre (ou NULL si problème)
node_t *bst_insert(node_t *tree, int val);
// efface le premier élément contenant la valeur val dans l'arbre
// et retourne l'arbre (ne fait rien si val est absente)
node_t *bst_delete(node_t *tree, int val);

// la valeur val est-elle présente dans l'arbre?
bool bst_is_present(node_t *tree, int val);
// retourne le noeud où la valeur val se trouve (NULL si absent)
node_t *bst_search(node_t *tree, int val);
// l'arbre est-il un arbre binaire de recherche?
bool bst_is_bst(node_t *tree);

// retourne le noeud avec la valeur minimale de l'arbre (NULL s'il y a pas)
node_t *bst_find_min_node(node_t *tree);
// retourne la valeur la plus petite stockée dans l'arbre (ou MIN_INT)
int bst_find_min(node_t *tree);
// retourne le noeud avec la valeur maximale de l'arbre (NULL s'il y a pas)
node_t *bst_find_max_node(node_t *tree);
// retourne la valeur la plus grande stockée dans l'arbre (ou MAX_INT)
int bst_find_max(node_t *tree);
```

Remarque 1

Vous devrez probablement écrire d'autres fonctions "utilitaires" dans votre librairie mais les fonctions ci-dessus sont celles que vous exposerez à l'utilisatrice (p.ex. la création d'un nœud, ou la vérification si l'arbre est vide).

Vous devrez également écrire un programme qui génère N nombres aléatoires et qui construit un arbre binaire de recherche. Il faut ensuite afficher les nombres triés en parcourant l'arbre par un parcours symétrique (ou infixé). Pour chaque nombre affiché il faudra aussi donner le niveau sur lequel il se trouve dans l'arbre.