Ce TP vous invite à programmer un arbre binaire de recherche.

1 Rappel : Fonctionnement

Un arbre binaire de recherche (binary search tree ou BST) est une structure récursive permettant facilement de gérer des listes de nombres en les maintenant triés.

Un arbre binaire de recherche est soit :

— vide (NULL)

— un nœud contenant une valeur (int), et deux fils, un gauche, et un droit, tel que toute valeur du fils gauche soit plus petite que la valeur du noeud et toute valeur du fils droit soit plus grande que la valeur du noeud. Sur la Figure 1, les noeuds vides ne sont pas représentés.

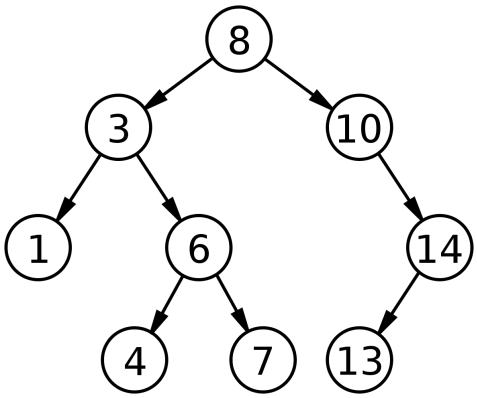

FIGURE 1 – Arbre binaire de recherche

2 Implémentation

On se donne le type suivant :

```
1  typedef struct arbre {
2      int val ;
3      struct arbre* fils_gauche ;
4      struct arbre* fils_droit ;
5  } *BST ;
```

On pourra ensuite créer des arbres avec les commandes suivantes :

```
1      BST B = malloc( sizeof(struct arbre) );
2      BST A =  malloc( sizeof(struct arbre) );
3      A -> val = 10;
4      B -> val = 11;
5      B -> fils_gauche = A;
```

Question 0 :

Pour éviter les fuites mémoires, écrivez une fonction récursive *delete* de type *void* qui supprime un arbre et libère la mémoire qui lui est allouée.

Question 1 :

Écrire une fonction *affiche* qui affiche les éléments d'un BST T donné en argument dans l'ordre croissant.

Question 2 :

Écrire une fonction *nb_ elem* de type *int* qui prend en entrée un BST T et renvoie le nombre d'éléments dans T.

Question 3 :

Écrire une fonction *est_ dans* de type *int* qui prend en entrée un entier x et un arbre binaire de recherche T, et qui renvoie 1 si $x \in T$ et 0 sinon.

Question 4 :

Écrire une fonction *verif* de type int qui prend en entrée un arbre et qui renvoie 1 si l'arbre est bien un BST, et 0 sinon.

Question 5 :

Écrire une fonction *ajoute* de type *void* qui prend en entrée un BST T et un entier x et ajoute x à T en maintenant la structure d'arbre binaire de recherche.

Question 6 :

Écrire une fonction *cree_ arbre* de type BST qui prend en entrée un entier n et une liste d'entier li de type $int*$ de taille n et qui crée l'arbre contenant ces entiers.

Question 7 :

Écrire une fonction *profondeur* de type *int* qui prend en entrée un BST T et renvoie sa profondeur.

Question 8 (difficile) :

Écrire une fonction *const_ opt* qui prend en entrée un entier n et une liste de li de n entiers et qui construit un BST de profondeur $\lceil \log(n) \rceil + 1$ contenant ces n entiers.

On pourra pour cela modifier la fonction *ajoute* pour maintenir une structure équilibrée à chaque ajout d'élément.