Exercice 1 : Liste simplement chaînée

On considère une liste simplement chaînée dont chaque noeud peut contenir un `double`. La liste est caractérisée par deux structures `Node` et `List`.

(a) Implémentez le type `Node` et la définition de son contenu.

```
typedef struct node {
  double value;
  struct node *next;
} Node;
```

(b) Implémentez le type `List` et la définition de son contenu. Cette dernière permet d'accéder le premier et le dernier élément de la liste et contient également le nombre d'éléments dans la liste.

```
typedef struct list {
  size_t count;
  Node *head;
  Node *tail;
} List;
```

(c) Implémentez la fonction `push` permettant d'insérer un nouvel élément dans la liste. Veillez aux cas particuliers, par exemple lorsque la liste est vide. La fonction retourne une valeur négative en cas d'erreur, zéro en cas de succès.

```
int push(List *list, double value) {
  Node *node = malloc(sizeof(Node));
  if (node == NULL) {
    return -1;
  }
  node->value = value;
  node->next = NULL;
  if (list->count == 0) {
    list->head = node;
    list->tail = node;
  } else {
    list->tail->next = node;
    list->tail = node;
  }
  list->count++;
  return 0;
}
```

(d) Implémentez la fonction `mean` permettant de calculer la valeur moyenne du contenu de la liste.

```
double mean(List *list) {
  double sum = 0;
  Node *node = list->head;
  while (node != NULL) {
    sum += node->value;
    node = node->next;
  }
  return sum / list->count;
}
```

(e) Implémentez une fonction de comparaison `compare` qui reçoit deux listes et retourne 0 si le contenu est identique, et 1 sinon.

```
int compare(List *a, List *b) {
  if (a->count != b->count) {
    return -1;
  }
  Node *node_a = a->head;
  Node *node_b = b->head;
  while (node_a != NULL) {
    if (node_a->value != node_b->value) {
      return -1;
    }
    node_a = node_a->next;
    node_b = node_b->next;
  }
  return 0;
}
```

Exercice 2 : Arbre de recherche binaire

Un BST (*Binary Search Tree*) est un arbre binaire respectant la propriété suivante : l'enfant de gauche est toujours plus petit que son parent et l'enfant de droite est toujours plus grand que son parent. Voici un exemple :

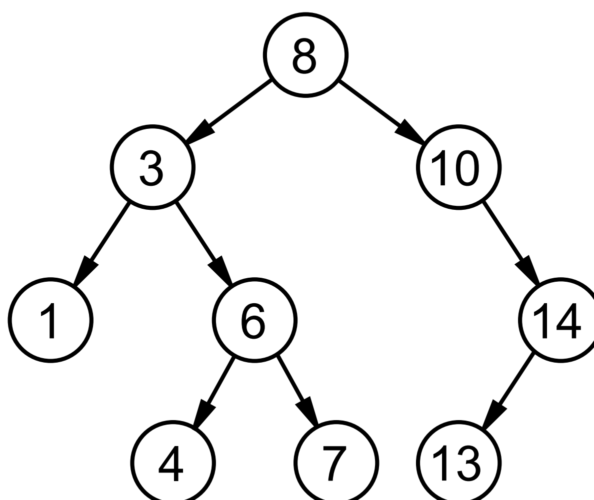

Figure 1 – BST

(a) Écrire la structure de donnée d'un noeud d'un BST contenant des valeurs entières non signées.

```
typedef struct node {
  unsigned int value;
  struct node *left;
  struct node *right;
} Node;
```

(b) Écrire une fonction récursive permettant de rechercher une valeur dans un bst. Si la valeur est trouvée la fonction retourne 1, sinon 0.

```
int search(Node *root, unsigned int value) {
  if (root == NULL) return 0;
  if (root->value == value) return 1;
  return search(
    root->value > value ? root->left : root->right, value);
}
```

(c) Quelle est la complexité en temps (Big-O) pour la recherche d'un élément dans un BST ?

$O(\log n)$