Assignment 1

This assignment has total 15 points. Each point amounts to one percent of the total module mark. You are not allowed to use any external library in your code. We might ask you to explain your code.

All assessment deadlines in this module are strict. Late submissions will get a mark of 0. All submissions must work on the virtual machine as specified.

The assignment consists of implementing a reversed binary search tree (BST) data structure in C. This part requires you to follow the lecture materials of Week 1 and 2.

Implementing a binary search tree (BST) in C

[Before starting this exercise, consider having a look at the linkedlist_basic.c source code file which is present inside Modules/Week2 on Canvas.]

Your task is to implement a BST for integers (int). In this exercise, we assume that there are **no duplicate nodes**. In bst.h, there is already a header file that declares the interface of that tree. You must implement the following functions for the tree operations in bst.c file. There is a file test_bst.c which is a simple testbench for testing your BST code. You may edit this file to include more test cases.

Implement the following functions in the bst.c file.

Task 1

```
Node* addNode(Node *root, int value);
```

This function is used to insert a new node in the tree as a leaf node, and returns the leaf. The value of the new node is assigned the input ‘value’. During the insertion, a traversal starts from the root of the tree. The traversal follows the left branch if the value of the new node is greater than the value of the currently visited node. Otherwise the traversal follows the right branch. For this question we will restrict to the case that all data values are unique (that is to say, there will be no duplicates) for simplicity. In case of duplicate, the function returns NULL and does not insert a node. Also, this function is used to create a root node, when its argument root is NULL.

Task 2

Node* removeNode(Node *root, int value);

This function is used to delete the node in the tree with the given value. For this question we will restrict to the case that all data values in the tree are unique. The function returns the new root, when is the root node to be deleted. Nothing happens if the value is not present.

Task 3

void displaySubtree(Node *N);

This function is for in-order printing of the node values. It prints

i) the values of the right subtree
ii) the value of the node
iii) finally, the values of the left
subtree

Thus, the function prints the node values in sorted ascending format.

The function should print one node value per line, for example:

```
2
3
4
5
```
Not 2 3 4 5.

Task 4

int numberLeaves(Node *N);

This function returns the number of leaves in the subtree rooted at node N.

Task 5

```
Node* removeSubtree(Node *root, int value);
```

This function deletes the entire subtree rooted at the node with the given value. The function returns the new root, like delete node. If the value is not present, then nothing should happen.

Task 6

```
int nodeDepth (Node * root, Node *N);
```

This function returns the depth of node N from root R. The depth is the exact number of edges between R and N. If N=R the depth is 0. If N does not belong to the tree rooted in R, then this function returns -1.