# Informatics II, Spring 2024, Solution Exercise 8

Publication of exercise: April 22, 2024
Publication of solution: April 29, 2024
Exercise classes: April 29 - May 3, 2024

**Learning Goal**

- Understanding Binary Trees and Binary Search Trees and how algorithms can operate on them.
- Knowing how to implement Insert, Delete and Traverse on Trees.
- Being able to apply learned methods to solve problems based on tree structure.

## Task 1 [Medium]

The structure of the tree node is defined as follows. The entry to a binary search tree is the root, an ordinary tree node just without parents.

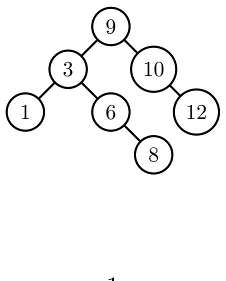
```
struct TreeNode {
  int val;
  struct TreeNode* left;
  struct TreeNode* right;
};
```

Write a C program that contains the following functions:

a) Write the function *struct TreeNode* insert(struct TreeNode** root, int val)* that inserts an integer `val` into the binary search tree.

b) Write the function *struct TreeNode* delete(struct TreeNode** root, int val)* that deletes the node with value `val` from the tree. If you have a choice between 2 nodes go for the node with the smaller value.

c) Write *void printTree(struct TreeNode* root)* which prints all edges with their *level* to the console in the format `Node A -- Node B : level`, each edge should be printed in a separate line. The ordering of the printed edges does not matter and may vary based on your implementation. Here, the *level* of an edge is defined by the larger depth of the two adjacent nodes.



Output of *printTree* for the example tree above should be:

```
9 −− 3 : 1
9 −− 10 : 1
3 −− 1 : 2
3 −− 6 : 2
10 −− 12 : 2
6 −− 8 : 3
```

d) Write *void traverseTree(struct TreeNode* root)* which prints the results of pre/in/post-order of the tree to the console in separate lines. Use a recursive approach.

e) **Bonus** [**Hard**] Write *void stack_traverseTree(struct TreeNode* root)* which prints the results of pre/in/post-order of the tree to the console in separate lines. Use stacks instead of a recursive approach.

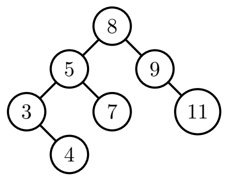
Test your program by performing the following operations:

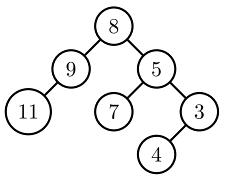
- Create a root node `root` and insert the values 9, 3, 10, 6, 12, 1, 8 in that order. This should produce the tree from above.
- Print the tree to the console.
- Print the traversals to the console.
- Delete the values 23, 1, 3, 9 from the tree in that order.
- Print the tree to the console.
- Print the traversals to the console.

Solution: see code task1.c

## Task 2 [Medium]

The mirror image of a binary (search) tree is obtained by swapping the left and right subtrees of each node. It essentially flips the tree horizontally. The tree below on the left produces the mirror image below on the right. Implement a C program to produce the mirror image of a given binary (search) tree. Test your program by making the tree below, mirroring it and printing the tree traversals of both the original and the mirrored tree.





Solution: see code task2.c

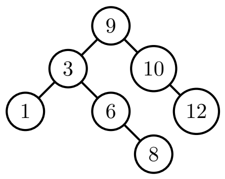
## Task 3 [Medium]

Write a C program, that checks whether two given binary trees are identical. Two binary trees are identical, if they contain the same node values and have the exact same structure.

Solution: see code task3.c

## Bonus Task [Hard]

The diameter of a tree is defined as the longest path between any two nodes in the tree. It may or may not pass through the root and there may be multiple diameter paths in a given tree (think for example of a perfect binary tree, where every leaf to leaf path passing through the root has the same diameter length). In the tree below, the diameter path passes through nodes 8 - 6 - 3 - 9 - 10 - 12 and has a length of 6 nodes. Write a C program to find such a diameter of a binary tree. **Hint:** Try implementing a divide and conquer approach.



Solution: see code task4_bonus.c