MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG213 Data Structures – Worksheet 10

Binary Search Tree ADT

Purpose:
This worksheet is concerned with binary search trees. You are not expected to complete the entire worksheet in class. Work on as many problems as you can; the remaining problems you can use for practice and to test your understanding of binary trees.

1) Implement a recursive function `RecursiveInsertBinarySearchTree()` for inserting elements into a binary search tree. Write a main program to test your function.

   You may use the following declaration for your binary tree:

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

Tree RecursiveInsertBinarySearchTree(int x, Tree t)
{
   if (t == NULL)
   {
         /* Create and return a one-node tree */
         t = (Tree) malloc(sizeof(struct tree_node));
         if (t == NULL)
               fatal_error("Out of space!!!");
         else
         {
               t->val = x;
               t->left = t->right = NULL;
         }
   }
   else if (x < t->val)
         t->left = RecursiveInsertBinarySearchTree(x, t->left);
   else if (x > t->val)
         t->right = RecursiveInsertBinarySearchTree(x, t->right);

   /* else x is in the tree already. We'll do nothing */
   return t; /* Don't forget this line!! */
}
```

2) Implement an iterative C function `IterativeFindMinimum()` for finding the minimum value in a binary search tree.
3) Implement an iterative C function `IterativeFindMaximum()` for finding the maximum value in a binary search tree.
4) Implement a recursive C function `RecursiveFindMaximum()` for finding the maximum value in a binary search tree.
5) Implement a recursive C function `RecursiveFindMinimum()` for finding the minimum value in a binary search tree.
6) Implement a C function called `findElement()` that takes a value and returns the node that includes that value.

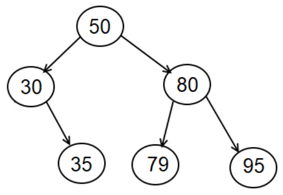
7) Implement a recursive C function `PrintTree()` for traversing a binary search tree in order printing the stored values along the way.

```
void printTree(struct Tree *p)
{
    if (p!=NULL){
    PrintTree(p->left);
    printf("%d\n", p->val);
    PrintTree(p->right);}
}
```

8) Implement an iterative C function `DeleteBinarySearchTree()` for deleting a value from a binary search tree.
9) Implement a recursive C function `PrintBinaryTreeStructure()` which shows the structure of the binary tree.
10) Implement a recursive function for `CountNodes()` counting the number of leaves in a binary search tree. **Hint:** Modify your function `PrintTree()` to keep track on the number of nodes visited.
11) Implement a recursive function for `CountLeaves()` counting the number of leaves in a binary search tree. **Hint:** Modify your function `PrintTree()` to also check whether or not a node is a leaf and to update the number of leaf nodes encountered.
12) Implement a recursive C function `TreeHeight()` for finding the height of a binary search tree. The height of a binary search tree is the length of the longest path from the root to a leaf. **Hint:** Modify your function `PrintTree()` to determine the height of each node's left and right subtree.
13) Implement a C function `CopyTree()` that takes as input a binary tree and makes a copy of the binary tree.
14) Write a C function `IdenticalTrees()` which checks whether two trees are identical. Two trees are considered identical if (1) they have the same structure and (2) all the values stored in corresponding nodes are identical.
15) Implement an iterative C function `IterativeInsertBinarySearchTree()` for inserting a new value into a binary search tree.
16) Implement in order traversal of a binary search tree **without** using recursion. **Hint:** Use a stack to keep track of the order in which nodes are visited for the first time in the tree.

Please note that from this worksheet, you need to submit the solution of the following question (17)

17) Write a program which will do following;
   a) A function `CreateTree()` that takes 6 elements from the user and creates corresponding BST. For instance; if the entered elements are 50,30,35,80,79,95 then the following BS will be created:



   b) A function `LevelOrderDisplay()` that takes the created tree as an input and displays tree elements in level-order (i.e. using breadth-first tree traversal). To implement this breadth-first tree traversal you should use queue data structures.

Please note that you can write helper functions.