CS29003 ALGORITHMS LABORATORY
Assignment No: 1
Last Date of Submission: 29–July–2015

Let T be a binary search tree (BST) with integer key values, and let x and y be two integers with $x \leq y$. The goal of this assignment is to design an algorithm to print all the key values v stored in T, that satisfy $x \leq v \leq y$. Let h be the height of T, and k the number of values printed. Your printing algorithm should run in $O(h + k)$ time, and use only O(1) additional variables (if you count the recursion stack, the space requirement would be $O(h)$).

First, define a data type to store a node in the BST. Each node should consist of an integer key value and two child pointers (left and right). A node in the tree must not store any additional element (like the parent pointer).

Now, solve the following parts in order to arrive at the desired printing algorithm.

Part 1: Write an *insert*() function for inserting a value x in a BST T. The function should return the modified tree after the insertion (or the original tree if x is already stored in T). Use the standard BST-insertion procedure.

Part 2: Write a function *printBST*() to print a binary search tree T. A BST is uniquely identified by its pre-order and in-order listings. So it suffices to print these two listings. You need to write two other functions *preorder*() and *inorder*() for this. A fancy printing is shown in the sample output. Do not waste time on implementing it. If you like this printing, try it offline as a take-home programming challenge.

Part 3: Write a function *search1*() that, given a BST T and an integer x as input, returns a pointer to the tree node storing the smallest value $\geq x$. Notice that x itself need not be present in T. In that case, a pointer to the tree node storing the closest value larger than x should be returned. If all the values stored in T are smaller than x, then the NULL pointer should be returned. The running time of this function should be $O(h)$.

Part 4: Write a function *search2*() that, given a BST T and an integer y as input, returns a pointer to the tree node storing the largest value $\leq y$. Notice that y itself need not be present in T. In that case, a pointer to the tree node storing the closest value smaller than y should be returned. If all the values stored in T are larger than y, then the NULL pointer should be returned. The running time of this function should be $O(h)$.

Part 5: Let p be a node in a BST T. There exists a unique path from the root of T to p. By an ancestor of p, we define any node on this path. That is, the ancestors of p are p itself, the parent of p, the grandparent of p, the grandgrandparent of p, and so on. Given two nodes p and q in T, the *lowest common ancestor* of p and q is a node r in T such that r is an ancestor of both p and q, and is farthest from the root among all the common ancestors of p and q. Since the root is a common ancestor of every node in T, the lowest common ancestor of any nodes p and q exists and is uniquely defined. Write a function *lca*() that, upon the input of a BST T and two pointers p, q to nodes in T, returns a pointer to the lowest common ancestor of p and q. The case $p = q$ should be allowed. The running time of this function should be $O(h)$.

Part 6: Write a function *printrange*(), that upon the input of a BST T and two integers x and y satisfying $x \leq y$, prints all the values v stored in T such that $x \leq v \leq y$. The values should be printed in sorted order (increasing). The running time and space requirement of this function should be as mentioned above, that is, $O(h + k)$ and $O(h)$, respectively.

Part 7: Write a *main*() function to do the following:

- Initialize T as an empty BST.
- Read n (the number of insertions) and integers a_1, a_2, …, a_n from the user. The integers a_1 through a_n are inserted one by one in T using the *insert*() function of Part 1. After all insertions are made, you print T using the function *printBST*() of Part 2.
- Read two integers x and y from the user. Run the two functions *search1*(x) and *search2*(y). Let the two pointers returned by these calls be p and q. Print the key values pointed to by p and q (see Parts 3 and 4).
- Run *lca*(p,q), and print the key value pointed to by the pointer returned by the call (see Part 5).
- Invoke the function *printrange*(T, x, y) of Part 6 in order to print the values in T in the range $[x, y]$.

Submit a single C/C++ file solving all the parts. Do not use global or static variables. Do not use the C++ STL.

Sample Output

```
n = 20
+++ Insert      : 86 58 82 78 48 85 28 18 14 69 11 3 37 50 17 96 77 11 43 56

+++ The BST created has the following listings
    Preorder    : 86 58 48 28 18 14 11 3 17 37 43 50 56 82 78 69 77 85 96
    Inorder     : 3 11 14 17 18 28 37 43 48 50 56 58 69 77 78 82 85 86 96

+++ The following fancy printing of the BST is not for submission.
+++ You may implement it as a take-home programming exercise.
    86
    L-->58
    |   L-->48
    |   |   L-->28
    |   |   |   L-->18
    |   |   |   |   L-->14
    |   |   |   |   |   L-->11
    |   |   |   |   |   |   L-->3
    |   |   |   |   |   |   |   L-->NULL
    |   |   |   |   |   |   |   R-->NULL
    |   |   |   |   |   |   R-->NULL
    |   |   |   |   |   R-->17
    |   |   |   |   |       L-->NULL
    |   |   |   |   |       R-->NULL
    |   |   |   |   R-->NULL
    |   |   |   R-->37
    |   |   |       L-->NULL
    |   |   |       R-->43
    |   |   |           L-->NULL
    |   |   |           R-->NULL
    |   |   R-->50
    |   |       L-->NULL
    |   |       R-->56
    |   |           L-->NULL
    |   |           R-->NULL
    |   R-->82
    |       L-->78
    |       |   L-->69
    |       |   |   L-->NULL
    |       |   |   R-->77
    |       |   |       L-->NULL
    |       |   |       R-->NULL
    |       |   R-->NULL
    |       R-->85
    |           L-->NULL
    |           R-->NULL
    R-->96
        L-->NULL
        R-->NULL

x = 30
y = 70

+++ search1(30) : 37
+++ search2(70) : 69

+++ lca(37,69)  : 58

+++ Values in T between 30 and 70 are: 37 43 48 50 56 58 69
```