CSC 212 Midterm 2 - Spring 2015

College of Computer and Information Sciences, King Saud University
Exam Duration: 90 Minutes

23/04/2015

Question 1 [35 points]

1. Write the static method *moveAfter* (user of the Stack ADT), that takes as input two stacks st_1, st_2 and an index i. It moves the elements of stack st_2 after the element at position i in stack st_1. Assume that i is within the range of stack st_1, and that the top element has an index of 0. The method signature is *public static <T>void moveAfter(Stack<T>st_1, Stack<T>st_2 , int i).*

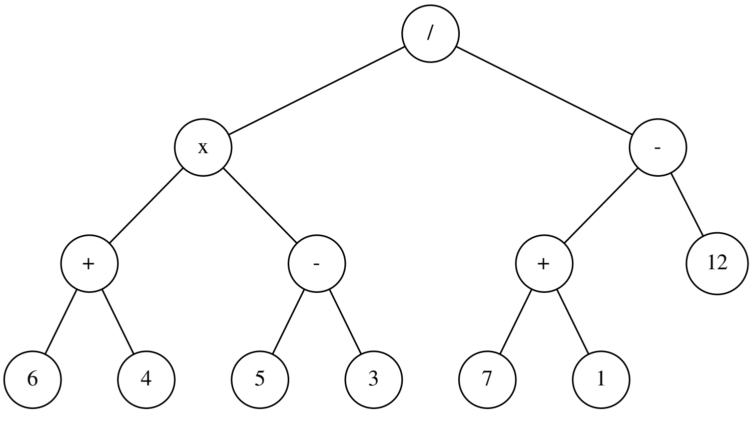
   Example 1.1. *If st_1 (top to bottom): 5, 2, 4, 1 and st_2 (top to bottom): 8, 9. After calling* moveAfter*(st_1, st_2 1), st_1 will be (top to bottom): 5, 2, 8, 9, 4, 1.*

2. Write the static method *countEquals* (user of the Stack ADT), that takes as input a stack *st*, and an element *e*. It returns the number of elements of stack *st* matching *e*. The stack st should **not change** after calling the method. The method signature is *public static<T>int countEquals(Stack<T>st, T e).*

   Example 1.2. *If st (top to bottom): 5, 2, 4, 1, 4, 2, 4. Then* countEquals*(st, 4) returns 3,* countEquals*(st, 2) returns 2, and* countEquals*(st, 7) returns 0.*

Question 2 [20 points]

1. Give the preorder, inorder and postorder traversals of the tree shown below.



2. Convert the following expression to postfix: ”3 + 4 × 9 - 4 × 6 × 7 - 3”.

3. Trace the evaluation of the following postfix expression: ”8 6 + 7 5 - × 6 8 + 7 - /” using a stack. Draw the stack after every push or pop operation (you have to **draw the stack 13 times** in total).

Question 3 [35 points]

1. Using the Binary Search Tree in Figure 1, insert the following:
   (a) 91 into the **Original tree**.
   (b) 85 into the **Original tree**.
   (c) 4 into the **Original tree**.
   (d) 15 into the **Original tree**.
   (e) 79 into the **Original tree**.

2. Using the Binary Search Tree in Figure 1, delete the following:
   (a) 92 from the **Original tree**.
   (b) 21 from the **Original tree**.
   (c) 89 from the **Original tree**.
   (d) 16 from the **Original tree**.
   (e) 80 from the **Original tree**.

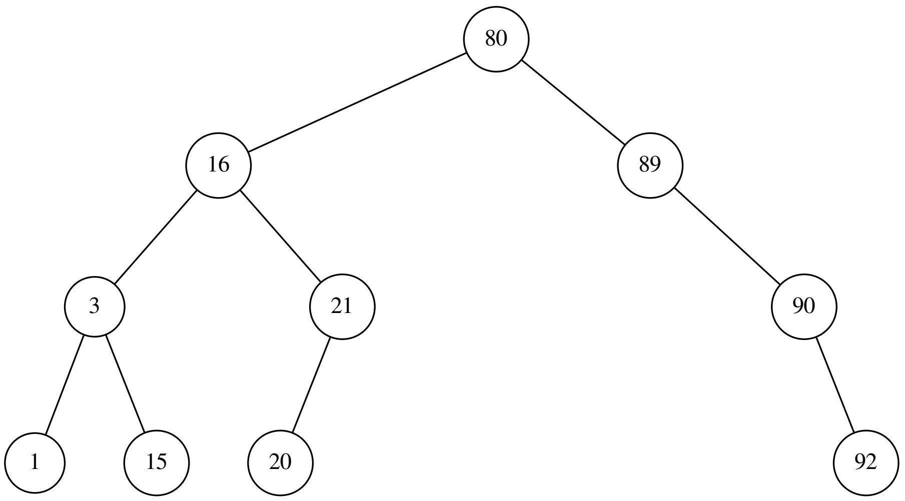
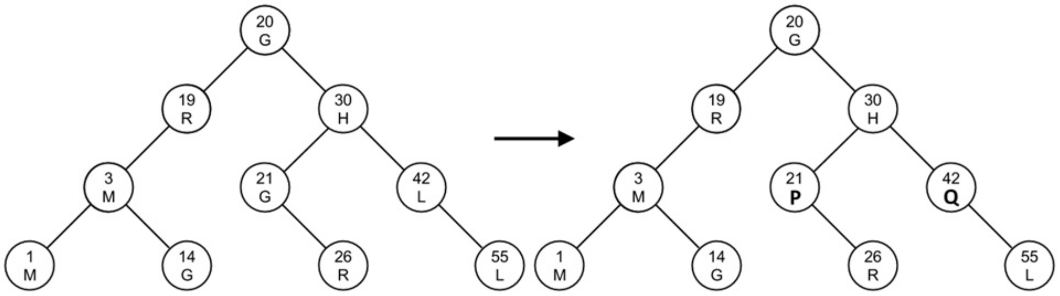


Figure 1: A BST.

3. Write the method *updateChildrenData*, member of the class BST, which takes as input a key k and two elements e_1 and e_2. Then it searches the tree for the key k. If not found, false is returned. When found, it updates its left child data with e_1 and its right child data with e_2 only if both of them exist then returns true. If one or both children were not found, false is returned. Assume that the tree is not empty. **Do not use any auxiliary data structures and do not call any methods**. The method signature is *public boolean updateChildrenData(int k, T* e_1*, T* e_2*).*

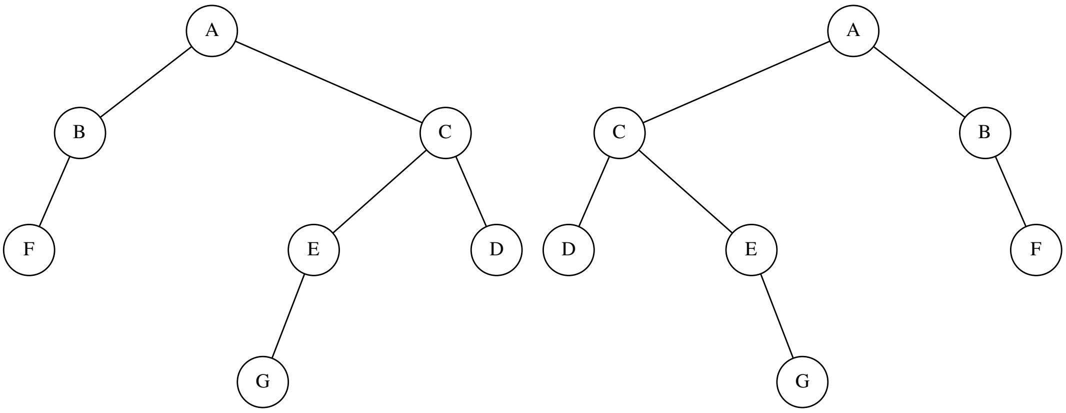
Example 3.1. *The call* updateChildrenData*(19,G,Z) on the tree shown below (on the left) returns false. The call* updateChildrenData*(18,O,F) returns false. The call* updateChildrenData *(30,P,Q) returns true, and the tree changes as shown to the right.*



Question 4 [10 points]

Write the **recursive** method *isMirror*, member of the class *BT* (Binary Tree), that takes as input a binary tree and returns true if the two trees are the mirror image of each other. The method signature is *public boolean isMirror(BT< T >bt)* (this method must call the private recursive method *recIsMirror*). **Important**: Non-recursive solutions are not accepted.

Example 4.1. *The two trees shown below are mirror images of each other.*



ADT Stack Specification

- Push (Type e): **requires**: Stack S is not full. **input**: Type e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
- Pop (Type e): **requires**: Stack S is not empty. **input**: **results**: the most recently arrived element in S is removed and its value assigned to e. **output**: Type e.
- Empty (boolean flag): **requires**: none. **input**: none. **results**: If Stack S is empty then flag is true, otherwise false. **output**: flag.
- Full (boolean flag): **requires**: none. **input**: none. **results**: If S is full then Full is true, otherwise Full is false. **output**: flag.