Name | ID | Seq. | Section

CSC 212 Midterm 2 - Fall 2017

College of Computer and Information Sciences, King Saud University
90 Minutes - 30/11/2017

Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q3.1	Q3.2	Total

Question 1 [30 points]

1. Trace the evaluation of the following expression **(draw the stack after every push operation**: `8 4 / 7 3 * - 4 2 * 1 - +`

   Answer:

2. Indicate the **preorder**, **inorder** and **postorder** traversals of the tree shown in **Figure 1**. **Choose only one answer for each traversal, for example, Preorder: 1, Inorder: 3, Postorder: 2.**

Preorder	Inorder	Postorder
1. F R A D S O I M L H	**1.** I D R L S M F H O A	**1.** I D M L S R H O A F
2. F R D I S M L A O H	**2.** D I M S L R F A H O	**2.** D I M L S R A H O F
3. F R D I S L M A O H	**3.** I D R M S L F A O H	**3.** I D L M S R H O A F
4. F R D S I M L A H O	**4.** D I R M S L F A H O	**4.** I D M L S H O R A F

Name | ID | Seq. | Section

Figure 1: Binary Tree.

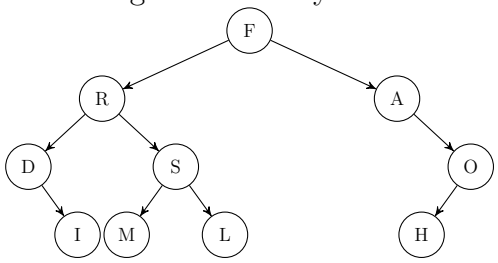
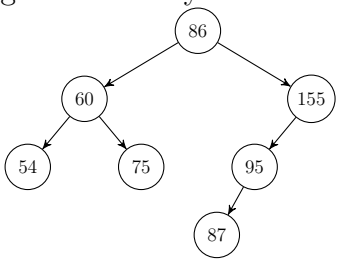


Figure 2: Binary Search Tree.



3. Given the initial BST shown in **Figure 2**, draw the resulting BST after each of the following sequences of operations. **For each sequence, you should draw one final tree result. Each sequence should be applied on the original tree.**

   (a) insert(40); findKey(80); insert(100); removeKey(155);
   (b) insert(90); removeKey(86); removeKey(87); insert(86);
   (c) removeKey(60); insert(58); insert(75); removeKey(75);

   Answer:

Question 2 [35 points]

1. Write the method **public static** `<T>` **void** `insert (Stack<T> st1, Stack<T> st2,` **int** `n)`, user of the ADT Stack, that inserts all elements of `st1` after the n^{th} element of `st2` without changing the order. The stack `st1` must not change after the insertion. Assume that `n` is valid and that the top element is the 1^{st} element in the stack . **You can use only one stack as an auxiliary data structure**.

   Example 2.1. *If* $st1 : X, Y, Z$ *(from top to bottom) and* $st2 : A, B, C, D, E, F, G$ *(from top to bottom), then after calling* `insert(st1,st2,4)`, $st2 : A, B, C, D, X, Y, Z, E, F, G$.

   Answer:

2. Write the method **public static** `<T>` **boolean** `insertAtEnds(BT<T> bt, T e, Relative rel)`, user of the ADT Binary Tree, that inserts the element `e` at the leftmost node and the rightmost node according to `rel`. If `e` is inserted at both ends, the method returns true, otherwise false. Assume that `bt` is not empty.

   Answer:

Question 3 [35 points]

1. Write the **recursive** method **private** `T findMin(BTNode<T> t)`, member of the class BT (binary tree), that receives a node `t` (where `t != null`), searches for the minimum value in the subtree rooted at `t` and returns it. Assume that the generic type `T` extends the interface `Comparable` and use the method compareTo() for comparison. **Do not call any methods of the class *BT*. Non-recursive methods are not accepted.**

   int `compareTo(T o)` compares the current object with object `o`. Returns a negative integer, zero, or a positive integer as the current object is less than, equal to, or greater than `o`.

   Answer:

2. Write the method **private void** `swapMaxMin(int k)`, member of the class BST (Binary search tree), that swaps the data of the node having the **maximum key** in the left subtree of the node with key `k` and the data of the node having the **minimum key** in the right subtree of the node with key `k`. If `k` does not exist or the corresponding node has less than two children nothing changes. **Do not call any methods of the class BST**.

   Answer:

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.

ADT Binary Tree Specification

boolean empty(): **Requires**: none. **Results**: returns true if the binary tree (BT) has no nodes.
boolean find (Relative rel): **Requires**: BT is not empty. **Results**: the current node of BT is determined by Relative and the current node prior to the operation as follows (always return true unless indicated so): (1) rel = Root: current = root (2) rel = Parent: if the current node has a parent then parent is the current node; otherwise returns false (3) rel = LeftChild: if the current node has a leftchild then it will be the current node; otherwise returns false (4) rel = RightChild: same as above but for rightchild.
boolean insert(Relative rel, Type val): **Requires**: either (1) BT is empty and rel = Root; or (2) BT not empty and rel= Root . **Results**: as follows: (1) rel = Root: create a root node with data = val. (2) rel = Parent: nonsense case. (3) rel = LeftChild: if current node does not have a leftchild then make one with data = val. (4) rel = RightChild: same as above but for rightchild. In all the above cases if the insertion was successful then it will be designated as current node and returns true, otherwise current remains unchanged and returns false.
void deleteSub(): **Requires**: BT is not empty. **Results**: the subtree whose root node was the current node before this operation is deleted from the tree. In case the resulting tree is not empty then current = root.

Note: Relative is enumerated type and is confined to the values Root, Parent, LeftChild, RightChild