1. ***Algorithm for constructing an expression binary tree:***
2. Convert expression to the list of operators and operands
3. Create a binary tree for the expression with ‘’ in the root and stack
4. Push tree to the stack and make it to be current tree
5. Read the list and:
6. If the current element is ‘(‘: create a blank left child for the current tree, push current tree to the stack, and switch current tree to the left child
7. If the element is an operand: set it as a root value of current tree and pop the element from the stack
8. If the element is operator: set it as a root value of current tree, insert blank right child to the current tree, push current tree to the stack and switch current to the right child
9. If element is ‘)’ pop an element from the stack

***Algorithm for evaluating:***

1. Traverse the tree using postorder traversal and obtain the list with the expression in postfix form
2. Use functions from previous HW to evaluate the postfix expression
3. **a)** One way is to use **preorder** traversal of the given tree because while reading this traversal it will be obvious where is the root of the whole tree and each subtree (which does not apply for inorder and postorder traversals) and the binary tree could be restored using the properties of BST after defining the root.

**b) *Algorithm for restoring the tree (from preorder traversal word-sequence):***

1. Put the sequence of words from the file to a list
2. Create a binary tree and put the first element from the list to its root
3. Remove first element from the list
4. Create a stack and push tree to the stack
5. Make the tree as a current
6. Read element from the list:
7. If element<top element of stack: insert the element as left child of the current tree, switch current tree to the left child and push current tree to the stack
8. Else: pop elements from stack until the top element is greater than current element from list. Switch current tree to the last element from the stack which is smaller then current list’s element. Insert current element to the right child of current tree. Switch tree to the right child. Push current tree to the stack
9. Repeat g-h until the end of the list

***Algorithm for level-printing:***

1. Create queue for parents and another queue for children
2. Push root to the parents-queue
3. While parents-queue is not empty print its elements and push children of every node from there to the children-queue
4. Switch queues
5. Repeat c-d until parents-queue is not empty