ALERT!

1. The objective of this lab is understanding tree data structure.
2. This is an individual lab, you are strictly **NOT** allowed to discuss your solution with fellow colleagues, even not allowed asking how is he/she is doing, it may result in negative marking. You can **ONLY** discuss with your TAs or with me.
3. Beware of memory leaks and dangling pointers.
4. Pay attention to **GOOD coding conventions** e.g.
   - Proper indentation.
   - Meaning variable and function names.
   - Use camelCase naming convention
   - Use meaningful prompt lines/labels for all input/output
5. **Anyone caught in act of plagiarism would be awarded an "F" grade in this Lab.**

Task 01: [15 Marks]

Implement the binary tree using linked implementation as we discussed in class.

```
// forward declaration of template class BTree
template<class T>
class BTree;

template<class T>
struct TreeNode
{
   T  data;
   TreeNode<T>*  left;
   TreeNode<T>*  right;
     // Methods…
};

template<class T>
class BTree
{
    TreeNode<T>* root;
     // Methods…
};
```

Implement following functions for BTree class:

1. **Constructor, destructor, Copy-constructor.**
2. **void setRoot (T value);**
3. **TreeNode<T>* getRoot ( );**
4. **void setLeftChild (TreeNode<T>* node, T value);**
5. **void setRightChild (TreeNode<T>* node, T value);**
6. **TreeNode<T>* getLeftChild (TreeNode<T>* node);**
7. **TreeNode<T>* getRightChild (TreeNode<T>* node);**
8. **void preOrder(TreeNode<T>* node);**
9. **void inOrder(TreeNode<T>* node);**
10. **void postOrder(TreeNode<T>* node);**
11. **void levelOrder(TreeNode<T>* node);**
12. **void delete (TreeNode<T>* node)**
    Deletes the given node and all its decedents.

Expression Trees: [15 Marks]

One of the applications of binary trees is an unambiguous representation of arithmetic, relational or logical expression. Once an expression tree is constructed from postfix expression string, different representations of the expression i.e. prefix notation, post fix notation and infix can be obtained by preorder, postorder and inorder traversals respectively. Your task is to generate an expression tree from a given postfix expression (by user) using Tree ADT that you have implemented in task 1. Also, implement a driver program that call all traversals on this expression tree.