# Lab 1 Analysis-Haoyu Qi

## Description

In this project, I create my own program to realize the ADT binary tree and use tree to do the recursion in order to solve the problem. As we all know, binary trees and recursion are inextricably linked, the preorder/postorder of the binary trees.

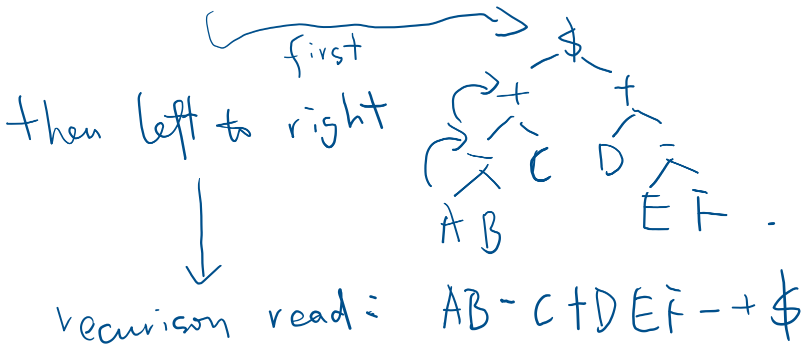
## Justification of Tree and Recursion

In this Lab's question, when we facing the prefix-postfix conversion, the tree is a good solution for us to deal with our problem because we the order of the leaves and node could represent the pre/in/postfix, like leaf leaf-node -right leaf is infix and left leaf-right leaf-node is the prefix. The operator should always be placed at the node. And the operons should be placed at the leaves. And the ADT binary tree is definitely a kind of recursion.

## Appropriateness of Application

In my application, I firstly realize the conversion of prefix to postfix using binary tree, this is the basic work. Plus, like the Lab 1, I also realize the input format mistake checking to make sure only the correct input prefix expression will be conversed to postfix expression, and for different kinds of mistake type, I gives different feedbacks. Also, I apply modular design in the application, separate functions in different class file, this makes the advance and modification easier.The other ADT can barely realize this function, because have strong connections between the elements make them hard to recursion, but actually the essence of recursive traversal helps us build the stack structure, and stack needs to remember the value of each node, also tree give me a very direct impression to generate the program.

For example:$+-ABC+D-EF



**Description and justification of your design decisions**

I use specific classes to define the binary tree and nodes, these files contains some other functions like addNode that may not be used in this lab, but still works.
The tree is actually be constructed after the node was set, and operons set into the leaves and operators set into the node from left to right.
The I/O is realized by BufferedRead and BufferedWrite as we shown in the Project0.
I wrote a distinct class to distinguish the character type because it is needed in both conversion and reading.
Also, I designed a methods to check the input mistake, the mistake includes containing invalid elements like “@, # ,&” and the operons in this lab contains characters and integers. The application can check 2 kinds of mistake input format, the incorrect numbers of operons and operaters, and the bad prefix format unable to convert to postfix.
I apply some user friendly designed in the application, like file input guild, error warning text or a as compact as possible output format.

**Efficiency with time and space**

The complexity can be divided into the binary tree construction and the binary tree read, the construction based on the nodes number and nodes only depends on the operators number so it is O(n). The traversal read complexity is also O(n) because it is also be depend on the numbers of operators and operons. So the total time complexity should be O(n)+O(n), still O(n).
The space complexity is depend on the nodes number and the as I said the system actually process like stack, so is should be also O(n).

**What I have learned**

Recursion is a extremely abstract concept and really hard for me to understand, but this lab gives me a inspiration about the connection between tree and recursion, though we have discussed this relationship in the class but the getting my hands on programming deepened my understanding. The recursion is beautiful and I found that after we construted the tree and node, the codes of the recursion seems more concise.

**What I might do differently next time**

After finishing Lab1 & Lab2, I may prefer stack in this problem's solution because the stack seems more easier for me. Also I know that my recursion code seems too fat, it still have potential to advance. And as we know the complexity of tree traversal can be O(logN) in binary search situation, so maybe my implement is not the best. Also the recursion is very easy to overflow, so when facing a long input how to make less recursion level is also a issue.