CSCI 2201, Spring 2019
Programming Assignment Two
Converting/Evaluating a simple math expression

This assignment is to ask you to (1) construct a binary tree structure to store a parsed math expression, (2) perform pre-order traversal, and write a pre-order stack machine to evaluate the expression, (3) perform post-order traversal, and write a post-order stack machine to evaluate the expression. You need to open the file, `expression.txt`, which contain a parsed math expression (a complete binary tree, the root node index is 0 and the left and right child nodes are indexed based on their parent node index). Please refer to the file `epression.txt` for the binary tree representation.

Input

A parsed math expression from the input file, `expression.txt`. This is a well formed math formula without syntax error. All operators, parentheses, and integer values are separated by a space.

Output

1) Pre-order traversal sequence and the evaluation result of the pre-order stack machine
2) Post-order traversal sequence and the evaluation result of the post-order stack machine.

Specifications

Your program must meet the following specifications:

1. Work on your own.
2. The name of the source code file must be Program02.xxx, where xxx can be either java, c/cpp (ANSI), or extension of other high level programming language.
3. Comments at the top with your name, e-Name (e-mail), date, course number, and short description of the program.
4. You need to implement your own stacks for your stack machines (methods, functions, or procedures for your two stack machines), applying existing package will not acquire credit.

Part I – Design document, data structure, algorithm, coding and implementation.

Part II – convert your algorithm to a program using your preferred high level programming language.

Example

A math expression `( 3 –  2 * 5 ) + ( ( 5 – 2 ) – ( 3 + 15 ) )` after parsing, one of the parsed binary tree will be: `+ - - 3 * - + _ _ 2 5 5 2 3 5`, where the character underscore (_) mean empty (null) node, which may be stored in the file `expression.txt`.

Assume your program is written in java
To test your program, I will first issue the command

```
javac Program02.java
```

then

```
java Program02 expression.txt
```

Assume your program is written in c (or cpp)
To test your program, I will first issue the command

```
gcc –o Program02 Programp2.c (cpp)
```

then

```
Program02 expression.txt      (under DOS or powershell)
./Program02 expression.txt    (under Unix/Linux)
```

Example of the input binary tree from the file `expression.txt`**:**

```
+ - - 3 * - + _ _ 2 5 5 2 3 15
```

The output of your program of the above parsed math expression should be:

```
Pre-Order:             + - 3 * 2 5 – - 5 2 + 3 15
Pre-Order Evaluation:  -22
Post-Order:            3 2 5 * - 5 2 – 3 15 + - +
Post-Order Evaluation: -22
```

Note: other parsed math expressions maybe used to test your program.