Brendan Bovenschen
CWID: A20323473

CS3353 Homework 4 Program - Expression Tree

Introduction

Expression trees are the most common solution for performing operations in a predetermined order. This system is implemented in calculators, computers, and any device utilizing some form of sequential processing capabilities. Using a tree, certain conventions can be used to compute the equation represented by the tree.

Build Tree

Before building the tree, we must first take divide the string from the input using a string stream. We are able to iterate through each element in the string using this stream. Using a set of rules based on what character is found, we can create a tree using a doubly linked list of nodes. These rules are as follows:

'(': if left == null, create a left node, else, create right
')': move currNode to its parent
operator: currNode’s parent value = operator, create right node from parent, this is now currNode
number: write the number to current node

currNode represents the current node that the system has saved a reference too and acts as a cursor for traversing the tree. These rules will create an expression tree with parent nodes containing operators and leaf nodes of integers. A syntax check is also implemented since the final node should be the head node. Every element of the list will only be iterated once, therefore, the big O running time of this sequence will be O(n) with n being the number of elements in the input string.

Create Postfix

The postfix notation is built on the post-order traversal of the tree. By starting at the head node, and saving a node after traversing the left and right node to a compilation string, we can obtain a string with each element in postfix order based separated by spaces. Since each element is iterated only once, the worst-case running time will follow O(n) with n being the number of element.

Solve Expression Through Tree

The expression tree can be traversed in postorder to obtain a final value being the solved expression. A function that takes a node and returns an integer can be implemented recursively to find the solved expression at each parent node. Starting from the head node, the function will check if the current node has children. If it does, this must be an operator, and the recursive function will be implemented on each of its children and the method will return the evaluated sum of the left and right child. If the node has no children, it is an integer and the integer value will be returned. This process will occur until a final value is found and each node has been traversed once. This means that the big O of this function will follow O(n).

Solve Expression Through Postfix

Using the previously found postfix form of our binary tree, we can resolve the same value found in the expression tree calculation from our postfix string. At each operator, we can replace it with the solved form of the two integers preceding it and erase these two integers. This will be done with every operator until the length of the list is '1'. We will see the same final answer as if we had solved it with the expression tree since the same operations will be performed in the same order. This will solve the expression tree in the worst time running time of O(n) when each element is traversed once.

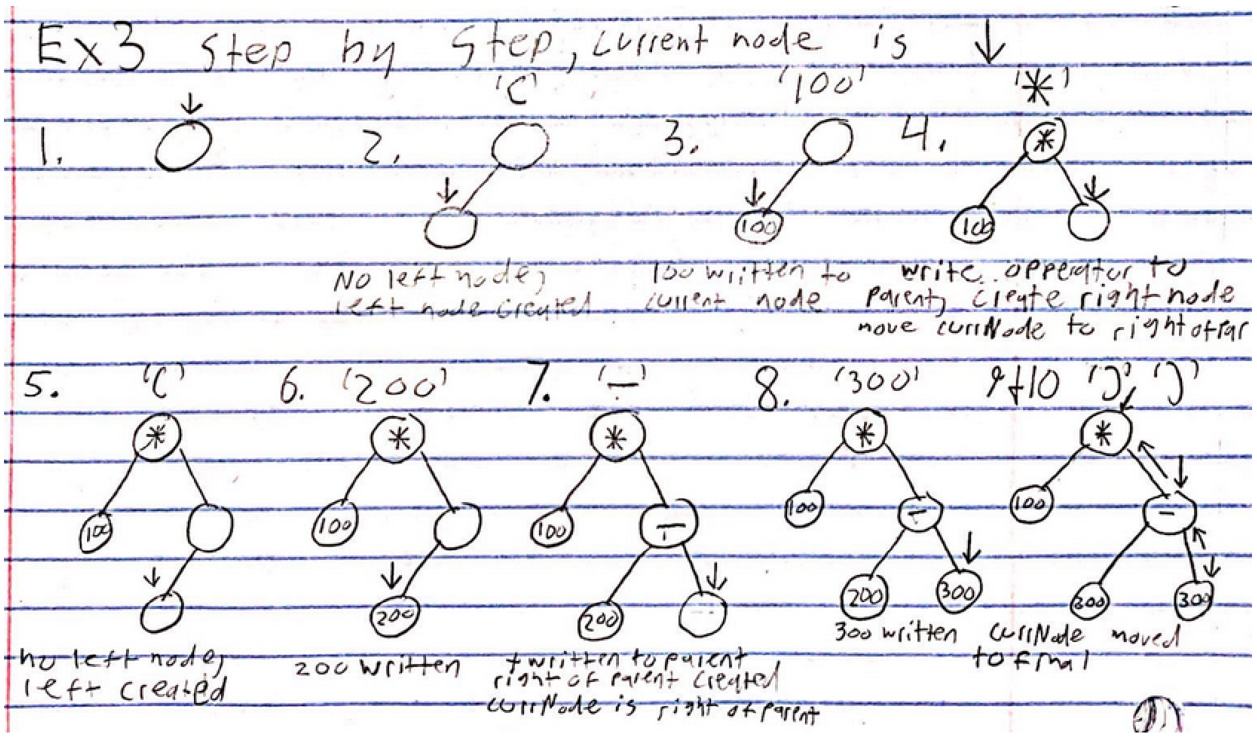
Example

Each of these examples processes will use the third expression given in the homework:

$$(100 * (200\ -\ 300))$$

Build Tree

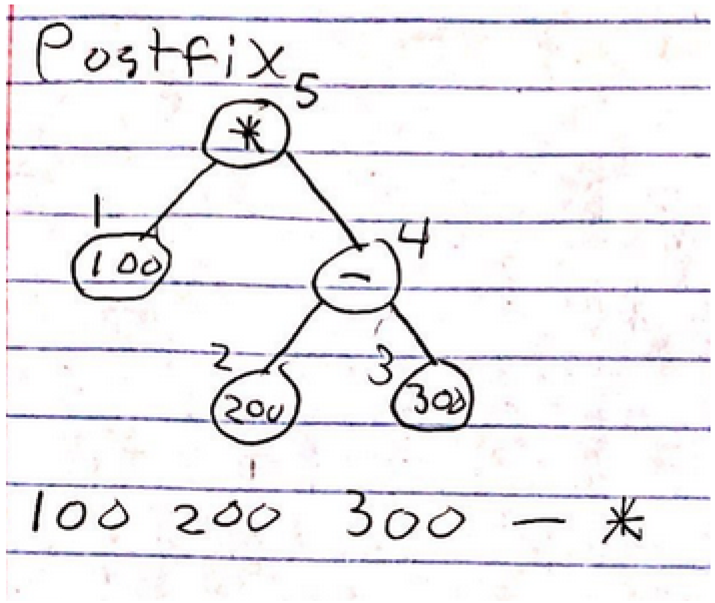
Each step in the process is shown below using the rules stated earlier for this method.


Ex3 step by step, current node is ↓
1.
2. '('
No left node, left node created
3. '100'
100 written to current node
4. '*'
write opperator to parent, create right node move currNode to right of par
5. '('
no left node, left created
6. '200'
200 written
7. '−'
− written to parent right of parent created currNode is right of parent
8. '300'
300 written
9+10 ')', ')'
currNode moved to final

Brendan Bovenschen
CWID: A20323473

Create Postfix

Using postorder traversal, we can visit each element to create the postfix form.



Postfix form: 100 200 300 - *

Solve Expression Through Tree

From the postorder traversal, we can solve the expression using our expression tree.

1. 100: return 100
2. 200: return 200
3. 300: return 300
4. -: get the value of each of its children and operate accordingly, 200 - 300, return -100
5. *: get the value of each of its children and operate accordingly, 100 * -100, return -10000

Final answer: -10000

Solve Expression Through Postfix

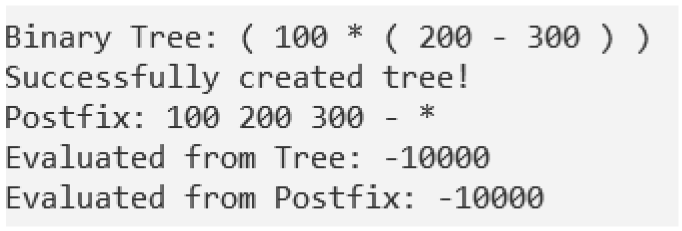
Using the postfix form, we can find solve our expression by operating on each element before an operator.

Step 1. 100 200 300 - * Initial postfix form
Step 2. 100 (200 300 -) * Operate on the two elements before the operator, 200 - 300 = -100
Step 3. 100 -100 * Replace the operator with the value found earlier and delete both predecessors
Step 4. (100 -100 *) Operate on the two elements before the operator, 100 * -100 = -10000
Step 5. -10000 Replace the operator with the value found earlier

Final answer: -10000

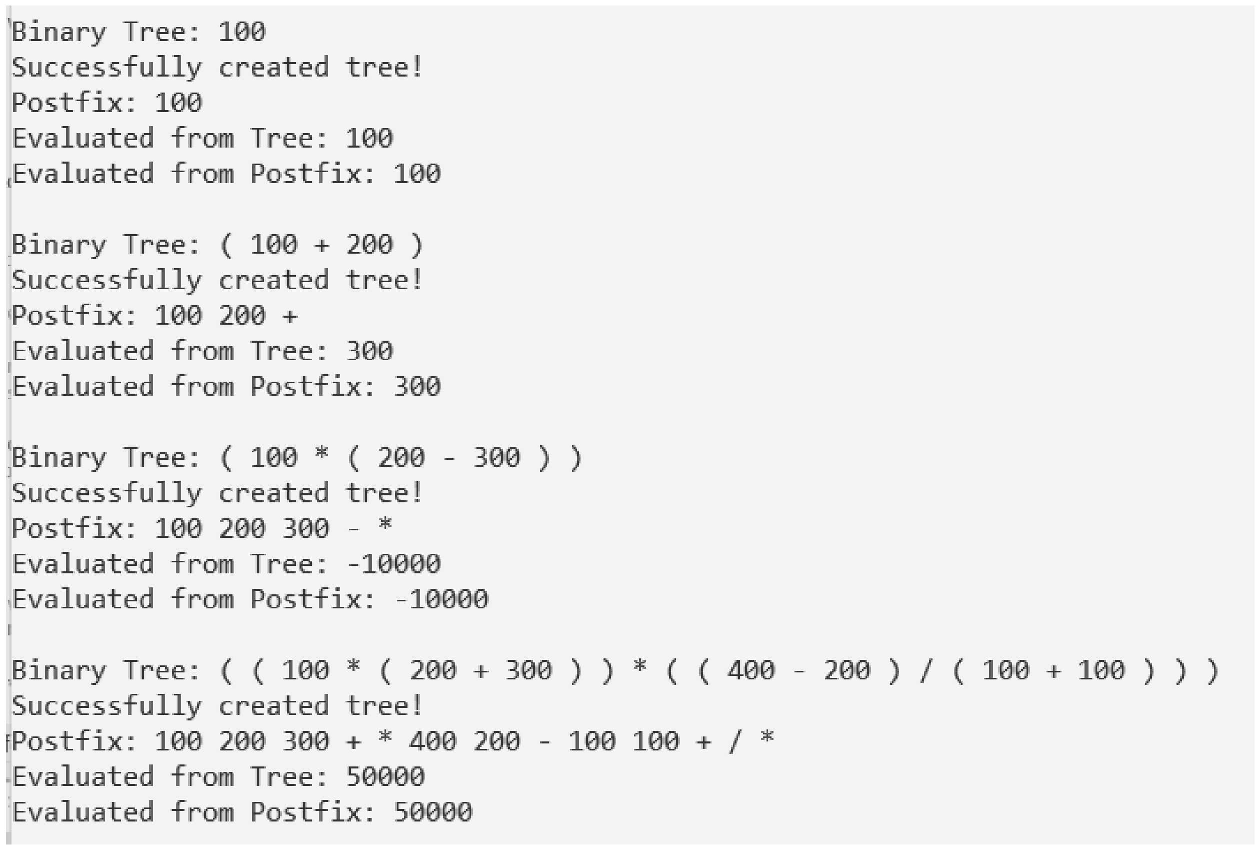
Brendan Bovenschen
CWID: A20323473

We can see that both final answers are the same in either method since they perform the same operations in the same order. These values have been replicated in the examples within this program as well as the other example cases stated in the homework.



```
Binary Tree: ( 100 * ( 200 - 300 ) )
Successfully created tree!
Postfix: 100 200 300 - *
Evaluated from Tree: -10000
Evaluated from Postfix: -10000
```

All Examples



```
Binary Tree: 100
Successfully created tree!
Postfix: 100
Evaluated from Tree: 100
Evaluated from Postfix: 100

Binary Tree: ( 100 + 200 )
Successfully created tree!
Postfix: 100 200 +
Evaluated from Tree: 300
Evaluated from Postfix: 300

Binary Tree: ( 100 * ( 200 - 300 ) )
Successfully created tree!
Postfix: 100 200 300 - *
Evaluated from Tree: -10000
Evaluated from Postfix: -10000

Binary Tree: ( ( 100 * ( 200 + 300 ) ) * ( ( 400 - 200 ) / ( 100 + 100 ) ) )
Successfully created tree!
Postfix: 100 200 300 + * 400 200 - 100 100 + / *
Evaluated from Tree: 50000
Evaluated from Postfix: 50000
```