DESIGN DOCUMENT

This Design Document lays out the structure and plan on implementing an expression tree on assignment 1b using results from assignment 1a.

ASSIGNMENT 1A PART2

COP 3530

DR. CHERYL RESCH

MOHAMMAD IMMAM

UFID 2989-1464

10TH SEPTEMBER, 2019

Contents

Design Document for Project 2

INTRODUCTION

Assignment 1A, was to validate an infix expression and convert it to a postfix expression. For the next assignment, we take in the postfix output from 1A to create an expression tree.

WHAT DATA STRUCTURE WILL YOU USE FOR THE EXPRESSION TREE?

The expression tree is mainly going to be implemented using a tree structure with roots and leaves.

HOW WILL YOU DESIGN YOUR NODE CLASS FOR THE EXPRESSION TREE? WHAT ARE THE PROPERTIES OF THE NODES? WHAT KIND OF DIFFERENT NODES WILL BE REQUIRED? HOW WILL YOU IMPLEMENT THESE DIFFERENT TYPES?

The whole Expression Tree is going to be implemented as a class containing the nodes required to store the data coming in. The wrapped expression tree will know the root and have necessary methods to operate itself. The node/struct class will have 2 node pointers initially pointing to null and later pointing to its children if assigned. Two basic types of nodes will be required:

1. Nodes that store strings- such trees will not require calculation on evaluation
2. Nodes that store integers that requires calculations while evaluation.

Nodes will also have a Boolean attribute storing if it is an operator (a parent to two children) or a leaf (with no children). The node will also store the original index of the data from the post fix expression. It will contain a type specifier which will determine the type of the data stored and assign value if an int or float type. An Evaluate function will apply the operator to its children and change its value if needed. illustration of the node is shown in figure 1.

Node	
Node* leftChild	
Node* rightChild	
string data	
String type	If the data stored is of int type or plain string that needs calculations or characters.
bool isLeaf	Boolean that is true if it is not operator and needs no evaluation.
float value	In case if the Node is of int type, it stores the value
int index	Original index number stored if needed
Node* parentPointer	A pointer pointing to its immediate ancestor.
Evaluate()	A function that will evaluate left child and right child according to parent and store the value or result

DESCRIBE THE ALGORITHM YOU WILL USE TO CREATE AN EXPRESSION TREE GIVEN THE POSTFIX EXPRESSION

```
ExpTreeConstructor(expression)
    Create an empty stack of nodes
    Root=null
    While there are more tokens in expression:
        If token isOperand:
            createNode(data)
            push createdNode to stack
            root=createdNode        //although this should not remain the root
        else if token isOperator:
            CreateNode(stacktop1, stacktop2)
            -pop out stacktops as you get them
            point the 2 children's parents to createdNode
            push the node to the stack
            root=createdNode
```

//Node struct will have two kinds of constructor. One parameter constructor creates a leaf node and two parameter constructor creates a non leaf node.

//Nodes will have datatype function which will determine if its an int type node or just string data

PSUEDOCODE1: CREATING A EXPRESSION TREE FROM POSTFIX

DESCRIBE THE ALGORITHM YOU WILL USE TO EVALUATE THE EXPRESSION

```
EvaluateTree(Node* root)
    root->evaluate();
Every Node will have an evaluate function of such:
Evaluate()
    If (thisNode is leaf):
        return value
    else:
        childOne->evaluate();
        childTwo->evaluate();
        (depending upon data type and the operator type)
            return result from operation of ChildOne and ChildTwo
```

Every Node will have an evaluate function that

- returns itself if it is a leaf
- otherwise evaluates its children and
- operates on them (depending on the data type and operator type we are dealing with)
- and finally returns the value.
- Evaluate is sort of a recursive call but the methods belong to different instances of the same object

PSUEDOCODE2: FOR EVALUATING TREES