WA6

WEEKLY ASSIGNMENT 6
BME 121, FALL 2016

Due Wed Nov 9 at 4:30 pm

ASSIGNMENT

This is a relatively simple assignment to complete but it is important to strive to understand its underlying concepts (objects linked using pointers, recursive traversal). The starting point is a nearly working C++ program in the file "wa6-start.cpp". By working with a C++ program, we hope to ease your transition into BME 122.

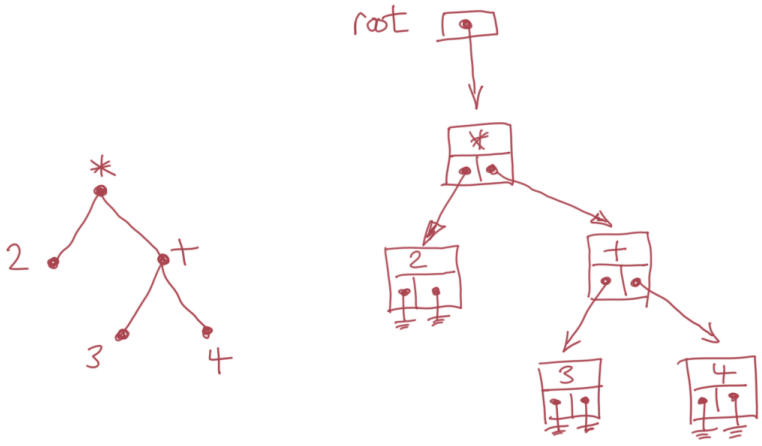
The provided program acts as a simple calculator which can add, subtract, multiply, and divide "double" numbers. The calculator takes its input in what is called RPN (reverse Polish notation) where an operation is specified by listing its two operands followed by the operator. Thus, "a + b" in the usual "infix" notation, becomes "a b +" in RPN. No parentheses are needed in RPN for these operators (a consequence of them each having a fixed number of operands). For example "2 * (3 + 4)" becomes "2 3 4 + *" in RPN.

The way the program is written, you enter one "token" of an RPN expression on each line. Each such token can be a "double" number, one of the four operators ("+", "-", "*", or "/") or the equals sign ("=") to end the expression. Once the "=" is entered, the program redisplays the expression in fully parenthesized infix form and evaluates the answer. The program does no error checking whatsoever so an invalid expression will probably cause it to crash.

In the given starting point, the evaluated answer is always zero. You need to complete the static "evaluate" method in the "Node" class so it returns the correct answer.

NODE CLASS

A binary tree is just a collection of nodes in parent-child relationships. The top node, often called the root, has no parent. Each node is either an interior node (having exactly two children) or a leaf node (having no children). Associated with each node is some piece of data. We use a Node object for each node in a binary tree. The Node class specifies that each object holds a piece of "string" data and two pointers, one to the left child Node and one to the right child Node. For a leaf node, the two pointers hold NULL. In our case, leaf nodes will hold numbers and interior nodes will hold operators. The expression "2 * (3 + 4)" results in the following binary tree. The pointer variable "root" points to the top Node.



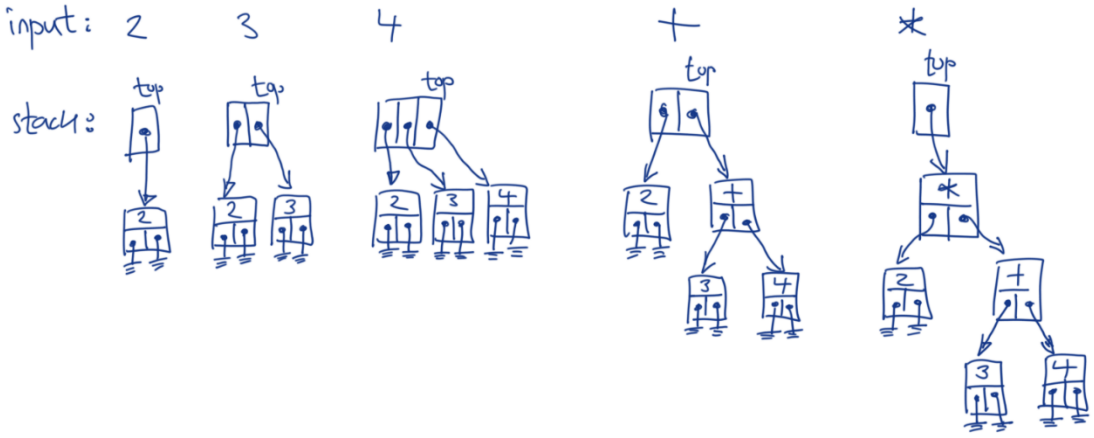
The Node class has two constructors. One takes only a string and creates a leaf node. The other takes a string and two pointers to other Nodes and creates an interior node.

STACK

The "stack< Node* >" type uses a library implementation of a stack where each element is a pointer to a Node. A stack only allows manipulation of its "top" element. The method called "top( )" returns the top element on the stack. The method called "pop( )" discards the top element from the stack. The method called "push( )" receives as a parameter a new element to place on the top of the stack (making it the new "top" element).

Constructing a binary tree from RPN input is quite easy. If the input token is a number, form a new leaf node and push it onto the stack. If the input token is an operator, remove two nodes from the stack, form a new interior node using these two nodes and the

operator, then push the result back onto the stack. The following diagram shows the stack at each iteration of the loop in the "main" method for input of the RPN expression "2 3 4 + *".



RECURSIVE WALK

The method "walk" in the Node class uses recursion to visit each node in the binary tree. The embedded "cout" statements result in the expression being displayed on the console in the standard infix notation with parentheses around every operation.

You need to write the body of the "evaluate" method in the Node class. It should also recursively visit Nodes in the following way. If the node is an operator, return the result of the expression "left operator right" where left is the evaluation of the left child subtree, right is the evaluation of the right child subtree. If the node is a leaf node (i.e., a number) return the conversion of the number (stored as a string) to a "double". In C++, the "stod" method will do this conversion just like "double.Parse" in C#.

The "~Node" method is the "destructor" method for the Node class. In "unmanaged" C++, you have to manually release the memory allocated to objects when you are done with them. The "delete" operator applied to an object accomplishes this. The delete operator results in a call to the destructor method of the class. In our case, this allows us to recursively delete the child nodes. Since C# only runs in a "managed" environment, it has no delete operator and no destructor method is required.

SUBMISSION

Name your C# program file as wa6.cpp. Include the standard doc-comment block. Submit wa6.cpp at the following url.

https://georgefreeman.ca/fileuploader