Isabella Morgan

CSCI 3333-06 Fall

Project Two: Binary Arithmetic Expression Trees

Project two requires us to put together a program that takes in and reads arithmetic expressions from an input file, in this instance, “expressions.txt” and “more_expressions.txt” these files contain mathematical expressions that contain varying, multiple operators and operands. This program must also analyze the syntax and semantics of the expressions, ensuring that the expressions are in fact computable. I had to construct a Binary Expression Tree using stacks, which then outputs the evaluated value along with its prefix, infix, and postfix notations.

This algorithm is based on stack-based expression parsing. The input expressions are read one at a time from the input files. For each expression, the program processes it token by token, either an operand or operator, to build a binary expression tree. Each line from the input file is read into a string, then the string is scanned character by character to identify numbers, parentheses, and operators. I use two stacks, the first is “opStack” which is used to store operators (i.e., *, +, -, (,)), and “treeStack” which is used to store tree node pointers. When an operand is read, a new tree node is created and is then pushed onto the treeStack. Though when an operator is encountered, if it has lower or equal precedence than the operator on top of opStack, nodes are popped from the stacks to form subtrees, and parentheses are used to control precedence. In class treeNode, I use three components which are: value/data that takes in the operator or operand as a string or char, left which is a pointer to the left child of a node, and right which is a pointer to the right child. Once the expression is completely processed, the remaining operator nodes are popped, connecting their corresponding operands. If there are no syntax or semantic errors like mismatched parentheses, division of zero, or missing operands, then the tree is evaluated recursively by traversing in prefix, infix, and postfix orders.

I also did the optional extra credit; it was just a matter of ensuring that calculator could not only handle arithmetic operators but also handle relational and logical operators. While the expressions from “expressions.txt” were evaluated into their numerical value, the expressions from “more_expressions.txt” were evaluated into either a 1 or 0, for true or false.