Raymond Arias

CS-240-02-sp15

Project 3

06/04/15

**Project Specification:**

My project 3 converts infix mathematical expressions to both postfix and prefix mathematical expressions. It does this by calling two static methods that use algortihms to do the conversions. The first static method is convertToPostfix which has a string array as a parameter, the infixExpression, and returns a string array, the postfix expression. The method uses the standard Stack interface provided by Java. The stack used in this method is a local variable, the method also has a local string array named retStr which is the string that will returned, and an integer counter used to increment the elements in retStr. Before the method processes the infix expression passed in, it sends the infix expression to two boolean functions. The first function named isBalanced verifies that the infix expression is a balanced expression, meaning that every opening parenthesis has a closing parenthesis. If the expression is not balanced the method returns false, then the convertToPostfix method throws an exception. The next function is named hasCorrectOperation, which verifies the infix expression has a correct number of operands and operators. If the infix expression has extra operators or operands the method returns false, and convertToPostfix throws an expception. If both boolean methods return true, convertToPostfix begins to run the algorithm described in the project three pdf. Once the algortithm is ran the method returns a copy of retStr with the length of counter. The second static method is convertToPrefix, which also has parameter string array passed in, the infix expression, and returns a string array, the prefix expression. The prefix conversion method runs in the nearly all the same way the postfix conversion. The only thing part of convertToPrefix that is different than convertToPrefix is it has a second local instance of stack to hold the operators and the operands. Beside that everything else is the same to convertToPostfix method, including the two boolean methods to verify if the infix expression has balanced parenthesis and correct number of operands or operators, it returns a string array, and uses the algorithm described in the pdf for project three. Other methods that were implemented for this project, a boolean method named higherPrecedence which returns true if the item being scanned has high precedence than the value at the top of the stack, another boolean method name samePrecedence that returns true if the value being scanned has the same precedence as the value on top of the stack, and a boolean method isOperator which returns true if the item being scanned is a mathematical operator, else returns false, this is mainly used to parse out spaces and other characters. That is how my project three converts infix expression into prefix and postfix mathematical expression.

**Testing Methodology**

**Test Case One:** Verify both methods can convert simple expression to postfix and prefix without any parenthesis and is mathematically correct, for example A+B. This test should not throw any exceptions and return a correct conversion from infix to postfix and prefix.

**Test Case Two:** Verify the both methods can convert infix expression with parenthesis to both postfix and prefix expression, for example (A\*B\*(C-D)). This test case should not throw an exception and return a correct conversion from infix to postfix and prefix.

**Test Case Three:** The test verifies that the method convertToPostfix can detect an unbalanced expression, for example (A\*B((C-D))). This test should cause convertToPostfix to throw an exception that prompt the expression is unbalanced.

**Test Case Four:** The test verifies that the method convertToPostfix can detect if the expression is missing an operand or operator, for example A+BC. This test should cause convertToPostfix to throw an exception.

**Test Case Five:** The test verifies that the method convertToPrefix can detect an unbalanced expression, for example (A\*B((C-D))). This test should cause convertToPrefix to throw an exception that prompts that the expression is unbalanced.

**Test Case Six:** The test verifies that the method convertToPrefix can detect if the expression is missing an operand or operator, for example A+BC. This test should cause convertToPrefix to throw an exception.

**Lesson Learned**

This project taught how using a stack can beneficial when breaking down expressions and other difficult problems. Trying to convert a infix expression to postfix or prefix would have been very difficult without using the stack data structure. This project also taught to try and catch all possible error in my code by using boolean methods or exceptions. This project overall increased my knowledge of the stack data structure and code robustness.