Homework 6

John Justis

12/7/2021

**Introduction:**

For this final homework assignment, we had to combine many of the skills we have learned over the past semester to create an extensive program that takes in expressions and can perform a multitude of actions on those expressions. This program is interactive meaning it takes input from the user and gives an output. The input would be a series of expressions separated be semicolons. These expressions could be arithmetic as well as assignment of variables. Expressions are a series of repeating operators and operands, and each expression had to be validated before it could be solved. For each expression it also had to be determined the postfix and prefix form as well as fully parenthesized. The program also needed to store the information for variable assignment and use that variable assignment to solve other expressions containing those variables.

**Design & Implementation:**

I started the design of this program with the expression and token classes that were created in a previous assignment. There had to be several additional functions added to the expression class and a few changes to the token class. The expression class already takes each character in the given string and tokenizes it into a vector. A postfix function takes the vector of tokens and sorts it into postfix form. The prefix function sorts using the same algorithm as the postfix by first reversing the original vector and then reversing again after the postfix. There is also an added function to parenthesize the vector of tokens. The parenthesize function adds an open and closed parentheses for each operator unless one is already given. I chose to implement all these functions in the expression set program and store them to be called later instead of calling each function when the user gives the action. The set function first calls an additional is valid function to ensure the function is a valid arithmetic function that could be postfixed and solved. The is valid function uses a case break structure and alternates between two states of expect operator and expect operand. The function should cycle back and forth between these until the end of the expression to ensure it is valid. Once the expression is validated the postfix, prefix, and parenthesize functions can be called, and the values stored in additional token vectors stored in the Expression class. When the user inputs the action calling for the postfix or prefix the appropriate vector is displayed.

The program also had to solve each of the given functions if they were valid solvable expressions. The expressions are solved using the postfix of the given expressions, the postfix has order of operations already set and no parenthesis which makes it easier for the computer to solve. The function takes the top two numbers from the vector as well as the top operator, converts to integers and solves them, the resulting answer is returned to the stack. If an assignment expression is inputted, it needs to be stored outside of the expression class and then used later in an expression containing that assignment. I used a linked list that contained data for both the name of the variable and the value assigned to it. When an expression is solved that contained an identifier it is checked through the linked list to see if the identifier matches any of the given assignments and if so, it is replaced in the vector with the value of that identifier. After the expression is solved, it is returned to its original expression without the replacement.

**Test Cases:**

To test my program, I created an additional user action “d” which is used to access the display function given in the original expression class. I added each addition to the program to his display function so it would show the postfix, prefix, etc. Throughout the process of implementing each function I used this to test and see the potential output of a given expression. I tested each new expression extensively using many variations of possible expressions. Once the entire program was completed with each function, I used the given example program and tested various expressions using that program and compared the results to my own to ensure they were the same. I also tested to make sure the dialog for errors or different outputs was correct compared to the original.

**Conclusion:**

This program was the most extensive we have done. It includes all the different things we have learned this semester and was the longest program I have written. The checkpoints were helpful as the full three weeks were needed to complete this program. Completing this program, however, was also very rewarding getting to see all the different elements we have been studying come together in an interactive program. This program is able to solve any expression and use data structures to store variables and use them in the solving of the expressions.

**Test Output:**

**=== expression evaluation program starts ===**

**input:a+b\*c;a\*b+c;( a + b)\*c; (a + (b - c) \* d / c)\*(e -f);**

**action:>**

**prefix of a+b\*c is: + a \* b c**

**prefix of a\*b+c is: + \* a b c**

**prefix of ( a + b)\*c is: \* + a b c**

**prefix of (a + (b - c) \* d / c)\*(e -f) is: \* + a / \* - b c d c - e f**

**action:<**

**postfix of a+b\*c is: a b c \* +**

**postfix of a\*b+c is: a b \* c +**

**postfix of ( a + b)\*c is: a b + c \***

**postfix of (a + (b - c) \* d / c)\*(e -f) is: a b c - d \* c / + e f - \***

**action:f**

**fully parenthesizing a+b\*c results: (a+(b\*c)**

**fully parenthesizing a\*b+c results: ((a\*b)+c)**

**fully parenthesizing ( a + b)\*c results: ((a+b)\*c)**

**fully parenthesizing (a + (b - c) \* d / c)\*(e -f) results: ((a+(((b-c)\*d)/c))\*(e-f))**

**action:=**

**a+b\*c = no result, some variable has undefined value**

**a\*b+c = no result, some variable has undefined value**

**(a+b)\*c = no result, some variable has undefined value**

**(a+(b-c)\*d/c)\*(e-f) = no result, some variable has undefined value**

**action:c**

**input:a+b-c\*d/e;a=1;b=2;c=3;d=4;e=5;**

**action:=**

**a+b\*c = 7**

**a\*b+c = 5**

**(a+b)\*c = 9**

**(a+(b-c)\*d/c)\*(e-f) = no result, some variable has undefined value**

**a+b-c\*d/e = 1**

**cannot evaluate a=1 which is not an arithmetic expression, but assignment.**

**cannot evaluate b=2 which is not an arithmetic expression, but assignment.**

**cannot evaluate c=3 which is not an arithmetic expression, but assignment.**

**cannot evaluate d=4 which is not an arithmetic expression, but assignment.**

**cannot evaluate e=5 which is not an arithmetic expression, but assignment.**

**action:>**

**prefix of a+b\*c is: + a \* b c**

**prefix of a\*b+c is: + \* a b c**

**prefix of (a+b)\*c is: \* + a b c**

**prefix of (a + (b - c) \* d / c)\*(e -f) is: \* + a / \* - b c d c - e f**

**prefix of a+b-c\*d/e is: - + a b / \* c d e**

**no prefix of a=1 which is not an arithmetic expression, but assignment.**

**no prefix of b=2 which is not an arithmetic expression, but assignment.**

**no prefix of c=3 which is not an arithmetic expression, but assignment.**

**no prefix of d=4 which is not an arithmetic expression, but assignment.**

**no prefix of e=5 which is not an arithmetic expression, but assignment.**

**action:<**

**postfix of a+b\*c is: a b c \* +**

**postfix of a\*b+c is: a b \* c +**

**postfix of (a+b)\*c is: a b + c \***

**postfix of (a + (b - c) \* d / c)\*(e -f) is: a b c - d \* c / + e f - \***

**postfix of a+b-c\*d/e is: a b + c d \* e / -**

**no postfix of a=1 which is not an arithmetic expression, but assignment.**

**no postfix of b=2 which is not an arithmetic expression, but assignment.**

**no postfix of c=3 which is not an arithmetic expression, but assignment.**

**no postfix of d=4 which is not an arithmetic expression, but assignment.**

**no postfix of e=5 which is not an arithmetic expression, but assignment.**

**action:f**

**fully parenthesizing a+b\*c results: (a+(b\*c)**

**fully parenthesizing a\*b+c results: ((a\*b)+c)**

**fully parenthesizing (a+b)\*c results: ((a+b)\*c)**

**fully parenthesizing (a + (b - c) \* d / c)\*(e -f) results: ((a+(((b-c)\*d)/c))\*(e-f))**

**fully parenthesizing a+b-c\*d/e results: ((a+b)-((c\*d)/e)**

**no fully parenthesizing of a=1 which is not an arithmetic expression, but assignment**

**no fully parenthesizing of b=2 which is not an arithmetic expression, but assignment**

**no fully parenthesizing of c=3 which is not an arithmetic expression, but assignment**

**no fully parenthesizing of d=4 which is not an arithmetic expression, but assignment**

**no fully parenthesizing of e=5 which is not an arithmetic expression, but assignment**

**action:q**

**=== expression evaluation program ends ===**