TEAM 1 PROJECT 2

Infix Expression Parser

Dawson Ploudre

Luke Janis

APRIL 10TH, 2023

TABLE OF CONTENTS

1. System Design 3

2. UML Diagram 4

3. Test Cases 5

4. Team Member Contribution 8

5. Possible Improvements 8

1. System Design

The program consists of three files, infixParserSource.cpp (main file), Expression.h, and Expression.cpp (object class).   
  
infixParserSource.cpp - Generates a new Expression object for each line in the input file. Each line represents an infix expression. Calls Expression.print() for each line.

Expression.h and .cpp - Defines an Expression object which stores the infix expression as a string. Houses all functions within the program, explained below.

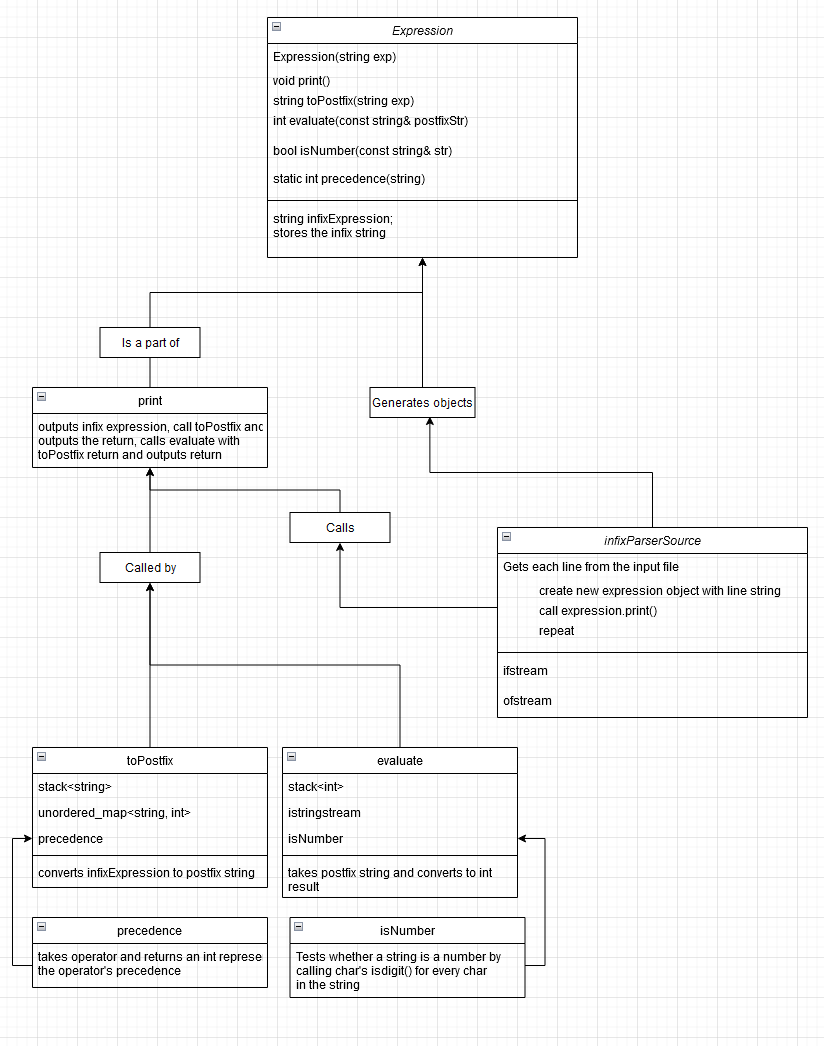
* print outputs infixExpression to console. Calls toPostfix and outputs its return. Calls evaluate with toPostfix’s return and outputs its return.
* toPostfix uses a stack<string> and an unordered\_map<string, int> to convert the infixExpression data field of the object to a postfix string and returns it.
* precedence is used by toPostfix to determine the precedence of operators
* evaluate uses a stack<int> and an input string stream of the postfix string to convert and return its int result.
* isNumber is used by evaluate to determine whether a token is an operand or operator

In Summary:  
 While new line in input file, get the line  
 Create new expression object with line string

Call expression.print()

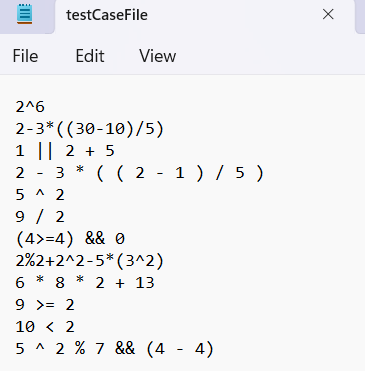
Expression.print() outputs infix string  
 Expression.print() calls toPostfix and outputs that string  
 Expression.print() calls evaluate with toPostfix result and outputs int  
 result  
 End while

1. UML Diagram



1. Test Cases

Here, we have our test case file with all of our inputs. First, let’s look at all of the inputs in the test case file. We did a mix of all operators to show that the program can process each operator effectively. We also added in different white spacing to show that the program can handle any expression despite different spacing.



Now, what do we expect the program to output?

For the infix expression, **2^6** we expect an evaluation of **64**.

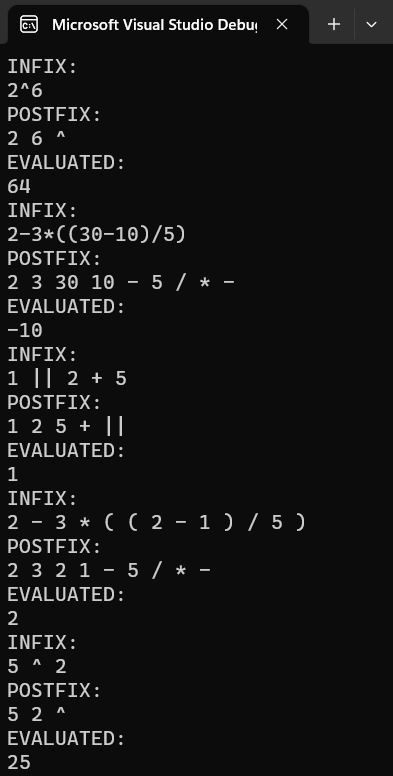
For the infix expression, **2-3\*((30-10)/5)** we expect an evaluation of (**-10**).

For the infix expression, **1 || 2 + 5** we expect a boolean evaluation of **1** as it is *true*.

For the infix expression, **2-3\*((2-1)/5)** we expect an evaluation of **2**,

while **(3\*((2-1)/5)) = 0.6 → 0** as we convert to the whole integer 0, therefore **2 - 0 = 2**.

For the infix expression, **5^2** we expect an evaluation of **25**.



As we can see, our program processed the first five of our test cases correctly and output the right solution. Now, back to the rest of the test cases.

For the infix expression, **9/2** we expect the integer value of **4**, as we don’t want 4.5 evaluated

For the infix expression, **(4>=4) && 0** we expect a boolean evaluation of **0** as it is *false*.

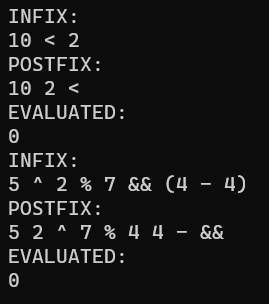
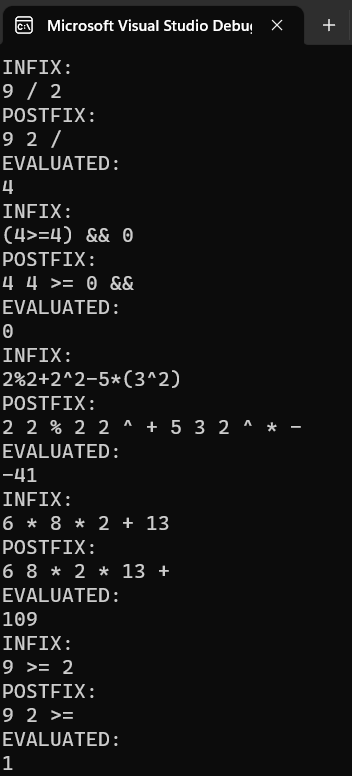
For the infix expression, **2%2 + 2^2 - 5\*(3^2)** we expect an evaluation of (**-41**).

For the infix expression, **6 \* 8 \* 2 + 13** we expect an evaluation of **109**.

For the infix expression, **9 >= 2** we expect a boolean evaluation of **1** as it is *true*.

For the infix expression, **10 < 2** we expect a boolean evaluation of **0** as it is *false*.

For the infix expression, **5 ^ 2 % 7 && (4 - 4),** expect a boolean evaluation of **0** as it is *false*.



As we can see, once again our program output all of the expected outputs. From all of these test cases we can see that our program effectively processes an infix expression into a postfix expression. It then evaluates the postfix expression effectively. It does all of this while ignoring white space in the input file, and assigning correct precedence to all of the input operators.

1. Team Member Contribution

Dawson Ploudre: Created the precedence function which assigned correct precedence values to all operators. Also created the toPostFix function which processed an infix expression into a postfix expression. Wrote the test cases and possible improvements for the project report.

Luke Janis: Hosted Github repository and created basic structure of the program  
 with Expression objects. Implemented infixParserSource to read input file and

create Expression objects. Implemented evaluate function (uses isNumber).

Wrote System Design and UML diagram in project report.

1. Possible Improvements

One possible improvement would be in the toPostfix function. In this function we start by removing all of the whitespace from the input string by using the erase function. This function runs in T(n) time as it iterates over the string to remove all white space. We then process the string into postfix form which takes T(n) time, effectively making the function run in T(2n). However, if we were to simply ignore the whitespace while processing the string we could get this down to T(n), saving time.

Another possible improvement would be in the evaluate function. In this code we use repeated if/else statements to evaluate boolean operators. Instead of this we could’ve used the ternary operate ‘?’. This would’ve condensed the code and improved the readability.

All in all, our code could use some slight improvements. However, the program as is is still effective and efficient.