

CPT-287
Team 2
Infix Expression Parser

| | |
|-----------------|---|
| 5 \$ 4 | Operator "\$" not recognized. Returned: 5 |
| 1+2*3 | 7 |
| 2+2^2*3 | 14 |
| 1==2 | 0 |
| 1+3>2 | 1 |
| (4>=4)&&0 | 0 |
| (1+2)*3 | 9 |
| 2%2+2^2-5*(3^2) | -41 |
| 2^3 | 8 |
| 6*2 | 12 |
| 6-2 | 4 |
| 6>5 | 1 |
| 6!=5 | 1 |
| 6>5&&4>5 | 0 |
| 1 0 | 1 |
| (3+4) 1 | 1 |
| (2>3)-2 | -2 |
| 5^2%7&&(4-4) | 0 |
| 3/(6*5-30) | Error: division by zero. Returned: 0 |

Chris Anderson

Luke Hunt

Steven Shackleford

Table of Contents

| | |
|---------------------------------|-------------|
| Table of Contents | 2 |
| System Design | 3 |
| Infix Expression parsing | 3 |
| Postfix conversion | 3 |
| Postfix Expression Evaluation | 3 |
| Infix Parser Methods | 4 |
| Precedence | 4 |
| addWhiteSpace | 4 |
| isInt | 4 |
| infixToPostfix | 4 |
| evaluatePostfix | 5 |
| Evaluate | 5 |
| UML Diagram | 6 |
| Test Cases | 7-11 |
| Team member contribution | 12 |
| Future Improvements | 13 |

System Design

In order to evaluate infix expressions, we convert the infix expressions read in from the input file into postfix expressions using stacks, and then evaluate the postfix expressions to print the values of the expressions to the standard console.

Infix Expression parsing

Our program starts by reading the input file line by line and then passing the line that was just read to the {infixToPostfix} method inside the [InfixParseMethods] class. This method is the one that takes the input string and converts it to a postfix expression using stacks, and then is properly evaluated.

Postfix conversion

The {infixToPostfix} method takes in a string as it's variable and then adds whitespace onto the end of the string that acts as a buffer. It then properly adds the operators and operands to the appropriate string/stack and then passes the new string to the {evaluatePostfix} method.

Postfix Expression Evaluation

In order to evaluate the postfix expression with the stack of operands and string of operators, we first pop the first two operands off the stack (right and then left), and then we take the first token from the string of operators to evaluate the operands that were popped from the stack.

Infix Parser Methods

Precedence

The precedence method is used to determine the correct order of operations based on the current operator. A series of if statements is used to compare a list of known cases. If a match is found, the integer value representing their precedence is returned. Otherwise an error is returned warning the user of an unknown operator.

addWhiteSpace

The addWhiteSpace method reads through the infix formatted string one character at a time and adds spaces between operators and operands to ensure consistent formatting. Since the program allows users to enter formulas with or without spaces, this method is a necessary step for the methods that follow.

isInt

The isInt method allows a properly error handled boolean operation before numerical methods are used that may cause the program to crash. The method uses a try-catch system to attempt to convert a string to an integer. If the process is successful, true is returned. If this causes a compiler error, false is returned.

infixToPostfix

After using the addWhiteSpace method to clean up input and using spaces as delimiters, this method parses the input string token by token to create a postfix equivalent formula. The method uses a stack to store operators during the algorithm. Operators are added to the output string based on

either explicit parentheses or order of operations as determined by the precedence method. Finally this method returns a string containing the new postfix expression with spaces between all tokens.

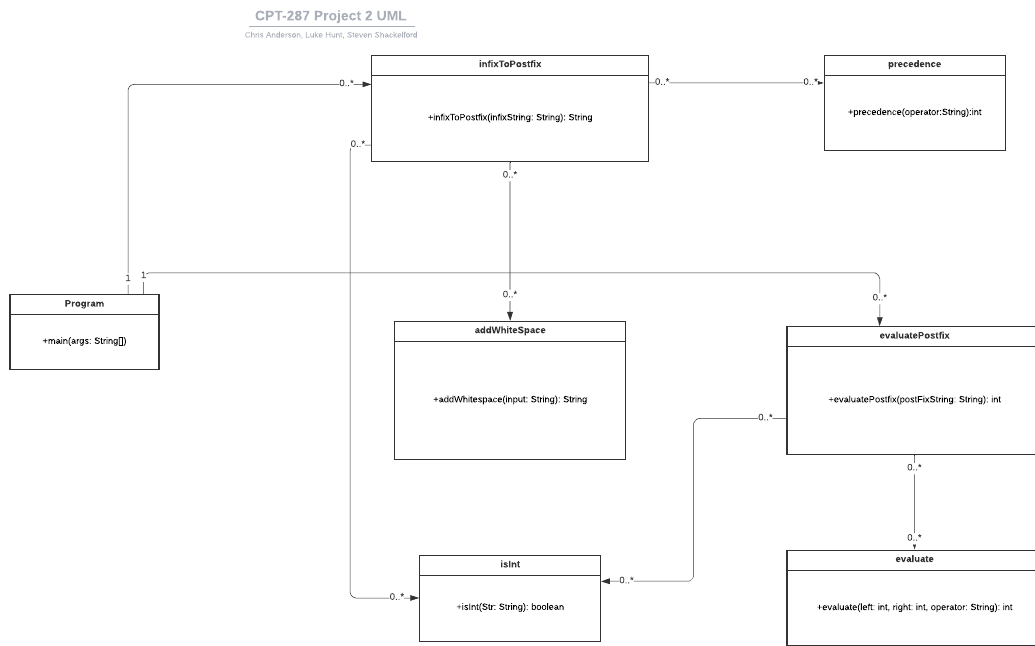
evaluatePostfix

This method takes in the converted postfix expression string and creates a stack to store the operands, and when the scanner parsing the string comes across an operator in the string, it pops the right and left operand out of the stack, calls the {evaluate} method with the left and right operand, and the operator [token], and then pushes the result into the stack. This method returns the final result.

Evaluate

This method takes in the left operand, right operand, and operator to determine which logical, comparison, or arithmetic operation should be performed. A series of if statements is used to compare the operator string to the known cases. The resulting value is returned if a match is found. Otherwise, an error is returned warning the user of an unknown operator


UML Diagram



Test Case 1:

Testing program for “divide by zero” error.

For this test we will create an input file with an expression of: “8 / (9 * 9 - 81)”. The program should not throw an in-built IDE error of any kind, instead line 142 of our “evaluate” method should catch this and print out to the console, “Error: division by zero. Returned: 0”.

 Project2testcase1.txt - Notepad

File Edit Format View Help

8 / (9 * 9 - 81)

```
137● public static int evaluate(int left, int right, String operator){
138     // Power
139     if(operator.equals("^")) {return (int) Math.pow(left, right);}
140     //Arithmetic
141     if(operator.equals("*")) {return left * right;}
142     if(operator.equals("/") && right == 0) {System.out.print("Error: division by zero. Returned: "); return 0;}
143     if(operator.equals("/")) {return left / right;}
144     if(operator.equals("%")) {return left % right;}
145     if(operator.equals("+")) {return left + right;}
146     if(operator.equals("-")) {return left - right;}
147     // Comparisons
148     if(operator.equals(">")) {
149         if(left > right) {return 1;}
150         else {return 0;}
151     }
152     if(operator.equals("<")) {
153         if(left < right) {return 1;}
154         else {return 0;}
155     }
156 }
```

As we run the program, we can see that it did not throw any IDE errors, but did output our error message to the console.

```
1 package project_2;
2
3 import java.io.FileInputStream;
4 import java.io.IOException;
5 import java.util.Scanner;
6
7 public class Program {
8
9
10 public static void main(String[] args) throws IOException {
11     FileInputStream inputFile = new FileInputStream("Project2testcase1.txt");
12     Scanner scanner = new Scanner(inputFile); // Input file scanner
13
14     while(scanner.hasNext()) {
15         String input = scanner.nextLine();
16         String postfix = InfixParserMethods.infixToPostfix(input); // convert string to postfix
17         System.out.println(InfixParserMethods.evaluatePostfix(postfix)); // evaluate converted string
18     }
19
20     scanner.close(); // close scanner
21     inputFile.close(); // close input file
22 }
23
24
25 }
26
```

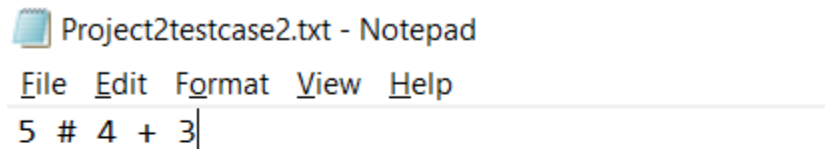
Console x

<terminated> Program [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (Nov 1, 2021, 6:18:41 PM – 6:18:42 PM)
Error: division by zero. Returned: 0

Test Case 2:

Test program to catch an unrecognized symbol.

In this second test we will create an input file that that has the expression: “5 # 4 + 3”. Like the previous test our program, specifically our “infixToPostfix” method, should catch this and and output a message to the console that says, “Operator “#” not recognized. Returned 5”.




```
InfixParserMethods.java | Program.java | Project2testcase2.txt
75
76 public static String infixToPostfix(String infixString) {
77     infixString = addWhiteSpace(infixString); // clean up input
78     Stack<String> stack = new Stack<>(); // initialize new stack
79     Scanner scanner = new Scanner(infixString); // create scanner to parse string
80     String postfixString = "";
81
82     while(scanner.hasNext()) {
83         String token = scanner.next();
84         // if token is a number
85         if(isInt(token)) {
86             postfixString += token + " "; // add to postfix string
87             continue; // move on to next item
88         } else if the token is an opening parentheses
89         } else if (token.equals("(")) {
90             stack.push(token); // push token onto stack
91             continue; // move on to next item
92         } else if the token is an operator
93         } else if (precedence(token) != -1) {
94             // while the stack is not empty AND top of stack is not opening parentheses AND precedence of current token <= top of stack
95             while(!stack.isEmpty() && !stack.peek().equals("(") && precedence(token) <= precedence(stack.peek())) {
96                 postfixString += stack.pop() + " ";
97             } // end while
98             stack.push(token); // push the current token onto the stack
99         } else if (token.equals(" ")) { // token is closing parentheses
100             while(!stack.isEmpty() && !stack.peek().equals("(")) { // while top of stack is not opening parentheses
101                 postfixString += stack.pop() + " ";
102             } // end while
103             stack.pop(); // pop opening parentheses off stack
104         } else {
105             System.out.printf("Operator \"%s\" not recognized. Returned: ", token);
106             break;
107         } // end else
108     } // end while
109     // pop remaining items in stack
110     while(!stack.isEmpty()) {postfixString += stack.pop() + " "};
111 }
```

As we run the program, we can see that it handles it as expected and outputs our error message to the console.

```
InfixParserMethods.java | Program.java | Project2testcase2.txt
75
76 public static String infixToPostfix(String infixString) {
77     infixString = addWhiteSpace(infixString); // clean up input
78     Stack<String> stack = new Stack<>(); // initialize new stack
79     Scanner scanner = new Scanner(infixString); // create scanner to parse string
80     String postfixString = "";
81
82     while(scanner.hasNext()) {
83         String token = scanner.next();
84         // if token is a number
85         if(isInt(token)) {
86             postfixString += token + " "; // add to postfix string
87             continue; // move on to next item
88         } else if the token is an opening parentheses
89         } else if (token.equals("(")) {
90             stack.push(token); // push token onto stack
91             continue; // move on to next item
92         } else if the token is an operator
93         } else if (precedence(token) != -1) {
94             // while the stack is not empty AND top of stack is not opening parentheses AND precedence of current token <= top of stack
95             while(!stack.isEmpty() && !stack.peek().equals("(") && precedence(token) <= precedence(stack.peek())) {
96                 postfixString += stack.pop() + " ";
97             } // end while
98             stack.push(token); // push the current token onto the stack
99         } else if (token.equals(" ")) { // token is closing parentheses
100             while(!stack.isEmpty() && !stack.peek().equals("(")) { // while top of stack is not opening parentheses
101                 postfixString += stack.pop() + " ";
102             } // end while
103             stack.pop(); // pop opening parentheses off stack
104         } else {
105             System.out.printf("Operator \"%s\" not recognized. Returned: ", token);
106             break;
107         } // end else
108     } // end while
109     // pop remaining items in stack
110     while(!stack.isEmpty()) {postfixString += stack.pop() + " "};
111 }
```

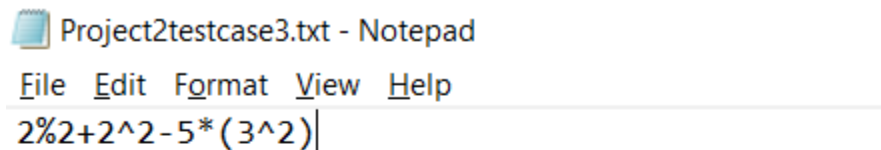
```
Console | Problems | Debug Shell
<terminated> Program [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (Nov 1, 2021, 7:05:52 PM - 7:05:52 PM)
Operator "#" not recognized. Returned: 5
```

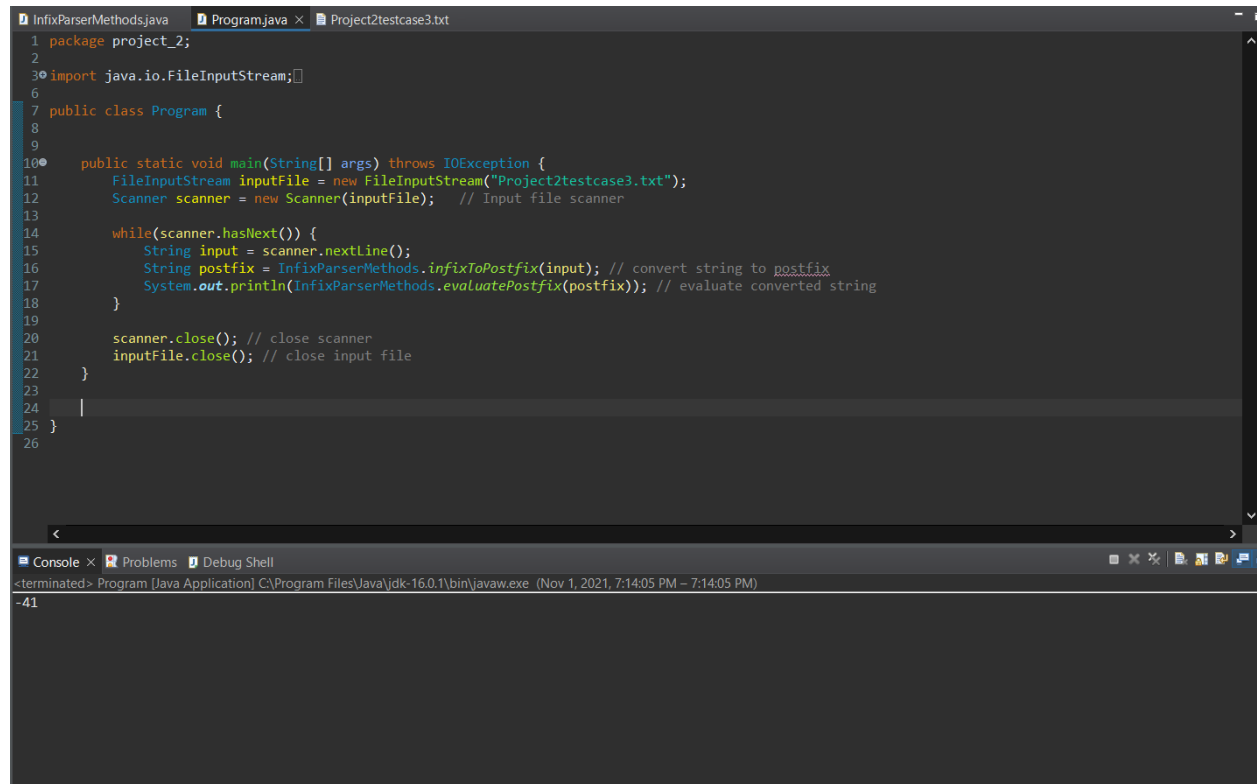
Test Case 3:

Test program to parse the expression and then evaluate.

In our third and final test we will create an input file that has an expression that can be parsed and evaluated, as it is an example provided in the project. Our input file will contain the following expression:

"2%2+2^2-5*(3^2)". The program should parse it and evaluate to -41.





```
1 package project_2;
2
3 import java.io.FileInputStream;
4
5
6
7 public class Program {
8
9
10     public static void main(String[] args) throws IOException {
11         FileInputStream inputFile = new FileInputStream("Project2testcase3.txt");
12         Scanner scanner = new Scanner(inputFile); // Input file scanner
13
14         while(scanner.hasNext()) {
15             String input = scanner.nextLine();
16             String postfix = InfixParserMethods.infixToPostfix(input); // convert string to postfix
17             System.out.println(InfixParserMethods.evaluatePostfix(postfix)); // evaluate converted string
18         }
19
20         scanner.close(); // close scanner
21         inputFile.close(); // close input file
22     }
23
24 }
25
26
```

Console × Problems × Debug Shell

<terminated> Program [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (Nov 1, 2021, 7:14:05 PM – 7:14:05 PM)

-41

As we run the program, it correctly parses and returns the evaluation of -41.

Team member contribution

All members: Wrote and discussed code and wrote it all on one machine to upload to github all at once.

Steven: System Design Document outline, System design block, github readme

Chris: UML chart, Test cases, work on system design document

Luke: collected discussed code and uploaded to github, work on system design document

Future Improvements

Possibly improve to be able to evaluate an infix expression without converting to postfix.

Fix the return value on incorrect operators / divide-by-zero error.