Assignment 2: Solving Expressions in Postfix Notation using Stacks

Cathal Lawlor – 21325456

Abstract task of assignment: Program takes numerical infix from user, converting it to postfix expression, printing result. Using stacks and basic maths rules of operation we convert and calculate using various methods.

Personal notes:

- Infix expressions is the normal presentation for math equations (how humans normally read equations), postfix is where operators (+, - etc) follow their operands (the numbers / variables)
- Only numerical expressions - single digits 0-9 and +, -, *, /, ^, (,)
- Minimum input of 3, maximum of 20
- Before algorithm, check if input is invalid, prompt user to re-enter if needs be
- Must use provided ArrayStack - provided
- Precedence for maths is - ^, * or /, + or –
- Utility method for returning value of operator, based on precedence, enabling comparing
- In order to carry out the mathematical operations, you will need to ensure that the operands are casted to an appropriate number type. ArrayStack works with Objects.
- Even though we are using single digit integers for the operands, result can be a decimal number, e.g. 3 * 4 / 5 = 2.4 o If we only cast using integers, then the result will be incorrectly given as 2. Return in decimal form?

Planning / Analysis / Design Notes:

There will be three main pieces to this, checking is the infix expression valid, converting expression to postfix, calculating using postfix expression.

My approach for error checking is:

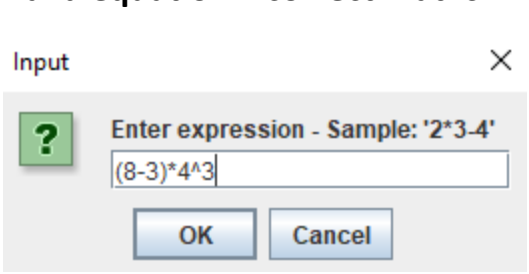
- to check the length of the string using string.Length > size
- if char is a digit, check following char if it is a digit, if so, throw an error
- otherwise we check if char is alphabetic (if so, return error) & if char is not in a specified list of special chars (*,/,(,),+,-,^) return error#

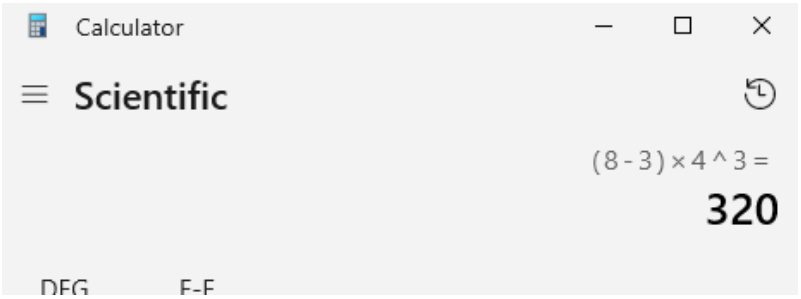
Converting to postfix, I use the rules specified in assignment document. I employ a precedence checker, using a case statement. This is used to employ the rules. This returns postfix as a string.

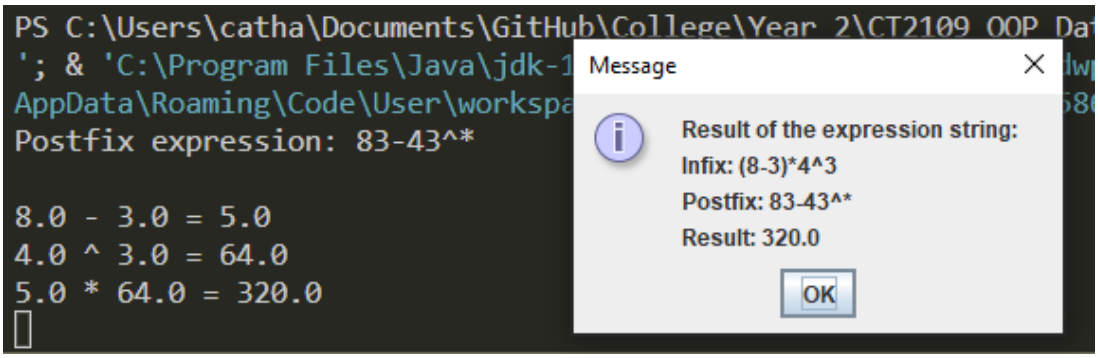
Calculating using postfix will be taking in the string, breaking it into chars, where operands are pushed to the stack, casted as double NUMBER objects. When an operator is encounter, top two operands are popped from stack, operation completed, and push the result to the stack. At the end of this with no more operands or operators, we return the result.

Testing:

Valid equation – correct maths

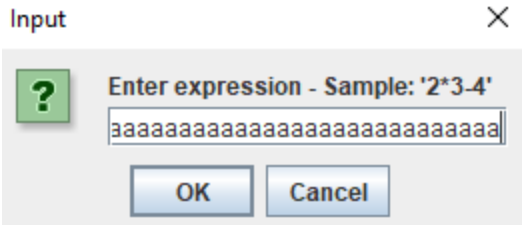




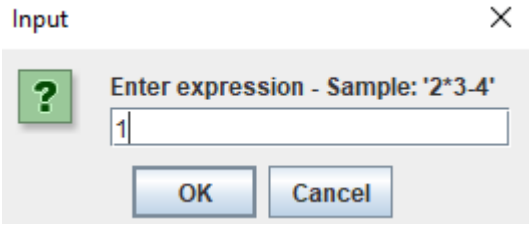


Invalid strings:

Invalid characters and over 20 characters

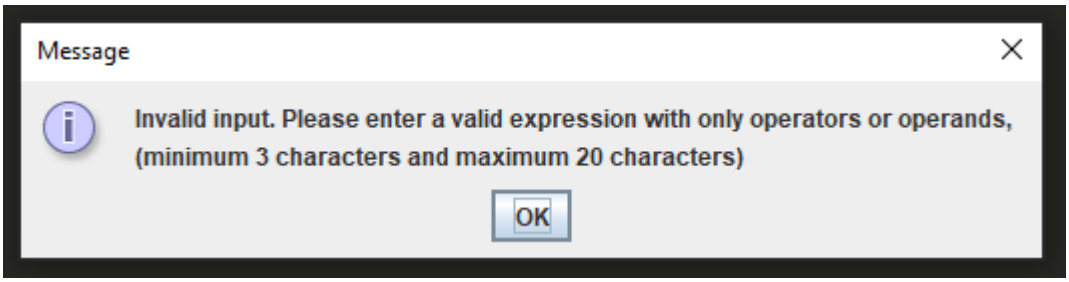


Under 3 characters



Output:

Correct error, try again



Code:

postfixGenerator.java

```
import javax.swing.JOptionPane;

public class postfixGenerator {


    public static void main(String[] args) {
        postfixGenerator runCalculator = new postfixGenerator();
        runCalculator.postFixCalculator(); //running my program
    }

    char[] specialCharacters = {'^', '/', '*', '+', '-', '(',')'}; //permitted characters
outside of digits

    public void postFixCalculator() {
        expressionToPostfixAlgo algorithm = new expressionToPostfixAlgo(); //instanciating
a new prefix to postfix algorithm
        postfixStringCalculator stringCalculated = new postfixStringCalculator();
//instanciating calculator for evaluating postfix notation expressions

        String inputInfixExpression = JOptionPane.showInputDialog("Enter expression -
Sample: '2*3-4'", null);//taking a new input from user

        char inputChar[] = inputInfixExpression.toCharArray(); //converting our string
into chars in an array

        while (!errorChecker(inputInfixExpression, inputChar)) { //while the string isn't
passing conditions - alert user and input new string

            JOptionPane.showMessageDialog(null, "Invalid input. Please enter a valid
expression with only operators or operands, \n(minimum 3 characters and maximum 20
characters)");
            inputInfixExpression = JOptionPane.showInputDialog("Enter expression - Sample:
'2*3-4'", null);
            inputChar = inputInfixExpression.toCharArray();
        }


        String postfixOutput = algorithm.stackManipulator(inputChar); //assigning postfix
String to the output of our postfix calculator
        System.out.println("Postfix expression: " + postfixOutput + "\n");




        double result = stringCalculated.postfixResult(postfixOutput);//assigning
calculations from the postfix expression
        JOptionPane.showMessageDialog(null, "Result of the expression string: \n" +
"Infix: " +  inputInfixExpression
```

```
                                                        + "\nPostfix: " +
postfixOutput + "\nResult: " + result);


    }

    public boolean errorChecker(String inputInfixExpression, char[] inputChar) { //passed
in string, checking if it's valid
        if (inputInfixExpression.length() > 20 || inputInfixExpression.length() < 3) {
//length checking for compliance
            return false;
        }

        for(int i = 0; i < inputInfixExpression.length() - 1; i++) { //going through all
the whole input string / char array

            if(Character.isDigit(inputChar[i])) { //if the char is a digit
                //we will check for a following digit, if we are not at end of an array
                if(  i != inputInfixExpression.length() &&
                    (Character.isDigit(inputChar[i]) && Character.isDigit(inputChar[i+1])
) ){
                    return false; //if there are two digits beside each other, return
false e.g. 93 is above 0-9
                }
            }
            else if(Character.isAlphabetic(inputChar[i])) { //if character is alphabetic,
return false
                return false;
            }
            else if (!specialcharChecker(inputChar[i])) { //if the character is not in the
special characters array, return false
                return false;
            }
        }
        return true; //return true if no invalid characters incurred
    }

    public boolean specialcharChecker(char charToCheck) {
        for (char elem : specialCharacters) { //for all the elements in specialCharacters,
we check if our char is there, returning true if it is
            if (charToCheck == elem) {
                return true;
            }
        }
        return false; //if not there, return false
    }

}
```

expressionToPostfixAlgo.java

```
public class expressionToPostfixAlgo {

    public String stackManipulator(char[] infixChars) {
        String postfixOutput = ""; //output string
        ArrayStack postfixStack = new ArrayStack(); //instanciating a stack to use
        char currentChar, topOfStack = '0';

        for(int i = 0; i < infixChars.length; i++) { //through the whole infixChars array
            currentChar = infixChars[i]; //assign the current char to a variable

            switch (currentChar) {
                case '+':
                case '-':
                case '*':
                case '/':
                case '^': //if operator found
                    if(!postfixStack.isEmpty()){ //if statment to stop us accessing array
if its empty
                        topOfStack = (char)postfixStack.top(); //top of stack char stored
                    }

                    //while the precedence of the current operator is the same or less of
the operand currently in the stack,
                    //we output everything to the output string, after we push current
operator
                    while(precedenceCalc((char)currentChar) <= precedenceCalc(topOfStack)
&& !postfixStack.isEmpty()) {
                        postfixOutput += postfixStack.pop();
                    }
                    postfixStack.push(currentChar);
                    break;

                case '(': //if (, we push
                    postfixStack.push(currentChar);
                    break;


                case ')': //if )
                    //while stack isn't empty, and we haven't encountered (, we print it
all to output & pop
                    while(!(postfixStack.isEmpty() || (char)postfixStack.top() == '(' ) )
{
                        postfixOutput += postfixStack.pop();
                    } //once we hit (, we stop outputting to string, and discard the (
                    postfixStack.pop(); //discarding the '(' from the stack
                break;

                default: //we have done sanatation on string input, we now assume all
remaining characters is a singular digit
                    postfixOutput += currentChar; //putting it to output
                    break;
```

```
            }
        }

        while(!postfixStack.isEmpty()) { //once the string is finished, we pop the rest of
stack to output
            postfixOutput += postfixStack.pop();

        }

        return postfixOutput; //return the output
    }




    public int precedenceCalc(char operator) {
        switch(operator) { //switch based on the operator passed in
            case '^': //Power of (^) returns highest precedence
                return 3;


            case '*': //Multiplication or division (* or /) returns next highest
precedence
            case '/':
                return 2;


            case '+': //Addition or subtraction (+ or -) returns lowest precedence
            case '-':
                return 1;

            default: //else we return a lower precednce for digits etc.
               return -1;
        }
    }
}
```

postfixStringCalculator.java

```
public class postfixStringCalculator {

    public double postfixResult(String inputString){ //input string of postfix expression
        char chr = '.';
        double result = 0.0;
        ArrayStack operandStack = new ArrayStack();

        for(int i = 0; i < inputString.length(); i++) { //for postfix input string length
            chr = inputString.charAt(i); //assign current char to chr variable

            if(Character.isDigit(chr)) { //if a digit, we push it to the stack
                operandStack.push((double)Character.getNumericValue(chr)); //pushed as
Number object as a double
            }

            else {  //otherwise, as an operator
                double temp1 = (double)operandStack.pop(); //assign our two temporary
variables as the top two numbers off the stack,
                double temp2 = (double)operandStack.pop(); //to be used in calculations

                switch(chr) { //depending on the operator we do the following maths
operation
                    case '^': //power of
                        result = Math.pow(temp2, temp1);
                        break;
                    case '/': //division
                        result = temp2 / temp1;
                        break;
                    case '*': //multiplication
                        result = temp2 * temp1;
                        break;
                    case '+': //addition
                        result = temp2 + temp1;
                        break;
                    case '-': //subtraction
                        result = temp2 - temp1;
                        break;
                }
                System.out.println(temp2 + " " + chr + " " + temp1 + " = " + result);
//prinitng results to the console

                operandStack.push((double)result); //push the result to the stack
            }
        }
        result = (Double)operandStack.pop(); //only item on stack should be the result,
which we pop

        return result; //return result
    }

}
```