CT2109 – Assignment 1 – Luke Gibbons - 15553883:

Solving Expressions in Postfix Notation Using Stacks

Problem Statement:

The problem requires taking in an infix expression (operator, operand, operator) and converting it into postfix notation (operator, operator, operand). This will then be calculated and the result returned. Both the conversion and the calculation will require the use of a stack class along with a set of instructions (i.e. what/when to push/pop). These instructions will be worked out in the design stage.

Analysis and Design:

My initial thought when thinking about solving the problem is to create a converter class. This can prompt the user to input an infix expression in its constructor method.

The next thing I will need to do is create an algorithm to verify if the input is valid. I believe a separate method would be the best bet for this. I will need to first convert the string into a char array. It will require the verification of 2 things: 1. the input is between 3 and 20 characters long and 2. It only uses single digits and valid operands.

The length verification will be the easiest part.

```
If(input >= 3 and input <=20){

        Verified= true

}

Else{

        Verified = false

        Print 'Must be between 3 and 20 characters'
```

}

The operator and single digit verification will take more work. I originally thought of creating a char array containing all the valid chars. I would then scan through the input array and return true if it matches with any of the chars in the valid array. However, having done more research I believe a better solution would be to use the built in function character.isDigit to verify if it is 0-9 and then a switch statement to verify the operator.

```
For(Items in char Array){
        If(item is a digit AND item+1 is a digit){
                Return false
}
        Else If(Item is a valid Operator and item+1 is NOT a digit){
                Return false
}
        Else If(item is NOT  a digit AND  item is NOT a valid Operator){
                Return false
}
        Else
                Return true
}
```

Once the input has been verified I can proceed to converting it to postfix. I will create another method for this and use the following instructions to properly convert the input using a stack.

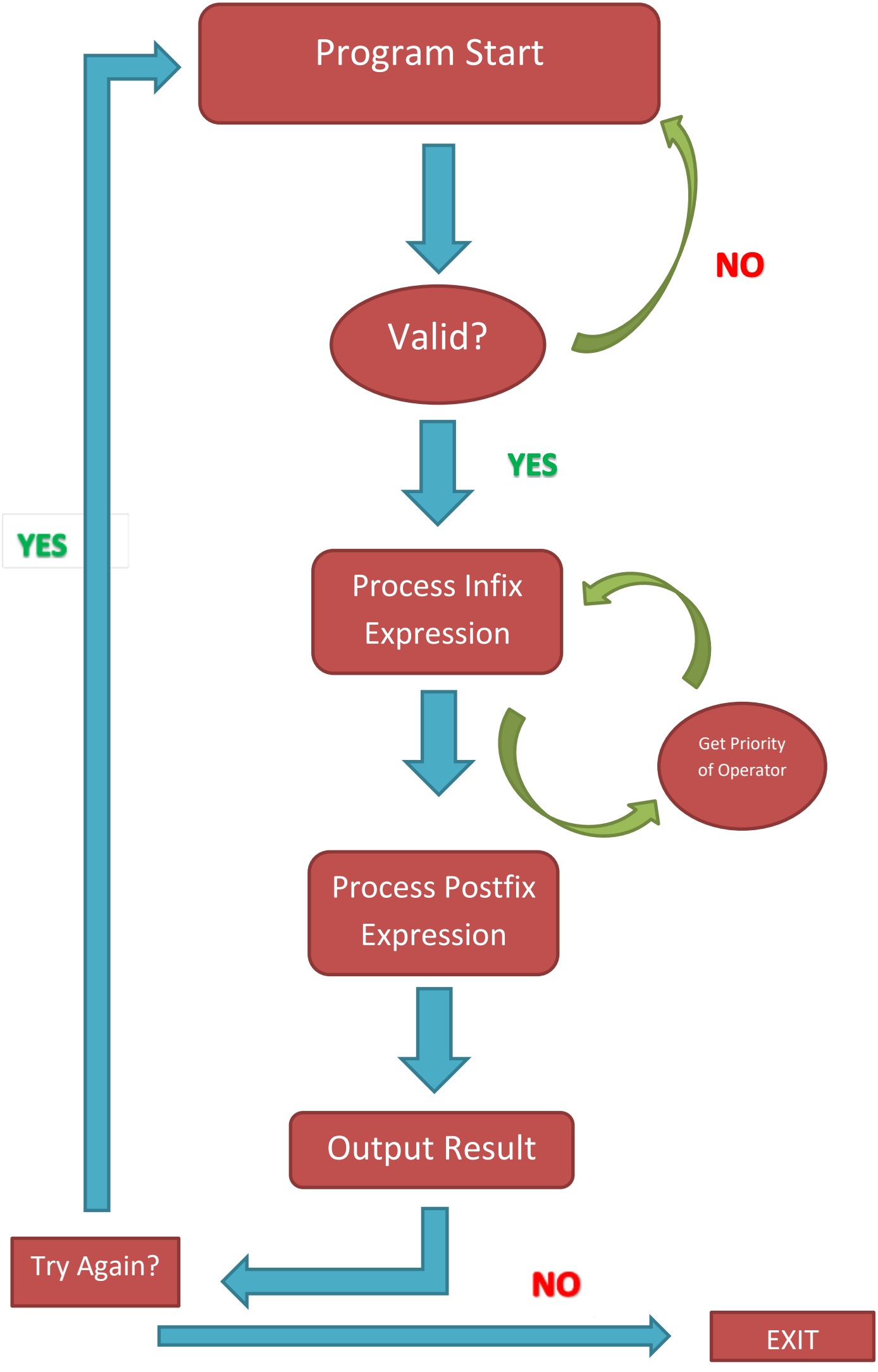
1. Create a switch statement which is called when an operator is encountered. It will return the priority of the operator as an integer from 1-3 with 3 being highest priority.
2. If it is a number send it straight to the stack
3. If it is an operand, call the switch statement from step 1 and compare it what is at the top of the stack. Pop the stack until an operator of lower priority is found.
4. If it is an open bracket, stack it.
5. If it is a closed bracket pop the stack (sending the operator into the result) until an open bracket is found. Then delete the open bracket and don't stack the closed bracket.
6. Finally, pop all remaining operators in the stack into the result

The final aspect will be to put this postfix expression into a method to be calculated. I will again need to convert it to a char array and go through each character. The instructions for this method will be:

1. If it is a digit push it into the stack
2. If it is an operator, pop the stack twice and store the values (I will need to cast the object to a double here)
3. Perform the operation on the 2 values
4. Push the result back into the stack
5. At the end of the expression, the number at the top of the stack is the result. (I will need to cast this as well).

An extra feature I would like to add if I have time is a try again option at the end of the program. To do this I will need to look into JOptionPane features to see how to achieve a GUI that receives a response.

The flow of control should be similar to the following flow chart:


Program Start
Valid?
NO
YES
YES
Process Infix Expression
Get Priority of Operator
Process Postfix Expression
Output Result
Try Again?
NO
EXIT

The Code:

Infix converter class

- This class contains the bulk of the code. When it is created, the program runs.

```
import javax.swing.*;

public class InfixConverter {

    String input;
    Boolean valid = false;
    char[] inputArray;
    Stack s = new ArrayStack(20); //Stack initialised


    public InfixConverter() {

        int begin = startUp(); //start up the program

        //Loop to allow user to try again if input is not valid
        while(begin==0){
            valid = false;
            begin = startUp();
        }

        System.exit(0);
    }

    //Method to begin the programme.
    public int startUp(){

        while(!valid) {
            input = JOptionPane.showInputDialog("Enter an infix numerical
expression between 3 & 20 characters.");
            if(validCheck(input)){
                valid = true;
            }
            else{
                JOptionPane.showMessageDialog(null, "Invalid Expression. Please
Enter Again.",
                                                "Input error",
JOptionPane.ERROR_MESSAGE);3*3
            }
        }

        String postfix = processInfix(inputArray);
        double result = processPostfix(processInfix(inputArray));
        JOptionPane.showMessageDialog(null, "The resulting expression is as
follows: "
                                        + "\nInfix: " + this.input + "\nPostfix: "
+ postfix + "\nResult: " + result,
                                        "Result", JOptionPane.PLAIN_MESSAGE);

        //Try again GUI. Returns 1 if NO and 0 if YES
        int n = JOptionPane.showConfirmDialog(
                null,
                "Would you like to try again?",
```

```
                "A Question For Thee",
                JOptionPane.YES_NO_OPTION);

        return n;

    }

    //Method validCheck makes sure the input:
    // 1. Is between 3 and 20 characters long and
    // 2. Only uses the valid characters (,),+,-,*,/,^ and single digits 0-9

    public boolean validCheck(String input) {

        int open= 0, closed = 0; //To count number of open and closed brackets

        if (input.length() >= 3 && input.length() <= 20) {
            inputArray = input.toCharArray(); //Convert the string to a Char Array

            if(!Character.isDigit(inputArray[0])) //If the first character isn't a
number, it's not valid e.g +3*6
                return false;

            for (int i = 0; i < inputArray.length; i++) {

                if(inputArray[i] == '(') //if open bracket increment the open
value..
                    open +=1;
                if(inputArray[i] == ')') //same but with closed bracket
                    closed+=1;

                if(i+1 < inputArray.length){ //to avoid array index out of bounds
exception
                    if (Character.isDigit(inputArray[i]) &&
Character.isDigit(inputArray[i + 1])) { //This makes sure no double digits
                        return false;
                    } else if ((validOperator(inputArray[i])) &&
(validOperator(inputArray[i + 1]) && inputArray[i+1] != '('){ //no 2 operators in
a row BUT ok if operator and open bracket
                        if(inputArray[i] != ')'){ //To allow operand after a closed
bracket e.g (3+1)+2
                            return false;
                        }

                    }
                }
                if (!Character.isDigit(inputArray[i]) &&
!validOperator(inputArray[i])) { //not a number AND not a valid operator
                    return false;
                }
            }

            if(open!= closed){ //Make sure the all open brackets are closed
                return false;
            }
            return true;

        }

        return false;
    }

    //Method validOperator takes in a char and checks to see if it is a valid
operator.

    public boolean validOperator(char op){
        switch(op){
            case '+':
            case '-':
```

```
            case '*':
            case '/':
            case '^':
            case ')':
            case '(':
                return true;
        }

        return false;
    }

    //Method getPriority takes the operator and returns an int that represents its
Priority Level. (3 being highest Priority)
    public int getPriority(char operator){
        switch(operator){
            case '^':
                return 3;
            case '*':
            case '/':
                return 2;
            case '+':
            case '-':
                return 1;
        }

        return 0;
    }

    //The method processInfix is where the inputted infix is converted into postfix
format

    public String processInfix(char[] input) {
        String result = ""; //Initialize the string. This will be the postfix
formatted input.

        for (int i = 0; i < input.length; i++) {

            //If it is a digit it is sent straight to the result
            if (Character.isDigit(input[i])) {
                result += input[i];
            }

            //If it is an open bracket it is pushed into the stack.
            else if (input[i] == '(') {
                s.push(input[i]);
            }

            //If it is a closed bracket, all items in the stack are popped until an
open bracket is found.
            //Both brackets are then discarded
            else if (input[i] == ')') {

                while (!s.isEmpty() && (char)s.top() != '(') {
                    result += (char)s.pop();
                }

                if (!s.isEmpty() && (char)s.top() != '(') {
                    System.out.println("Invalid Expression. A '(' character must be
followed by a ')' in the Expression.");
                } else {
                    s.pop(); //get rid of ( from the eq.
                }
            }

            //Else it must be an operand...
            else {
                //loop checks:
                // 1. that the stack is not empty and
```

```
                // 2. that the current token is less or equal in priority to the
one on the top of the stack
                // If it both are true then the operator is sent to the result
                while(!s.isEmpty() && getPriority(input[i]) <=
getPriority((char)s.top())){
                    result += (char)s.pop();
                }
                s.push(input[i]);
            }
        }

        //Pop the remainder of the stack to the result
        while(!s.isEmpty()){
            result += (char)s.pop();
        }

        return result;
    }


    //Method processPostfix takes the postfix expression as an argument, calculates
it and returns the result
    //The steps are
    // 1. If it is a digit push into stack
    // 2. If it is an operator, pop the stack twice, preform operations on the 2
values and push the result into stack

    public double processPostfix(String postfix){
        char[] postfixArr = postfix.toCharArray(); //Convert string to Char Array

        //loop through the array, looking at each value
        for(int i = 0; i < postfixArr.length; i++){

            //If digit, push it into the stack
            if(Character.isDigit(postfixArr[i])){
                s.push(postfixArr[i]);
            }
            else{
                double d1 = convertToDbl(s.pop()); //convert to double
                double d2 = convertToDbl(s.pop());

                //switch to determine what operation should be performed - printed
out for testing
                switch(postfixArr[i]){
                    case '+':
                        System.out.println(d2 + " + " + d1); s.push(d2+d1);
System.out.println("= " + (d2+d1)); break;
                    case '-':
                        System.out.println(d2 + " - " + d1); s.push(d2-d1);
System.out.println("= " + (d2-d1)); break;
                    case '*':
                        System.out.println(d2 + " * " + d1); s.push(d2 * d1);
System.out.println("= " + (d2*d1));break;
                    case '/':
                        System.out.println(d2 + "/" + d1); s.push(d2/d1);
System.out.println("= " + (d2/d1));break;
                    case '^':
                        System.out.println(d2 + " ^ " + d1); s.push(Math.pow(d2,
d1)); System.out.println("= " + (Math.pow(d2,d1))); break;
                    default:
                        System.out.println("ERROR IN CODE!");
                }
            }


        }

        double result = convertToDbl(s.top()); //return the item at top of the
```

```
stack (converted to a double)
        s.pop();
        return result;
    }

    //The Method convertToDbl takes in an object(from the stack) and casts it to a
double
    private double convertToDbl(Object a) {
        double result = 0;

        //If the object is a Character...
        if(a instanceof Character){
            char x = (Character)a;
            result = (double)(x - '0'); // -'0' as it will convert the ascii number
(e.g 4 is 52 in ascii) to the standard number
        }

        //If it is a Double...
        else if(a instanceof Double){
            result = (Double)a;
        }

        return result;
    }
}
```

Other classes used are the given ArrayStack and a very simple test class to run the code.

Test class to run code:

```
public class StackTest {

    public static void main(String[] args)
    {
        InfixConverter convertmebaby = new InfixConverter();

    }
}
```

Testing section on next page

TEST	Expected Outcome	Actual Outcome	Works?
Not enough characters entered	Error screen along with a try again prompt	Error screen along with a try again prompt See figure 1 & 2	✓
Too many characters entered	Error screen along with a try again prompt	Error screen along with a try again prompt See figure 1 & 2	✓
Non valid operator entered	Error screen along with a try again prompt	Error screen along with a try again prompt See figure 1 & 2	✓
Double digit number entered	Error screen along with a try again prompt	Error screen along with a try again prompt See figure 1 & 2	✓
Double operands entered	Error screen along with a try again prompt	Error screen along with a try again prompt See figure 1 & 2	✓
Convert infix expression 4*4+2 into postfix notation	44*2+ (Found using online converter)	44*2+ See figure 3	✓
Convert infix expression 6*3+4*(6*4/3)-9+7 To postfix notation	63*464*3/*+9-7+ (Found using online converter)	63*464*3/*+9-7+ See figure 4	✓
Open bracket entered without a closing one	Error screen along with a try again prompt	Error screen along with try again prompt. See fig 1 & 2	✓
Postfix expression 4*4+2 calculated	18	18 See figure 3	✓
Postfix expression 6*3+4*(6*4/3)-9+7 calculated	48	48 See figure 4	✓

Testing inputs and outputs

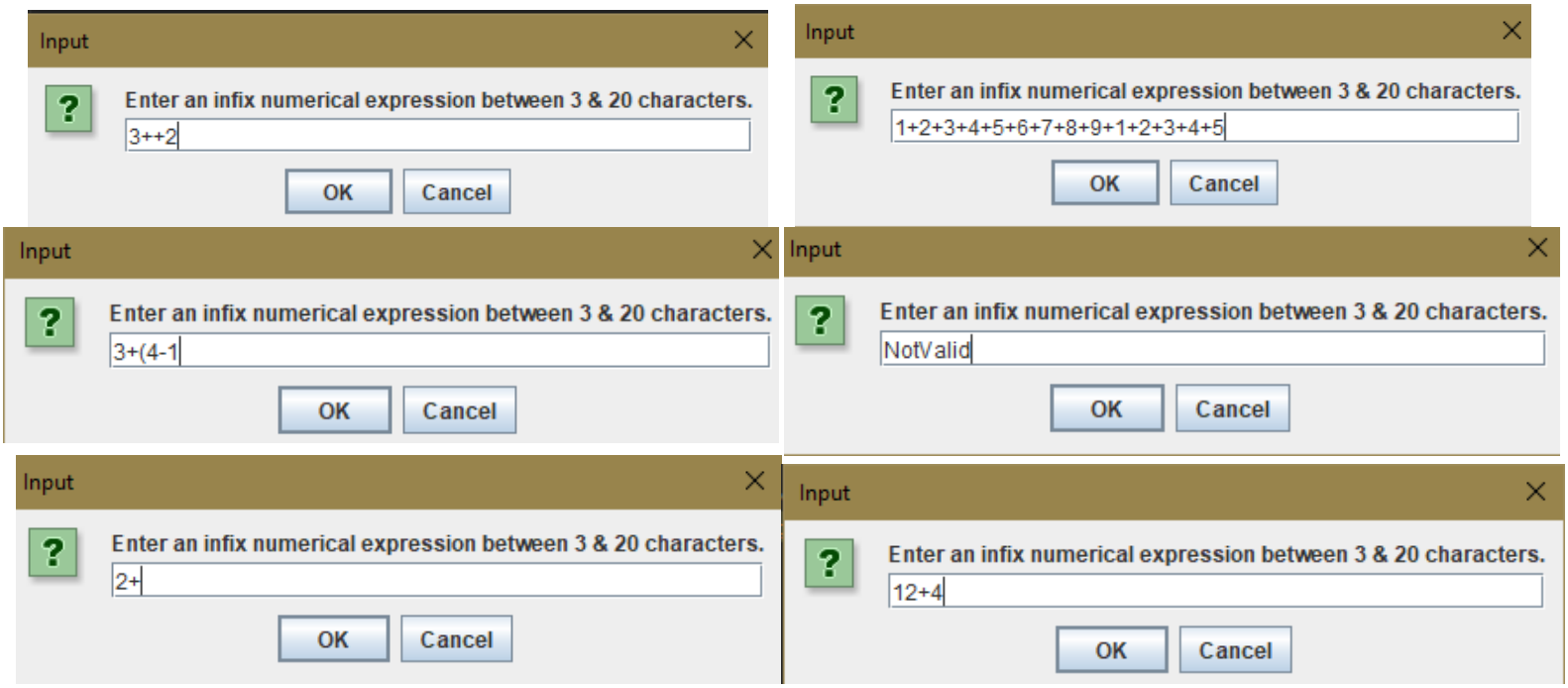


Figure 1: Non valid inputs

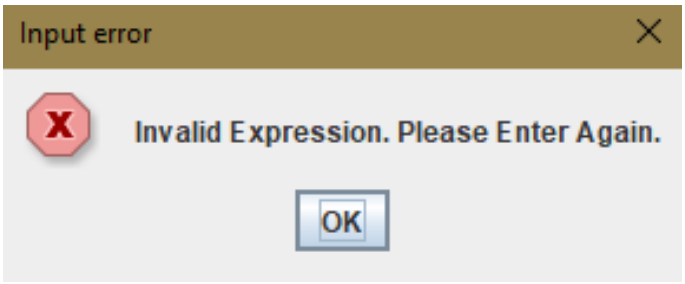


Figure 2: Error screen if non valid input entered

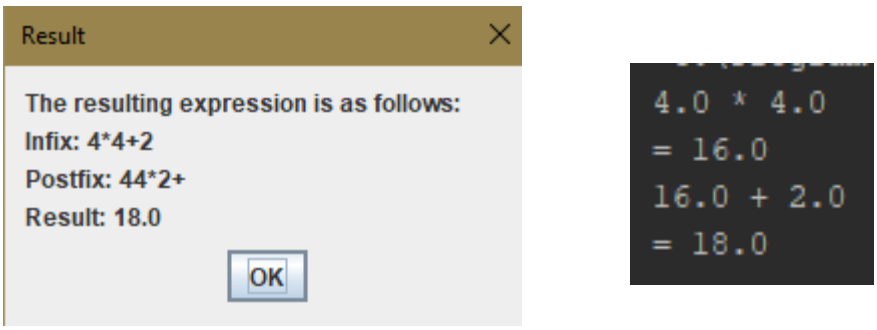


Figure 3: Outputted result and calculation in console (for testing)

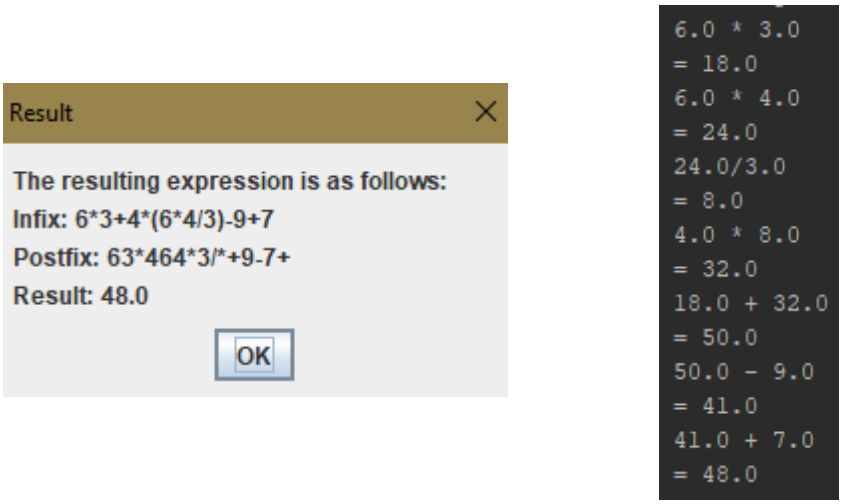


Figure 4: Outputted result and calculation in console (for testing)

Sources

Online Convertor Used for Testing Conversion - https://www.mathblog.dk/tools/infix-postfix-converter/