In-fix to Post-fix converter and calculator

(Author - Abishek Bupathi)

I. Problem statement

The function of the program is to get an infix expression as input from the user and check whether it consists only of digits and operators or else print an incorrect message.

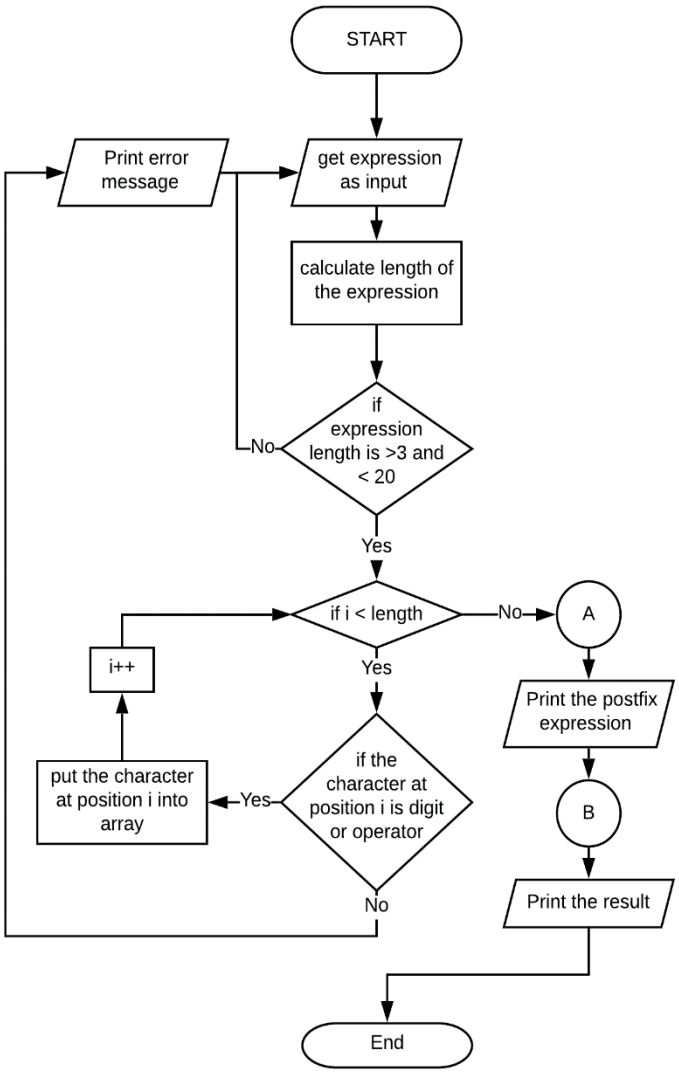
After checking for the correctness of the input, the infix expression must be converted to postfix expression and print it. Then the postfix expression needs to be evaluated and the result must be printed.

II. Analysis and Design

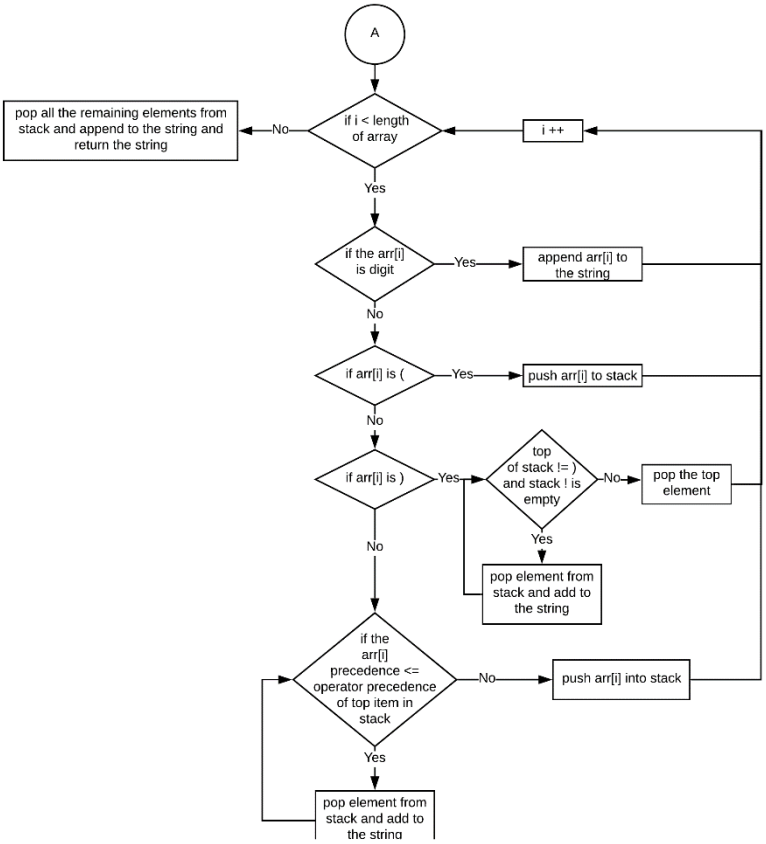
For the program I chose to have 2 separate functions to carry out the 2 different processes:

- postfix_convertor () – to convert infix expression to postfix expression
- evaluate () – to calculate the result using the postfix expression

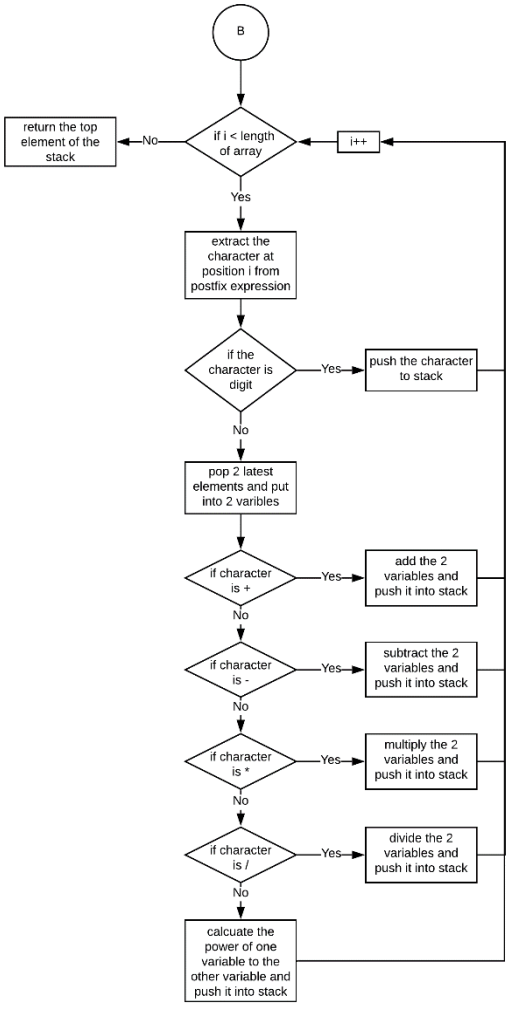
a) Flowchart for main Method:



b) Flowchart for postfix_converter() method:



c) Flowchart for evaluate() method:



There is also another function operator_precedence () that returns an integer value for the operator that is sent in the arguments so that 2 operators can be compared while popping and pushing into the stack.

III. Code

```
import java.util.Scanner;

public class InfixToPostfix {

    // initializing scanner class
    public static Scanner obj= new Scanner(System.in);

    // initializing required class variables
    static int top = -1,top_evaluate = -1, size = 100;
    static char  stack[] = new char[size];
    static double stack_evaluate[] = new double[size];

    public static void main(String args[]){

        // initializing required variables
        String input;
        int input_length;
        String postfix_expression = "";
        char current_character;
        boolean flag = false;

        while(flag != true) {

            System.out.println("Enter the Infix Expression");

            input = obj.next(); //getting input from user

            // computing the length of the input
            input_length = input.length();
            // creating array to store characters of the input
            char arr[] = new char[input.length()];

            // checking the length of the expression
            if (input_length >= 3 && input_length <= 20)
            {
               // for loop to put character from string into array
               for (int i = 0; i < input_length; i++) {

                current_character = input.charAt(i);

                // checking for the correctness of the expression
                if (operator_precedence(current_character) != 0 ||
Character.isDigit(current_character) || current_character == '(' || current_character == ')') {
                    arr[i] = current_character;
                } else {
                    flag = true;
                    break;
                }
             }

            } else flag = true;

            if (flag == false) {

                // calling postfix_converter function and storing it
                postfix_expression = postfix_converter(arr, input_length);

                System.out.println("Postfix Expression: " + postfix_expression);

                // calling evaluate function and storing it
                Double result = evaluate(postfix_expression, postfix_expression.length());

                System.out.printf("Result = %.2f", result);
                flag = true;

            } else {
                // printing failure message for incorrect expression
                System.out.println("Invalid input please try again");
                flag = false;
            }
        }

    }

/*
postfix_converter function get in the array and it's length as arguments and converts the
infix expression into postfix and returns as a string
 */
    public static String postfix_converter(char arr[], int input_length){
```

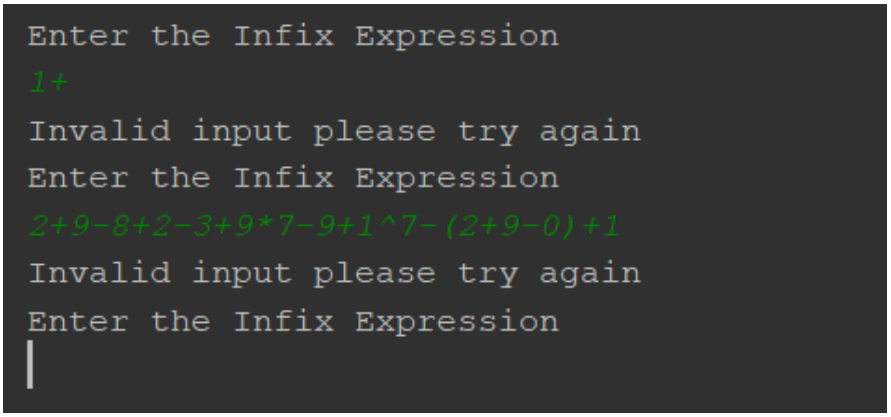
```
        // initializing local variable
        String postfix = new String("");

        // loop to iterate through each elements in the array
        for (int i = 0; i<input_length; i++) {

            // checking whether current element is a digit
            if (Character.isDigit(arr[i]))
                // appending it to the postfix string
                postfix += arr[i];

            // checking if the current element is (
            else if (arr[i] == '(')
                //appending it the string
                push(arr[i]);

            // checking for )
            else if (arr[i] == ')')
            {
                // popping out all the elements before ( in the stack and appending it into string
                while (!isEmpty() && getTop() != '(')
                    postfix += pop();

                // discarding (
                 pop();
            }

            else {

                // checking whether the operator precedence of the current element is less than of
that present on the top of the stack
                while (operator_precedence(arr[i]) <= operator_precedence(getTop()) )
                    // popping elements in the stack and adding it to string
                    postfix += pop();

                // if the operator precedence of the current element is greater than of that present
on the top of the stack the above while loop is skipped and current element is pushed
                push(arr[i]);
            }
        }

        // popping out all the elements from the stacks
        while(!isEmpty()) {
            postfix += pop();
        }

        return postfix;
    }

    // evaluate function gets postfix string and its length as arguments and return the evaluated
value of the expression
    public static double evaluate(String postfix, int length){

        // initalizing local variables
        char c;
        double a,b;

        // loop to iterate through each character in the String
        for(int i=0; i< length; i++) {

            c = postfix.charAt(i);

            // checking for digits
            if (Character.isDigit(c))
                //pushing the digit into stack array
                push_evaluate((double) (c - '0'));
            else {
                // popping last 2 elements and assigning the values to variables a and b if a
operator is detected
                a = pop_evaluate();
                b = pop_evaluate();

                // if the character is +, a and b are added and pushed back into the stack
                if (c == '+') {
                    push_evaluate(a+b);

                } else
                    // if the character is -, b and a are subtracted and pushed back into the stack
                    if (c == '-') {
                    push_evaluate(b-a);

                } else
                        // if the character is *, a and b are multiplied and pushed back into the
stack
                    if (c == '*') {
```

```
                    push_evaluate(a*b);

                } else
                        // if the character is /, b and a are divided and pushed back into the stack
                    if (c == '/') {

                    push_evaluate(b/a);

                } else
                        // if the character is +, b to the power of a is calculated and pushed back
into the stack
                    if (c == '^'){

                    push_evaluate(Math.pow(b,a));

                }
            }
        }

        // returning the calculated value
        return getTop_evaluate();
    }
    // operator_precedence method gets operator as argument and return the precedence of the operator
    public static int operator_precedence(char operator){
        int value = 0;

        if(operator == '^')
            // if the operator is ^ value 3 is returned
            value = 3;
        else if(operator == '*' || operator == '/')
            // if the operator is * or / value 2 is returned
            value = 2;
        else if(operator == '+' || operator == '-')
            // if the operator is + or - value 1 is returned
            value = 1;

        return value;
    }
    //push method pushes the element in the stack array
    public static void push(char item){

        if(top > size-1 )
            System.out.println("Stack Overflow");
        else {
            top++;
            stack[top] = item;
        }
    }
    // pop element remove and returns the top element of the stack array
    public static char pop(){

        char item;

        if(top < 0 ) {
            System.out.println("Stack underflow");
            return ' ';
        }else {

            item = stack[top];
            stack[top] = ' ';
            top--;
            return item;
        }
    }
    // pop_evaluate removes and return the top element of the stack_evaluate array
    public static double pop_evaluate(){

        double item;

        if(top_evaluate < 0 ) {
            System.out.println("Stack underflow");
            return ' ';
        }else {
            item = stack_evaluate[top_evaluate];
            stack[top_evaluate] = ' ';
            top_evaluate--;
            return item;
        }
    }
    //push method pushes the element into stack_evaluate array
    public static void push_evaluate(double item){

        if(top_evaluate > size-1 )
            System.out.println("Stack Overflow");
```

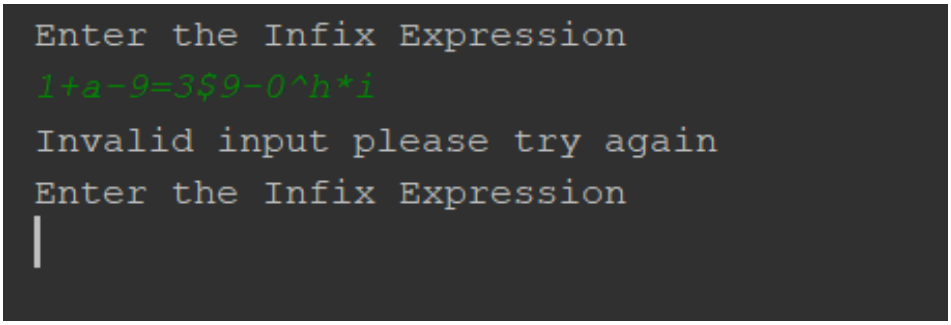
```
            else {
                top_evaluate++;
                stack_evaluate[top_evaluate] = item;
            }
        }
        // getTop function returns the top value of stack array
        public static char getTop(){
            if (top == -1)
                return ' ';
            else
                return stack[top];
        }
        //getTOP_evaluate function returns the top value of stack_evaluate array
        public static double getTop_evaluate(){
            if (top_evaluate == -1)
                return ' ';
            else
            return stack_evaluate[top_evaluate];
        }
        // isEmpty checks whether the stack array is empty
        public static boolean isEmpty(){
            if (top==-1)
                return true;
            else
                return false;
        }
}
```

IV. Testing

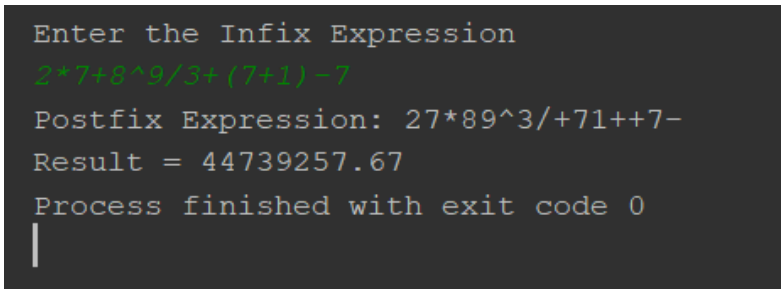
Scenario 1 - when the input is less than 3 or greater than 20:

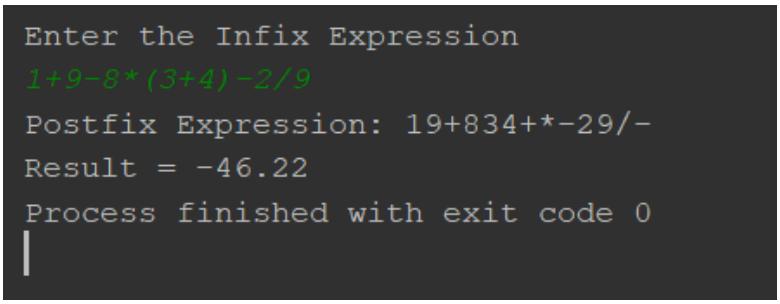


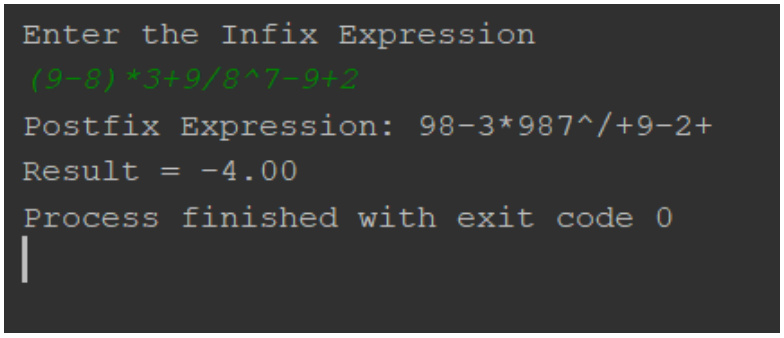
Scenario 2 - When characters other than 0-9 and +, -, *, /, ^, (,) is entered:



Scenario 3 – When correct expression is entered:







INPUT	Expected Output	Actual Output
1+	Invalid input please try again	Invalid input please try again
2+9-8+2-3+9*7-9+1^7-(2+9-0)+1	Invalid input please try again	Invalid input please try again
1+a-9=3$9-0^h*i	Invalid input please try again	Invalid input please try again
2*7+8^9/3+(7+1)-7	Postfix Expression: 27*89^3/+71++7- Result = 44739257.67	Postfix Expression: 27*89^3/+71++7- Result = 44739257.67
1+9-8*(3+4)-2/9	Postfix Expression: 19+834+*-29/- Result = -46.22	Postfix Expression: 19+834+*-29/- Result = -46.22
(9-8)*3+9/8^7-9+2	Postfix Expression: 98-3*987^/+9-2+ Result = -4.00	Postfix Expression: 98-3*987^/+9-2+ Result = -4.00