**Infix, Prefix and Postfix Notation**

We are accustomed to write arithmetic expressions with the operation between the two operands: **a+b**or**c/d**.  If we write **a+b\*c**, however, we have to apply precedence rules to avoid the ambiguous evaluation (add first or multiply first?).

There's no real reason to put the operation between the variables or values.  They can just as well precede or follow the operands.  You should note the advantage of prefix and postfix: the need for precedence rules and parentheses are eliminated.

|  |  |  |
| --- | --- | --- |
| **Infix** | **Prefix** | **Postfix** |
| a + b | + a b | a b + |
| a + b \* c | + a \* b c | a b c \* + |
| (a + b) \* (c - d) | \* + a b - c d | a b + c d - \* |

Postfix expressions are easily evaluated with the aid of a stack.



## Program to convert an infix expression to postfix

import java.util.Stack;

class Test

{

    // A utility function to return precedence of a given operator

    // Higher returned value means higher precedence

    static int Prec(char ch)

    {

        switch (ch)

        {

        case '+':

        case '-':

            return 1;

        case '\*':

        case '/':

            return 2;

        case '^':

            return 3;

        }

        return -1;

    }

    // The main method that converts given infix expression

    // to postfix expression.

    static String infixToPostfix(String exp)

    {

        // initializing empty String for result

        String result = new String("");

        // initializing empty stack

        Stack<Character> stack = new Stack<>();

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

        {

            char c = exp.charAt(i);

             // If the scanned character is an operand, add it to output.

            if (Character.isLetterOrDigit(c))

                result += c;

            // If the scanned character is an '(', push it to the stack.

            else if (c == '(')

                stack.push(c);

            //  If the scanned character is an ')', pop and output from the stack

            // until an '(' is encountered.

            else if (c == ')')

            {

                while (!stack.isEmpty() && stack.peek() != '(')

                    result += stack.pop();

                if (!stack.isEmpty() && stack.peek() != '(')

                    return "Invalid Expression"; // invalid expression

                else

                    stack.pop();

            }

            else // an operator is encountered

            {

                while (!stack.isEmpty() && Prec(c) <= Prec(stack.peek())){

                    if(stack.peek() == '(')

                        return "Invalid Expression";

                    result += stack.pop();

             }

                stack.push(c);

            }

        }

        // pop all the operators from the stack

        while (!stack.isEmpty()){

            if(stack.peek() == '(')

                return "Invalid Expression";

            result += stack.pop();

         }

        return result;

    }

    // Driver method

    public static void main(String[] args)

    {

        String exp = "a+b\*(c^d-e)^(f+g\*h)-i";

        System.out.println(infixToPostfix(exp));

    }

}

## Postfix Evaluation Algorithm

Assume we have a string of operands and operators, an informal, by hand process is

1. Scan the expression left to right
2. Skip  values or variables (operands)
3. When an operator is found, apply the operation to the preceding two operands
4. Replace the two operands and operator with the calculated value (three symbols are replaced with one operand)
5. Continue scanning until only a value remains--the result of the expression

The time complexity is O(n) because each operand is scanned once, and each operation is performed once.

/ Java proram to evaluate value of a postfix

// expression having multiple digit operands

import java.util.Stack;

class Test1

{

    // Method to evaluate value of a postfix expression

    static int evaluatePostfix(String exp)

    {

        //create a stack

        Stack<Integer> stack = new Stack<>();

        // Scan all characters one by one

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

        {

            char c = exp.charAt(i);

            if(c == ' ')

            continue;

            // If the scanned character is an operand

            // (number here),extract the number

            // Push it to the stack.

            else if(Character.isDigit(c))

            {

                int n = 0;

                //extract the characters and store it in num

                while(Character.isDigit(c))

                {

                    n = n\*10 + (int)(c-'0');

                    i++;

                    c = exp.charAt(i);

                }

                i--;

                //push the number in stack

                stack.push(n);

            }

            // If the scanned character is an operator, pop two

            // elements from stack apply the operator

            else

            {

                int val1 = stack.pop();

                int val2 = stack.pop();

                switch(c)

                {

                    case '+':

                    stack.push(val2+val1);

                    break;

                    case '-':

                    stack.push(val2- val1);

                    break;

                    case '/':

                    stack.push(val2/val1);

                    break;

                    case '\*':

                    stack.push(val2\*val1);

                    break;

            }

            }

        }

        return stack.pop();

    }

    // Driver program to test above functions

    public static void main(String[] args)

    {

        String exp = "100 200 + 2 / 5 \* 7 +";

        System.out.println(evaluatePostfix(exp));

    }

}