Stack Application:

2 * (3 + 3) – 8 / 4 / (1 + 1)

- Convertion of expressions.
- Expression evaluation.

Example: a + b => 2 + 3 -> infix expressions

1. Infix expressions -> <operand1> <operator> <operand2> Eg: A+B
2. Postfix expression -> <operand1> <operand2> <operator> Eg: ab+
3. Prefix expression -> <operator> <operand1> <operand2> Eg: +ab

Why Postfix and Prefix?

1. These expressions have only operands and operators
2. Don’t have to follow any precedence
3. Easy to evaluate these expression

1. How to convert infix expression to postfix expression
2. How to evaluate Postfix expression

Infix to Postfix conversion: Follow the precedence

2 * 3 + 3 – 8 / 4 / (1 + 1)

1) 2 * 3 + 3 – 8 / 4 / T1

2) T2 + 3 – 8 / 4 / T1

3) T2 + 3 – T3 / T1

4) T2 + 3 – T4

5) T5 – T4

6) T5 T4 -

T1 = {1 1 + }

2 * 3 => 2 3 *

T2 = 2 3 *

8 / 4 => 8 4 /

T3 => 8 4 /

T3 / T1 => T3 T1 /

T4 = T3 T1/

T2 + 3 => T2 3 +

T5 = T2 3 +

T5 T4 -

T2 3 + T3 T1 / -

2 3 * 3 + 8 4 / 1 1 + / -

Convertion using stack:

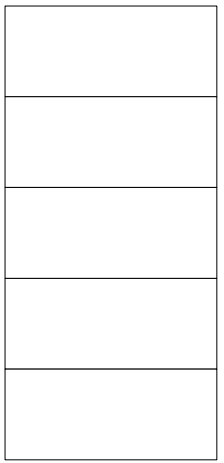
infix_exp

2 * 3 + 3 – 8 / 4 / (1 + 1) \0

postfix_exp

2 3 * 3 + 8 4 / 1 1 + / -

1. Scan the infix expression from left to right
2. Check it is operator or operand
3. If operand, Store it in the postfix expression.
4. if operator, then push the operator into stack when the stack is empty
5. If stack is not empty, then check the precedene of the operator
   5.1 if (stack not empty and stack(opr) >= infix(opr)) -> You cannot push
   pop that operator from stack and store it in postfix expression
   5.2 else
   Push operator into stack
6. if ‘(‘, then push ‘(‘ into stack
7. if ‘)’, then pop the operators from the stack and store it to
Postfix expression until you reach ‘(‘, discard ‘(‘.



stack

8. EOE, then pop all the operators from the stack and store it in the postfix exp

isalnum()

Postfix evaluation:

postfix_exp

2 3 * 3 + 8 4 / 1 1 + / -

1. Scan the expression from left to right and check
It is operator or operand.

2. if operand, then push the operand into stack

3. If operator, then pop 2 operands from stack
   And the order of operands, (operand2, operand1)
4. Perform the operation based on operator

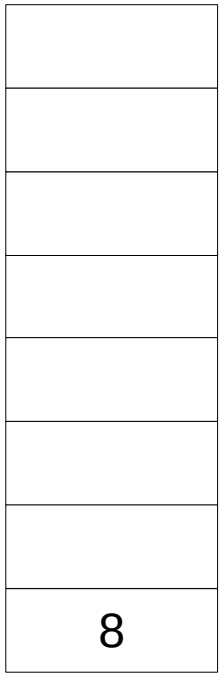
5. Push the result back to stack

6. If EOE, then pop the result from the stack.

Op2 = 1

Op1 = 9

Op1 opr op2
2 * 3 = 6
6 + 3 = 9
8 / 4 = 2
1 + 1 = 2
2 / 2 = 1
9 – 1 = 8



stack

```
precedence(opr)
{
        switch(opr)
        {
                '+' :
                '-':  return 1;
                *:
                /: return 2;
        }
}
```

```
+     -> 1
-
*     -> 2
/
```