CS & IT ENGINEERING

Data Structures Through Python

STACK

Lecture No.- 02

By- Satya sir

Recap of Previous Lecture

- Stack Operations
  - Push → Stack Full / overflow : TOP == N-1
  - Pop
  - Peek } stack Empty / underflow : TOP == -1
- Stack Permutations
  - Static / fixed sequence for Push : $\frac{^{2n}C_n}{n+1}$ Permutations
  - Dynamic / Variable / Random Push : $n!$ Permutations

Topics to be Covered

Applications of Stack

- Expression Conversion

  - Infix to Postfix Conversion

Topic : Expression Conversion - 1

- Applications of Stack (Last-In-First-Out)

  - Browsing history
  - undo operation
  - Tree | Graph Traversals : Depth-First Traversal
  - Recursion Implementation : $f(5) \rightarrow f(4) \rightarrow f(3) \rightarrow f(2) \rightarrow f(1) \rightarrow f(0)$
  - Function Implementation :

$$\begin{tikzcd} f() \arrow[r] & g() \arrow[r] \arrow[l, bend left] & h() \arrow[r] \arrow[l, bend left] & fun() \arrow[l, bend left] \end{tikzcd}$$

  - Backtracking
  - ✓ Expression Conversion
  - ✓ Expression Evaluation ---
  - Bracket balancing --

Topic : Expression Conversion - 1

Expression Conversion

Expression : Any Executable statement

- input/output
- function definitions
- Declarations Stmts
- Assignment Stmts
- Computational stmts → Arithmetic Expression
  x = 7 + 4 and 3 / 2 * 5 - 1
- Conditional stmts
- Iterative stmts

Arithmetic Expression

which uses 5 operators, along with ( )

1) + addition
2) − Subtraction
3) * multiplication
4) / division
5) ^ Exponentiation (raised to)

Default Precedence : ^ > /, * > +, −

Associativity : L TO R   L TO R

Topic : Expression Conversion - 1

$(7+3)*5/(2-1)$

Expression Representations

① Infix Notation / Representation : Operand1 Operator Operand2 $a+b$

② Prefix Notation / Polish Notation : Operator operand1 operand2 $+ab$

③ Postfix Notation | Reverse Polish Notation : Operand1 operand2 operator $ab+$

Conversion Possibilities :

1. Infix to Postfix ✓
2. Infix to Prefix ✓
3. Postfix to Infix
4. Postfix to Prefix
5. Prefix to Infix
6. Prefix to Postfix

Topic : Expression Conversion - 1

Infix to Postfix Conversion Procedure

① Let 'X' be infix ExP, 'Y' be resultant Postfix ExP, 'S' be stack.

② Scan 'X' from Left to Right, one Element at a time.

③ a) If scanned Element == Operand, Add it to Y.

b) If scanned Element == '(', Push it on stack S.

c) If Scanned Element == ')', keep Popping elements from S, add them to Y, until first '(' is encountered, ignore Paranthesis.

d) If Scanned Element == Operator (OP_S)

i) If stack is Empty, Push OP_S onto Stack.

ii) If TOS == '(', Push OP_S on to Stack

iii) If TOS == Operator(OP_T) : compare(OP_S, OP_T)

→ If $OP_T >= OP_S$, Pop OP_T, add it to Y. Push OP_S

→ If $OP_T < OP_S$, Push OP_S onto stack.

④ Repeat step ③ until Complete Expression is scanned.

⑤ once, Expression is over, make Stack Empty, by Popping and adding to Y.

⑥ The Resultant Postfix Exp. is Y.

Topic : Expression Conversion - 1

Example-1 :

Convert Infix expression X: A+(B−C/D)∧(E*F)+G/H to Postfix expression.

Scanned Element : A, +, (, B, −, C, /, D,), ∧, (, E, *, F,), +, G, /, H

OP_S (under /) OP_S (under +)

Stack :	+	~~(~~	−	/	~~)~~	∧	~~(~~	*	~~)~~	+	/	
	OP_T		OP_T			OP_T						
	Pop		Pop	Pop		Pop		Pop		Pop	Pop	

Y : ABCD/−EF*∧+GH/+

(Postfix Exp)

Example-2 : Convert to Postfix Expression

X : (3+(5*7/2)^(9-1)*5^(3^4))

Scanned Element : (, 3, +, (, 5, *, 7, /, 2,), ^, (, 9, -, 1,), *, 5, ^, (, 3, ^, 4,),)

Stack :

~~(~~	+	~~(~~	*	/	~~)~~	^	~~(~~	-	~~)~~	*	^	~~(~~	^	~~)~~	~~)~~
	Pop		Pop	Pop		Pop		Pop		Pop	Pop		Pop		

Y : 357*2/91-^534^^*+

Ex: 3

The resultant Postfix Expression for an infix Expression,

$$P \wedge (Q \wedge R / S) * (T \wedge U + V) - W \wedge (X / Y * Z) \text{ is } \_\_\_\_\_\_$$

Stack :	∧	~~(~~	∧	/	~~)~~	*	~~(~~	∧	+	~~)~~	−	∧	~~(~~	/	*	~~)~~
	Pop		Pop	Pop		Pop		Pop	Pop		Pop	Pop		Pop	Pop	

$y : P Q R \wedge S / \wedge T U \wedge V + * W X Y / Z * \wedge -$

Topic : Expression Conversion - 1

```
#Python Code to convert infix to Postfix Expression

Operators = set(['+', '-', '*', '/', '(', ')', '^'])

Priority = {'+':1, '-':1, '*':2, '/':2, '^':3}#1: Low, 3: High

def infixToPostfix(expression):
    stack = []
    output = ‘ '
    for character in expression:
        if character not in Operators:
            output+= character
        elif character=='(':
            stack.append('(')
        elif character==')':
            while stack and stack[-1]!= '(':
                output+=stack.pop()
            stack.pop()
```

Topic : Expression Conversion - 1

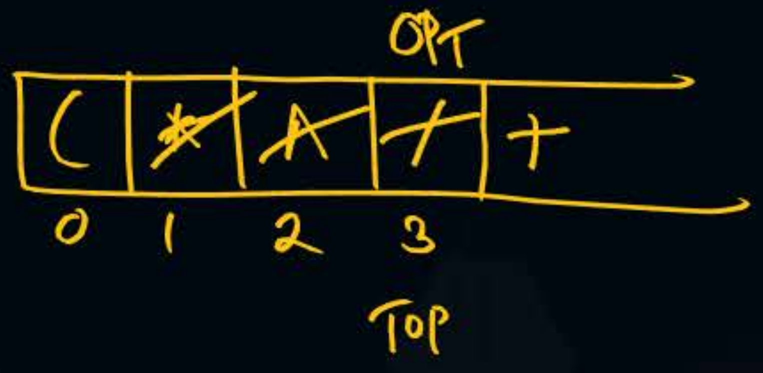
Stack is Not Empty

```
else:
        while stack and stack[-1]!='(' and Priority[character]<=Priority[stack[-1]]:
            output+=stack.pop()
        stack.append(character)
    while stack:
        output+=stack.pop()
    return output

expression = input('Enter infix expression ')
print('infix notation: ',expression)
print('postfix notation: ',infixToPostfix(expression))
```

Top Element

Scanned character = '+' OP_S



$OP_T \geq OP_S$ Pop, append, Push OP_S on S

2 mins Summary

- Applications of stack
  - Exp Conversion
    - Infix to Postfix conversion.

H/W Problem : X : $\left(a + b - \left(c / d * (e \wedge f \wedge g) / h\right) \wedge (i * j + k)\right)$

Y : ______________

t.me/ Satyasirpw

THANK - YOU