Sunbeam Institute of Information Technology

Pune and Karad

Module – Data Structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Applications – Stack and Queue

Stack

- Parenthesis balancing
- Expression conversion and evaluation
- Function calls
- Used in advanced data structures for traversing

- **Expression conversion and evaluation:**
  - Infix to postfix
  - Infix to prefix
  - Postfix evaluation
  - Prefix evaluation

Queue

- Jobs submitted to printer
- In Network setups – file access of file server machine is given to First come First serve basis
- Calls are placed on a queue when all operators are busy
- Used in advanced data structures to give efficiency.
- Process waiting queues in OS

Expression : combination of operands & operators

Type :

1) Infix : a + b (human)

2) Prefix : + a b

3) Postfix : a b +

} (computer)

$$\begin{tikzcd}\text{(computer)} \arrow[d] \\ \text{CPU} \arrow[d] \\ \text{ALU}\end{tikzcd}$$

Operation :

( )

$

* / %

+ −

Postfix Evaluation

- Process each element of postfix expression from left to right
- If element is operand
  - Push it on a stack
- If element is operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op2 – first popped element
    - Op1 – second popped element
  - Perform current element (Operator) operation between Op1 and Op2
  - Again push back result onto the stack
- When single value will remain on stack, it is final result
- e.g. 4 5 6 * 3 / + 9 + 7 -

Postfix evaluation

Postfix expression : 4 5 6 * 3 / + 9 + 7 -

$l \longrightarrow r$

Result : 16

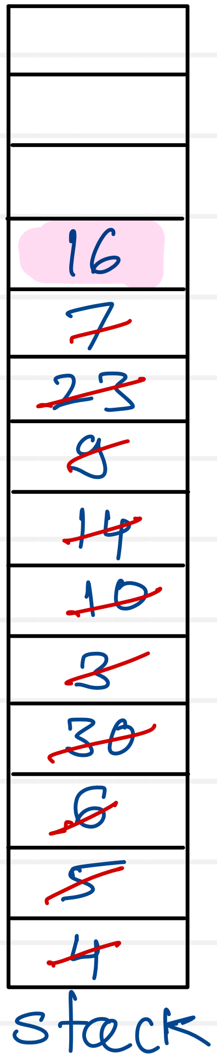
⑤ 23 − 7 = 16

④ 14 + 9 = 23

③ 4 + 10 = 14

② 30 / 3 = 10

① 5 * 6 = 30



'4' = 4

'0' = 48	'0' − '0' = 0
'1' = 49	'1' − '0' = 1
'2' = 50	'2' − '0' = 2

Prefix Evaluation

- Process each element of prefix expression from right to left
- If element is operand
  - Push it on a stack
- If element is operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op1 – first popped element
    - Op2 – second popped element
  - Perform current element (Operator) operation between Op1 and Op2
  - Again push back result onto the stack
- When single value will remain on stack, it is final result
- e.g. - + + 4 / * 5 6 3 9 7

Prefix evaluation

Prefix expression : - + + 4 / * 5 6 3 9 7

$l \longleftarrow r$

Result : 16

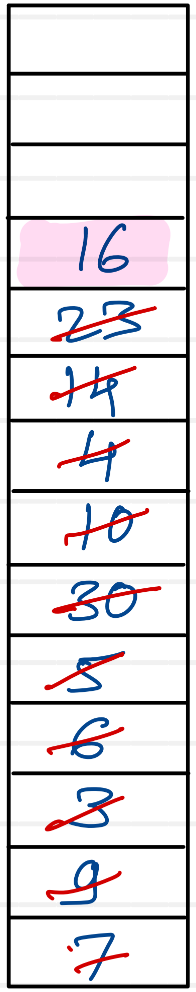
⑤ $23 - 7 = 16$

④ $14 + 9 = 23$

③ $4 + 10 = 14$

② $30 / 3 = 10$

① $5 * 6 = 30$



Infix to Postfix Conversion

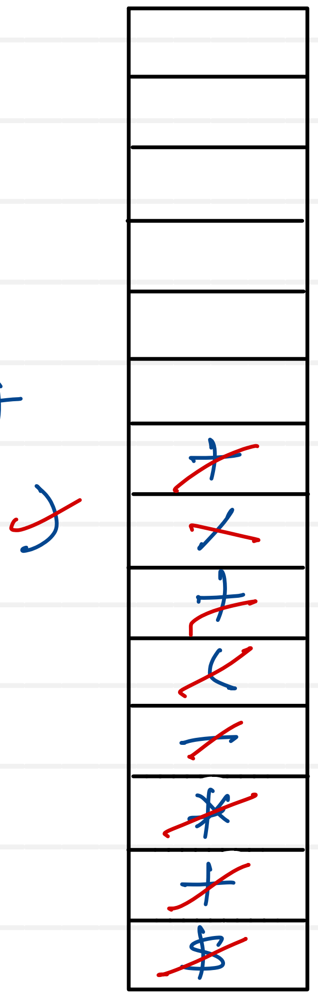
- Process each element of infix expression from left to right
- If element is Operand
  - Append it to the postfix expression
- If element is Operator
  - If priority of topmost element (Operator) of stack is greater or equal to current element (Operator), pop topmost element from stack and append it to postfix expression
  - Repeat above step if required
  - Push element on stack
- Pop all remaining elements (Operators) from stack one by one and append them into the postfix expression
- e.g. a * b / c * d + e – f * h + i

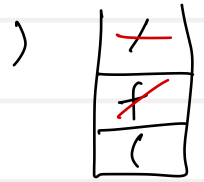
Infix to Postfix conversion

Infix expression : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7

$l \xrightarrow{\qquad\qquad\qquad} r$

Postfix expression : 1 9 $ 3 4 * + 6 8 2 / + - 7 +





```
while(st.peek() != '(')
    st.pop()
st.pop();
```

Infix to Prefix Conversion

- Process each element of infix expression from right to left
- If element is Operand
  - Append it to the prefix expression
- If element is Operator
  - If priority of topmost element of stack is greater than current element (Operator), pop topmost element from stack and append it to prefix expression
  - Repeat above step if required
  - Push element on stack
- Pop all remaining elements (Operators) from stack one by one and append them into the prefix expression
- Reverse prefix expression
- e.g. a * b / c * d + e – f * h + i

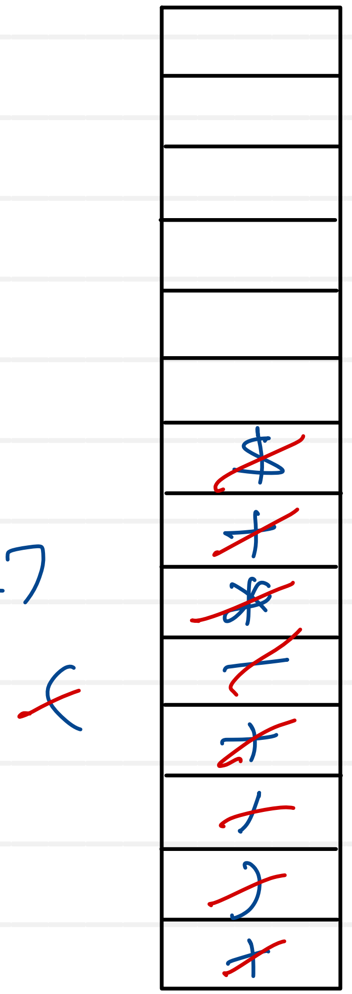
Infix to Prefix conversion

Infix expression : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7

$l \xleftarrow{\quad\quad\quad} r$

Expression : 7 2 8 / 6 + 4 3 * 9 1 $ + - +

Prefix expression : + - + $ 1 9 * 3 4 + 6 / 8 2 7



Prefix to Postfix

- Process each element of prefix expression from right to left
- If element is an Operand
  - Push it on to the stack
- If element is an Operator
  - Pop two elements (Operands) from stack, in such a way that
    - Op1 – first popped element
    - Op2 – second popped element
  - Form a string by concatenating Op1, Op2 and Opr (element)
  - String = “Op1+Op2+Opr”, push back on to the stack
- Repeat above two steps until end of prefix expression.
- Last remaining on the stack is postfix expression
- e.g. * + a b – c d

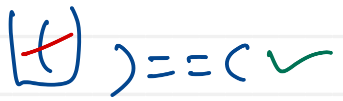
Postfix to Infix

- Process each element of postfix expression from left to right
- If element is an Operand
    - Push it on to the stack
- If element is an Operator
    - Pop two elements (Operands) from stack, in such a way that
        - Op2 – first popped element
        - Op1 – second popped element
    - Form a string by concatenating Op1, Opr (element) and Op2
    - String = “Op1+Opr+Op2”, push back on to the stack
- Repeat above two steps until end of postfix expression.
- Last remaining on the stack is infix expression
- E.g. a b c - + d e – f g – h + / *

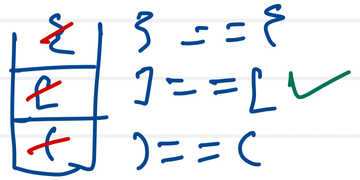
Valid Parantheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:
- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.
- Every close bracket has a corresponding open bracket of the same type.

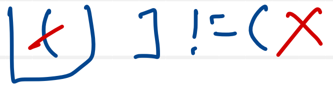
Example 1:
	Input: s = "()"
	Output: true



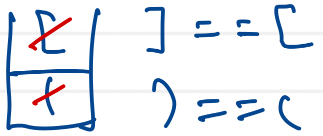
Example 2:
	Input: s = "()[]{}"
	Output: true



Example 3:
	Input: s = "(]"
	Output: false



Example 4:
	Input: s = "([])"
	Output: true



```
boolean isValid(String s) {
  Stack<Character> st = new Stack<>();
  for(i=0; i< s.length(); i++) {
      char ele = s.charAt(i);
     if(ele == '(' || ele == '[' || ele == '{')
        st.push(ele);
    else if(ele == ')' && !st.isEmpty && st.peek() == '(')
        st.pop();
    else if(ele == ']' && !st.isEmpty && st.peek() == '[')
        st.pop();
    else if(ele == '}' && !st.isEmpty && st.peek() == '{')
        st.pop();
   else
       return false;
  }
  if(!st.isEmpty())
     return false;
  return true;
}
```

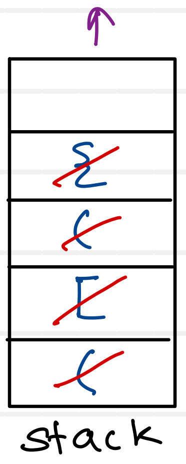
5+([9-4]*(8-{6/2}))

] == [

} == {

) == (

) == (

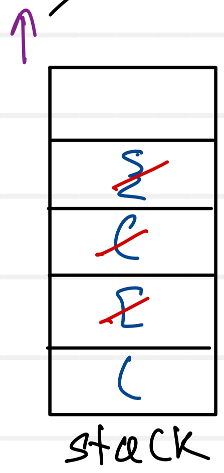


5+([9-4]*(8-{6/2}])

] == [

} == {

] != (



opening	(	[	{
	0	1	2
closing	)	]	}
	0	1	2

string
↓
indexOf( )
↓
returns index of char
returns -1 if char not found

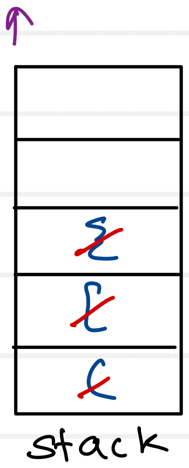
5+([9-4]*8-{6/2}))

] == [

} == {

) == (

) == ?

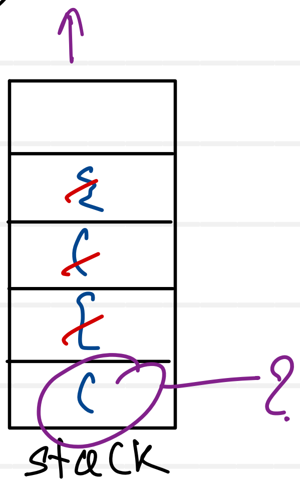


5+([9-4]*(8-{6/2})

] == [

} == {

) == (



SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com