Today's Content

1. Infix to Postfix
2. Evaluate Postfix

Expression Evaluation:

We write expression in infix:

infix: operator between operands

Issue is system cannot Evaluate Infix, so it converts infix → postfix & evaluate postfix expression.

postfix: operator after operands.

Q: Given Infix Expression

1. Convert to postfix
2. Evaluate postfix.

Infix Expression	Postfix Expression		
$a + b$	$ab +$		
$a - b$	$ab -$		
a / b	$ab /$		
$a * b$	$ab *$		
$a + \underbrace{b * c}$	$\underbrace{a} + \underbrace{bc*}$	$a \; bc* \; +$	
$4 + 3 * 3 - 2$	$4 + 33* - 2$	$\underbrace{4 \; 33* +} - \underbrace{2}$	$4 \; 33* + \; 2 \; -$
$6 + 3 * (\underbrace{3 + 2}) - 5$	$6 + \underbrace{3} * \underbrace{32+} - 5$	$\underbrace{6} + \underbrace{3 \; 32+ *} - 5$	$6 \; 3 \; 32+ * + 5 \; -$

#obs: In postfix no ()

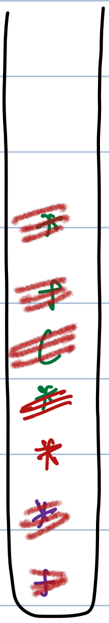
Given an Infix, convert to Postfix Expression.

#Note: Infix expression is vector of strings, where each string can be operand/ operator/ bracket.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Infix:	A	+	B	*	C	-	D	*	(	F	+	G	*	K	)

Postfix: A B C * + D F G K * + * -

#Idea: Infix ⟶ Postfix TC: O(N) SC: O(N)



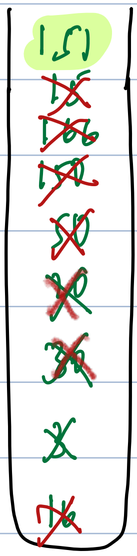
```
Iterate on Infix:
    if Infix[i] is operand: Postfix.push_back(Infix[i])
    if Infix[i] is open bracket: st.push(Infix[i])
    if Infix[i] is closing bracket:
        while( st.top() != open bracket) {
            Postfix.push_back( st.top());
            st.pop();
        }
        st.pop();
    if Infix[i] is operator,
        while( st.size() > 0 && st.top() != open bracket &&
                                pred(st.top()) >= pred(Infix[i])){
            Postfix.push_back( st.top());
            st.pop();
        }
        st.push(Infix[i])
}

while( st.size() > 0) {
    Postfix.push_back( st.top());
    st.pop();
}
```

Evaluate Post Expression:

#Note: Post expression is vector of strings, where each string can be operand/operator/bracket.

	0	1	2	3	4	5	6	7	8	9	10
Infix :	16	+	3	*	(	30	+	20	)	-	15;

	0	1	2	3	4	5	6	7	8
postfix:	16	3	30	20	+	*	+	15	-
	→	→	→	→	→	→	→	→	→



	A	operator	B		
#Note:	2nd pop()	$\textcircled{+}$	1st pop()		
operator :	30	+	20	= 50	push st
operator :	3	*	50	= 150	push st
operator :	16	+	150	= 166	push st
operator :	166	-	15	= 151	push st

Return top of stack = 151.

#Idea: Evaluate Postfix Expression. TC: O(N) SC: O(N)

```
stack<int> st;
Iterate on Postfix:
    if Postfix[i] is operand : st.push( stoi( postfix[i] ) );
    if Postfix[i] is operator:
        int B = st.top(); st.pop();
        int A = st.top(); st.pop();
        st.push(A operator B)
    }
}
return st.top();
```