STACK

By Er. Kushal Ghimire

Stack

- Non Primitive and Linear Data Structure
- LIFO [Last In First Out] principle
- Can be implemented using Array or Linked List
- A variable called TOP is used to keep track of the top of the stack.
- Stack Overflow = When we try to push a data on to the stack but the stack is already full.
- Stack Underflow = When we try to pop a data from the stack but the stack is already empty.

Algorithm for PUSH

- Check if the stack is full as **top == MAX_SIZE - 1**
- If the stack is full then display the appropriate message such as “Stack Overflow” and terminate the program
  - Stack Overflow
  - exit(0)
- Else
  - Increment the top variable as **top = top + 1**
  - Push the element to the top of the stack as stack[top] = data
- Terminate the program

Algorithm for POP

- Check if the stack is empty as **top < 0**
- If the stack is empty then display the appropriate message such as “Stack Underflow” and terminate the program
  - Stack Underflow
  - exit(0)
- Else
  - Pop the element of the top of the stack as data = stack[top]
  - Decrement the top variable as **top = top - 1**
- Terminate the program

Algorithm for PEEK

- Check if the stack is empty as **top < 0**
- If the stack is empty then display the appropriate message such as “Stack Underflow” and terminate the program
  - Stack Underflow / Stack Empty
  - exit(0)
- Else
  - Display the element of the top of the stack as **data = stack[top]**
- Terminate the program

Applications of stack

- Computer internally uses stack when we implement a recursive function.
- Internet Browser uses stack to keep track of the history of websites you visited.
- Calculation of postfix expression is done using stack.

Precedence

Operators	Symbols	Priority / Precedence
Exponent	^, $	Highest
Division and Multiplication	/, *	Second Highest
Addition and Subtraction	+ , -	Lowest

Algorithm for Infix to Postfix Expression Conversion

- Scan the character from left to right
- If the scanned character is ‘**(**’
  - PUSH TO THE STACK
- If the scanned character is an OPERAND
  - ADD TO THE POSTFIX STRING
- If the scanned character is an OPERATOR
  - Check Precedence
    - If precedence of the scanned character is greater than to the precedence of the operator in the stack then PUSH the operator to the stack
    - Else POP the operator from the stack and add the popped operator to the postfix string.Then, PUSH the scanned operator onto the stack.
- If the scanned character is ‘**)**’
  - POP ALL THE OPERATOR INSIDE PARENTHESIS “()” AND ADD IT TO THE POSTFIX STRING

Input Expression	Stack	Postfix Expression
K		K
+	+	
L	+	K L
-	-	K L+
M	-	K L+ M
*	- *	K L+ M
N	- *	K L + M N
+	+	K L + M N* K L + M N* -
(	+ (	K L + M N *-
O	+ (	K L + M N * - O
^	+ (^	K L + M N* - O
P	+ (^	K L + M N* - O P
)	+	K L + M N* - O P ^
*	+ *	K L + M N* - O P ^
W	+ *	K L + M N* - O P ^ W
/	+ /	K L + M N* - O P ^ W *
U	+ /	K L + M N* - O P ^W*U
/	+ /	K L + M N* - O P ^W*U/
V	+ /	KL + MN*-OP^W*U/V
*	+ *	KL+MN*-OP^W*U/V/
T	+ *	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T* KL+MN*-OP^W*U/V/T*+
Q	+	KL+MN*-OP^W*U/V/T*Q
		KL+MN*-OP^W*U/V/T*+Q+

Algorithm for Infix to Prefix Expression Conversion

- Scan the character from right to left.
- If the scanned character is ‘**(**’
  - PUSH TO THE STACK
- If the scanned character is an OPERAND
  - ADD TO THE POSTFIX STRING
- If the scanned character is an OPERATOR
  - Check Precedence
    - If precedence of the scanned character is greater than to the precedence of the operator in the stack then PUSH the operator to the stack
    - Else POP the operator from the stack and add the popped operator to the postfix string.Then, PUSH the scanned operator onto the stack.
- If the scanned character is ‘**)**’
  - POP ALL THE OPERATOR INSIDE PARENTHESIS “()” AND ADD IT TO THE POSTFIX STRING

Finally, Reverse the result postfix to obtain prefix expression.

Infix to prefix : K + L - M * N + (O^P) * W/U/V * T + Q

Input expression	Stack	Prefix expression
Q		Q
+	+	Q
T	+	QT
*	+*	QT
V	+*	QTV
/	+*/	QTV
U	+*/	QTVU
/	+*//	QTVU
W	+*//	QTVUW
*	+*//*	QTVUW
)	+*//*)	QTVUW
P	+*//*)	QTVUWP
^	+*//*)^	QTVUWP
O	+*//*)^	QTVUWPO
(	+*//*	QTVUWPO^
+	++	QTVUWPO^*//*
N	++	QTVUWPO^*//*N
*	++*	QTVUWPO^*//*N
M	++*	QTVUWPO^*//*NM
-	++-	QTVUWPO^*//*NM*
L	++-	QTVUWPO^*//*NM*L
+	++-+	QTVUWPO^*//*NM*L
K	++-+	QTVUWPO^*//*NM*LK
		QTVUWPO^*//*NM*LK+-++

Thus, The reverse of QTVUWPO^*//*NM*LK+-++ is **++-+KL*MN*//*^OPWUVTQ** which is the required prefix expression.

Evaluation of postfix expression

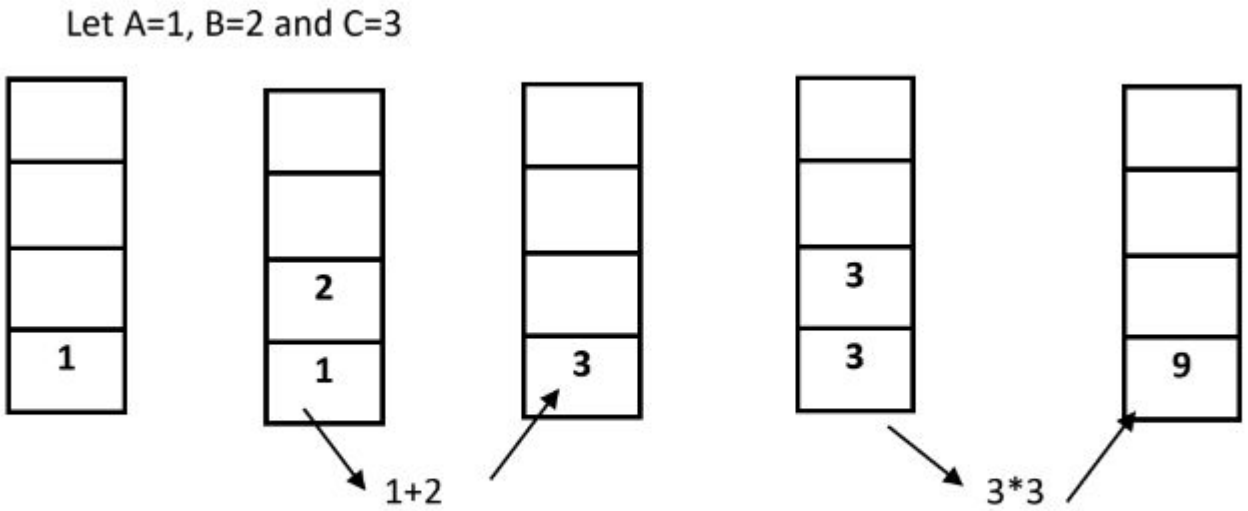
Algorithm to evaluate the postfix expression:

1. Start
2. If character is number

   Push on the stack

3. If character is operator(op)
   a. Val1=pop
   b. Val2=pop
   c. Perform result=val2 op val1
   d. Push the result into stack
4. Repeat 2 to 3 until postfix expression is scanned.
5. Output the result
6. Stop

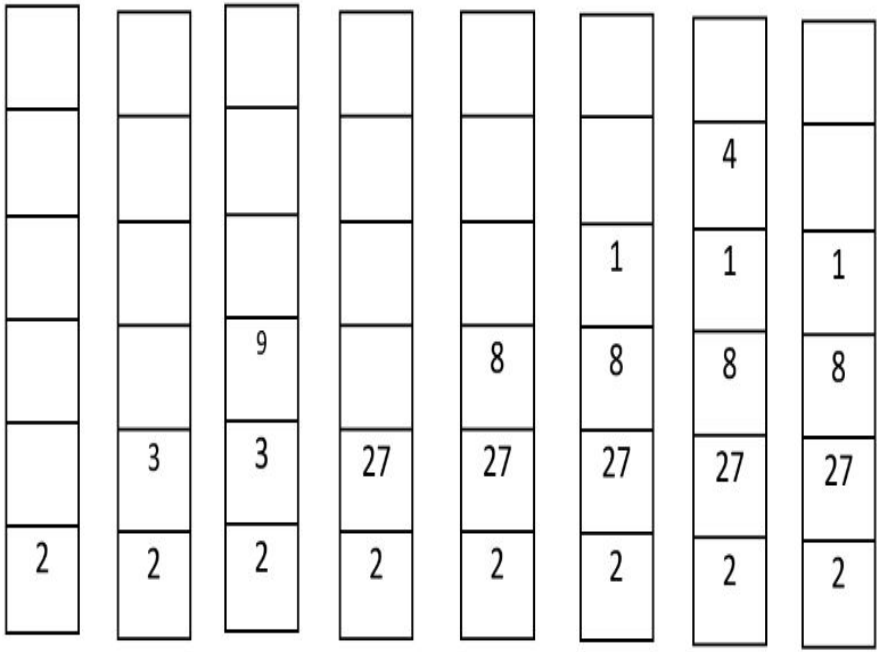
Evaluate AB+C*

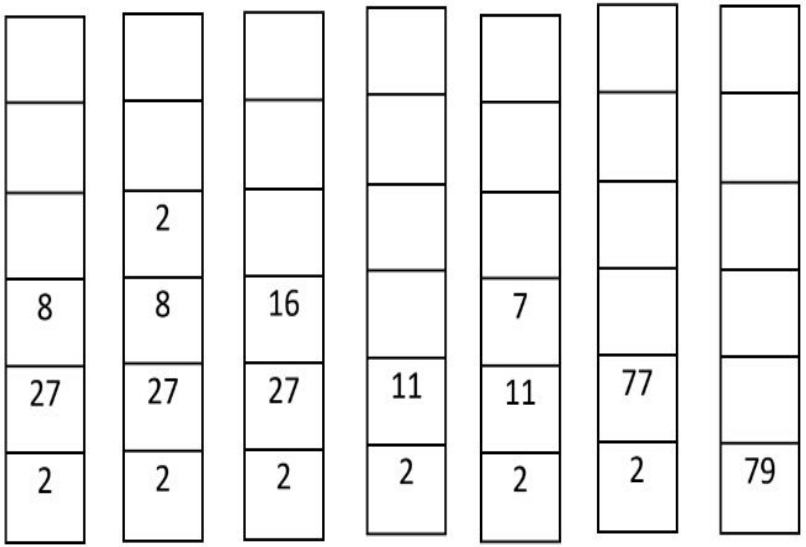


Ans: 9

Evaluate ABC*DEF^/G*-H*+

Let A= 2, B= 3, C=9, D=8, E=1, F=4, G= 2, H=7





Ans: 79