

Prefix, Postfix, Infix Notation

CSE 207

Infix Notation

To add A, B, we write

$$A+B$$

To multiply A, B, we write

$$A*B$$

The operators ('+' and '*') go in between the operands ('A' and 'B')

This is "*Infix*" notation.

Prefix Notation

Instead of saying "A plus B", we could say "add A,B " and write

+ A B

"Multiply A,B" would be written

* A B

This is *Prefix* notation.

Postfix Notation

Another alternative is to put the operators after the operands as in

$A B +$

and

$A B *$

This is *Postfix* notation.

Pre A In B Post

The terms
go between

operators

Parentheses

Evaluate $2+3*5$.

+ First:

$$(2+3)*5 = 5*5 = 25$$

* First:

$$2+(3*5) = 2+15 = 17$$

Infix notation requires Parentheses.

What about Prefix Notation?

$$+ 2 * 3 5 =$$

$$= + 2 \underline{* 3 5}$$

$$= \underline{+ 2 15} = 17$$

$$* + 2 3 5 =$$

$$= * \underline{+ 2 3 5}$$

$$= \underline{* 5 5} = 25$$

No parentheses needed!

Postfix Notation

$$2\ 3\ 5\ *\ + =$$

$$= 2\ \underline{3\ 5\ *} +$$

$$= \underline{2\ 15} + = 17$$

$$2\ 3 + 5\ * =$$

$$= \underline{2\ 3 +} 5\ *$$

$$= \underline{5\ 5\ *} = 25$$

No parentheses needed here either!

Conclusion:

Infix is the only notation that requires parentheses in order to change the order in which the operations are done.

Fully Parenthesized Expression

A FPE has exactly one set of Parentheses enclosing each operator and its operands.

Which is fully parenthesized?

$$(A + B) * C$$

$$((A + B) * C)$$

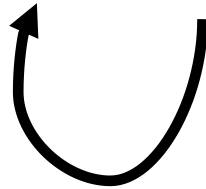
$$((A + B) * (C))$$



Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

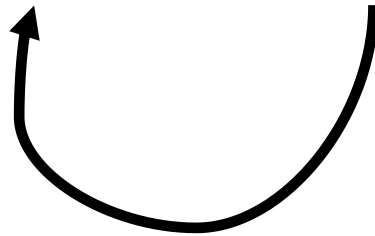
$$((A + B) * (C + D))$$



Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

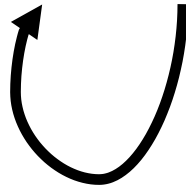
(+ A B * (C + D))



Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B (C + D)



Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B + C D

Order of operands does not change!

Infix to Postfix

$(((A + B) * C) - ((D + E) / F))$

A B + C * D E + F / -

Operand order does not change!

Operators are in order of evaluation!

Computer Algorithm

FPE Infix To Postfix

Assumptions:

1. Space delimited list of tokens represents a FPE infix expression
2. Operands are single characters.
3. Operators $+$, $-$, $*$, $/$

FPE Infix To Postfix

Initialize a Stack for operators, output list

Split the input into a list of tokens.

for each token (left to right):

- if it is operand: append to output

- if it is '(': push onto Stack

- if it is ')': pop & append till '('

FPE Infix to Postfix

$(((A + B) * (C - E)) / (F + G))$

stack: <empty>

output: []

FPE Infix to Postfix

$((A + B) * (C - E)) / (F + G)$

stack: (

output: []

FPE Infix to Postfix

$(A + B) * (C - E) / (F + G)$

stack: ((

output: []

FPE Infix to Postfix

$A + B) * (C - E)) / (F + G))$

stack: (((

output: []

FPE Infix to Postfix

+ B) * (C - E)) / (F + G))

stack: (((

output: [A]

FPE Infix to Postfix

$B) * (C - E)) / (F + G))$

stack: (((+

output: [A]

FPE Infix to Postfix

) * (C - E)) / (F + G))

stack: (((+

output: [A B]

FPE Infix to Postfix

* (C - E)) / (F + G))

stack: ((

output: [A B +]

FPE Infix to Postfix

$(C - E) / (F + G)$

stack: ((*

output: [A B +]

FPE Infix to Postfix

$C - E)) / (F + G))$

stack: ((* (

output: [A B +]

FPE Infix to Postfix

- E)) / (F + G))

stack: ((* (

output: [A B + C]

FPE Infix to Postfix

E)) / (F + G))

stack: ((* (-

output: [A B + C]

FPE Infix to Postfix

)) / (F + G))

stack: ((* (-

output: [A B + C E]

FPE Infix to Postfix

) / (F + G))

stack: ((*

output: [A B + C E -]

FPE Infix to Postfix

$/(F + G))$

stack: (

output: [A B + C E - *]

FPE Infix to Postfix

(F + G))

stack: (/

output: [A B + C E - *]

FPE Infix to Postfix

F + G))

stack: (/ (

output: [A B + C E - *]

FPE Infix to Postfix

+ G))

stack: (/ (

output: [A B + C E - * F]

FPE Infix to Postfix

G))

stack: (/ (+

output: [A B + C E - * F]

FPE Infix to Postfix

))

stack: (/ (+

output: [A B + C E - * F G]

FPE Infix to Postfix

)

stack: (/

output: [A B + C E - * F G +]

FPE Infix to Postfix

stack: <empty>

output: [A B + C E - * F G + /]

Problem with FPE

Too many parentheses.

Establish precedence rules:

My Dear Aunt Sally

We can alter the previous program to use the precedence rules.

Infix to Postfix

Initialize a Stack for operators, output list

Split the input into a list of tokens.

for each token (left to right):

- if it is operand: append to output

- if it is '(': push onto Stack

- if it is ')': pop & append till '('

- if it in '+-*/':

 - while peek has precedence \geq it:

 - pop & append

 - push onto Stack

pop and append the rest of the Stack.

Conversion of Infix to Postfix

Example to Convert Infix to Postfix using stack

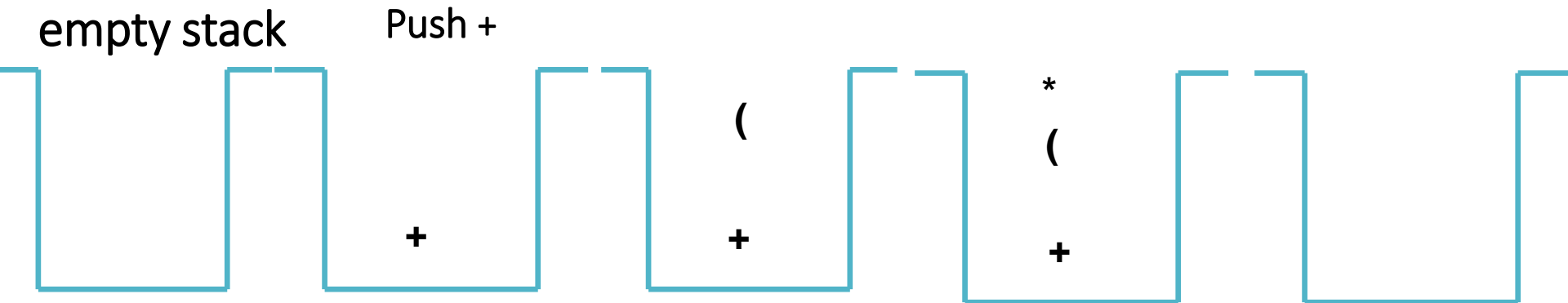
$$a + (b * c)$$

Read character	Stack	Output
a	Empty	a
+	+	a
(+(a
b	+(ab
*	+(*	ab
c	+(*	abc
)	+	abc*
		abc*+

Conversion of Infix to Postfix

Example how to Convert Infix to Postfix

Infix expression: $a + (b * c)$

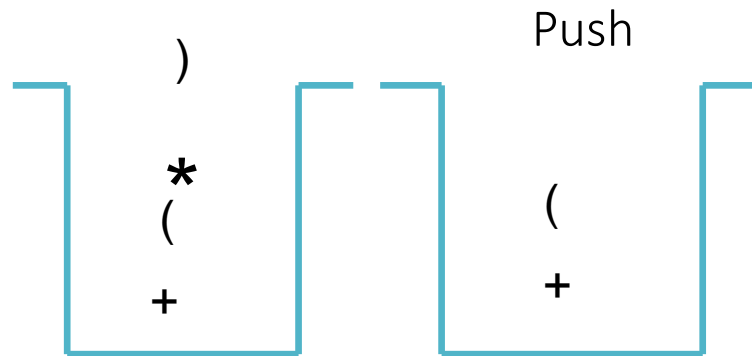


Postfix expression:

Conversion of Infix to Postfix

Example how to Convert Infix to Postfix







Infix expression: $a + (b * c)$



Postfix expression: $a b c$

Conversion of Infix to Postfix

TABLE 5.7Conversion of $x1 + 2.5 * \text{count} / 3$

Next Token	Action	Effect on operatorStack	Effect on postfix
x1	Append x1 to postfix.		x1
+	The stack is empty Push + onto the stack		x1
2.5	Append 2.5 to postfix		x1 2.5
*	precedence(*) > precedence(+), Push * onto the stack		x1 2.5
count	Append count to postfix		x1 2.5 count
/	precedence(/) equals precedence(*) Pop * off of stack and append to postfix		x1 2.5 count *

Conversion of Infix to Postfix

TABLE 5.7Conversion of $x1 + 2.5 * count / 3$ (continued)

Next Token	Action	Effect on operatorStack	Effect on postfix
/	precedence(/) > precedence(+), Push / onto the stack	<div style="border: 1px solid black; padding: 5px; display: inline-block;">/ +</div>	x1 2.5 count *
3	Append 3 to postfix	<div style="border: 1px solid black; padding: 5px; display: inline-block;">/ +</div>	x1 2.5 count * 3
End of input	Stack is not empty, Pop / off the stack and append to postfix	<div style="border: 1px solid black; padding: 5px; display: inline-block;">+</div>	x1 2.5 count * 3 /
End of input	Stack is not empty, Pop + off the stack and append to postfix	<div style="border: 1px solid black; padding: 5px; display: inline-block;"></div>	x1 2.5 count * 3 / +

Conversion of Infix to Postfix

Next Character	Postfix	Operator Stack (bottom to top)
<i>a</i>	<i>a</i>	
<i>/</i>	<i>a</i>	<i>/</i>
<i>b</i>	<i>a b</i>	<i>/</i>
<i>*</i>	<i>a b /</i>	
	<i>a b /</i>	<i>*</i>
<i>(</i>	<i>a b /</i>	<i>* (</i>
<i>c</i>	<i>a b / c</i>	<i>* (</i>
<i>+</i>	<i>a b / c</i>	<i>* (+</i>
<i>(</i>	<i>a b / c</i>	<i>* (+ (</i>
<i>d</i>	<i>a b / c d</i>	<i>* (+ (</i>
<i>−</i>	<i>a b / c d</i>	<i>* (+ (−</i>
<i>e</i>	<i>a b / c d e</i>	<i>* (+ (−</i>
<i>)</i>	<i>a b / c d e −</i>	<i>* (+ (</i>
	<i>a b / c d e −</i>	<i>* (+</i>
<i>)</i>	<i>a b / c d e − +</i>	<i>* (</i>
	<i>a b / c d e − +</i>	<i>*</i>
	<i>a b / c d e − + *</i>	

Exercise

Explain how to convert an infix expression to postfix expression?

Convert the following infix expression to postfix expression?

(a) $(a + b * c) / (c - d)$

(b) $a / (b + c) + d * (e - f)$

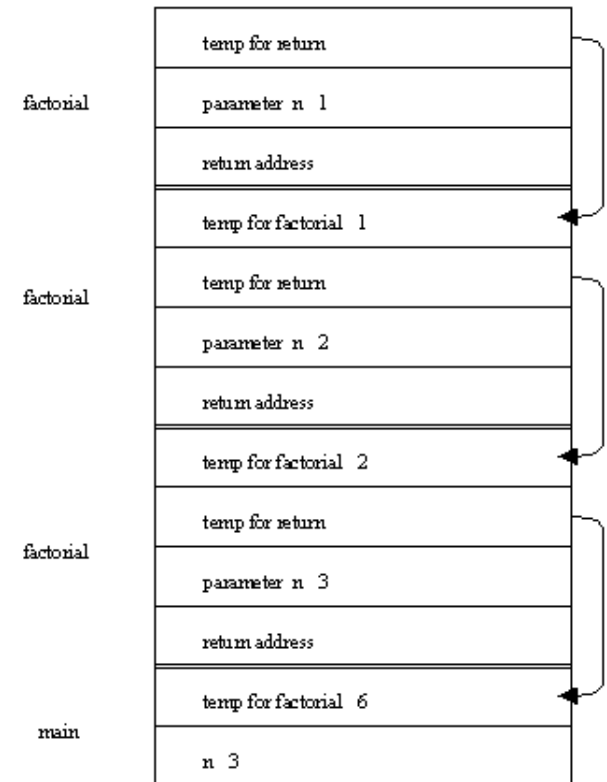
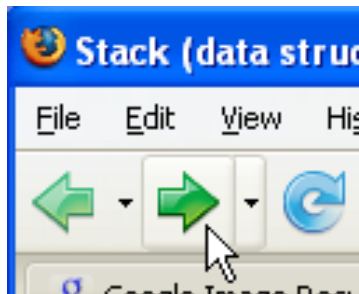
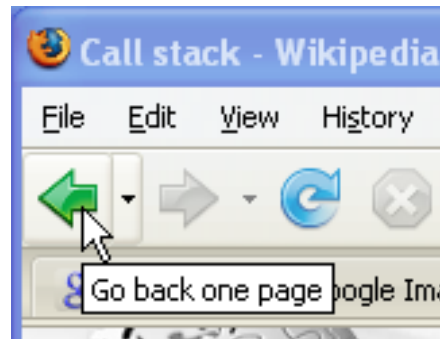
Explain an algorithm for converting to infix postfix?

Uses of Stacks

The runtime stack used by a process (running program) to keep track of methods in progress

Search problems

Undo, redo, back, forward



Simple Example

Infix Expression: $3 + 2 * 4$

PostFix Expression:

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression: $+ 2 * 4$

PostFix Expression: 3

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression: $2 * 4$

PostFix Expression: 3

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression: * 4

PostFix Expression: 3 2

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression: 4

PostFix Expression: 3 2

Operator Stack: + *

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression:

PostFix Expression: 3 2 4

Operator Stack: + *

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression:

PostFix Expression: 3 2 4 *

Operator Stack: +

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Simple Example

Infix Expression:

PostFix Expression: 3 2 4 * +

Operator Stack:

Precedence Table

Symbol	Off Stack Precedence	On Stack Precedence	
+	1	1	
-	1	1	
*	2	2	
/	2	2	
^	10	9	
(20	0	

Algorithm for Infix to Postfix

- 1) Examine the next element in the input.
- 2) If it is **operand**, output it.
- 3) If it is **opening parenthesis**, push it on stack.
- 4) If it is an **operator**, then
 - i) If stack is empty, push operator on stack.
 - ii) If the top of stack is opening parenthesis, push operator on stack
 - iii) If it has higher priority than the top of stack, push operator on stack.
 - iv) Else pop the operator from the stack and output it, repeat step 4
- 5) If it is a **closing parenthesis**, pop operators from stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
- 6) If there is **more input** go to step 1
- 7) If there is **no more input**, **pop** the remaining operators to output.

Evaluation a postfix expression

Each operator in a postfix string refers to the previous two operands in the string.

Suppose that each time we read an operand we push it into a stack. When we reach an operator, its operands will then be top two elements on the stack

We can then pop these two elements, perform the indicated operation on them, and push the result on the stack.

So that it will be available for use as an operand of the next operator.

Evaluating Postfix Notation

- Use a stack to evaluate an expression in postfix notation.
- The postfix expression to be evaluated is scanned from left to right.
- Variables or constants are pushed onto the stack.
- When an operator is encountered, the indicated action is performed using the top elements of the stack, and the result replaces the operands on the stack.

Evaluating a postfix expression

Initialise an empty stack

While token remain in the input stream

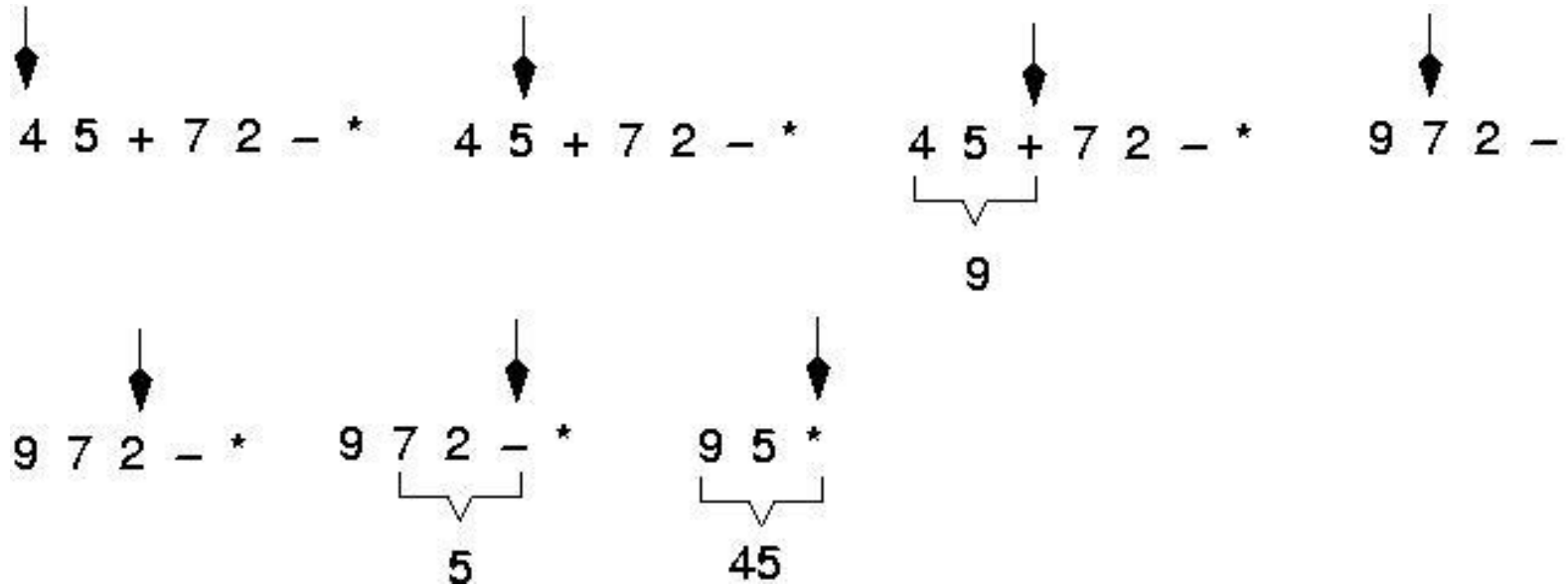
- Read next token

- If token is a number, push it into the stack

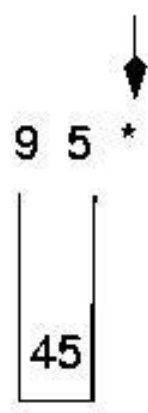
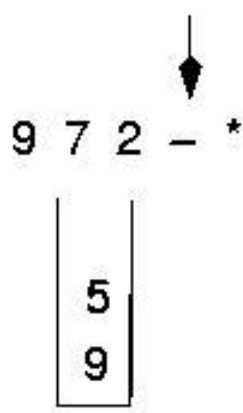
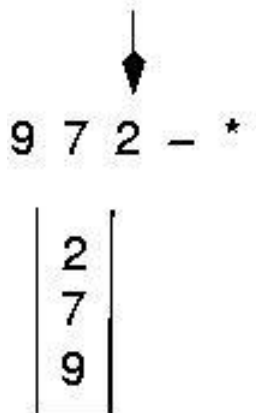
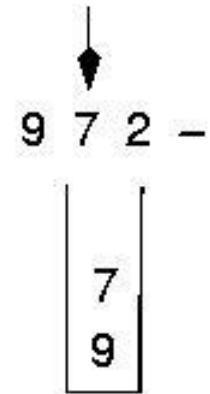
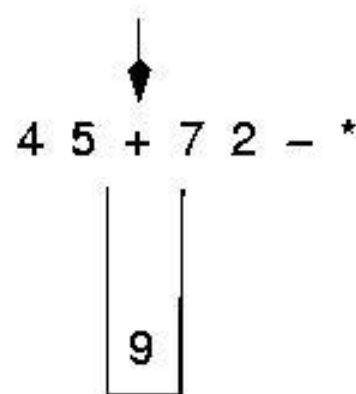
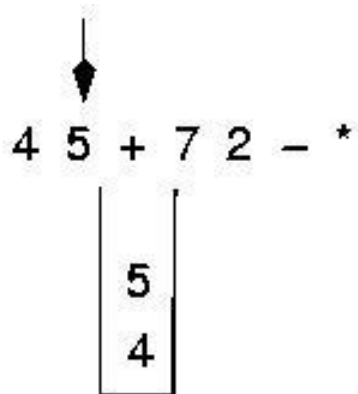
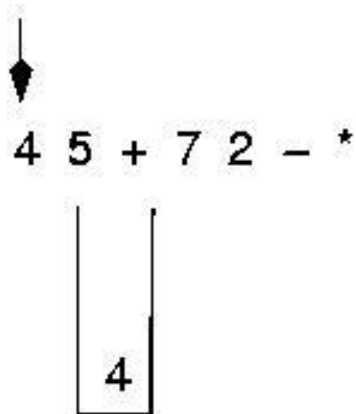
- Else, if token is an operator, pop top two tokens off the stack, apply the operator, and push the answer back into the stack

Pop the answer off the stack.

Example: postfix expressions (cont.)



Postfix expressions: Algorithm using stacks (cont.)



Algorithm for evaluating a postfix expression (Cond.)

WHILE more input items exist

{

 If *symb* is an operand

 then *push (opndstk,symb)*

 else *//symbol is an operator*

 {

Opnd1=pop(*opndstk*);

Opnd2=pop(*opndstk*);

Value = result of applying *symb* to *opnd1* & *opnd2*

Push(opndstk,value);

 }

//End of else

} // end while

Result = pop (opndstk);

Question : Evaluate the following expression in postfix :

623+-382/+*2^3+

Final answer is

- 49
- 51
- 52
- 7
- None of these

Evaluate- $623+-382/+*2^3+$

Symbol	opnd1		opnd2	value	opndstk
6					6
2					6,2
3					6,2,3
+		2	3	5	6,5
-	6	5	1	1	
3	6	5	1		1,3

Evaluate- 623+-382/+*2^3+

Symbol	opnd1	opnd2	value	opndstk
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
^	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52