Algebric Expressions Implementation using Stack -Part 2

Fakhera Nazir

Algorithm for Infix to Postfix

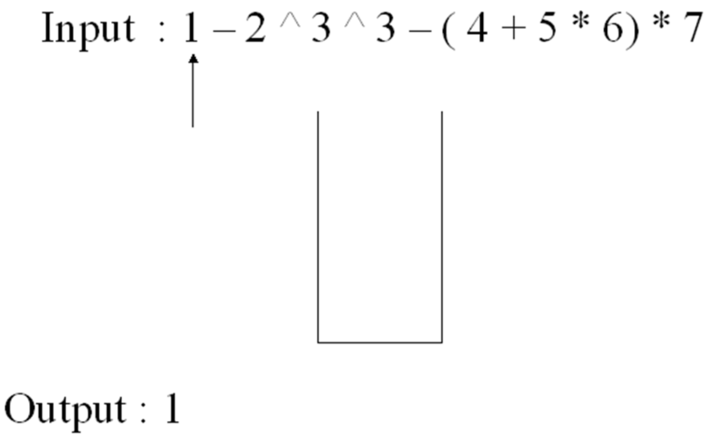
1) Examine the next element in the input.
2) If it is operand, output it.
3) If it is opening parenthesis, push it on stack.
4) If it is an operator, then
    - i) If stack is empty, push operator on stack.
    - ii) If the top of stack is opening parenthesis, push operator on stack
    - iii) If it has higher priority than the top of stack, push operator on stack.

      Else if its priority is less than or equal to stack's top, 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.

Application of Stacks – Infix to Postfix Conversion (cont’d)

```
while(stack is not empty){
        y = stack.pop( );
        Append y to postfixString;
}
```

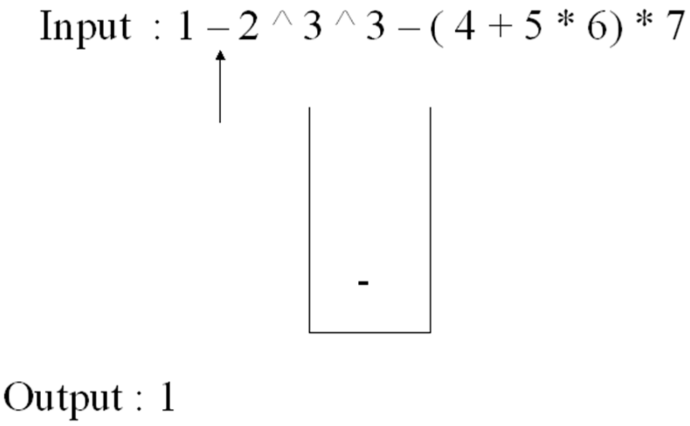
step1:



Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1

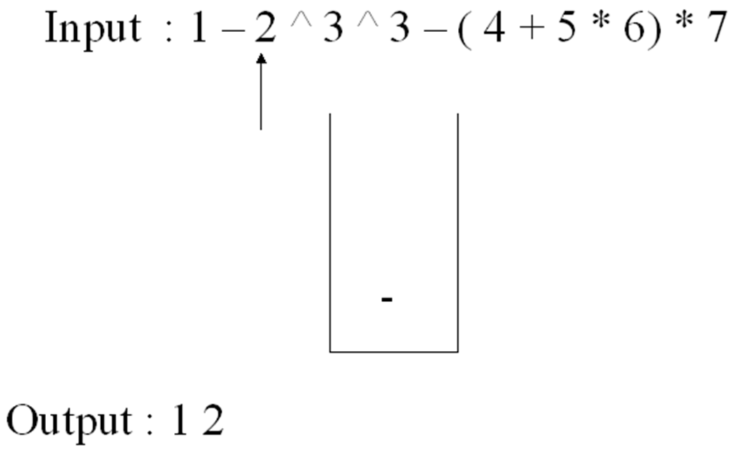
step2:



Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1

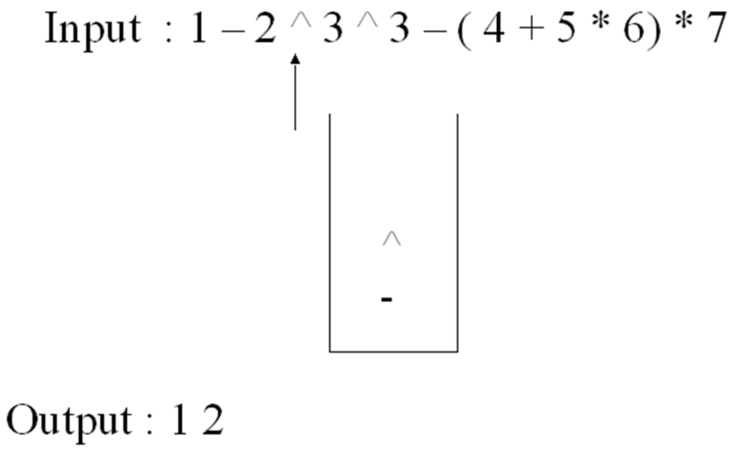
step3:



Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1 2

step4:



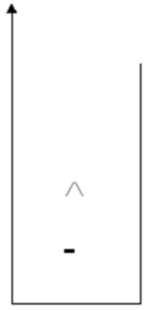
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1 2

Application of Stacks – Infix to Postfix Conversion (cont'd)

step5:

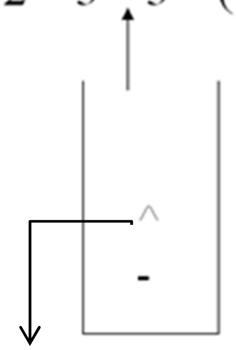
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3

step6:

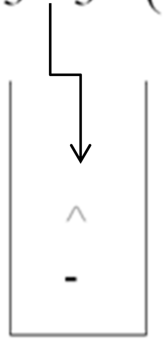
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3

step7:

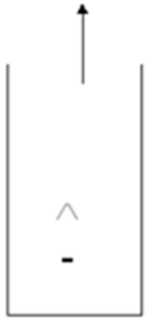
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^

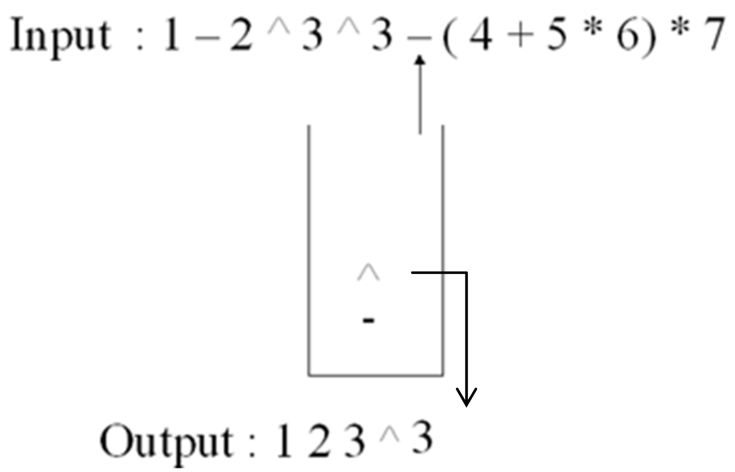
Step9:

Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

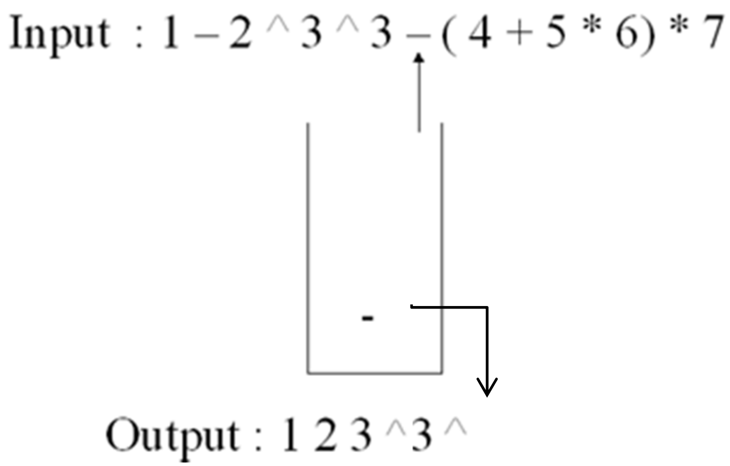


Output : 1 2 3 ^ 3

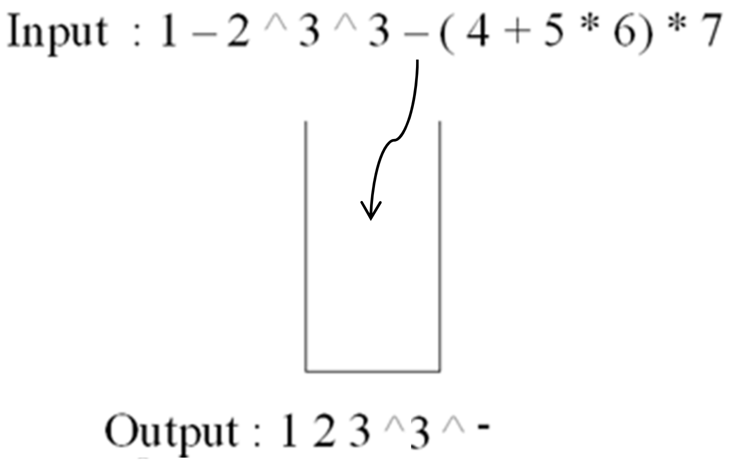
step8:


Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

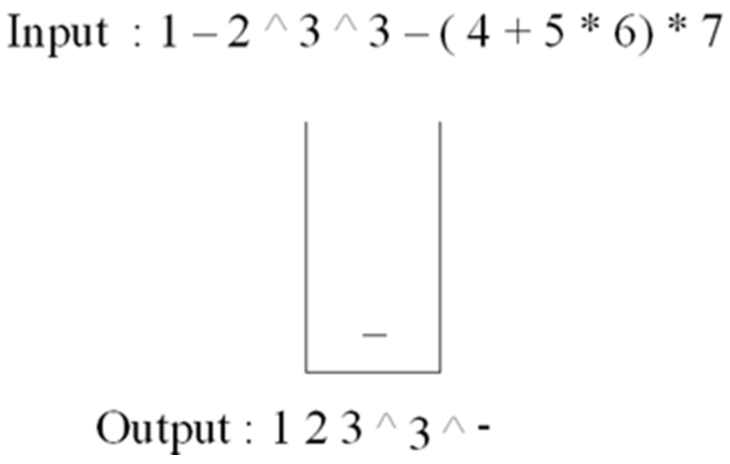
Output : 1 2 3 ^ 3


Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1 2 3 ^3 ^


Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1 2 3 ^3 ^ -

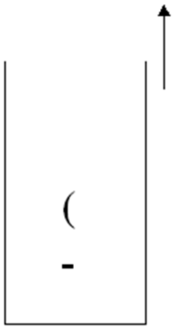

Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

Output : 1 2 3 ^ 3 ^ -

Application of Stacks – Infix to Postfix Conversion (cont’d)

step9:

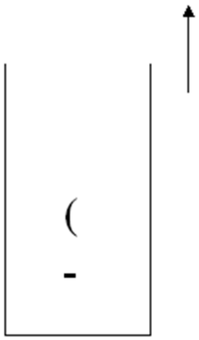
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^ 3 ^ -

step10:

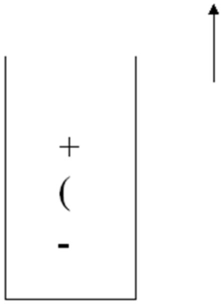
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^ 3 ^ - 4

step11:

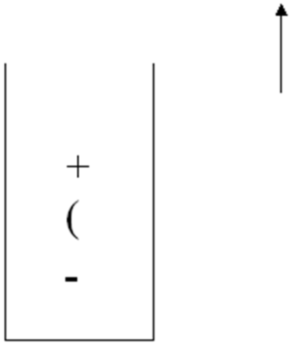
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^ 3 ^ - 4

step12:

Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

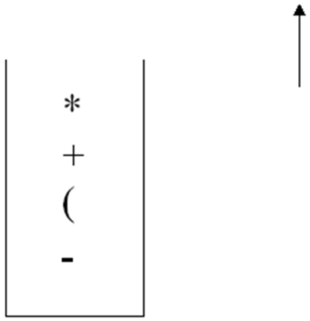


Output : 1 2 3 ^ 3 ^ - 4 5

Application of Stacks – Infix to Postfix Conversion (cont’d)

step13:

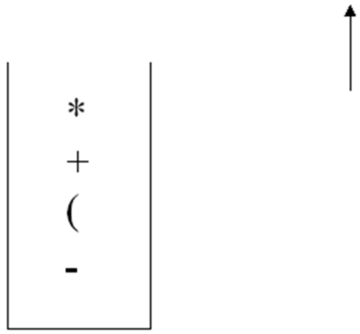
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^3 ^ - 4 5

step14:

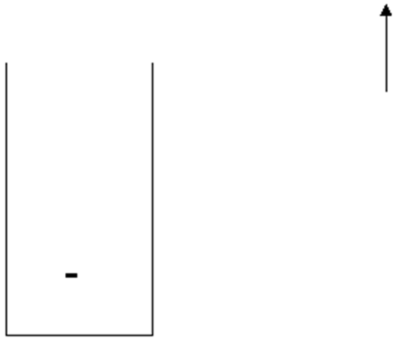
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^3 ^ - 4 5 6

step15:

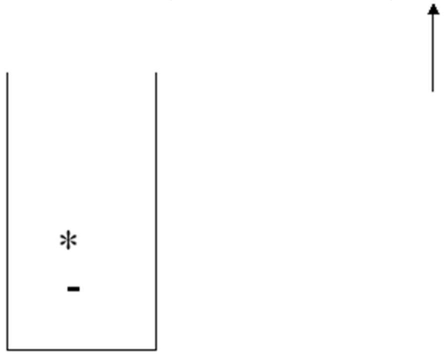
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^3 ^ - 4 5 6 * +

step16:

Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7

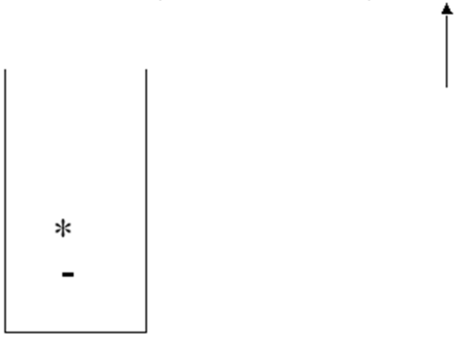


Output : 1 2 3 ^3 ^ - 4 5 6 * +

Application of Stacks – Infix to Postfix Conversion (cont’d)

step17:

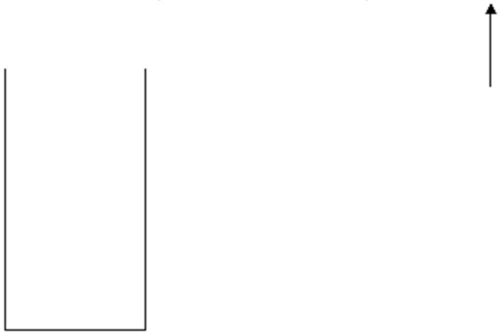
Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^ 3 ^ - 4 5 6 * + 7

step18:

Input : 1 – 2 ^ 3 ^ 3 – (4 + 5 * 6) * 7



Output : 1 2 3 ^ 3 ^ - 4 5 6 * + 7 * -

FPE Infix to Postfix

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

- stack: <empty>
- output: []

FPE Infix to Postfix

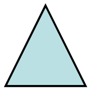
((A + B) * (C - E)) / (F + G))

- stack: (
- output: []

FPE Infix to Postfix

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

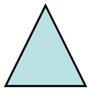
- stack: ((
- output: []



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 + /]

Suppose we want to convert 2*3/(2-1)+5*3 into Postfix form,

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

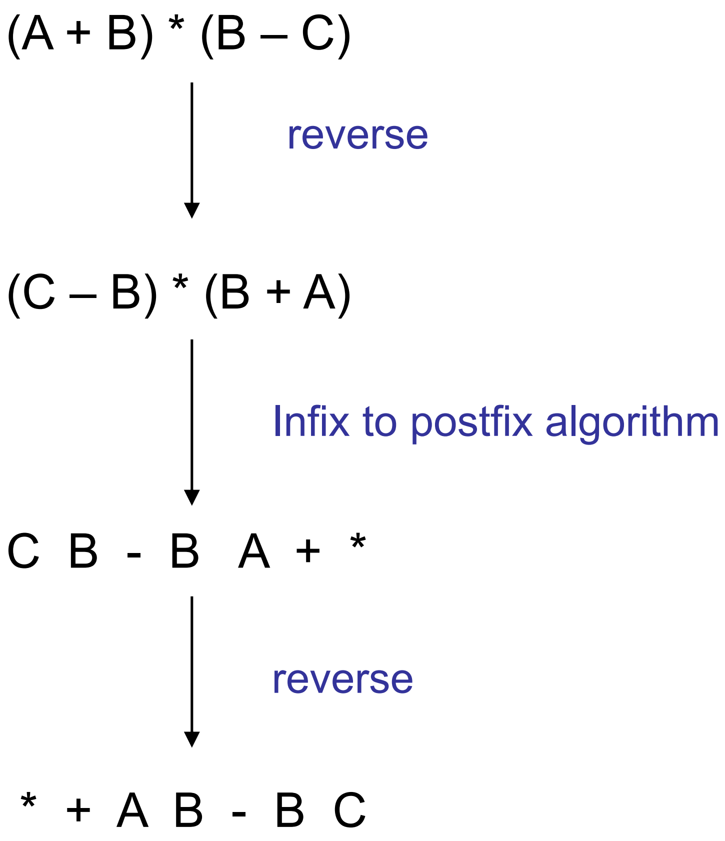
So, the Postfix Expression is 23*21-/53*+

Application of Stacks – Infix to Prefix Conversion

An infix to prefix conversion algorithm:

1. Reverse the infix string
2. Perform the infix to postfix algorithm on the reversed string
3. Reverse the output postfix string

Example:



(A + B) * (B – C)

reverse

(C – B) * (B + A)

Infix to postfix algorithm

C B - B A + *

reverse

* + A B - B C