Fundamentals of
Data Structures using C

Infix to Postfix Conversion

B.Bhuvaneswaran, AP (SG) / CSE

9791519152

bhuvaneswaran@rajalakshmi.edu.in

RAJALAKSHMI
ENGINEERING COLLEGE

Introduction

- To evaluate an arithmetic expression, first convert the given infix expression to postfix expression and then evaluate the postfix expression using stack.

Infix to Postfix Conversion

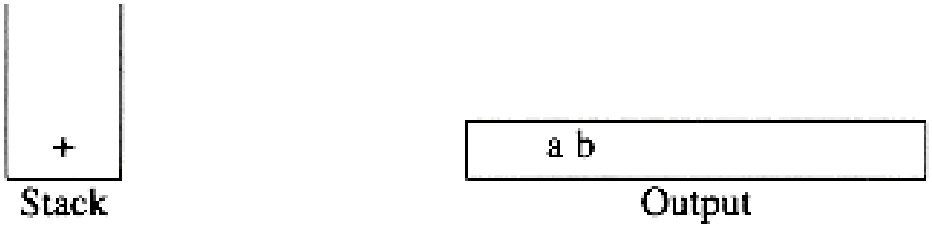
- Not only can a stack be used to evaluate a postfix expression, but we can also use a stack to convert an expression in standard form (otherwise known as infix) into postfix.

Infix to Postfix Conversion-Algorithm

- Make an empty stack.
- Read the infix expression one character at a time until it encounters end of expression.
- If the character is an operand, place it onto the output.
- If the character is an operator, push it onto the stack. If the stack operator has a higher or equal priority than input operator, then pop that operator from the stack and place it onto the output.
- If the character is a left parenthesis, push it onto the stack.
- If the character is a right parenthesis, pop all the operators from the stack till it encounters left parenthesis, discard both the parenthesis in the output.

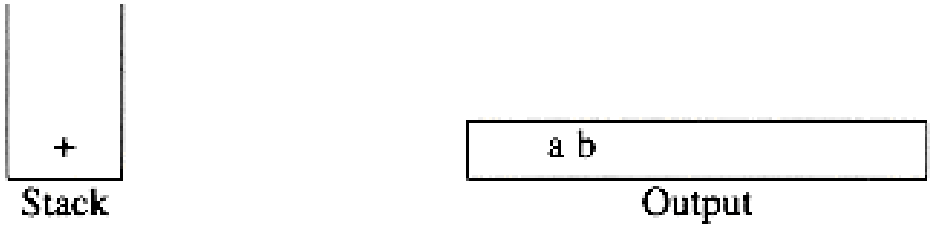
Example: a + b * c + (d * e + f) * g

- First, the symbol a is read, so it is passed through to the output.
- Then '+' is read and pushed onto the stack.
- Next b is read and passed through to the output.
- The state of affairs at this juncture is as follows:



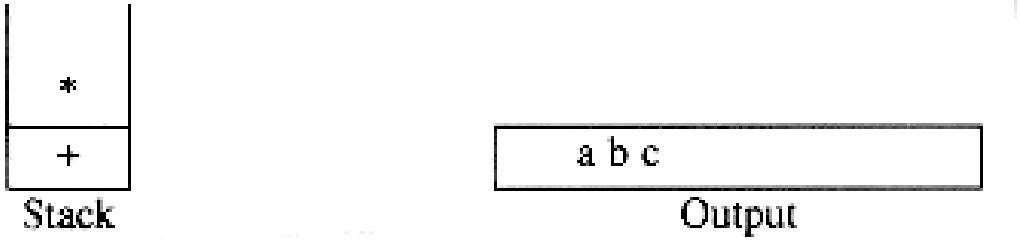
Example: a + b * c + (d * e + f) * g

- Next a '*' is read. The top entry on the operator stack has lower precedence than '*', so nothing is output and '*' is put on the stack.
- Next, c is read and output.
- Thus far, we have



Example: a + b * c + (d * e + f) * g

- Next a '*' is read. The top entry on the operator stack has lower precedence than '*', so nothing is output and '*' is put on the stack.
- Next, c is read and output.
- Thus far, we have

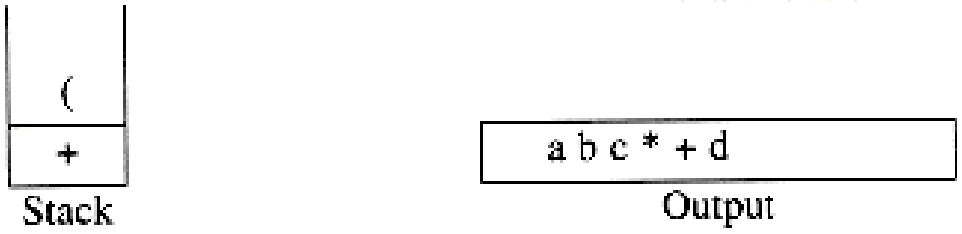


Example: a + b * c + (d * e + f) * g

- The next symbol is a '+'.
- Checking the stack, we find that we will pop a '*' and place it on the output, pop the other '+', which is not of lower but equal priority, on the stack, and then push the '+'.

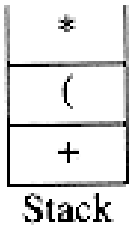
Example: a + b * c + (d * e + f) * g

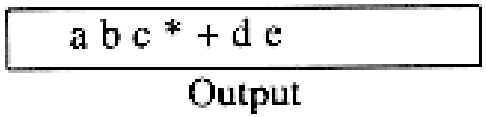
- The next symbol read is an '(', which, being of highest precedence, is placed on the stack.
- Then d is read and output.



Example: a + b * c + (d * e + f) * g

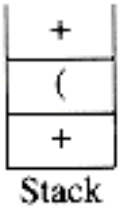
- We continue by reading a '*'.
- Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output.
- Next, e is read and output.

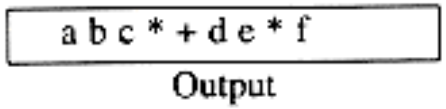




Example: a + b * c + (d * e + f) * g

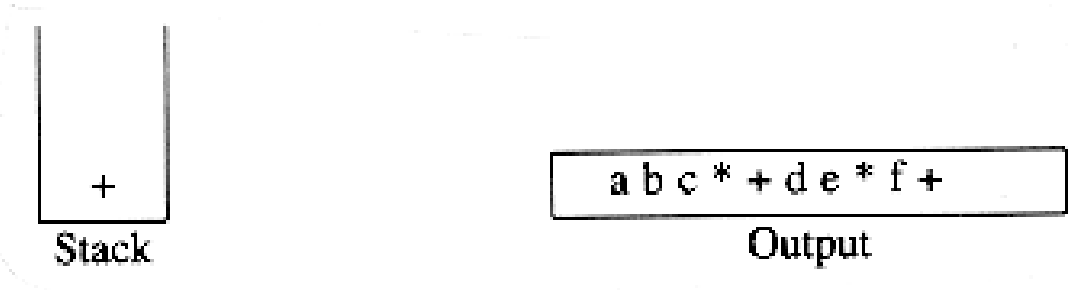
- The next symbol read is a '+'.
- We pop and output '*' and then push '+'.
- Then we read and output.





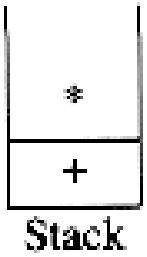
Example: a + b * c + (d * e + f) * g

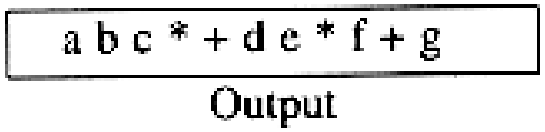
- Now we read a ')', so the stack is emptied back to the '('.
- We output a '+'.



Example: a + b * c + (d * e + f) * g

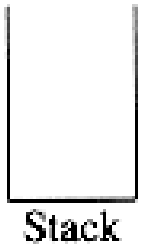
- We read a '*' next; it is pushed onto the stack.
- Then g is read and output.

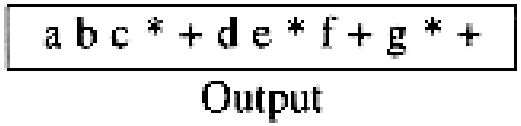




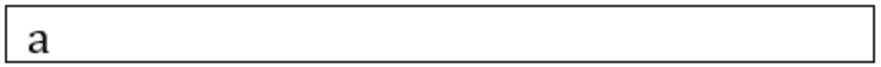
Example: a + b * c + (d * e + f) * g

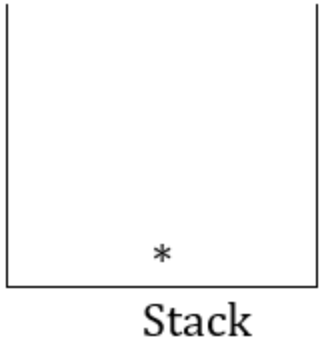
- The input is now empty, so we pop and output symbols from the stack until it is empty.

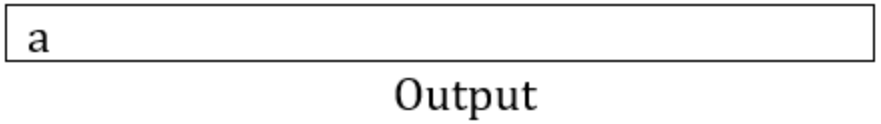




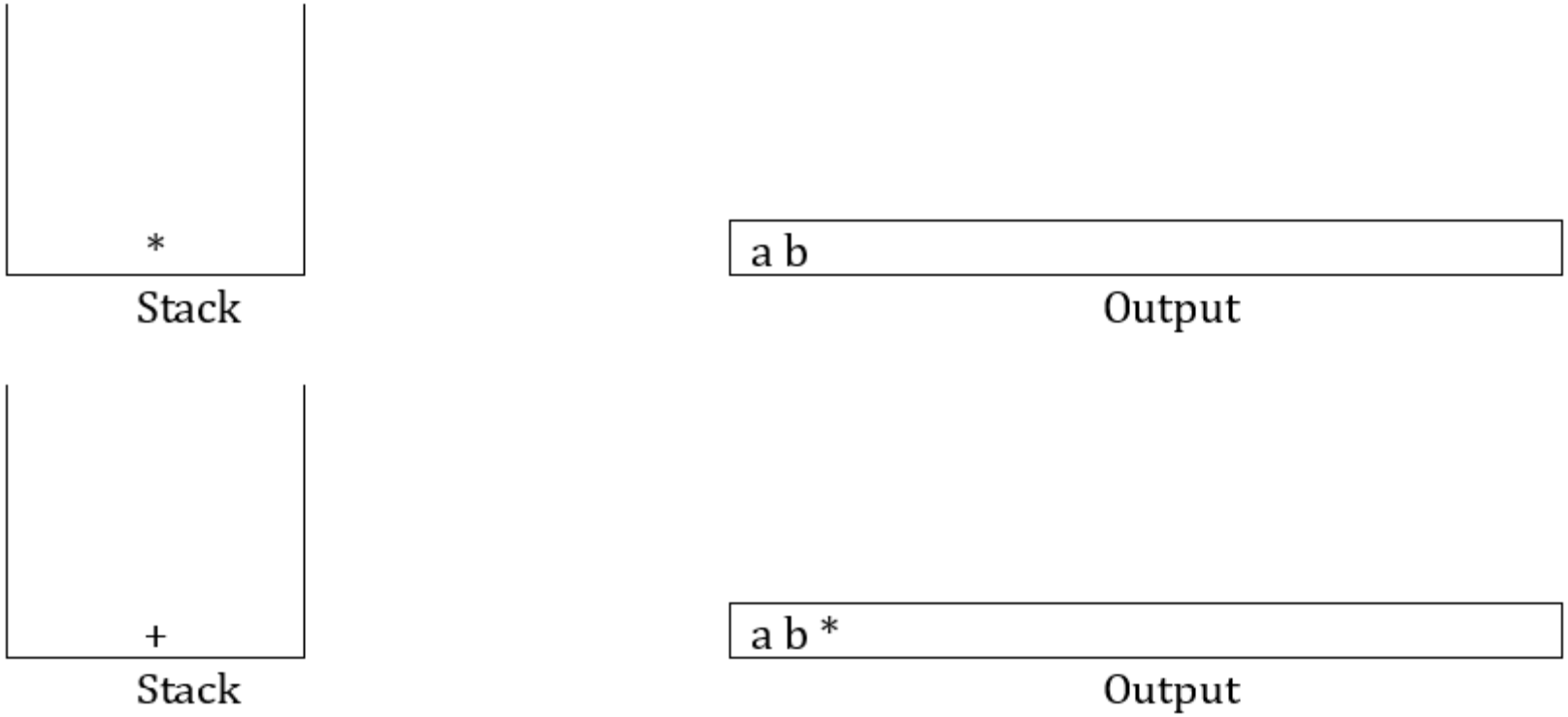
Example: a * b + (c - d / e)



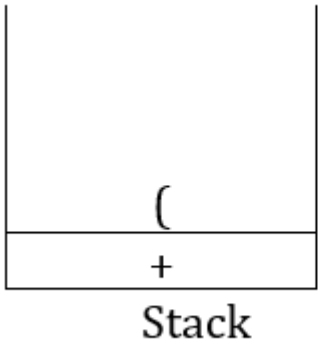




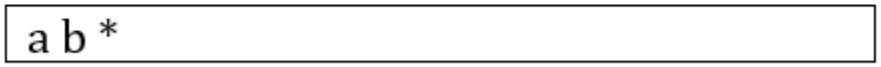
Example: a * b + (c - d / e)



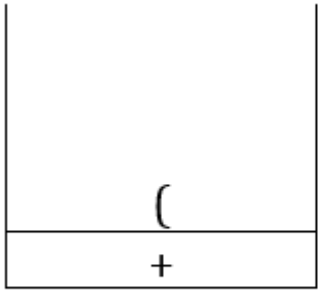
Example: a * b + (c - d / e)



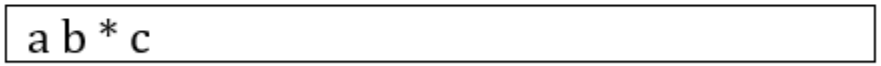
Stack



Output

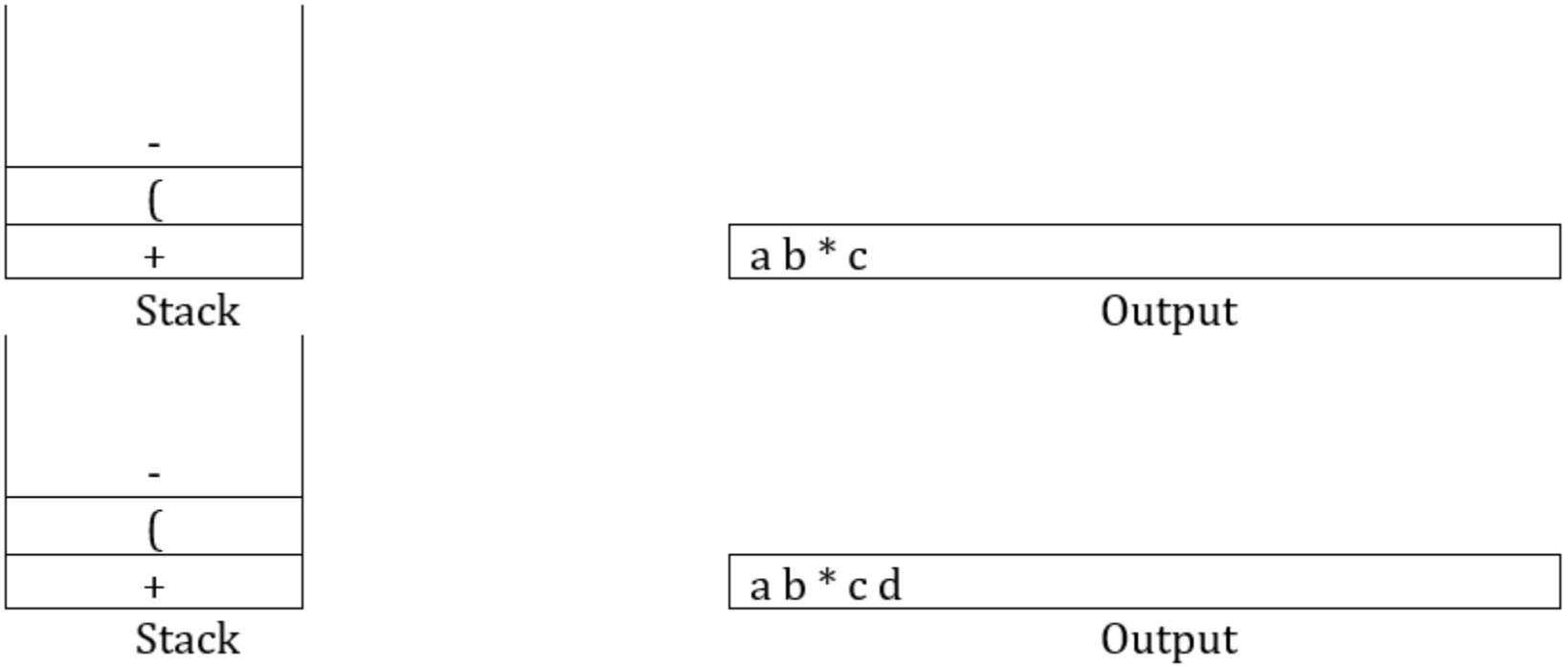


Stack

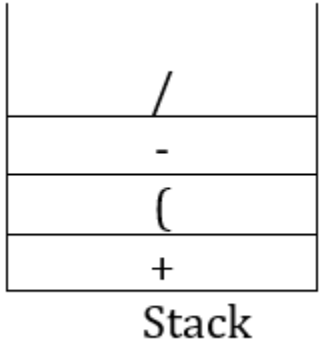


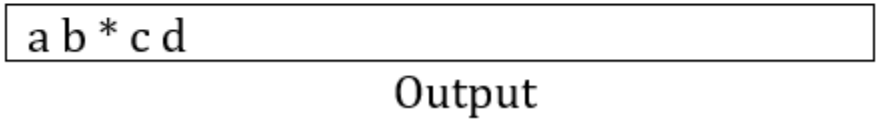
Output

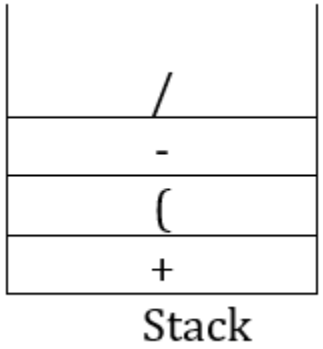
Example: a * b + (c - d / e)

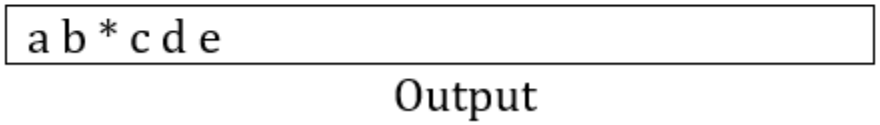


Example: a * b + (c - d / e)



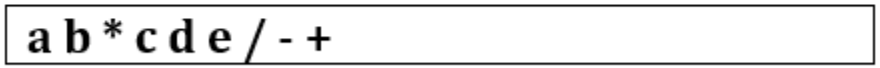






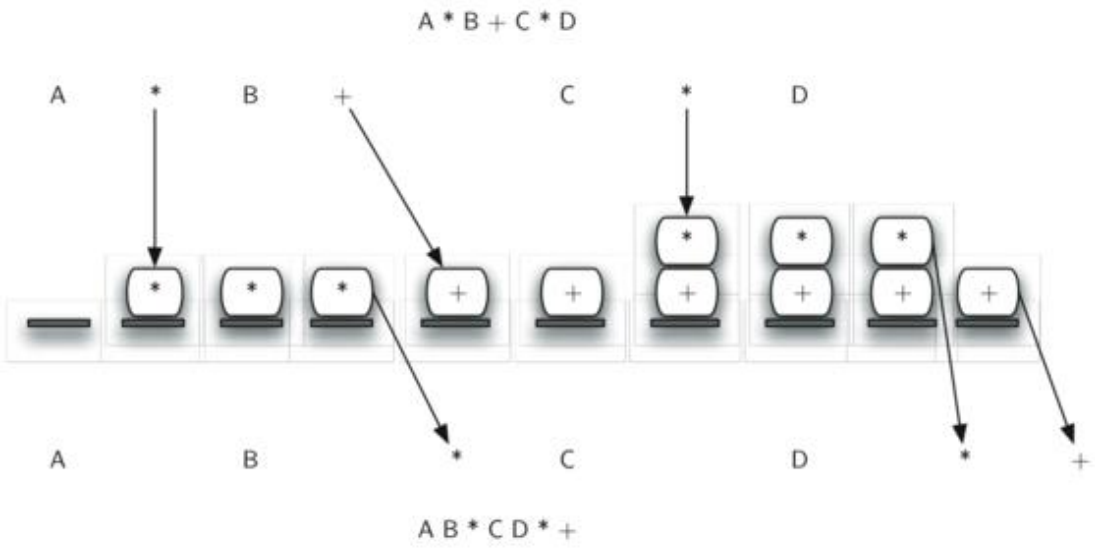
Example: a * b + (c - d / e)

Stack



Output

Example: A * B + C * D



Examples

Infix Expression	Prefix Expression	Postfix Expression
a + b	+ a b	a b +
a + b * c	+ a * b c	a b c * +
(a + b) * c	* + a b c	a b + c *
a + b * c + d	+ + a * b c d	a b c * + d +
(a + b) * (c + d)	* + a b + c d	a b + c d + *
a * b + c * d	+ * a b * c d	a b * c d * +
a + b + c + d	+ + + a b c d	a b + c + d +

Queries?

Thank You!