Data Structures – CST 201
Module - 2

Syllabus

- Searching
  - Linear Search
  - Binary Search
- Polynomial representation using Arrays
- Sparse matrix
- Stacks
- Queues
  - Circular Queues
  - Double Ended Queues
  - Priority Queues
- **Evaluation of Expressions**

APPLICATIONS OF STACK

1. Evaluation of arithmetic expressions
2. Implementation of Recursion
3. String reversal

APPLICATIONS OF STACK

1. **Evaluation of arithmetic expressions**
2. Implementation of Recursion
3. String reversal

Evaluation of arithmetic expressions

- An arithmetic expression consist of operands and operators
- Operands are variables or constants
- Operators are various types such as arithmetic unary and binary operators and Boolean operators
- Parenthesis are also used
- Eg1: a+b*c/d
- Eg2: (a+((b*c)/d))

Evaluation of arithmetic expressions

- Problem to evaluate these expression is the order of evaluation.
- We can assign each operator a precedence and associativity

Operator	Precedence	Associativity
+,- (unary)	1	
^	2	Right to left
*, /	3	Left to right
+, - (binary)	4	Left to right

EXPRESSIONS

- The way to write arithmetic expression is known as notation
- Arithmetic expressions can be written in 3 different ways
  - Infix notation
  - Postfix(Polish) notation
  - Prefix(Reverse-Polish) notation

Evaluation of arithmetic expressions

- Evaluation of expression is a two step process
  1. Convert given infix expression into postfix/prefix expression
  2. Evaluate the postfix/prefix expression

- In each procedure stack is the main data structure that we used to accomplish its task

INFIX NOTATION

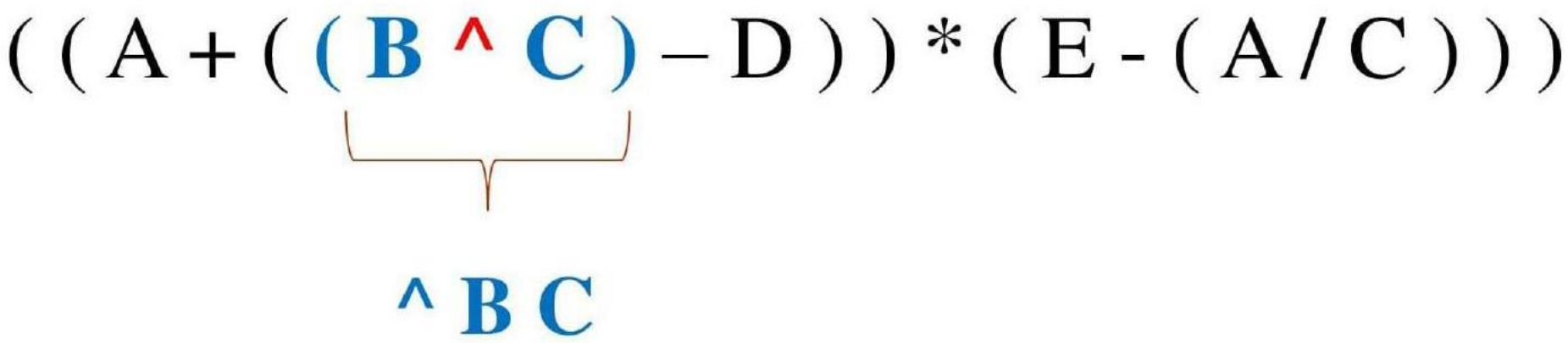
- Operators are written between the operands they act on.
- Ex: 5+2
- In infix notation, parentheses surrounding groups of operands and operators are necessary to indicate the intended order in which operations are to be performed.
- In the absence of parentheses, certain precedence rules determine the order of operations.

PREFIX NOTATION

- Operator precede the Operands
- Parentheses are not used
- Ex: +52

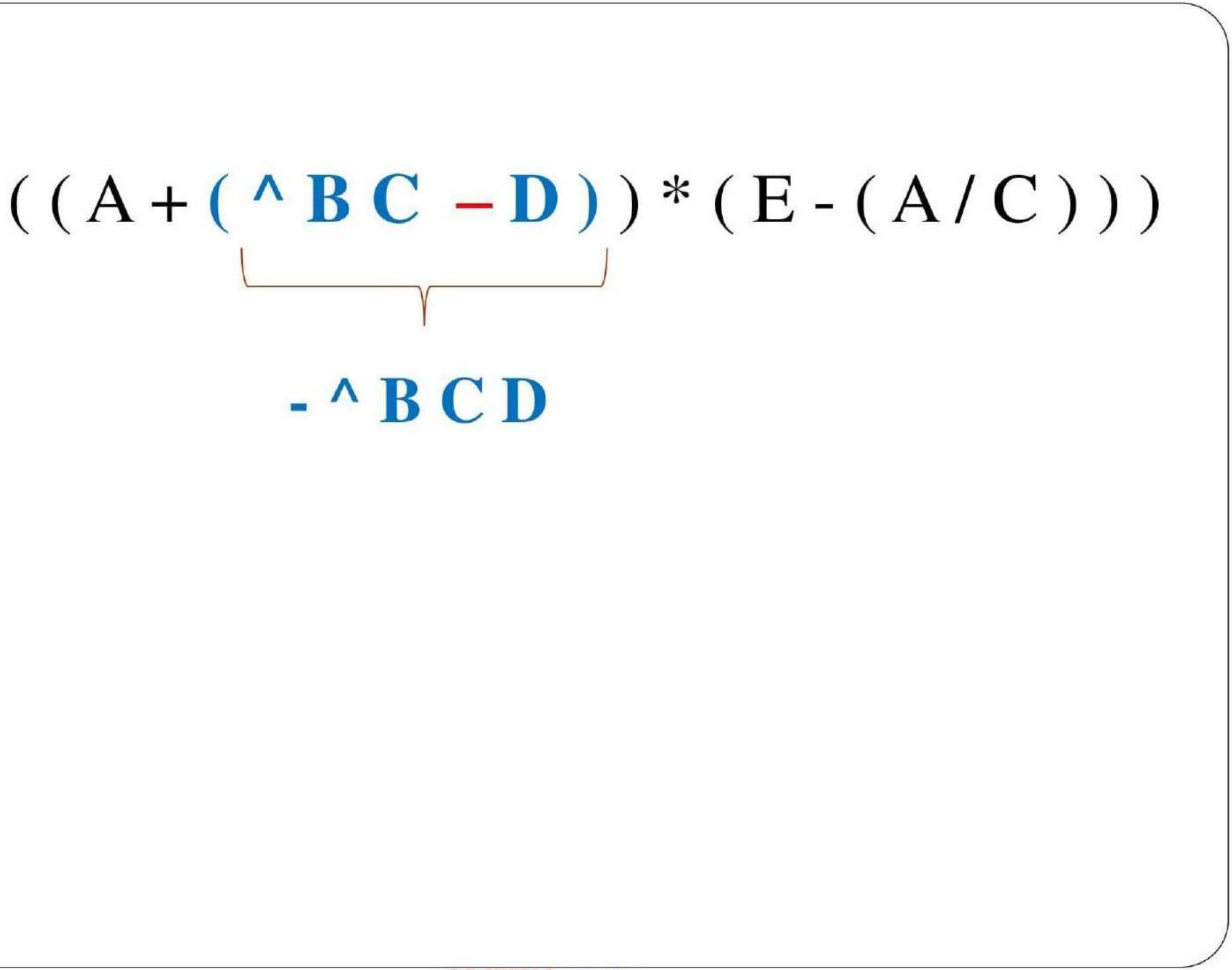
$$((A+((B\ \hat{}\ C)-D))*(E-(A/C)))$$

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


((A + ((B ^ C) – D)) * (E - (A / C)))
^ B C

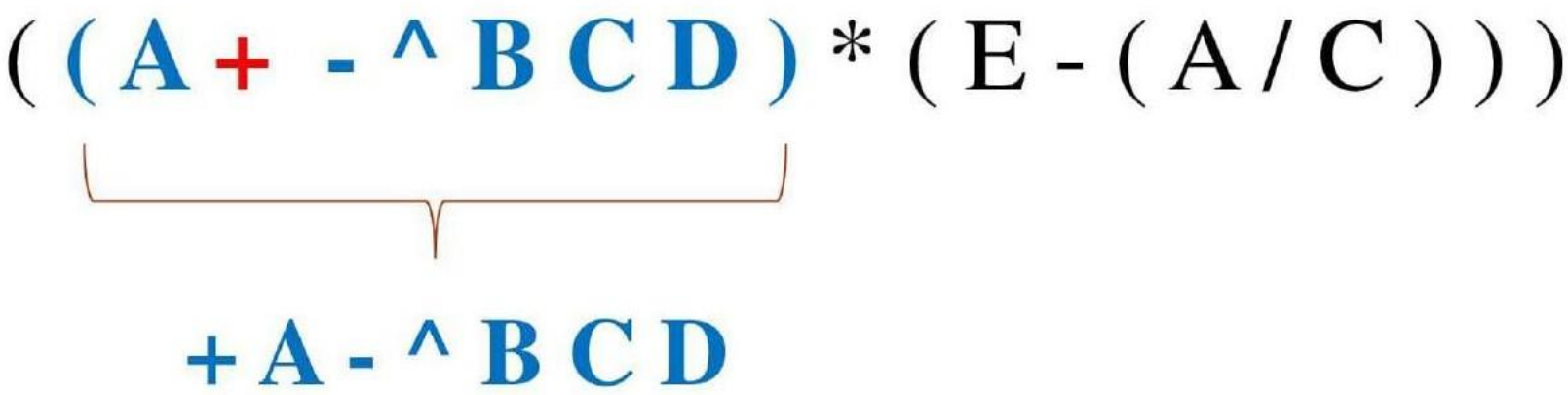
((A + (^ B C – D)) * (E - (A / C)))

((A+(^BC-D))*(E-(A/C)))


((A + (^ B C – D)) * (E - (A / C)))
- ^ B C D

((A + - ^ B C D) * (E - (A / C)))

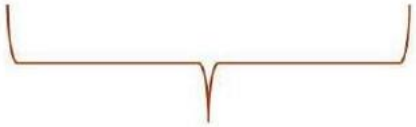
((A + - ^ B C D) * (E - (A / C)))


((A + - ^ B C D) * (E - (A / C)))
+ A - ^ B C D

(+A-^BCD *(E-(A/C)))

(+A-^BCD *(E-(A/C)))

(+ A - ^ B C D * (E - (A / C)))



/ A C

(+ A - ^ B C D * (E - / A C))

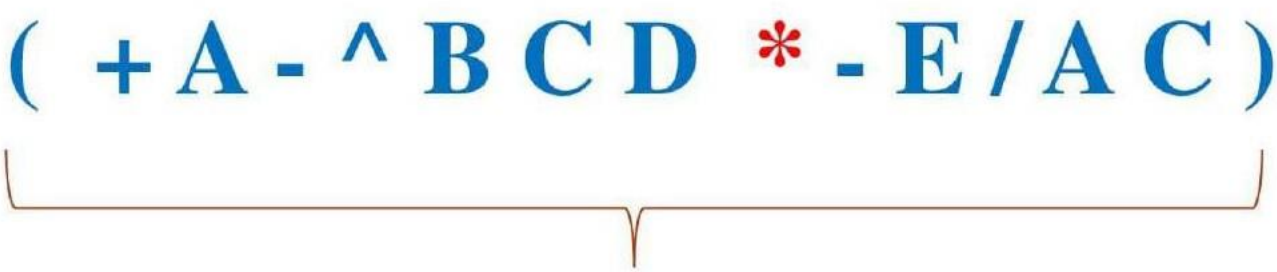
(+ A - ^ B C D * (E - / A C))

(+ A - ^ B C D * (E - / A C))

- E / A C

(+ A - ^ B C D * - E / A C)

(+ A - ^ B C D * - E / A C)


(+ A - ^ B C D * - E / A C)
* + A - ^ B C D - E / A C

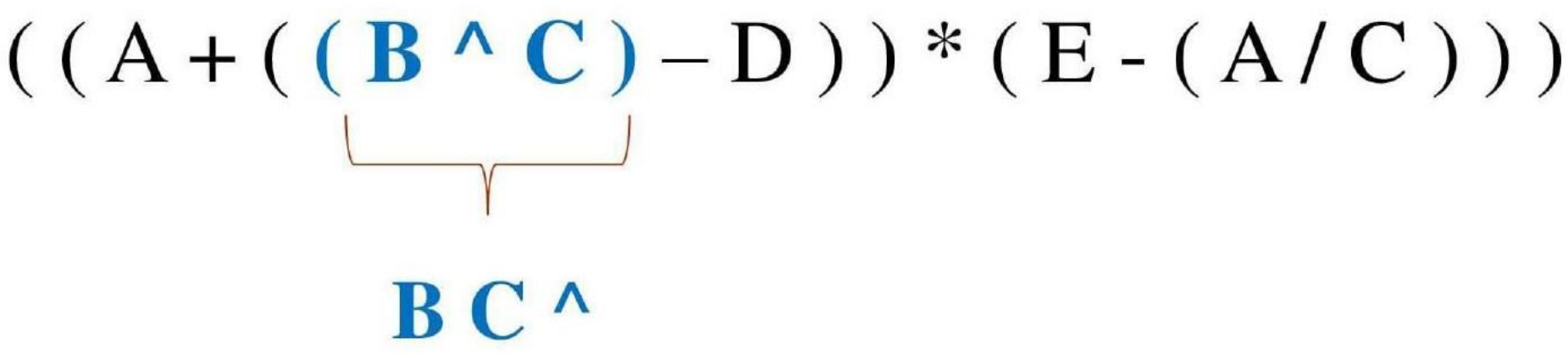
* + A - ^ B C D - E / A C

POSTFIX NOTATION

- Operands precede the operator
- Parentheses are not used
- Ex: 52+

$$((A+((B\ \hat{}\ C)-D))*(E-(A/C)))$$

((A + ((B ^ C) – D)) * (E - (A / C)))


((A + ((B ^ C) – D)) * (E - (A / C)))
B C ^

((A+(B C ^ −D))*(E-(A/C)))

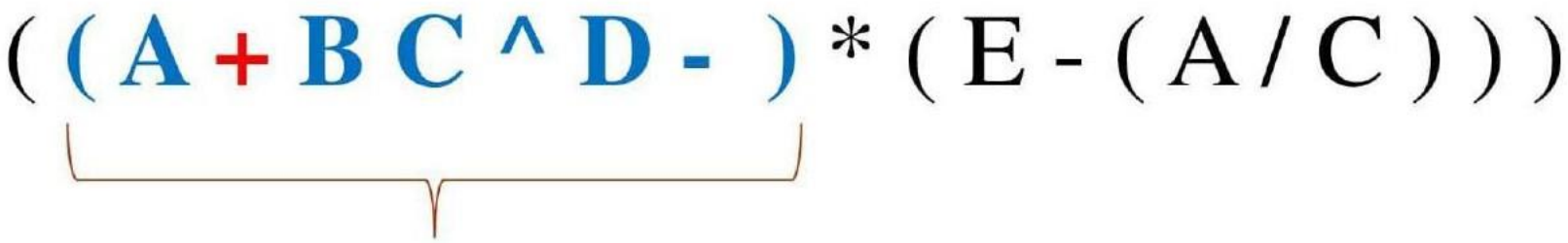
((A+(BC^ -D))*(E-(A/C)))

((A+(BC^ –D))*(E-(A/C)))

B C ^ D -

((A + **B C ^ D -**) * (E - (A / C)))

((A + B C ^ D -) * (E - (A / C)))



A B C ^ D - +

(**A B C ^ D - +** * (E - (A / C)))

(A B C ^ D - + * (E - (A / C)))

(A B C ^ D - + * (E - (A / C)))

A C /

(A B C ^ D - + * (E - A C /))

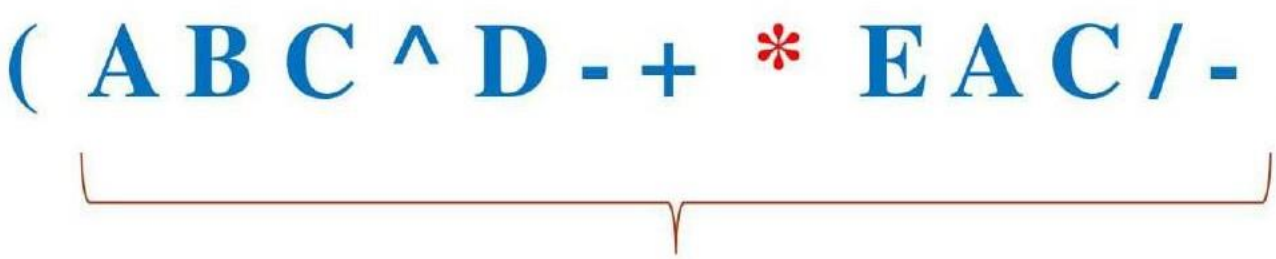
(A B C ^ D - + * (E - A C /))

(A B C ^ D - + * (E - A C /))

E A C / -

(A B C ^ D - + * **E A C / -**)

(A B C ^ D - + * E A C / -)

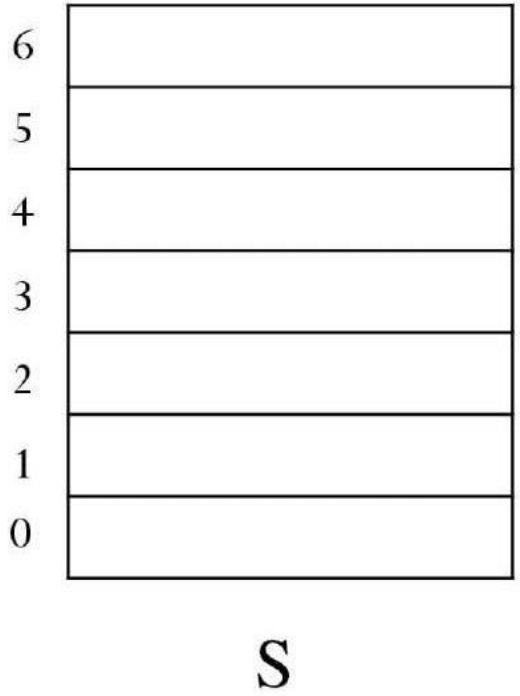

(A B C ^ D - + * E A C / -)
A B C ^ D - + E A C / - *

A B C ^ D - + E A C / - *

Infix Expression	Prefix Expression	Postfix Expression
A+B	+AB	AB+
A+B-C	-+ABC	AB+C-
(A+B)*(C-D)	*+AB-CD	AB+CD-*
((A+B)*C)-D	-*+ABCD	AB+C*D-
A+B*C-D+E/F/(G+H)	+-A*BCD//EF+GH	ABC*+D-EF/GH+/+
((A+B)*C-(D-E))/(F+G)	/-*+ABC-DE+FG	AB+C*DE--FG+/
A-B/(C*D/E)	-A/B/*CDE	ABCD*E//-

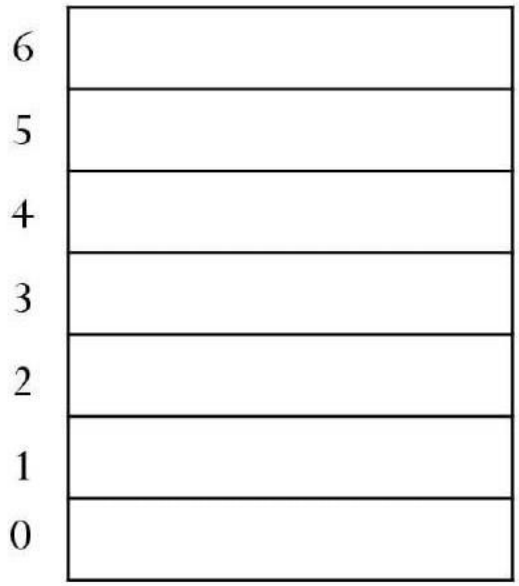
INFIX TO POSTFIX CONVERSION ALGORITHM

A – B / (C * D / E)



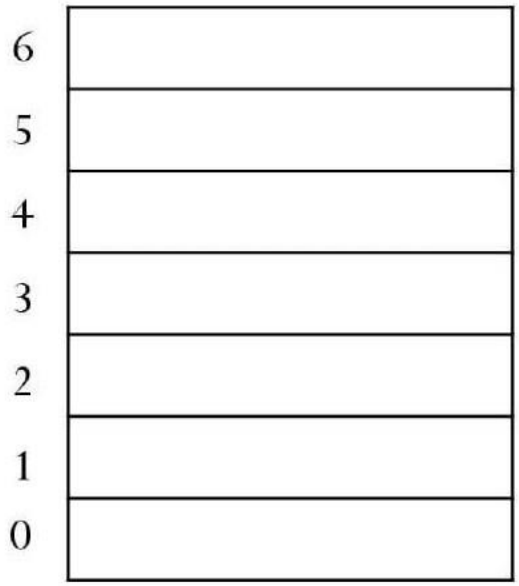
Postfix Expression:

A – B / (C * D / E)



Postfix Expression:

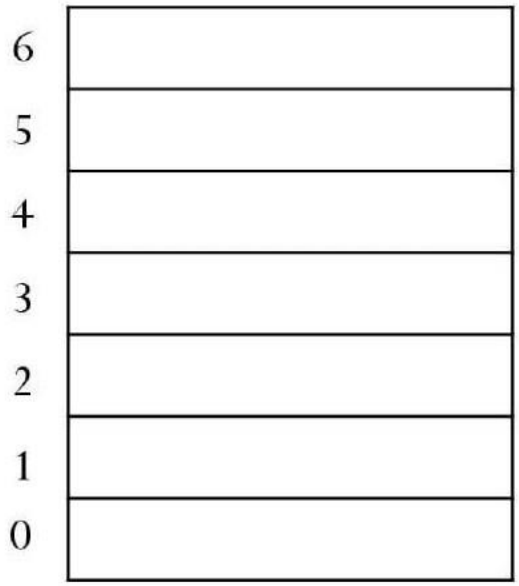
A – B / (C * D / E)



Postfix Expression:

A

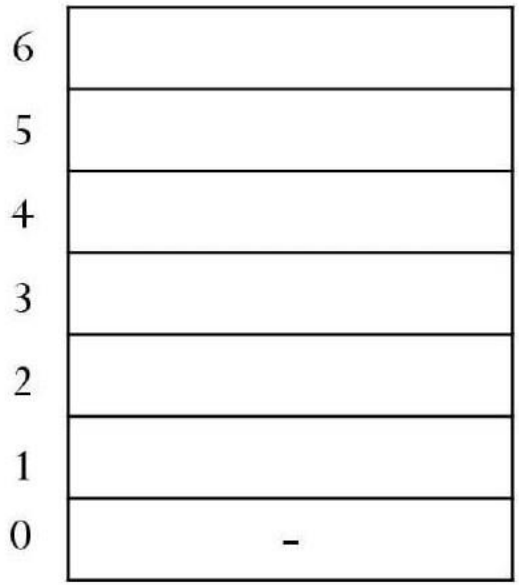
A – B / (C * D / E)



Postfix Expression:

A

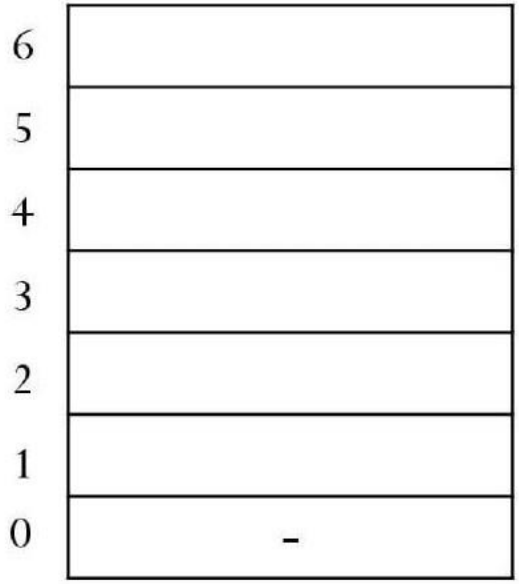
A – B / (C * D / E)



Postfix Expression:

A

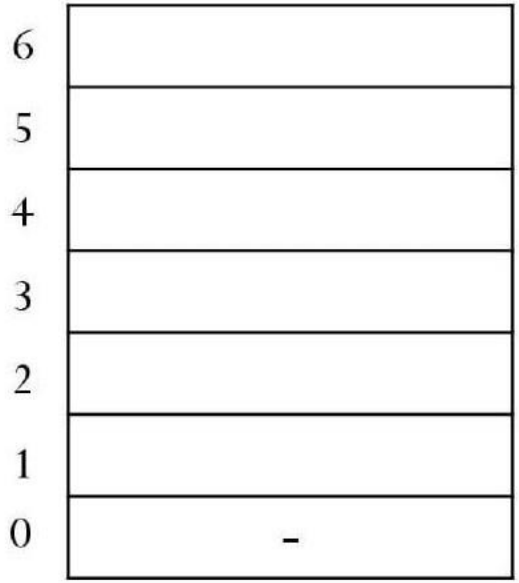
A – B / (C * D / E)



Postfix Expression:

A

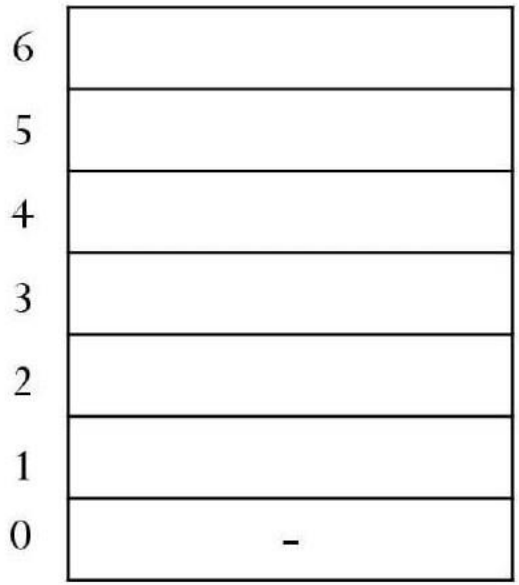
A – B / (C * D / E)



Postfix Expression:

A B

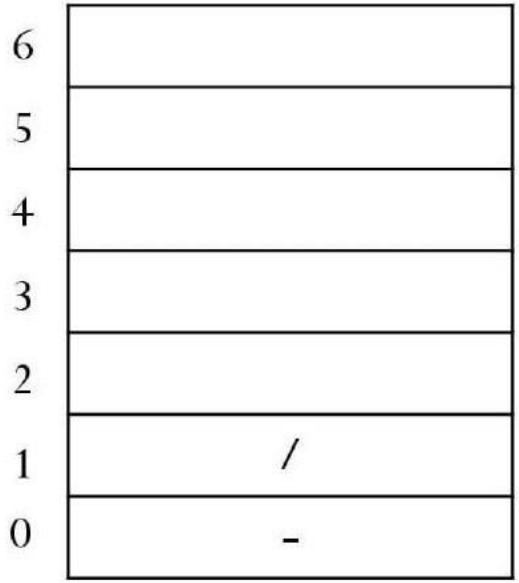
A – B / (C * D / E)



Postfix Expression:

A B

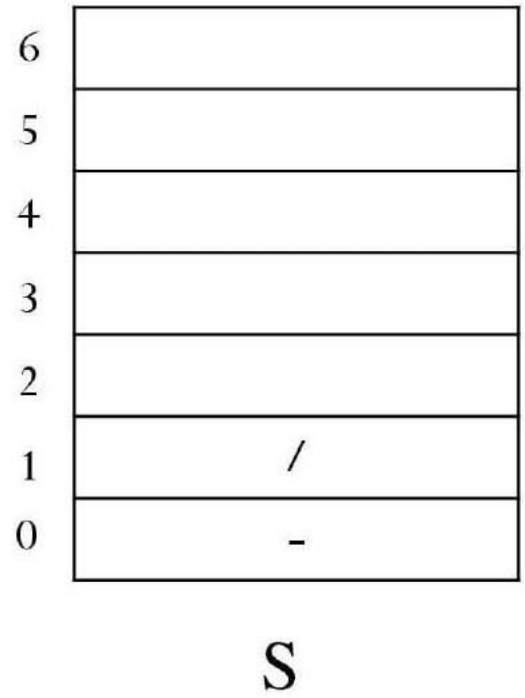
A – B / (C * D / E)



Postfix Expression:

A B

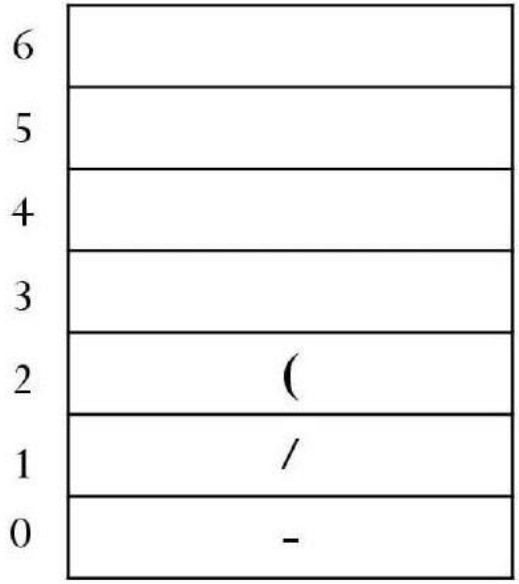
A – B / (C * D / E)



Postfix Expression:

A B

A – B / (C * D / E)

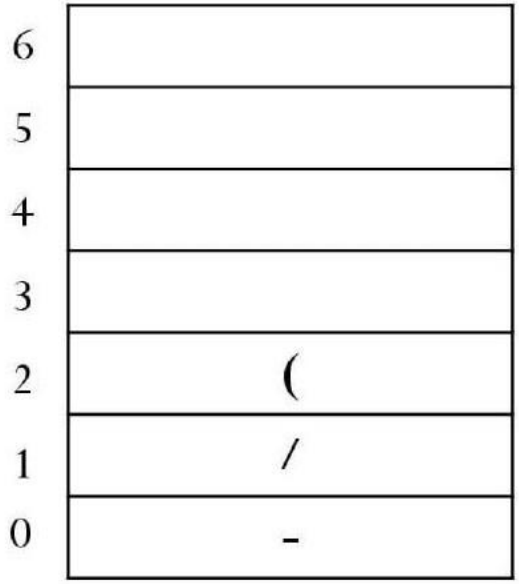


S

Postfix Expression:

A B

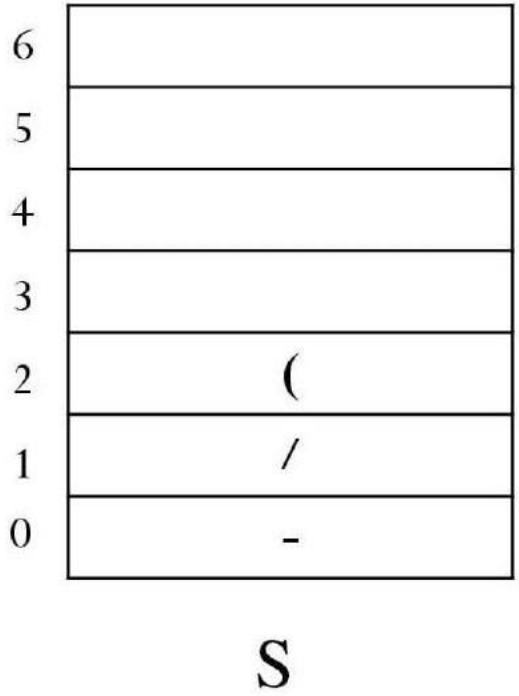
A – B / (C * D / E)



Postfix Expression:

A B

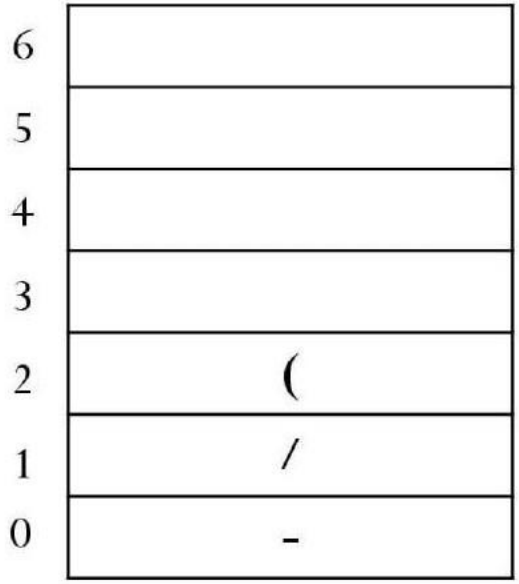
A – B / (C * D / E)



Postfix Expression:

A B C

A – B / (C * D / E)

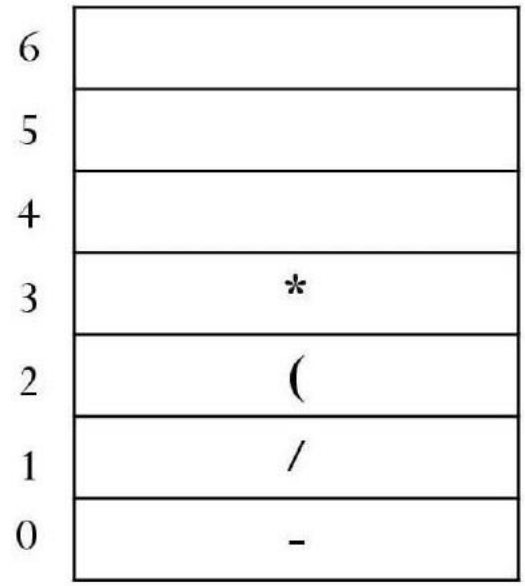


S

Postfix Expression:

A B C

A – B / (C * D / E)

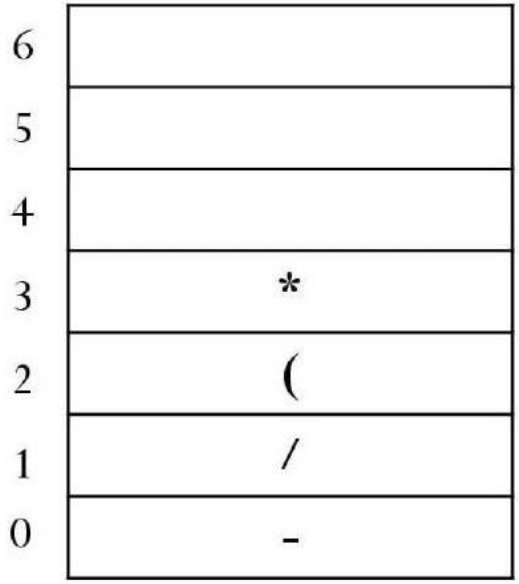


S

Postfix Expression:

A B C

A – B / (C * D / E)

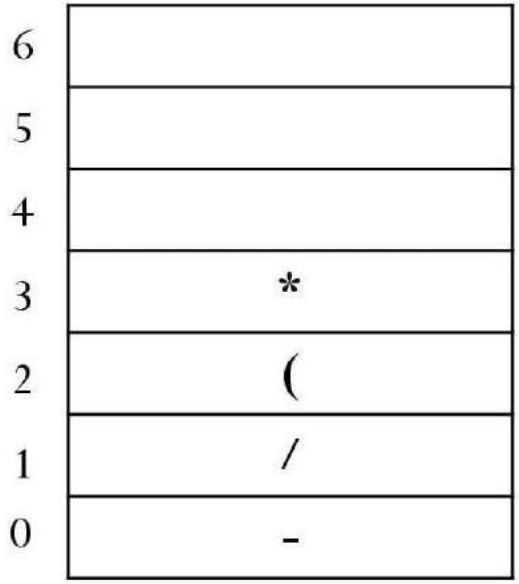


S

Postfix Expression:

A B C

A – B / (C * D / E)

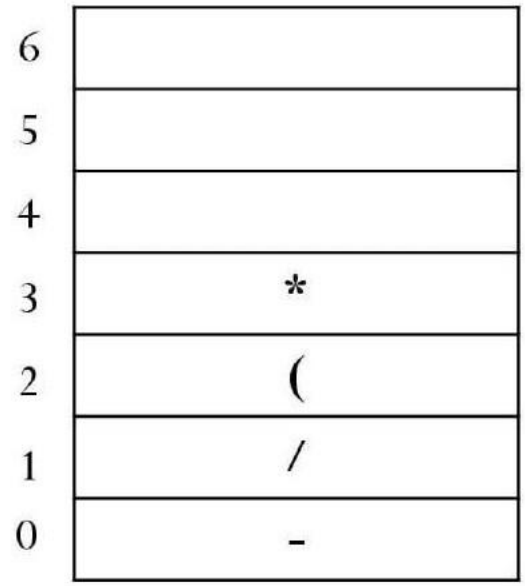


S

Postfix Expression:

A B C D

A – B / (C * D / E)

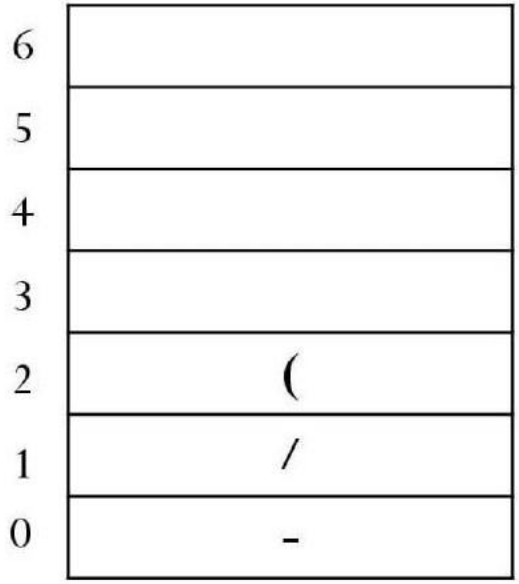


S

Postfix Expression:

A B C D

A – B / (C * D / E)

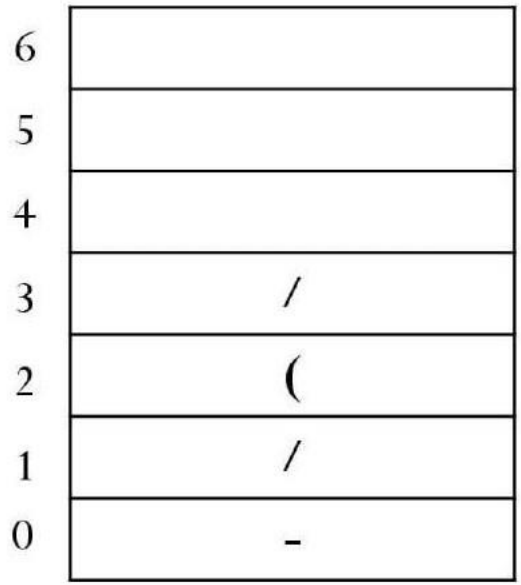


S

Postfix Expression:

A B C D *

A – B / (C * D / E)

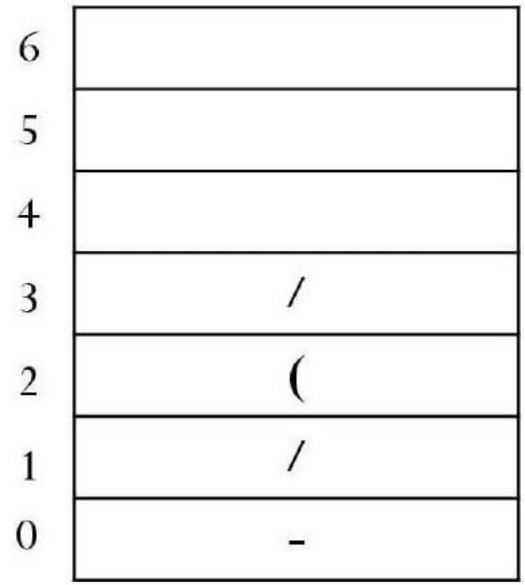


S

Postfix Expression:

A B C D *

A – B / (C * D / E)

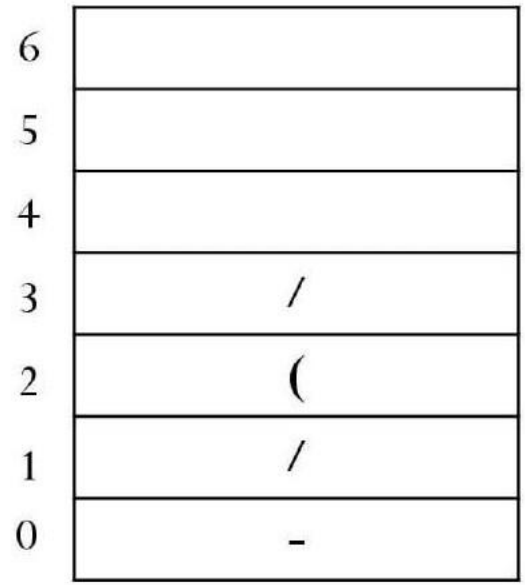


S

Postfix Expression:

A B C D *

A – B / (C * D / E)

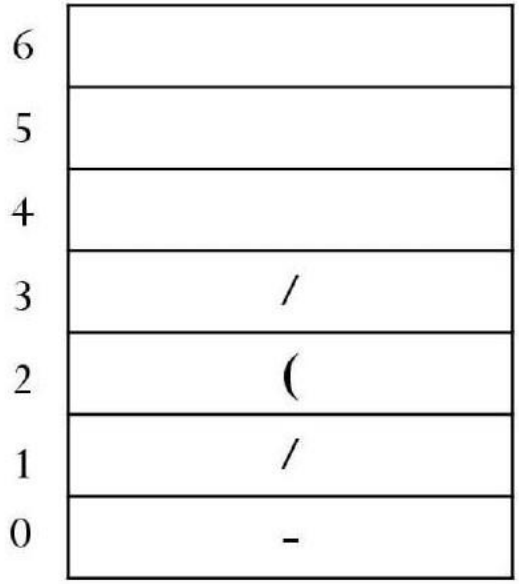


S

Postfix Expression:

A B C D * E

A – B / (C * D / E)

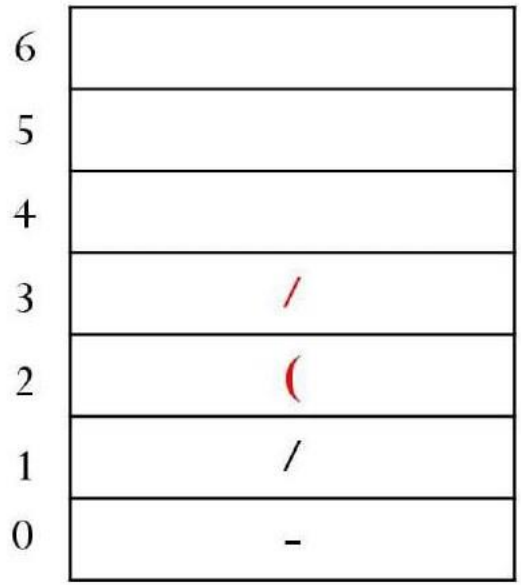


S

Postfix Expression:

A B C D * E

A – B / (C * D / E)

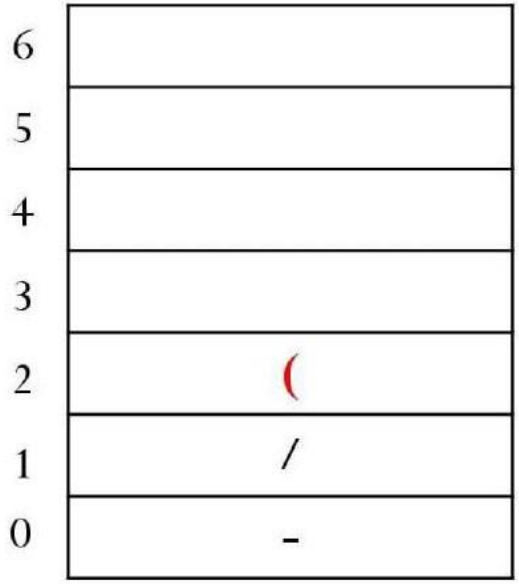


S

Postfix Expression:

A B C D * E

A – B / (C * D / E)

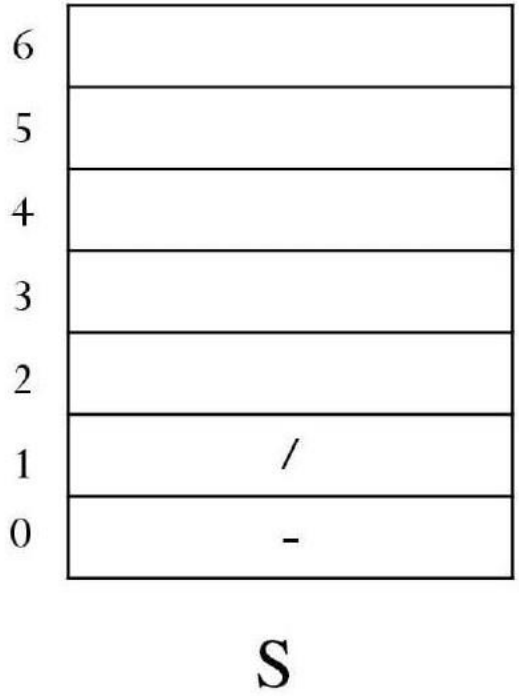


S

Postfix Expression:

A B C D * E /

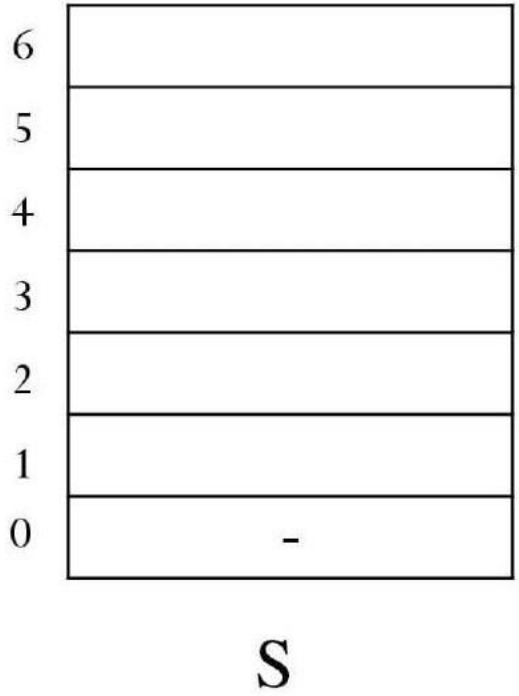
A – B / (C * D / E)



Postfix Expression:

A B C D * E /

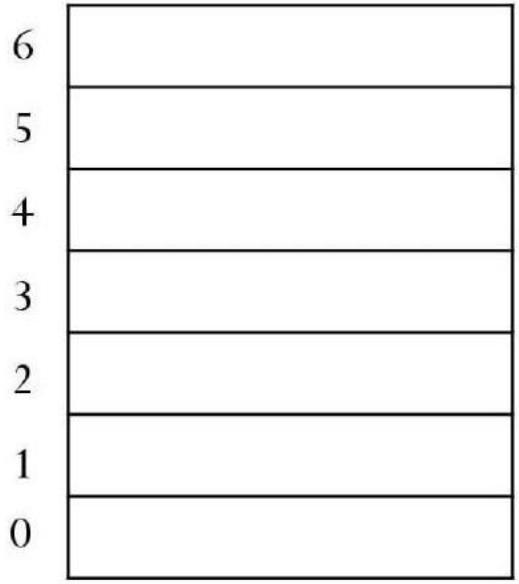
A – B / (C * D / E)



Postfix Expression:

A B C D * E / /

A – B / (C * D / E)



Postfix Expression:

A B C D * E / / -

INFIX TO POSTFIX -ALGORITHM

Algorithm infix_to_postfix(A, P, S)

```
{
   i=0
   while A[i]!='\0' do
   {
         if A[i] is an operand then
                ENQUEUE A[i] to P
         else if A[i] ='(' then
                PUSH A[i] to S
         else if A[i] = ')' then
            Repeatedly POP elements from S until the poped item is '('
            ENQUEUE all these poped items into P except '('
```

```
else
{   x=A[i]                  y=S[top]
    if top=-1 or y='('  then
                PUSH x in to S
    else
    {   while precedence(x)<=precedence(y) do
        {       POP an item from S and ENQUEUE it in P
                y=S[top]
                if top=-1 or y='(' then                 break
        }
        PUSH x in to S
    }
  }
}
Repeatedly POP remaining elements from S and ENQUEUE it in P
Print P
}
```

INFIX TO PREFIX CONVERSION ALGORITHM

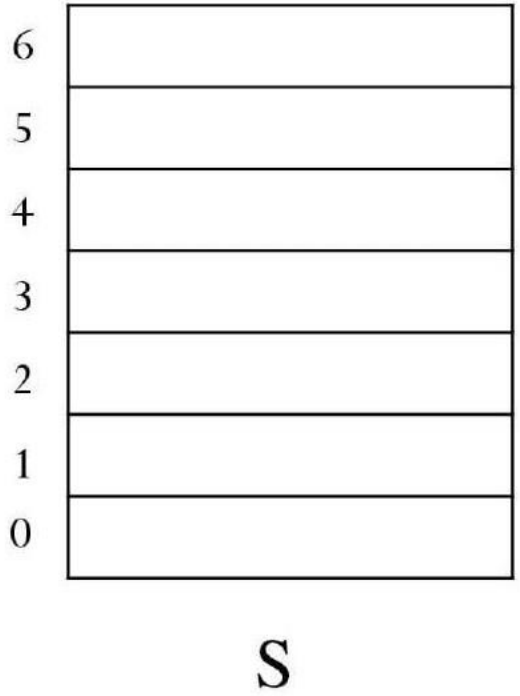
A – B / (C * D / E)

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

A – B / (C * D / E)

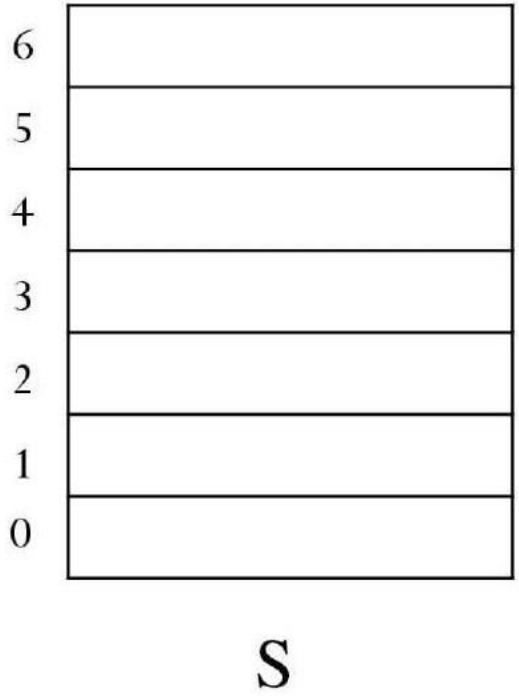
Reverse the string: **) E / D * C (/ B - A**



Prefix Expression:

A – B / (C * D / E)

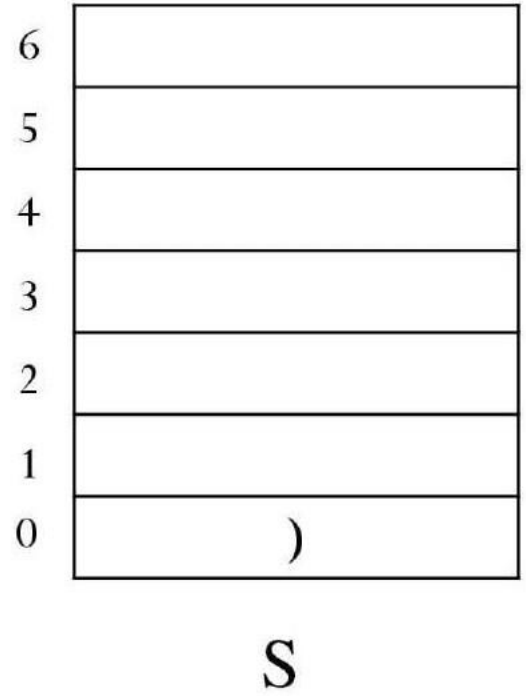
Reverse the string:) E / D * C (/ B - A



Prefix Expression:

A – B / (C * D / E)

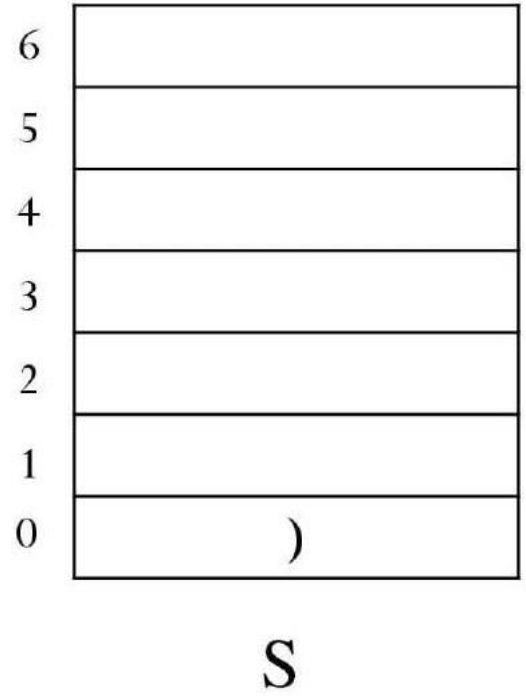
Reverse the string:) E / D * C (/ B - A



Prefix Expression:

A – B / (C * D / E)

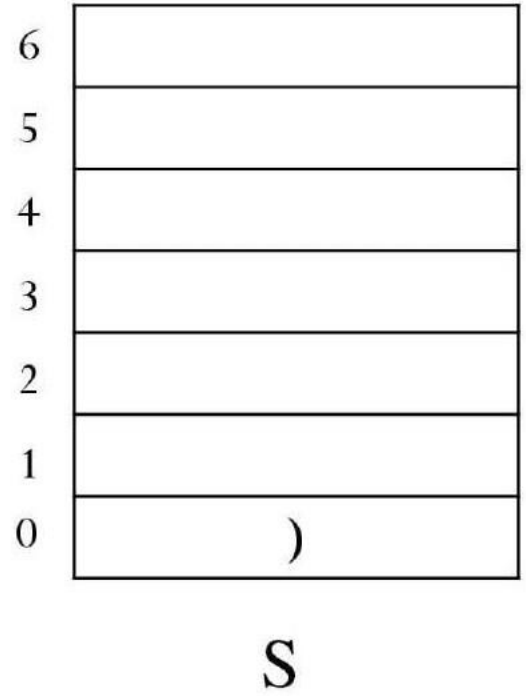
Reverse the string:) E / D * C (/ B - A



Prefix Expression:

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

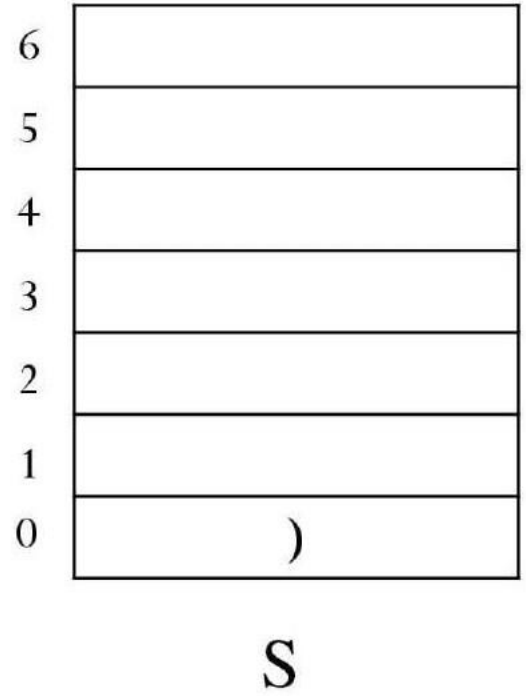


Prefix Expression:

E

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

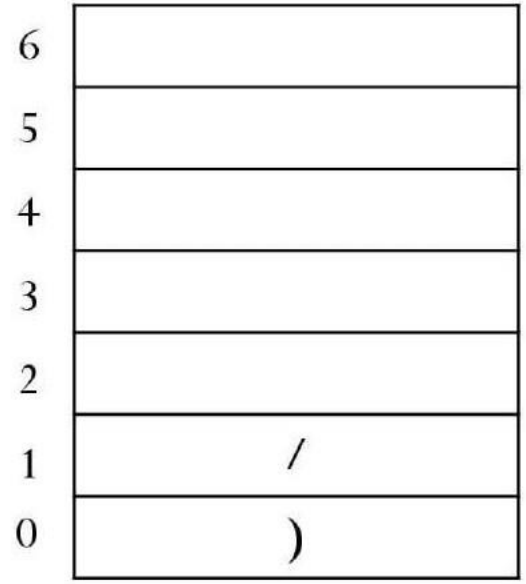


Prefix Expression:

E

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

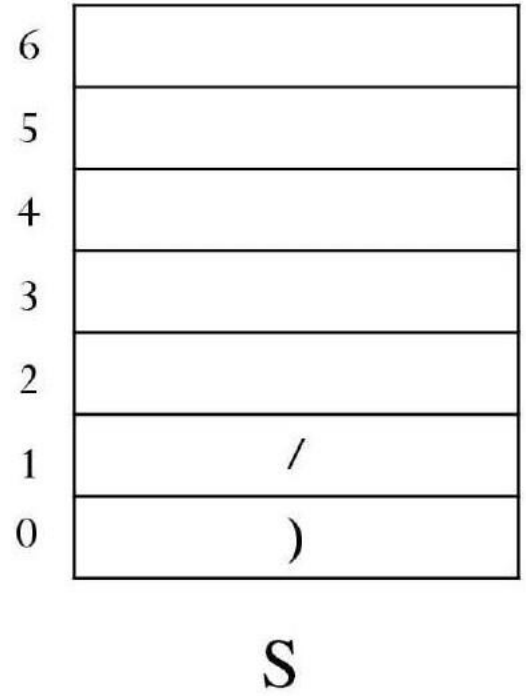


Prefix Expression:

E

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

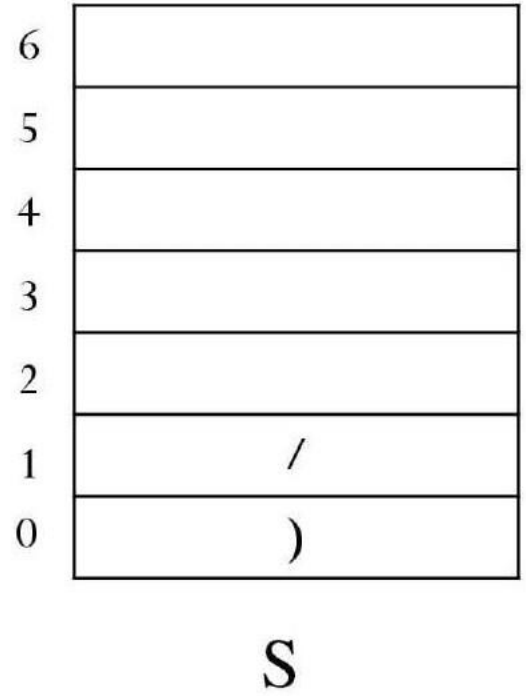


Prefix Expression:

E

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

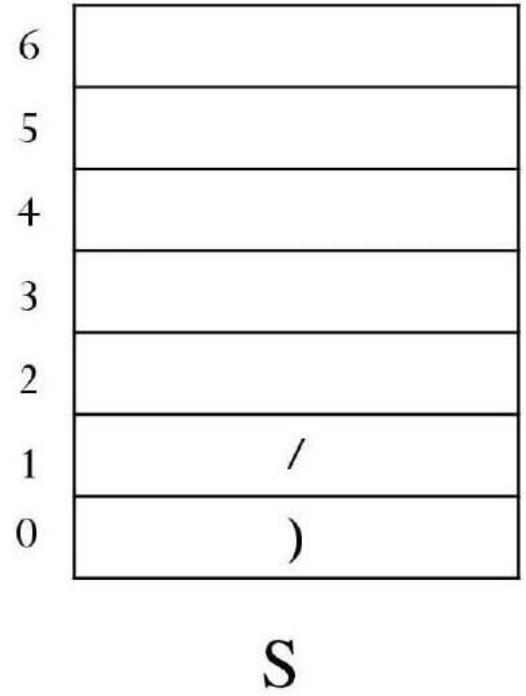


Prefix Expression:

E D

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

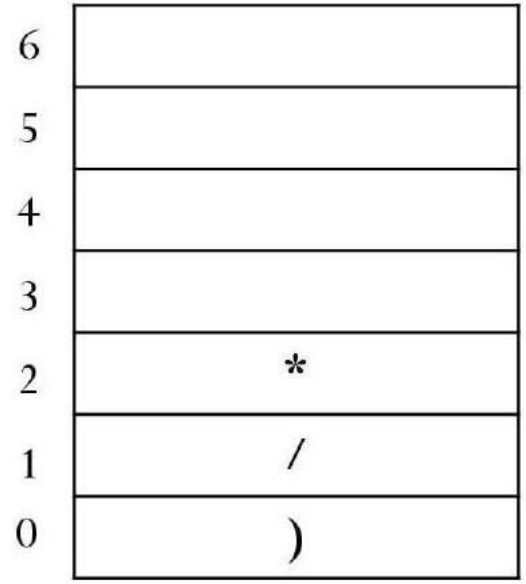


Prefix Expression:

E D

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

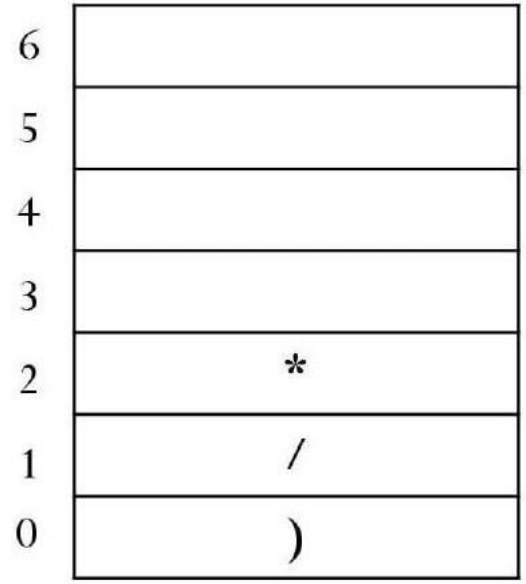


Prefix Expression:

E D

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

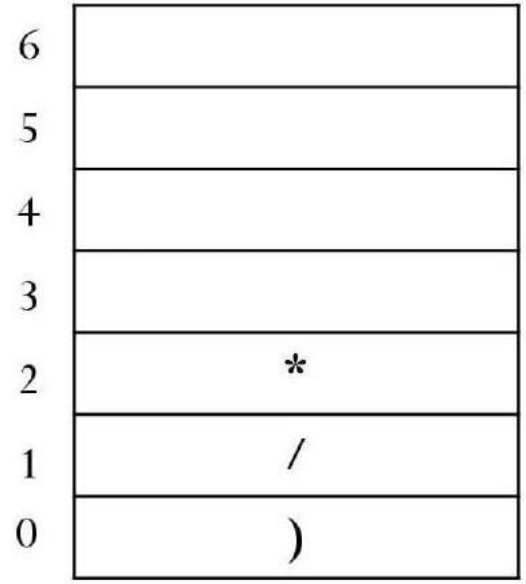


Prefix Expression:

E D

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

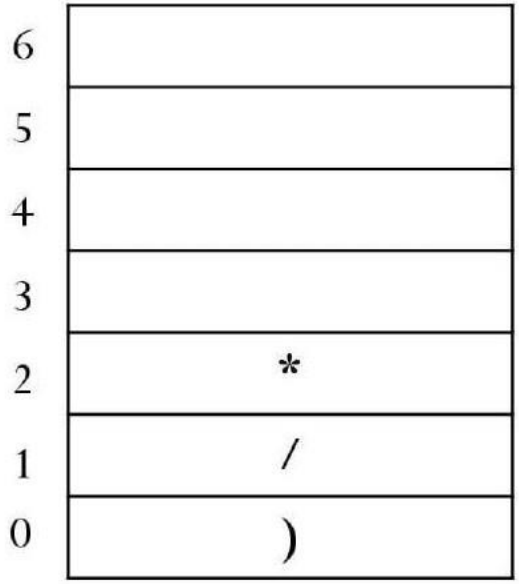


Prefix Expression:

E D C

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

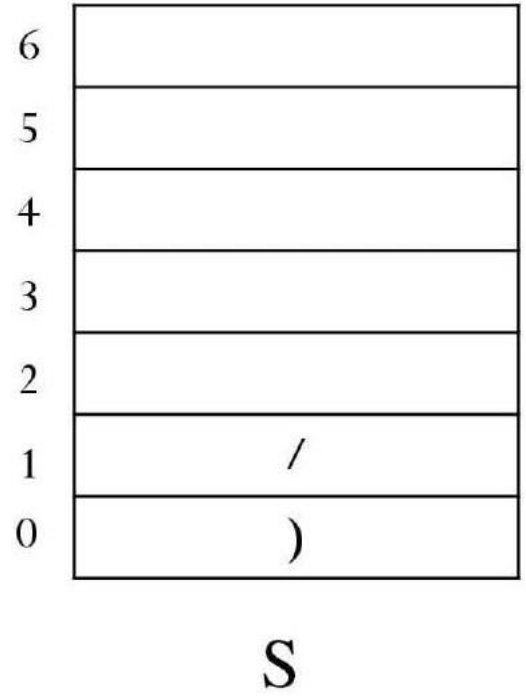


Prefix Expression:

E D C

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

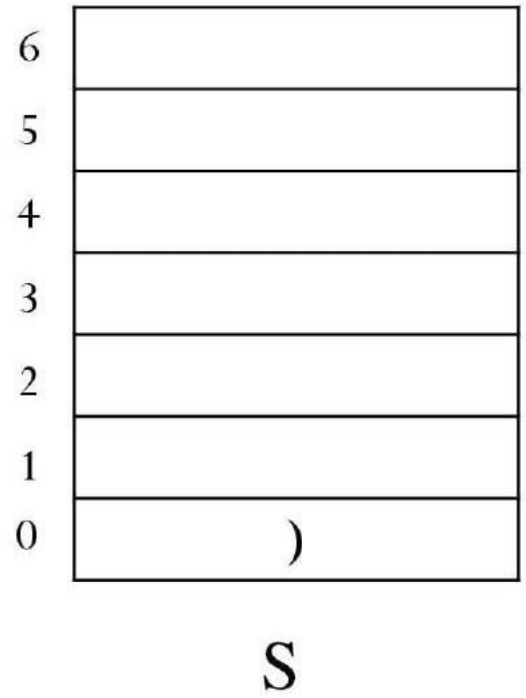


Prefix Expression:

E D C *

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

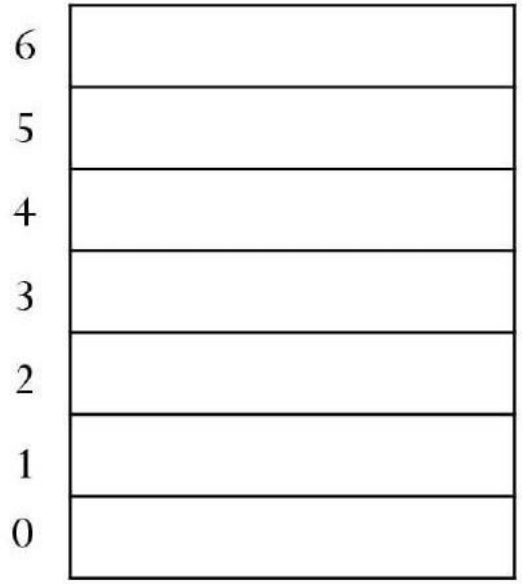


Prefix Expression:

E D C * /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

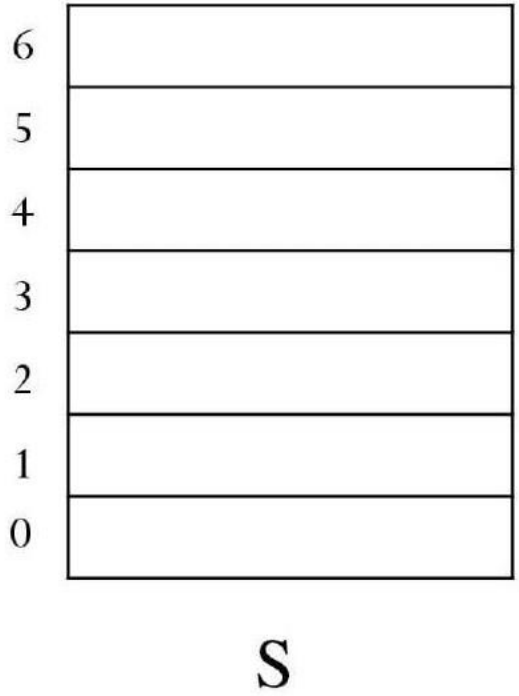


Prefix Expression:

E D C * /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

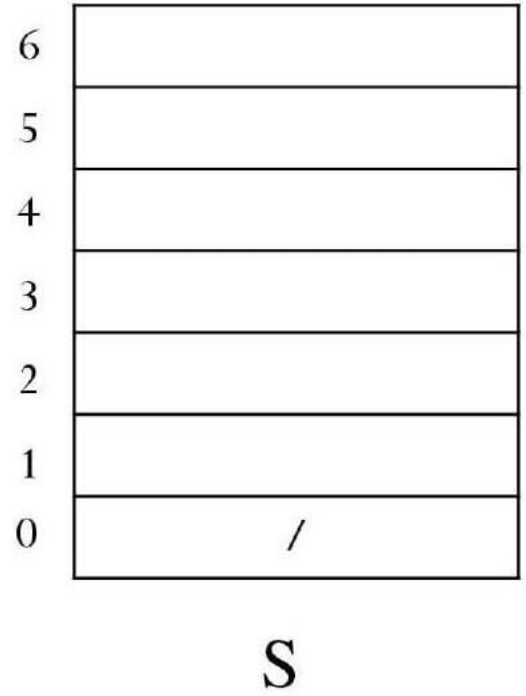


Prefix Expression:

E D C * /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

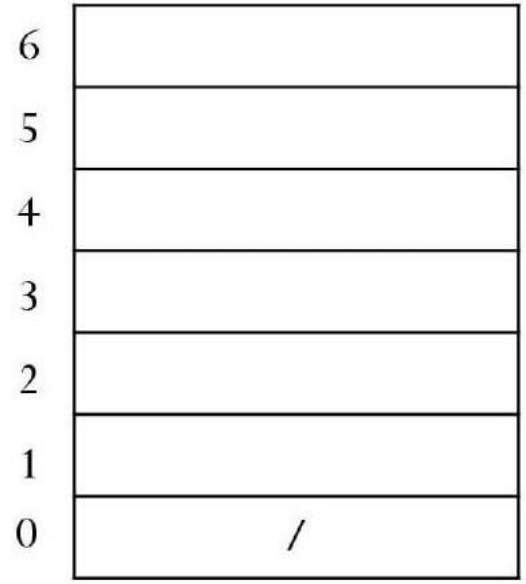


Prefix Expression:

E D C * /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

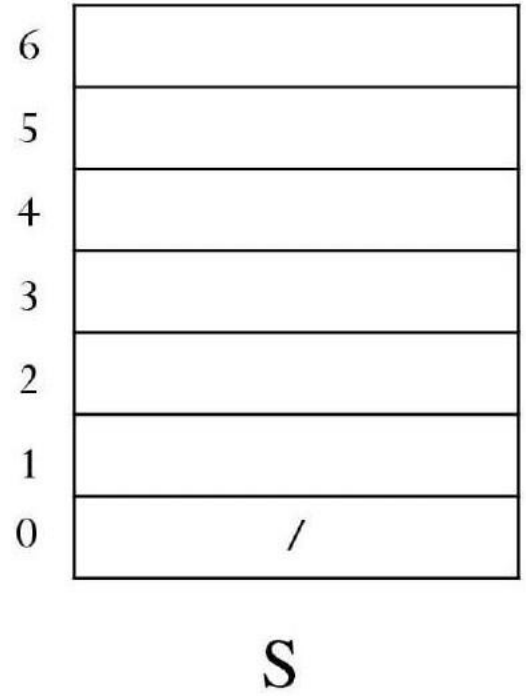


Prefix Expression:

E D C * /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

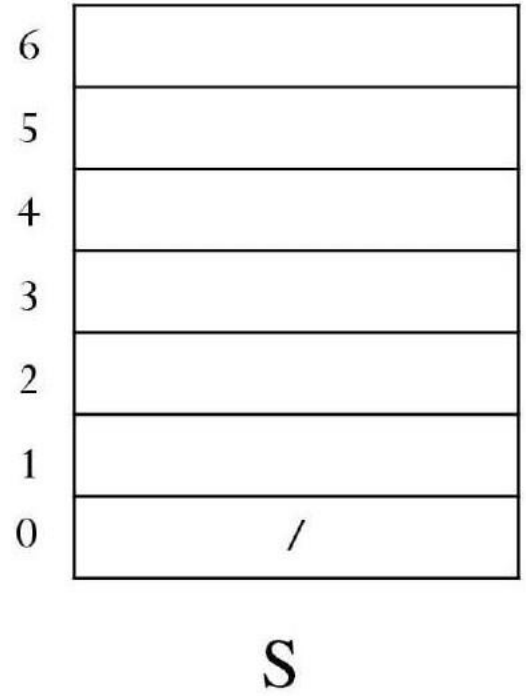


Prefix Expression:

E D C * / B

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

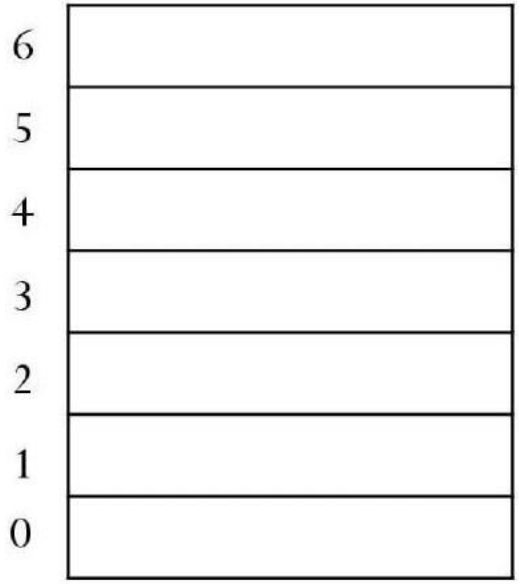


Prefix Expression:

E D C * / B

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A



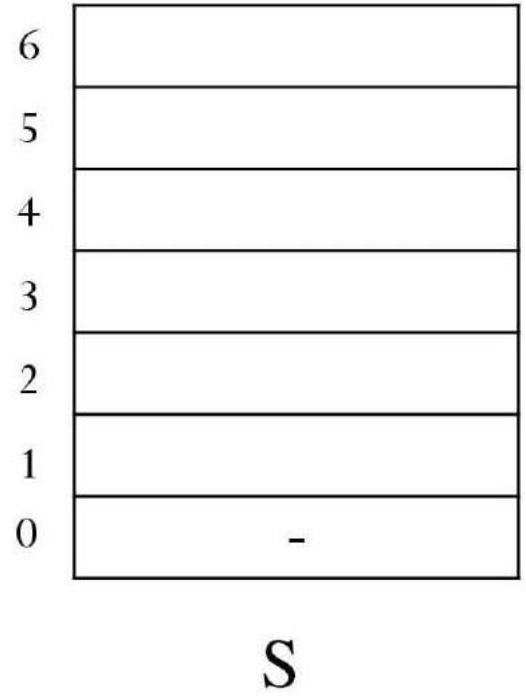
S

Prefix Expression:

E D C * / B /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

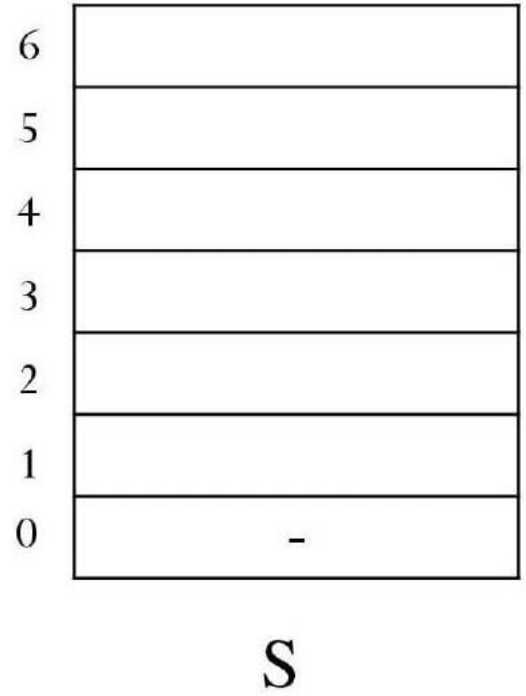


Prefix Expression:

E D C * / B /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

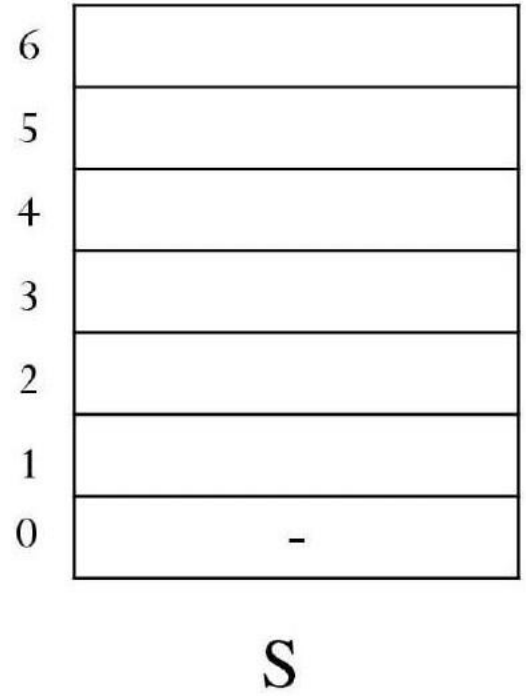


Prefix Expression:

E D C * / B /

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

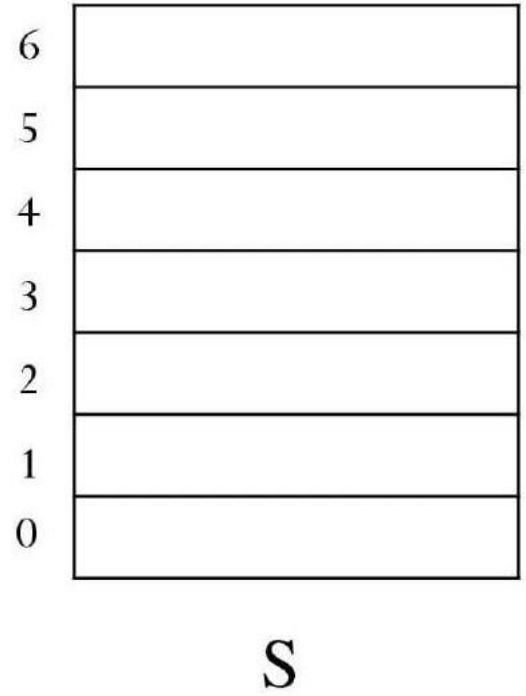


Prefix Expression:

E D C * / B / A

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A

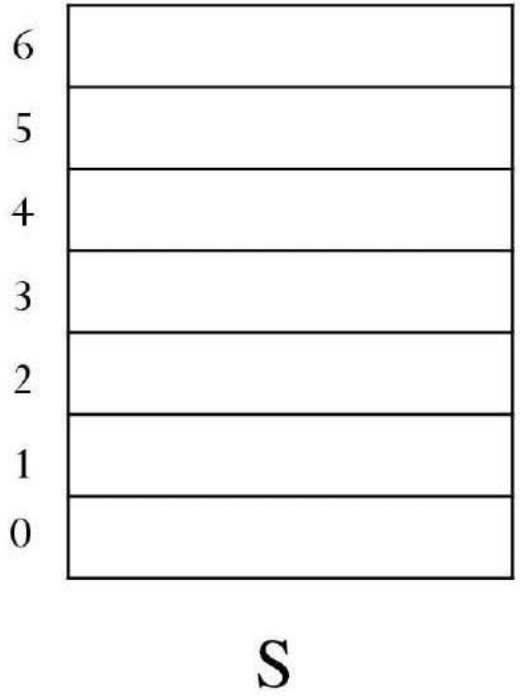


Prefix Expression:

E D C * / B / A -

A – B / (C * D / E)

Reverse the string:) E / D * C (/ B - A



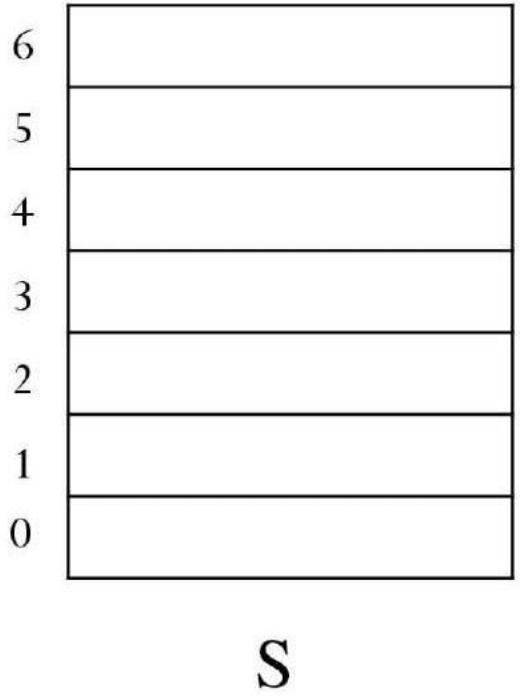
Prefix Expression:

E D C * / B / A -

Reverse the above Expression:

A – B / (C * D / E)

Reverse the string: **) E / D * C (/ B - A**



Prefix Expression:

E D C * / B / A -

Reverse the above Expression:

- A / B / * C D E

INFIX TO PREFIX -ALGORITHM

Algorithm infix_to_prefix(A, P, S)

```
{  Reverse A and place it in R
   i=0
   while R[i]!='\0' do
   {
        if R[i] is an operand then
              ENQUEUE R[i] to P
        else if R[i] =')' then
              PUSH R[i] to S
        else if R[i] = '(' then
           Repeatedly POP elements from S until the poped item is ')'
           ENQUEUE all these poped items into P except ')'
```

```
else
{   x=R[i]                  y=S[top]
    if top=-1 or y='(' then
                PUSH x in to S
    else
    {   while precedence(x)<precedence(y) do
        {       POP an item from S and ENQUEUE it in P
                y=S[top]
                if top=-1 or y='(' then              break
        }
        PUSH x in to S
    }
  }
}
Repeatedly POP remaining elements from S and ENQUEUE it in P
Reverse P and Print it
}
```

POSTFIX EXPRESSION EVALUATION - ALGORITHM

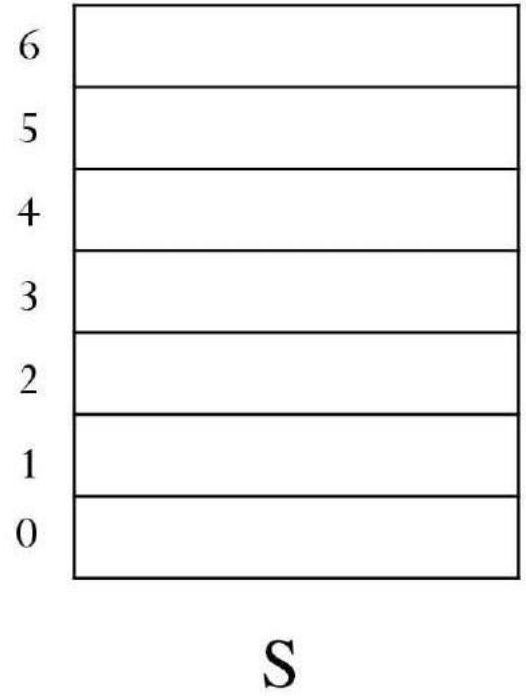
Infix Expression : 10 – 500 / (20 * 2 / 4)

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

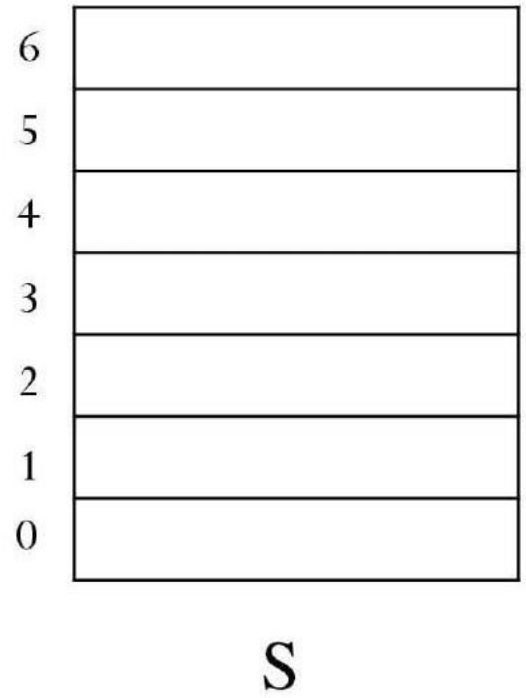
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



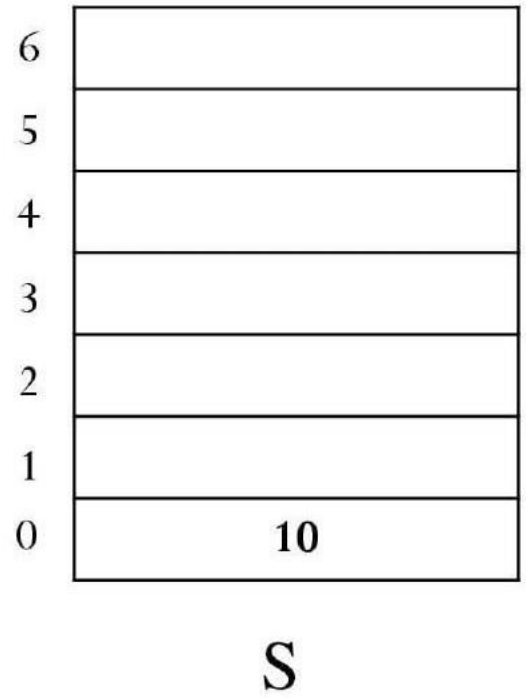
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



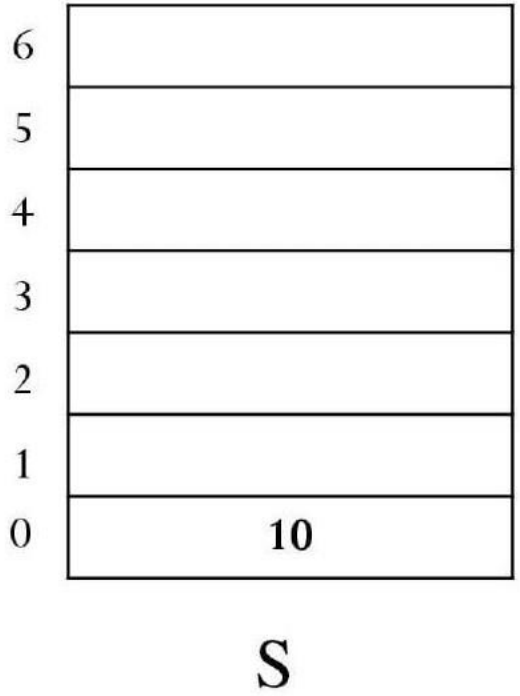
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



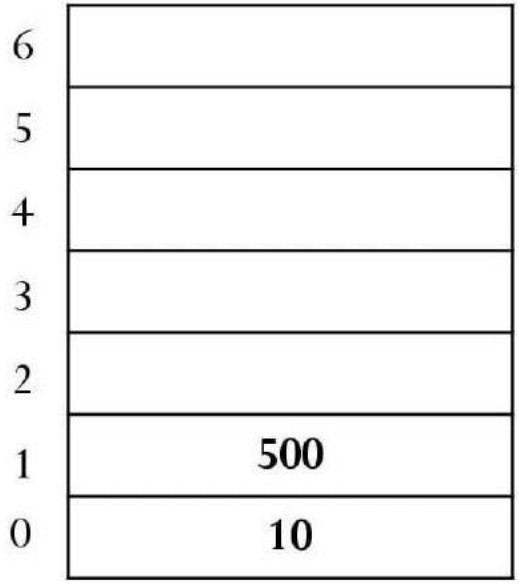
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



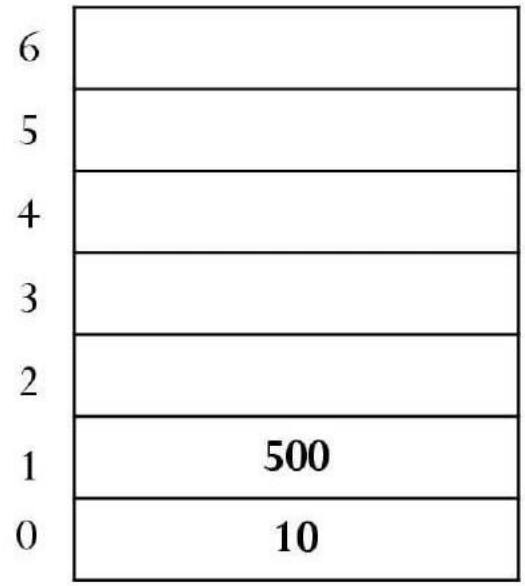
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



Infix Expression : 10 – 500 / (20 * 2 / 4)

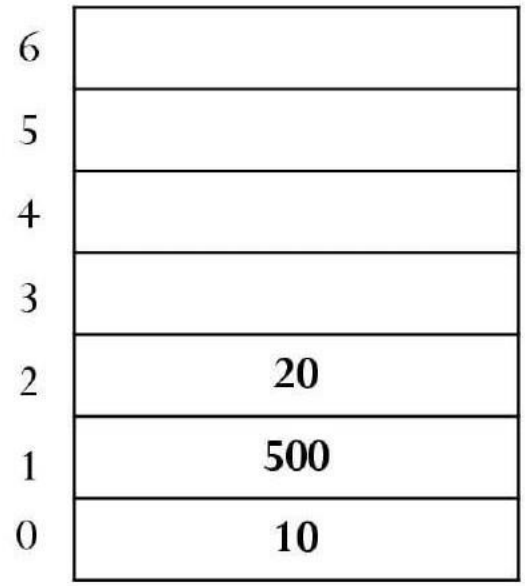
Postfix Expression : 10 500 20 2 * 4 / / -



S

Infix Expression : 10 – 500 / (20 * 2 / 4)

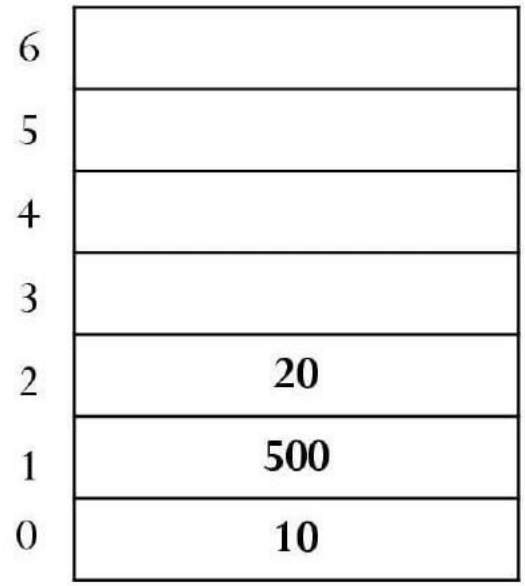
Postfix Expression : 10 500 20 2 * 4 / / -



S

Infix Expression : 10 – 500 / (20 * 2 / 4)

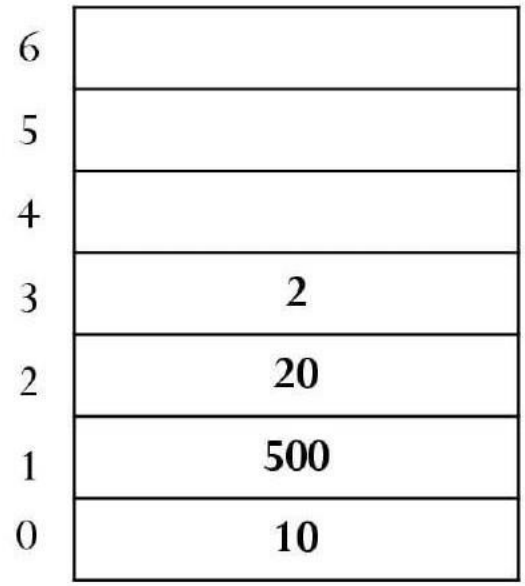
Postfix Expression : 10 500 20 2 * 4 / / -



S

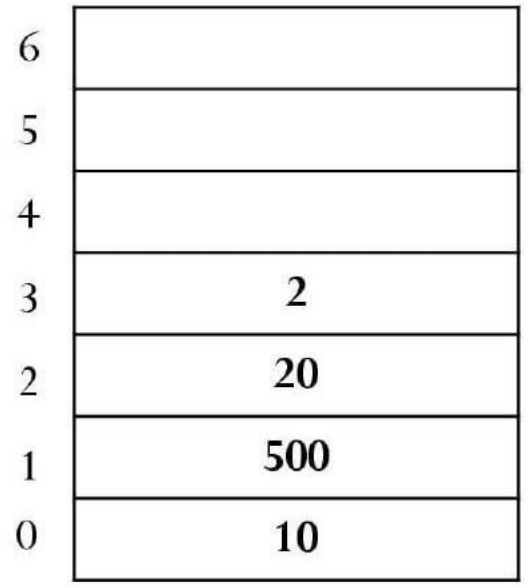
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



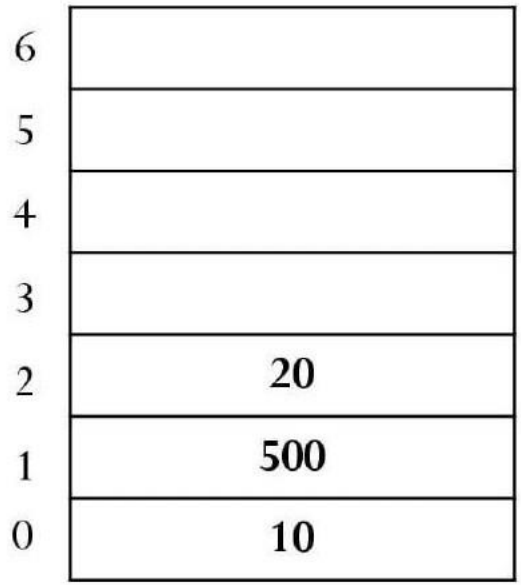
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

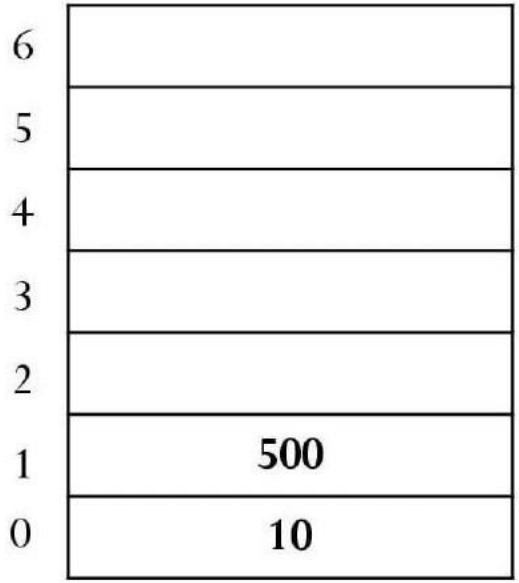


S

* 2

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

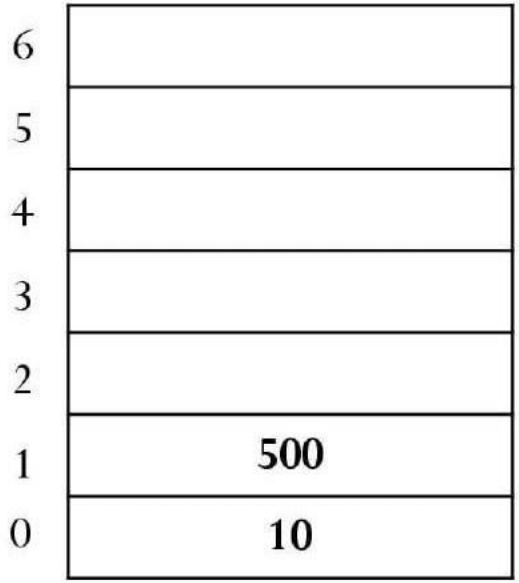


S

20 * 2

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

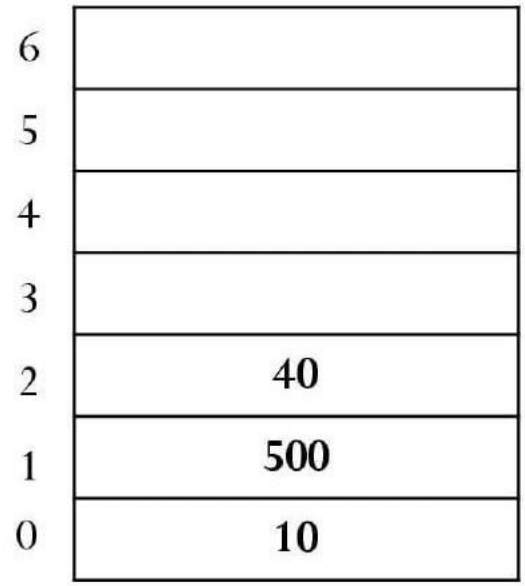


S

20 * 2 = 40

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

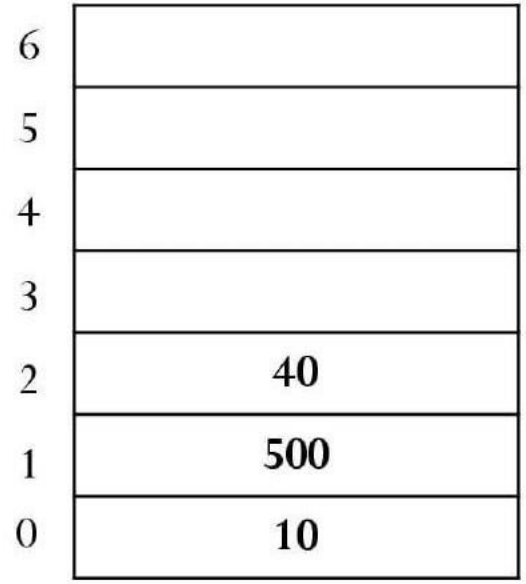


S

20 * 2 = 40

Infix Expression : 10 – 500 / (20 * 2 / 4)

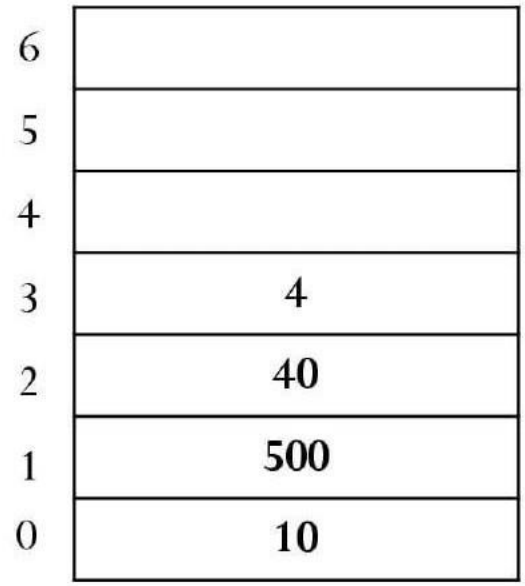
Postfix Expression : 10 500 20 2 * 4 / / -



S

Infix Expression : 10 – 500 / (20 * 2 / 4)

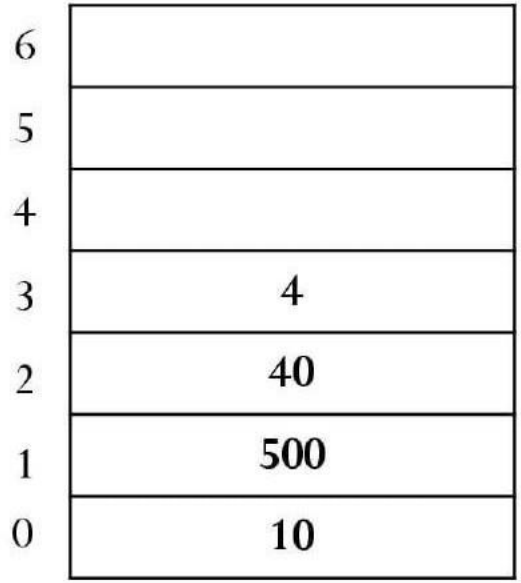
Postfix Expression : 10 500 20 2 * 4 / / -



S

Infix Expression : 10 – 500 / (20 * 2 / 4)

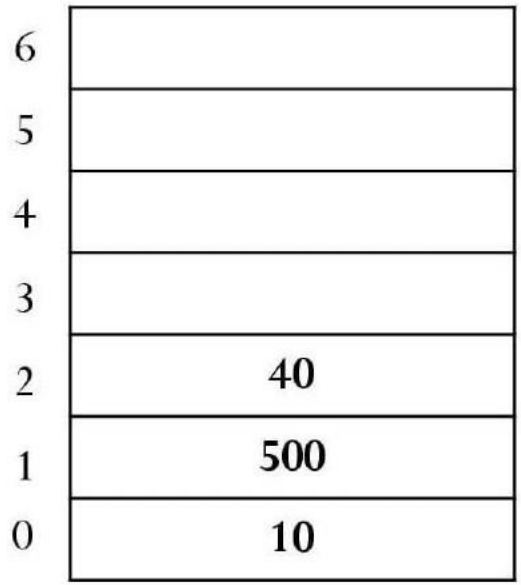
Postfix Expression : 10 500 20 2 * 4 / / -



S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

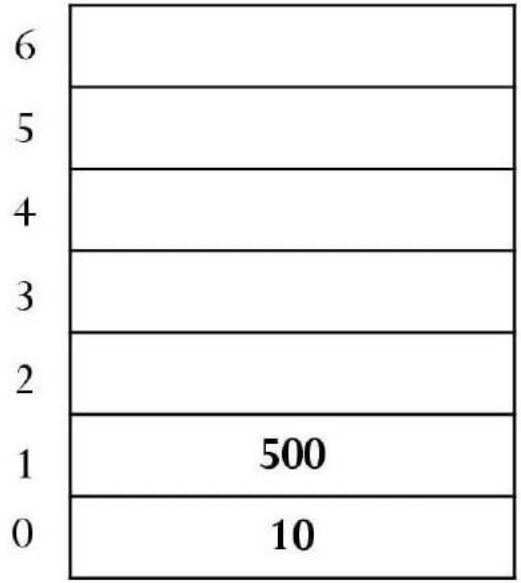


S

/ 4

Infix Expression : 10 – 500 / (20 * 2 / 4)

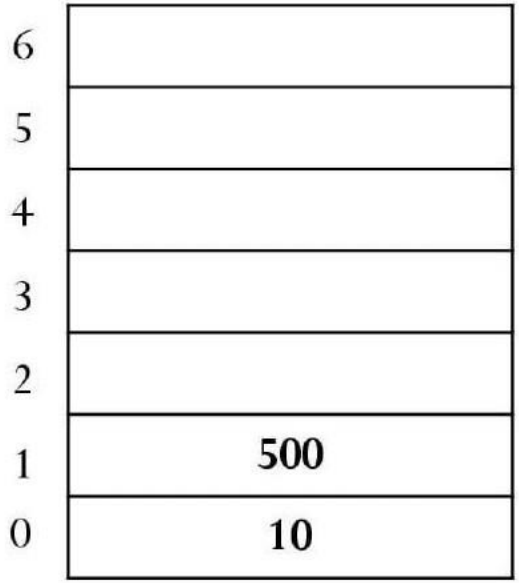
Postfix Expression : 10 500 20 2 * 4 / / -



40 / 4

Infix Expression : 10 – 500 / (20 * 2 / 4)

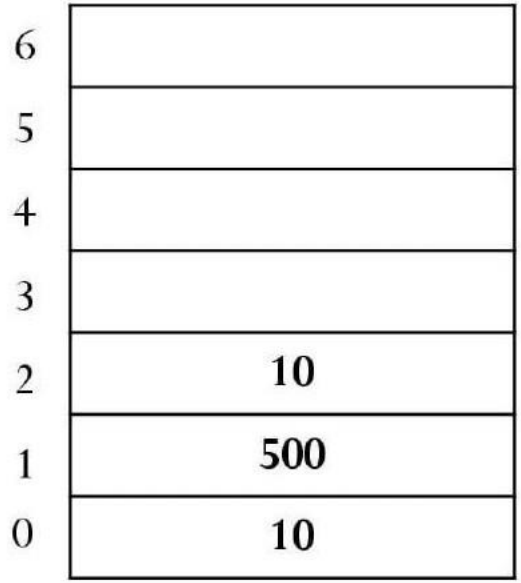
Postfix Expression : 10 500 20 2 * 4 / / -



40 / 4 = 10

Infix Expression : 10 – 500 / (20 * 2 / 4)

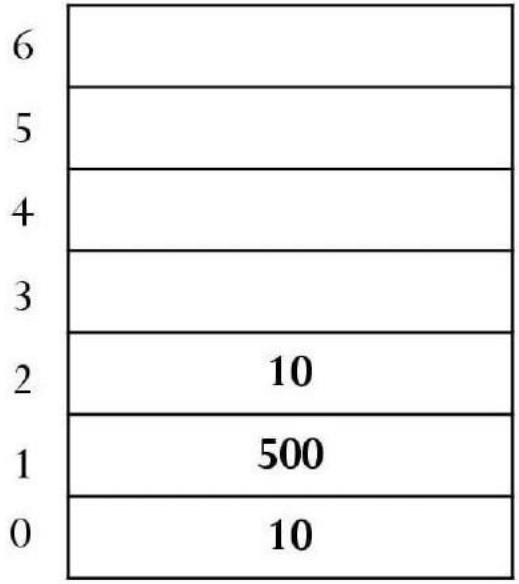
Postfix Expression : 10 500 20 2 * 4 / / -



40 / 4 = 10

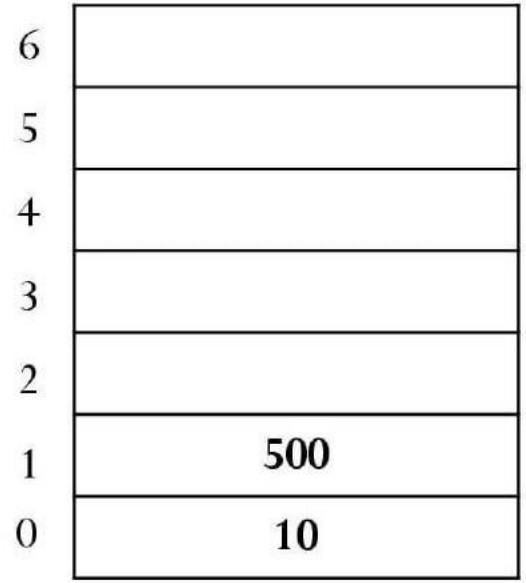
Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

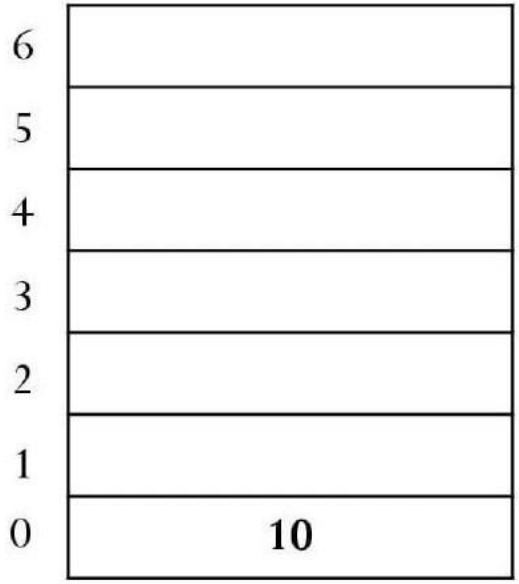


S

/ 10

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

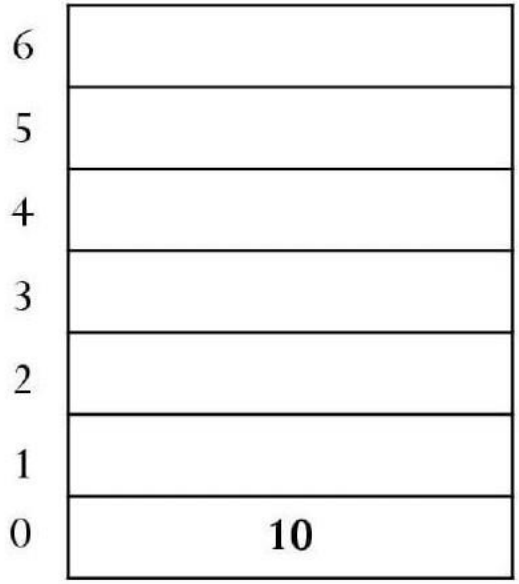


S

500 / 10

Infix Expression : 10 – 500 / (20 * 2 / 4)

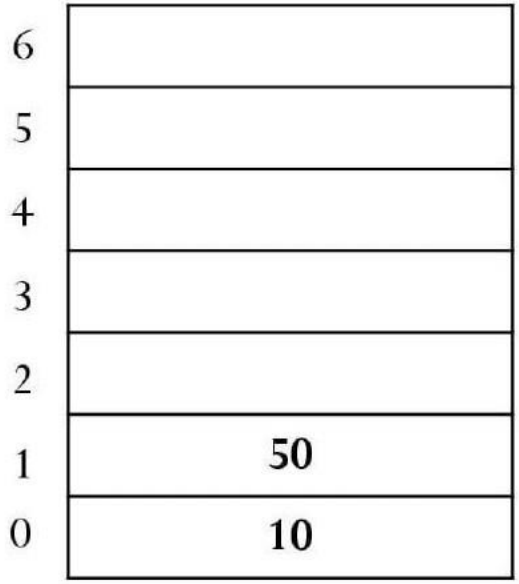
Postfix Expression : 10 500 20 2 * 4 / / -



500 / 10 = 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

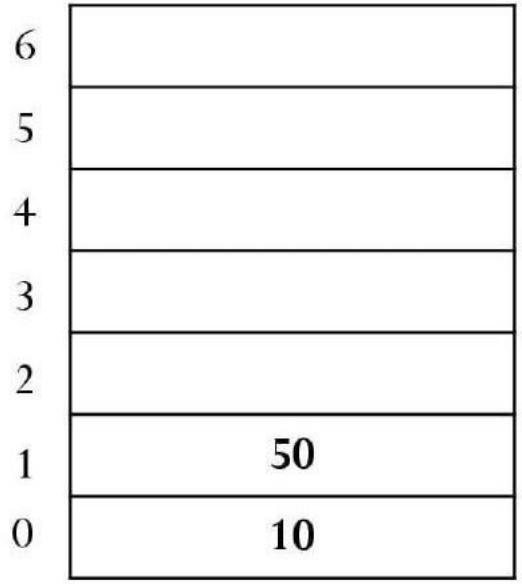
Postfix Expression : 10 500 20 2 * 4 / / -



500 / 10 = 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

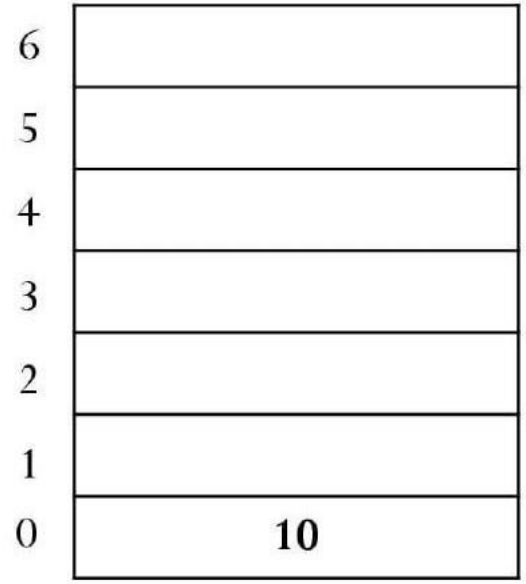
Postfix Expression : 10 500 20 2 * 4 / / -



S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

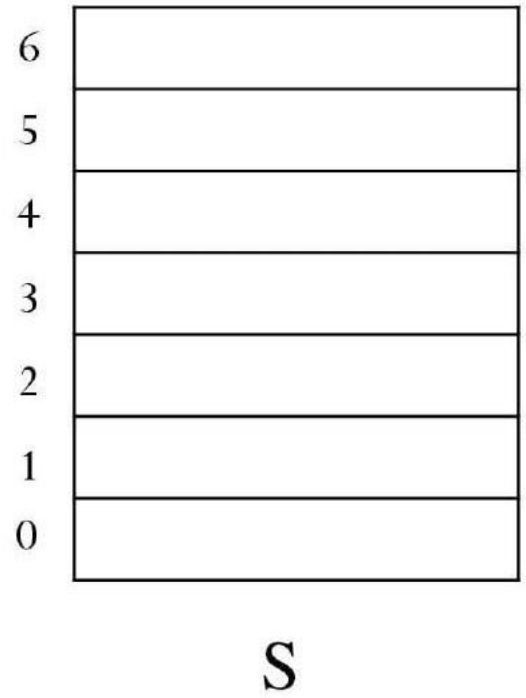


S

- 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

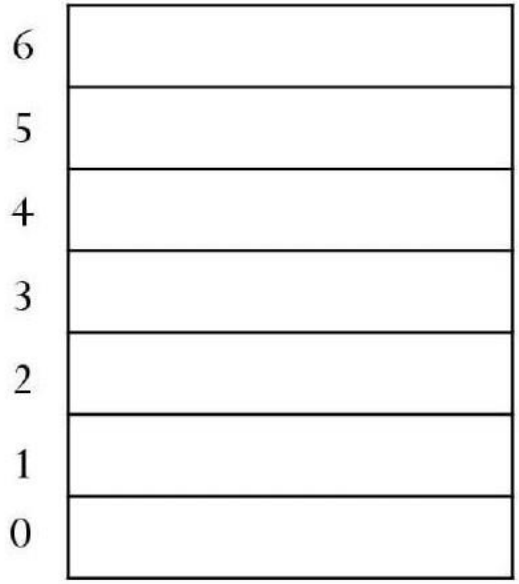
Postfix Expression : 10 500 20 2 * 4 / / -



10 - 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -

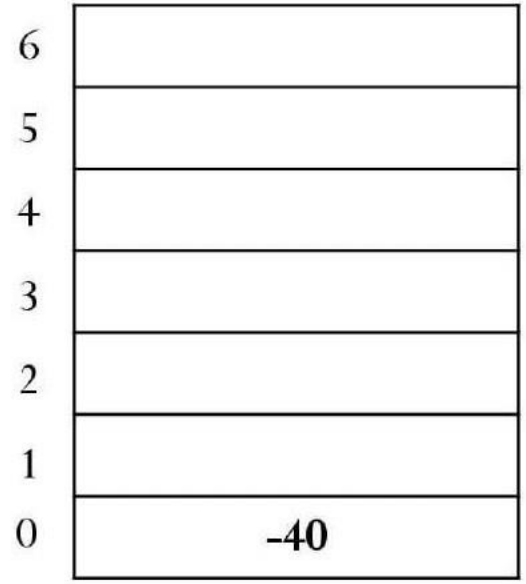


S

10 - 50 = -40

Infix Expression : 10 – 500 / (20 * 2 / 4)

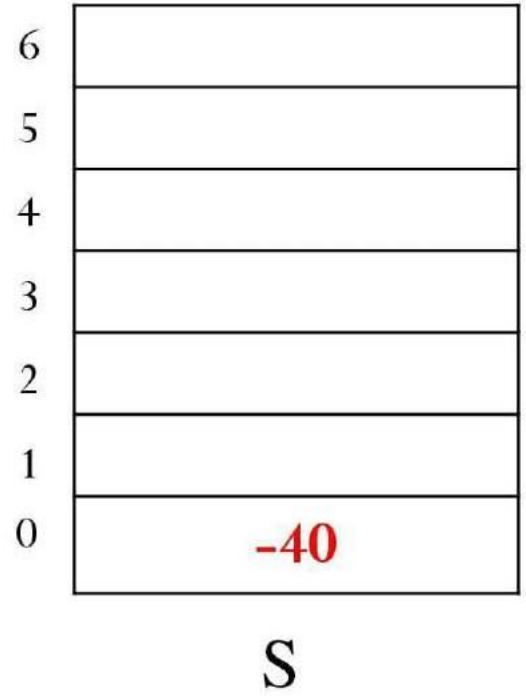
Postfix Expression : 10 500 20 2 * 4 / / -



10 - 50 = -40

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : 10 500 20 2 * 4 / / -



POSTFIX EVALUATION - ALGORITHM

Algorithm postfix_eval(P)

```
{       for i=0 to length(P)-1 do
        {       if P[i] is an operand then
                        PUSH P[i] to S
                else if P[i] is an operator then
                {       op2=POP()
                        op1=POP()
                        result = op1  P[i]  op2
                        PUSH  result in to S
                }
        }
        Print S[0]
}
```

PREFIX EXPRESSION EVALUATION - ALGORITHM

Infix Expression : 10 – 500 / (20 * 2 / 4)

Infix Expression : 10 – 500 / (20 * 2 / 4)

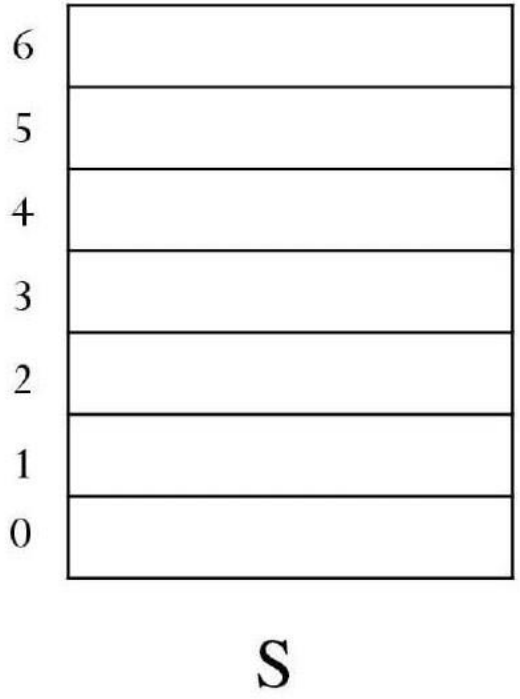
Postfix Expression : - 10 / 500 / * 20 2 4

Infix Expression : 10 – 500 / (20 * 2 / 4)
Postfix Expression : - 10 / 500 / * 20 2 4
Reverse the Expression:4 2 20 * / 500 / 10 -

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

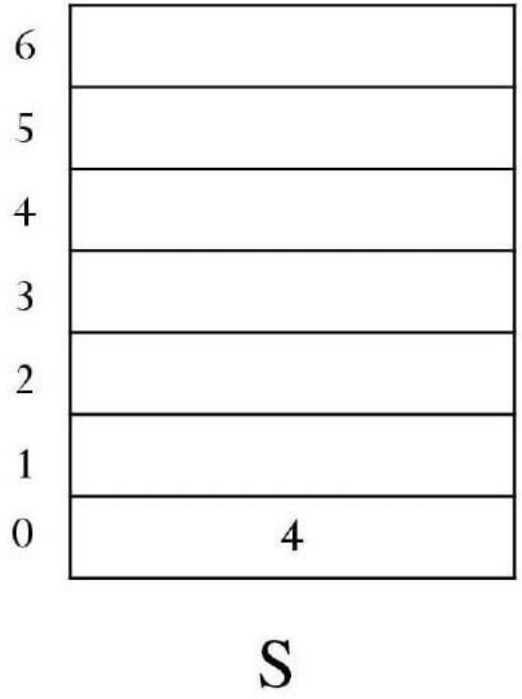
Reverse the Expression: 4 2 20 * / 500 / 10 -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

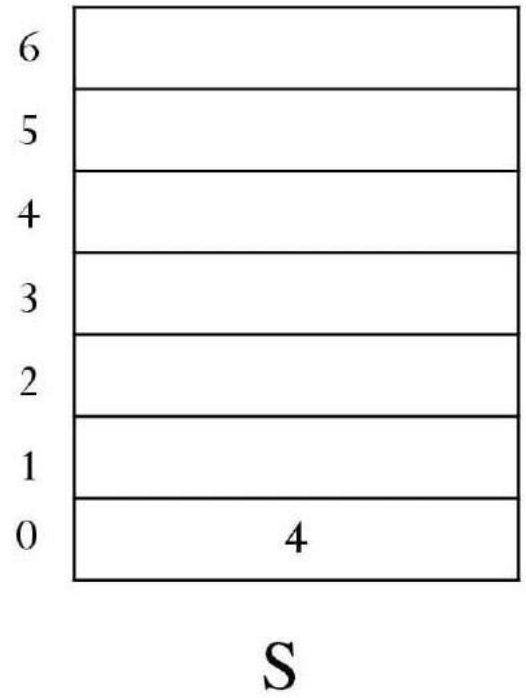
Reverse the Expression: 4 2 20 * / 500 / 10 -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

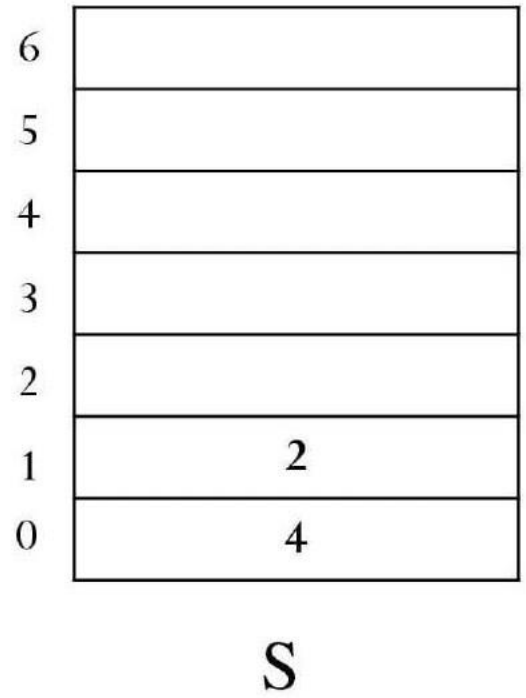
Reverse the Expression:4 2 20 * / 500 / 10 -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

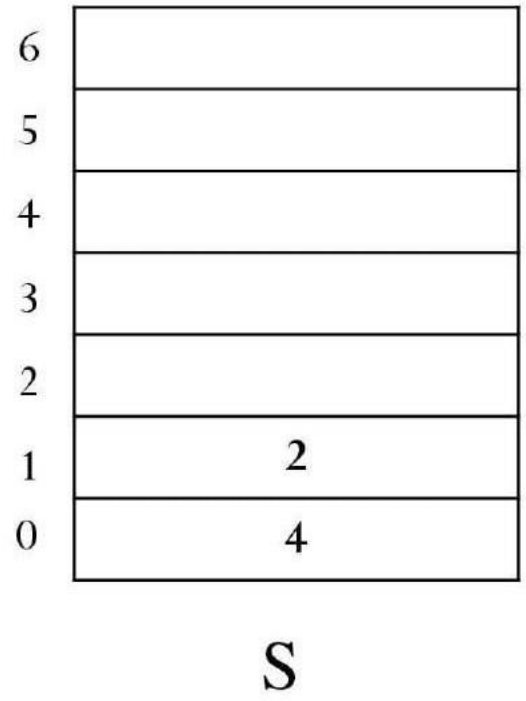
Reverse the Expression:4 2 20 * / 500 / 10 -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

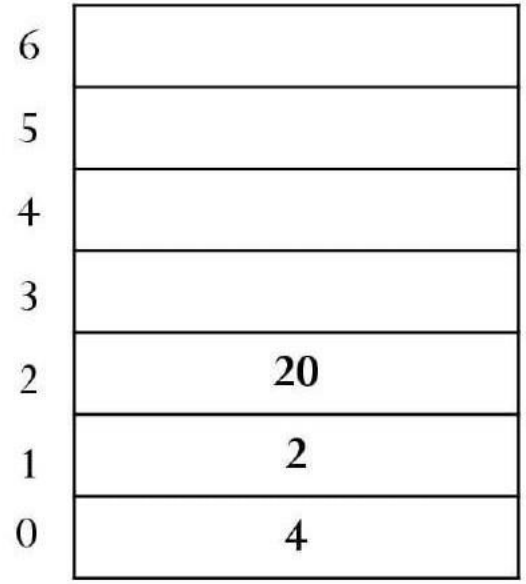
Reverse the Expression:4 2 20 * / 500 / 10 -



Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

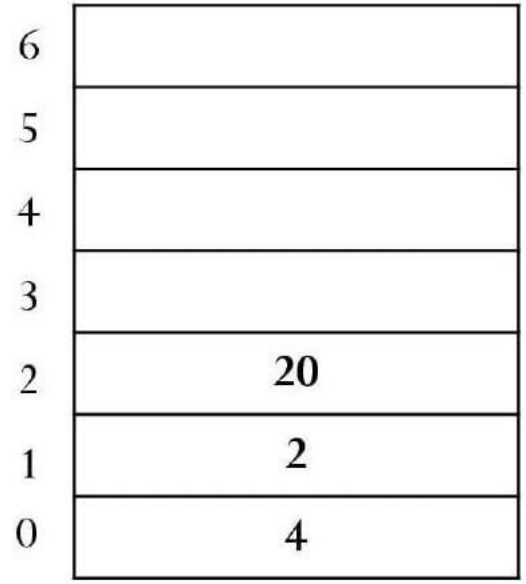


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

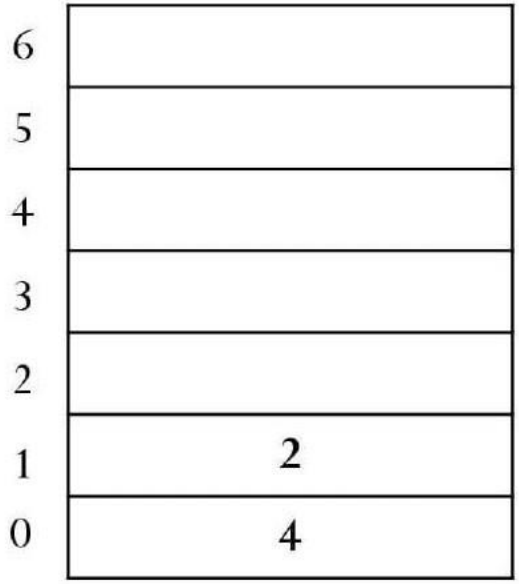


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



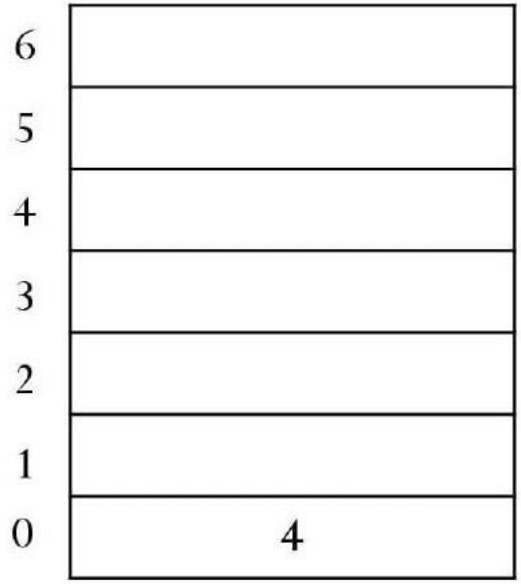
S

20 *

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



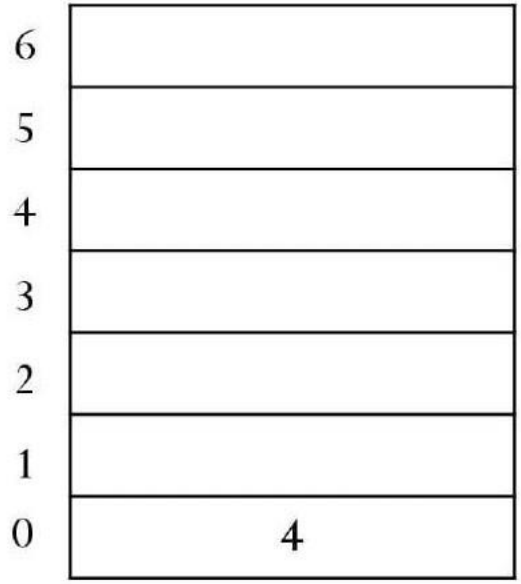
S

20 * 2

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression: 4 2 20 * / 500 / 10 -

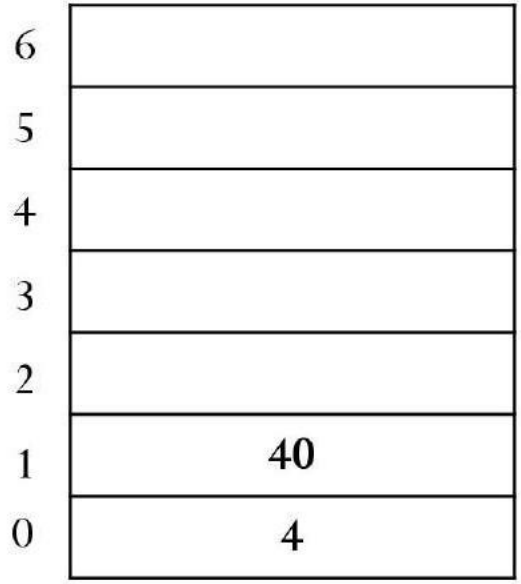


20 * 2 = 40

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression: 4 2 20 * / 500 / 10 -

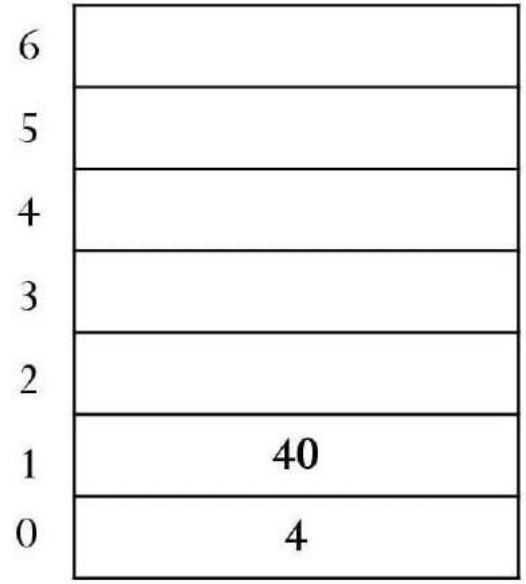


20 * 2 = 40

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

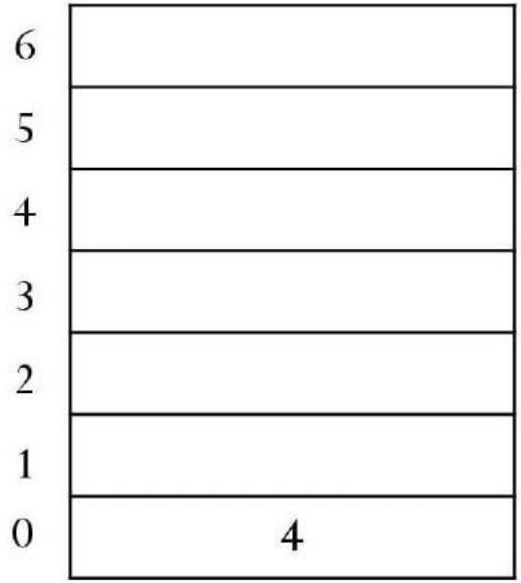


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



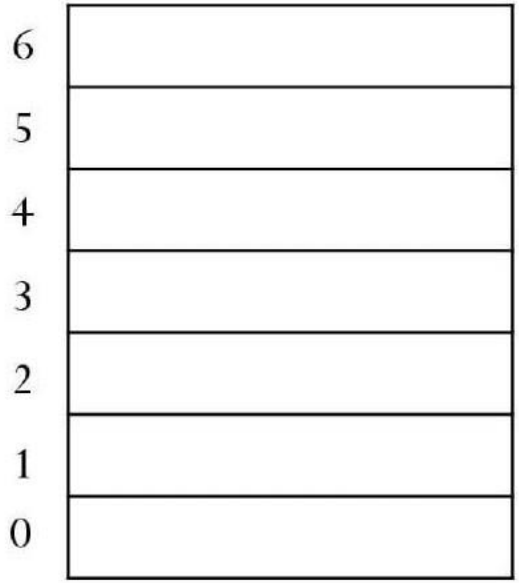
S

40 /

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



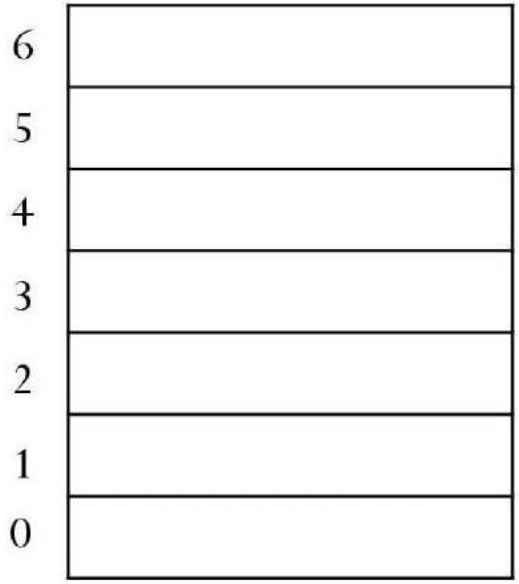
S

40 / 4

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression: 4 2 20 * / 500 / 10 -

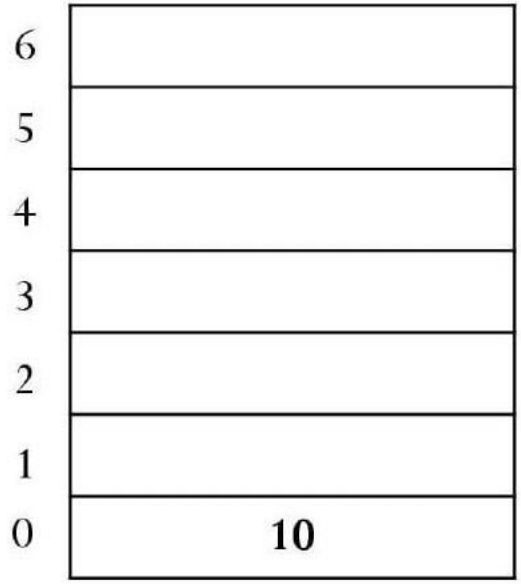


40 / 4 = 10

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

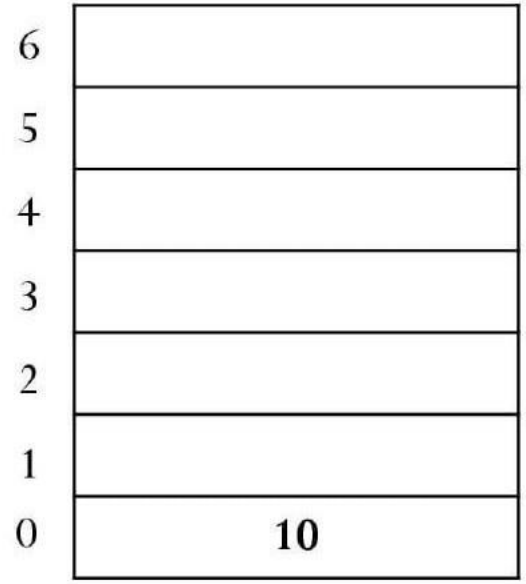


40 / 4 = 10

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

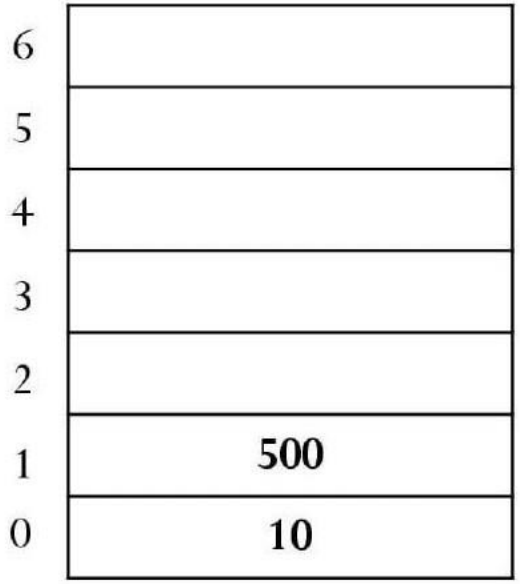


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

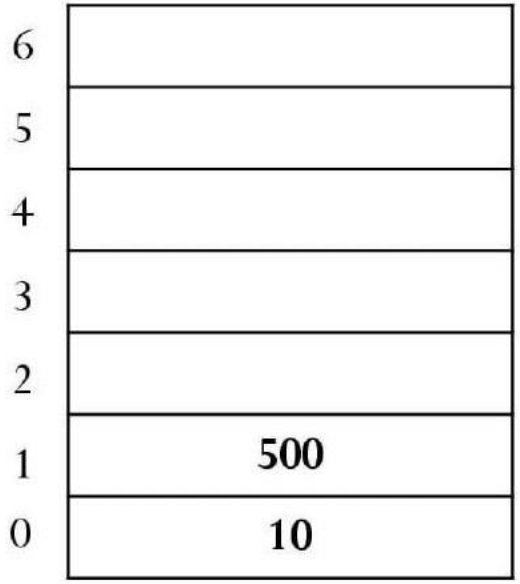


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

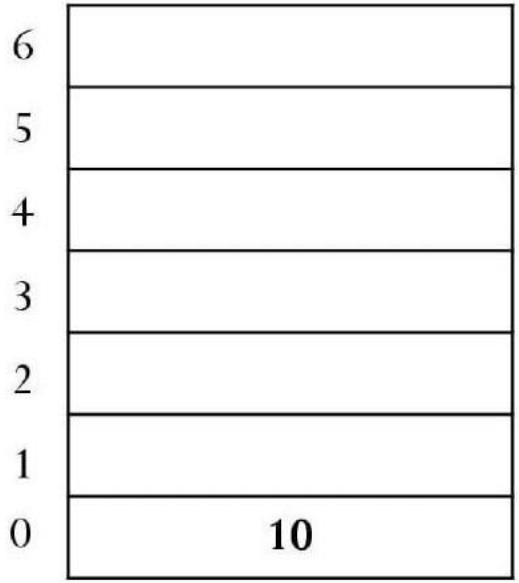


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



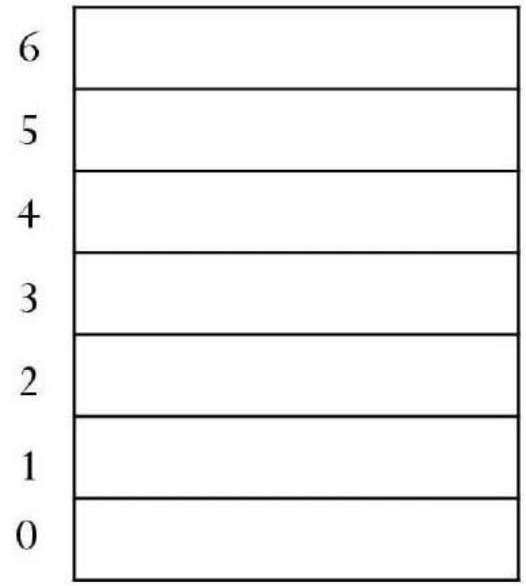
S

500 /

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



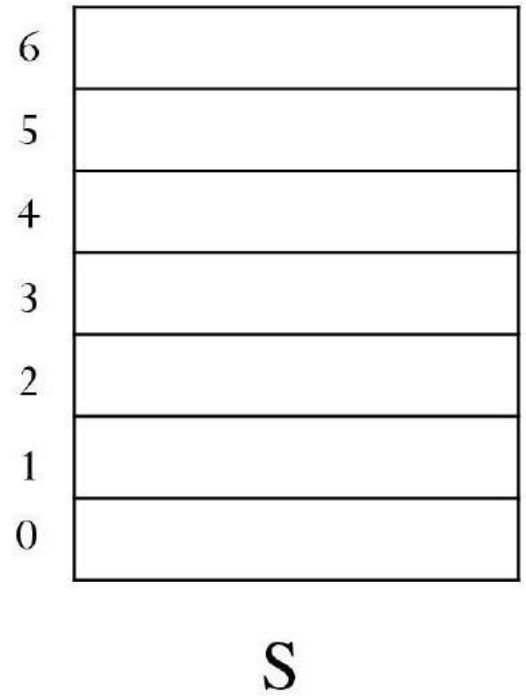
S

500 / 10

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

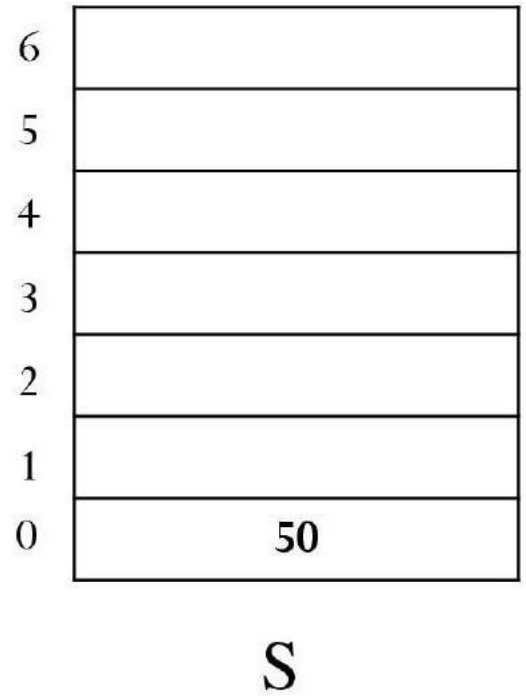


500 / 10 = 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

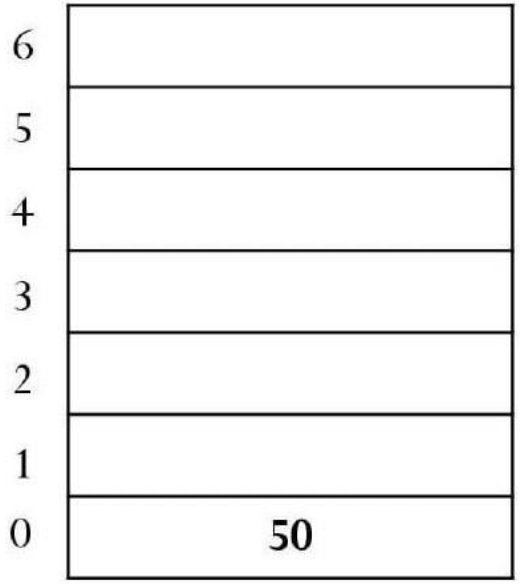


500 / 10 = 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

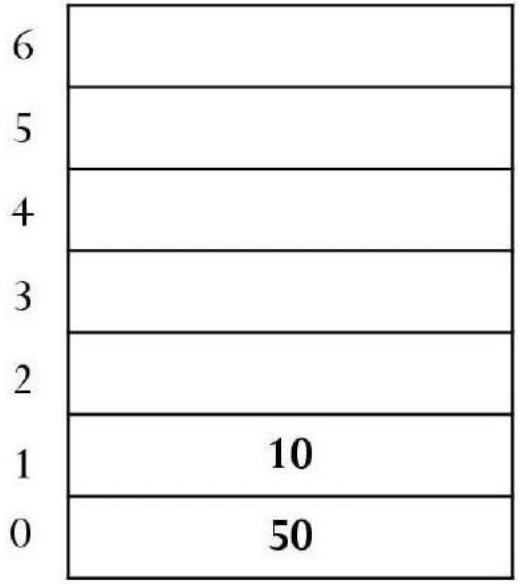


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

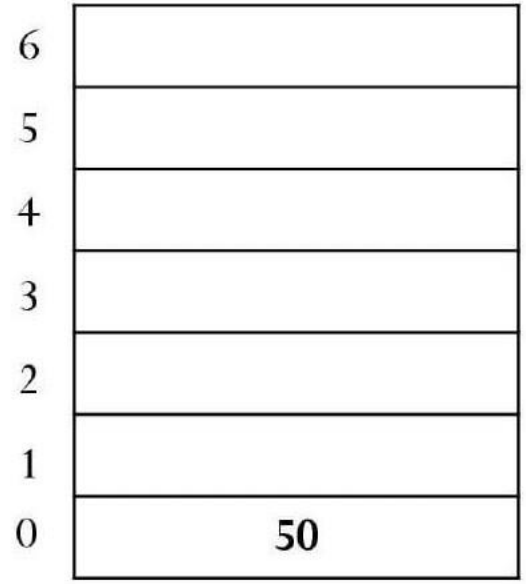


S

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



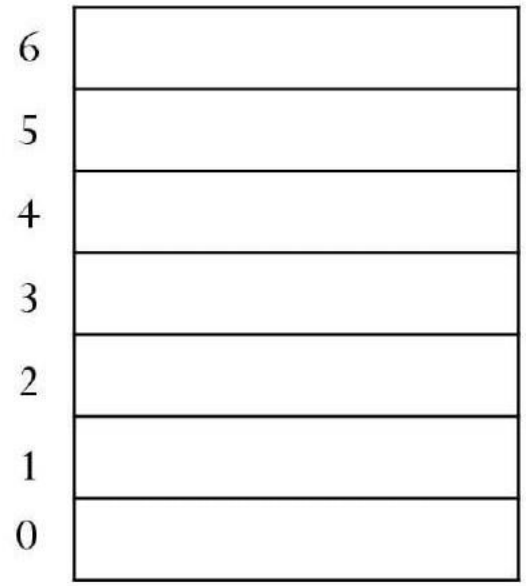
S

10 -

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

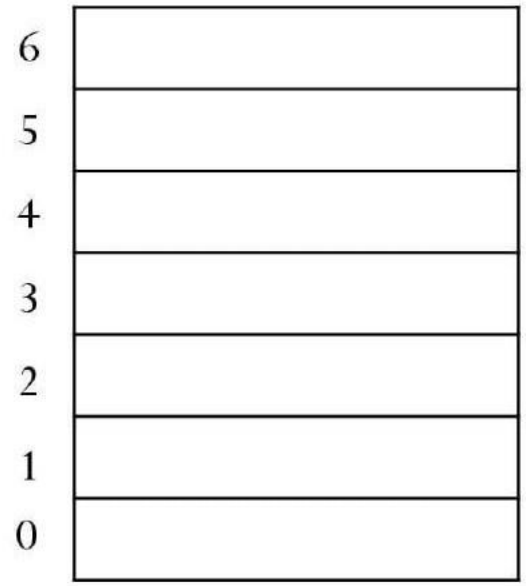


10 - 50

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

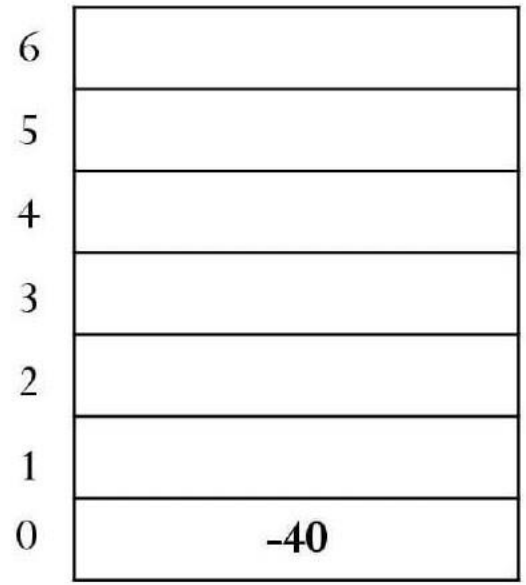


10 - 50 = -40

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -

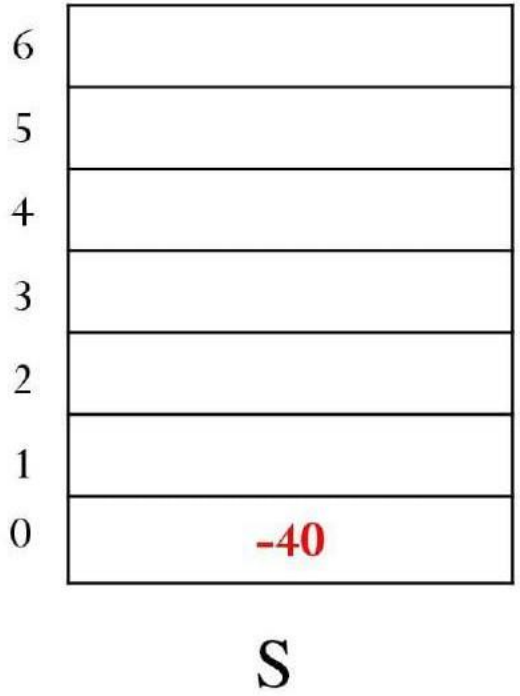


10 - 50 = -40

Infix Expression : 10 – 500 / (20 * 2 / 4)

Postfix Expression : - 10 / 500 / * 20 2 4

Reverse the Expression:4 2 20 * / 500 / 10 -



PREFIX EVALUATION - ALGORITHM

Algorithm prefix_eval(P)

```
{       for i= length(P)-1 to 0 do
        {       if P[i] is an operand then
                        PUSH P[i] to S
                else if P[i] is an operator then
                {       op1=POP()
                        op2=POP()
                        result = op1  P[i]  op2
                        PUSH result in to S
                }
        }
        Print S[0]
}
```

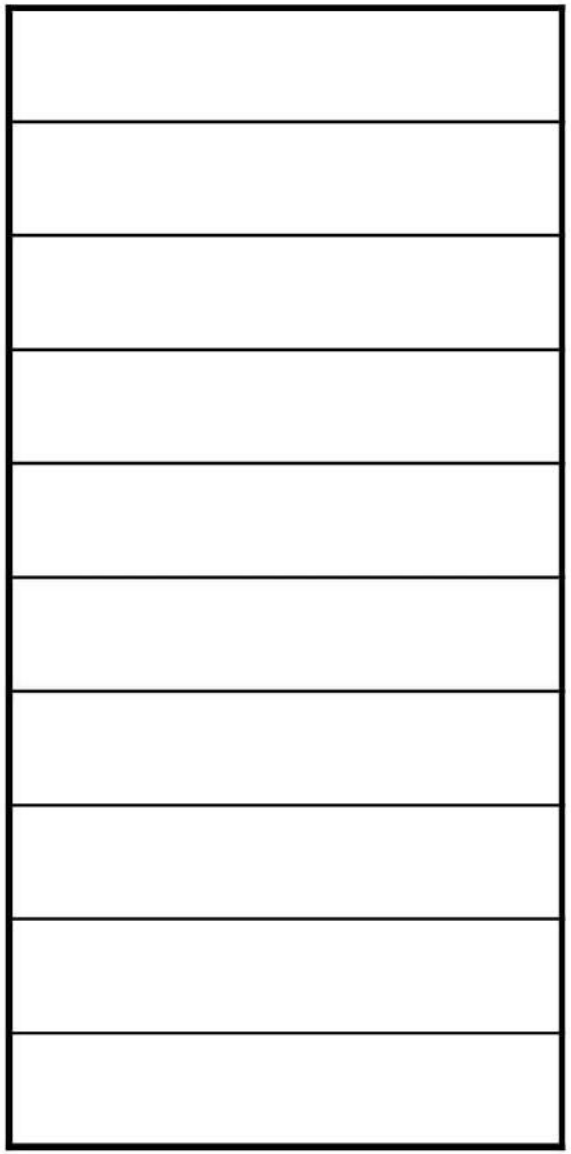
APPLICATIONS OF STACK

1. Evaluation of arithmetic expressions
2. **Implementation of Recursion**
3. String reversal

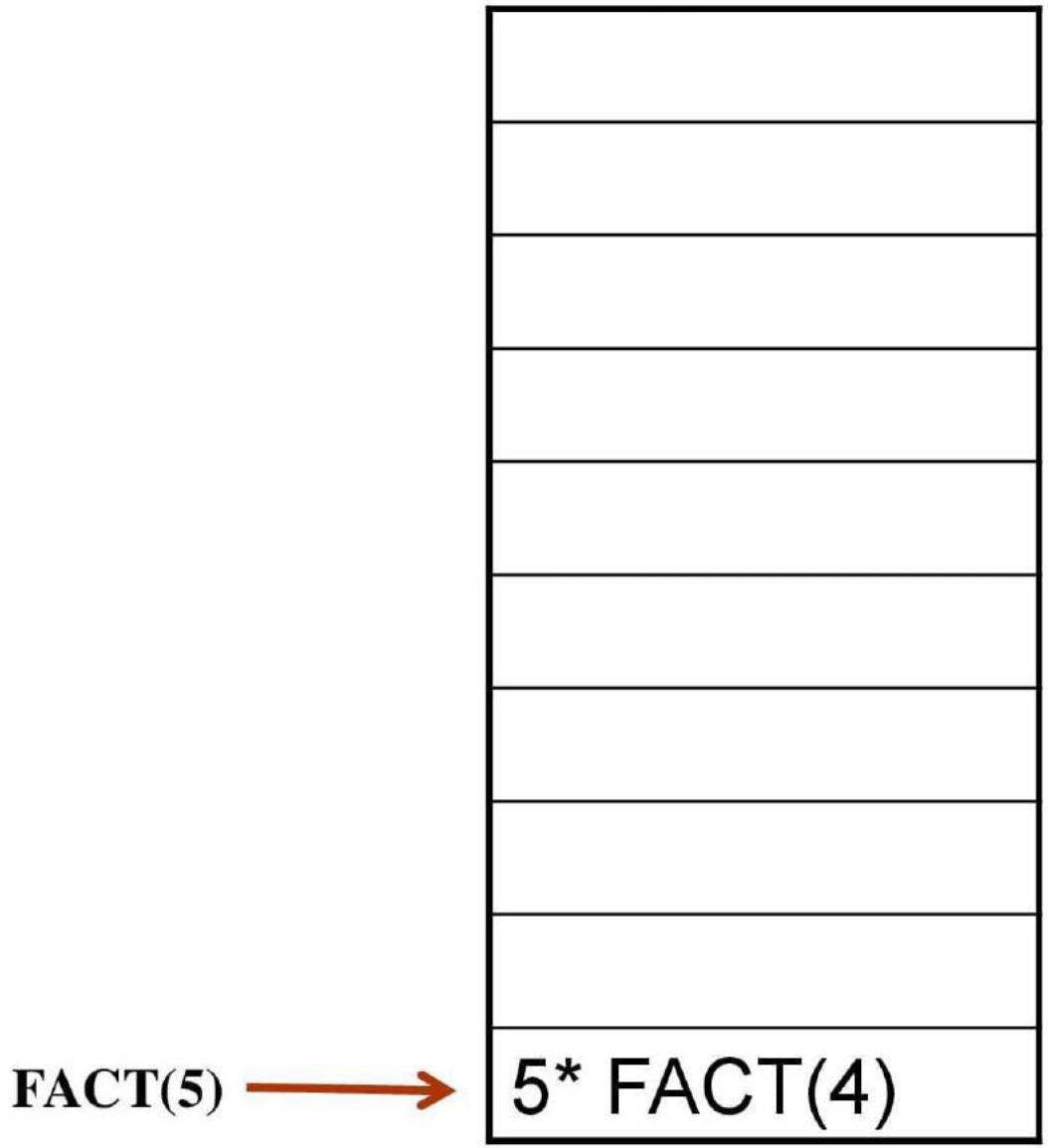
Program to find the factorial of a number using recursion

```
int FACT(int n)
{
    if (n == 0 || n == 1)
            return 1;
    else
            return  n * FACT(n-1);
}
```

FACT(5)= 5*FACT (4)

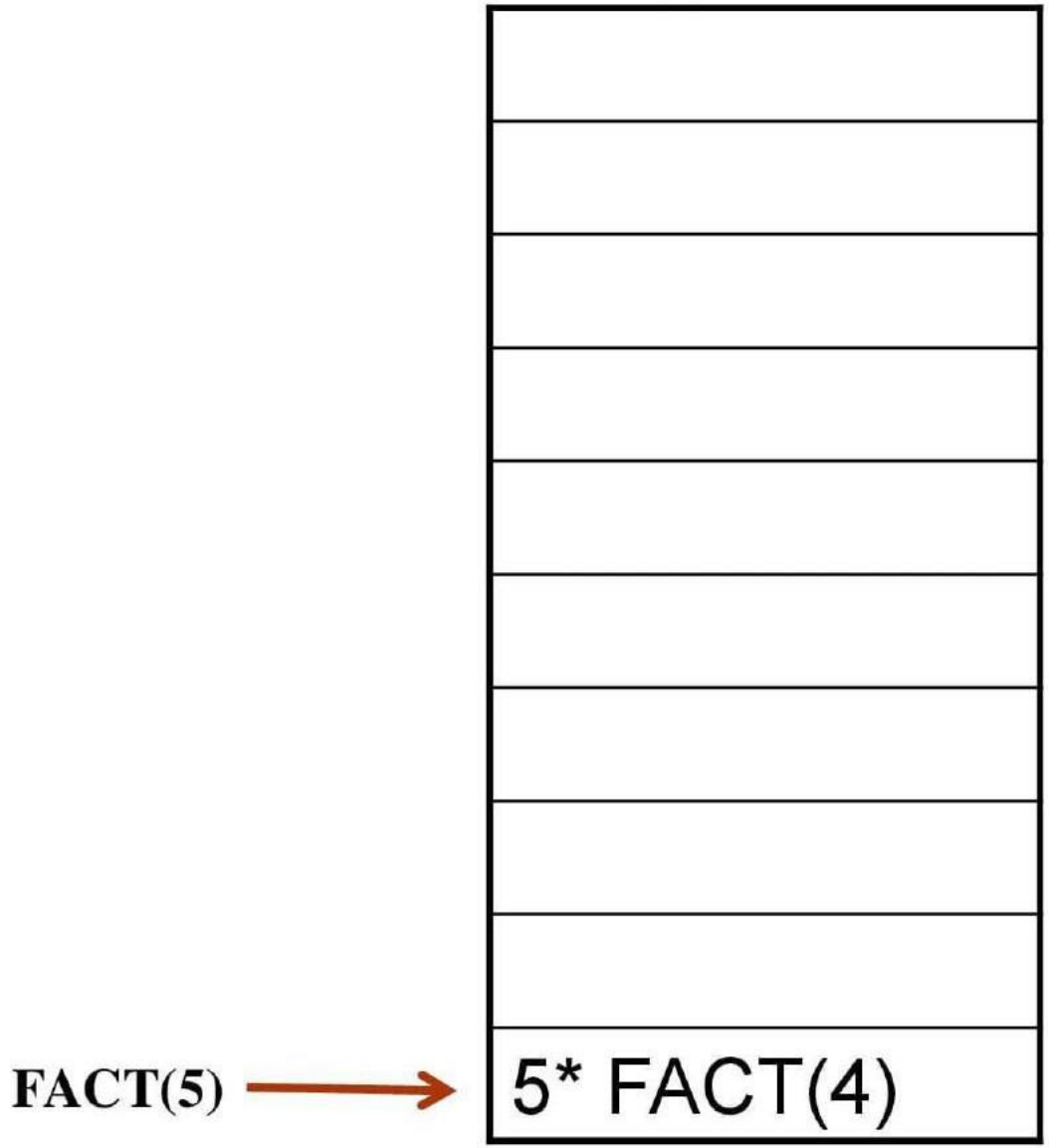


FACT(5)= 5*FACT (4)



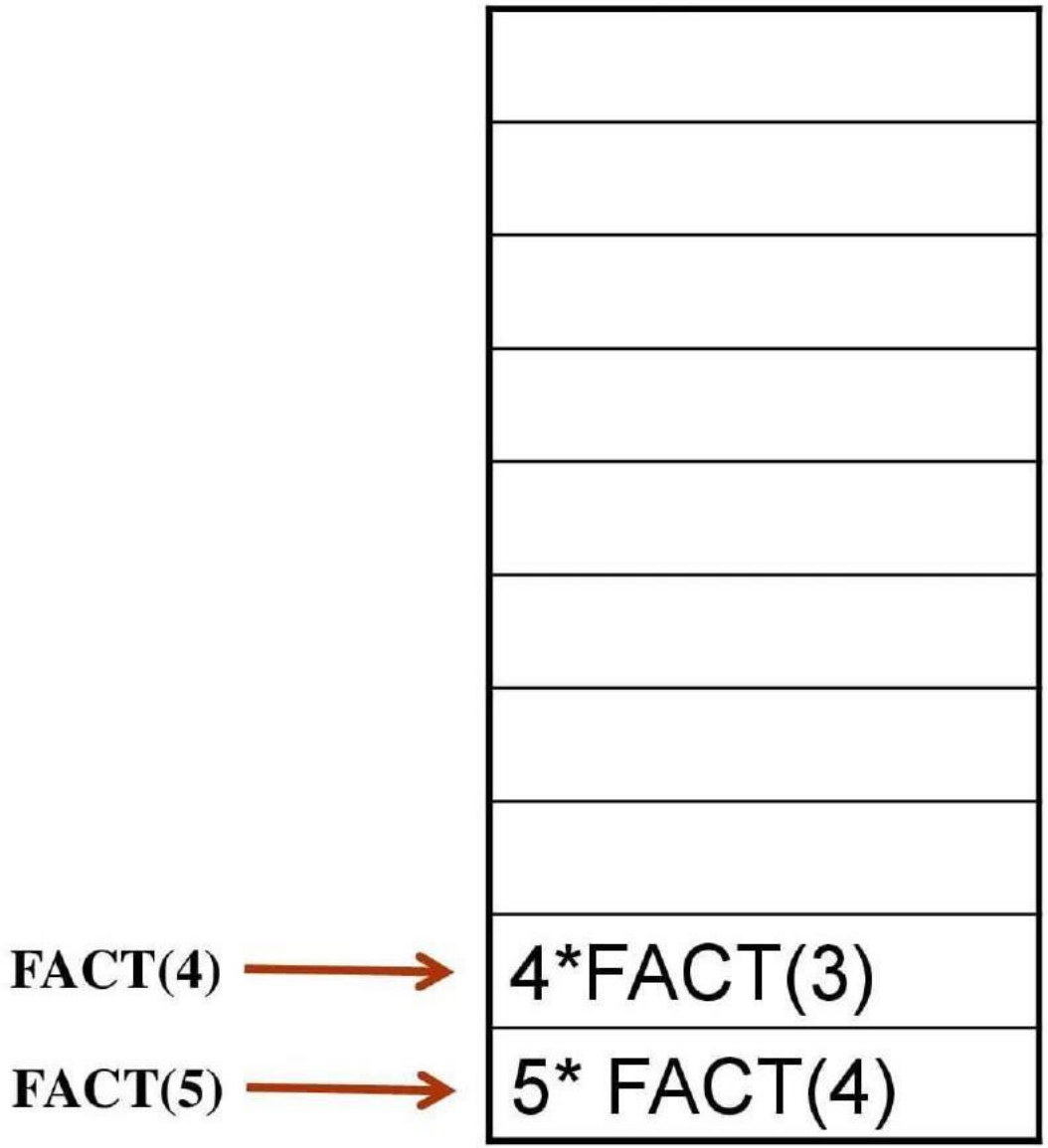
FACT(5)= 5*FACT (4)

=4*FACT(3)



FACT(5)= 5*FACT (4)

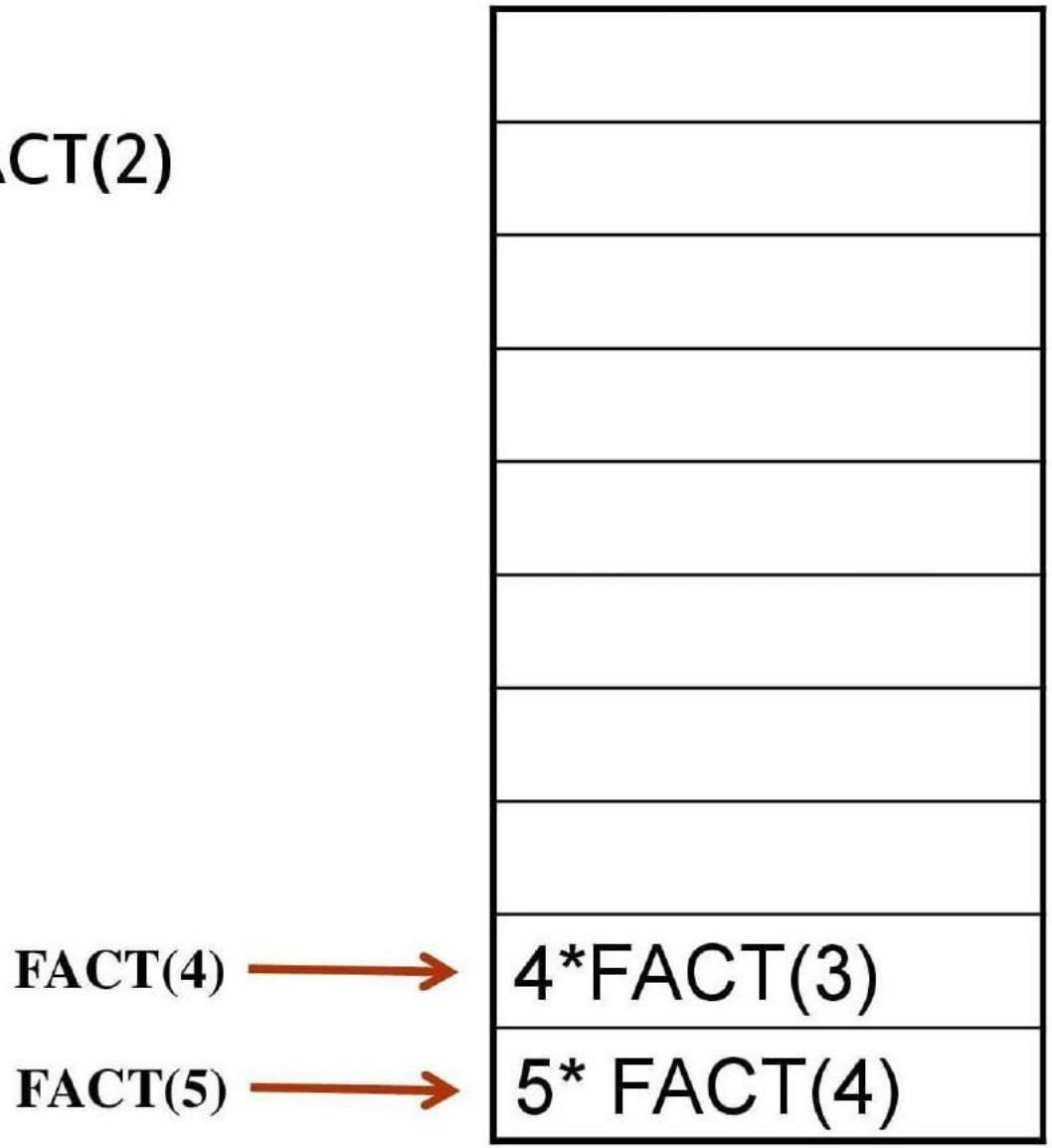
=4*FACT(3)



FACT(5)= 5*FACT (4)

=4*FACT(3)

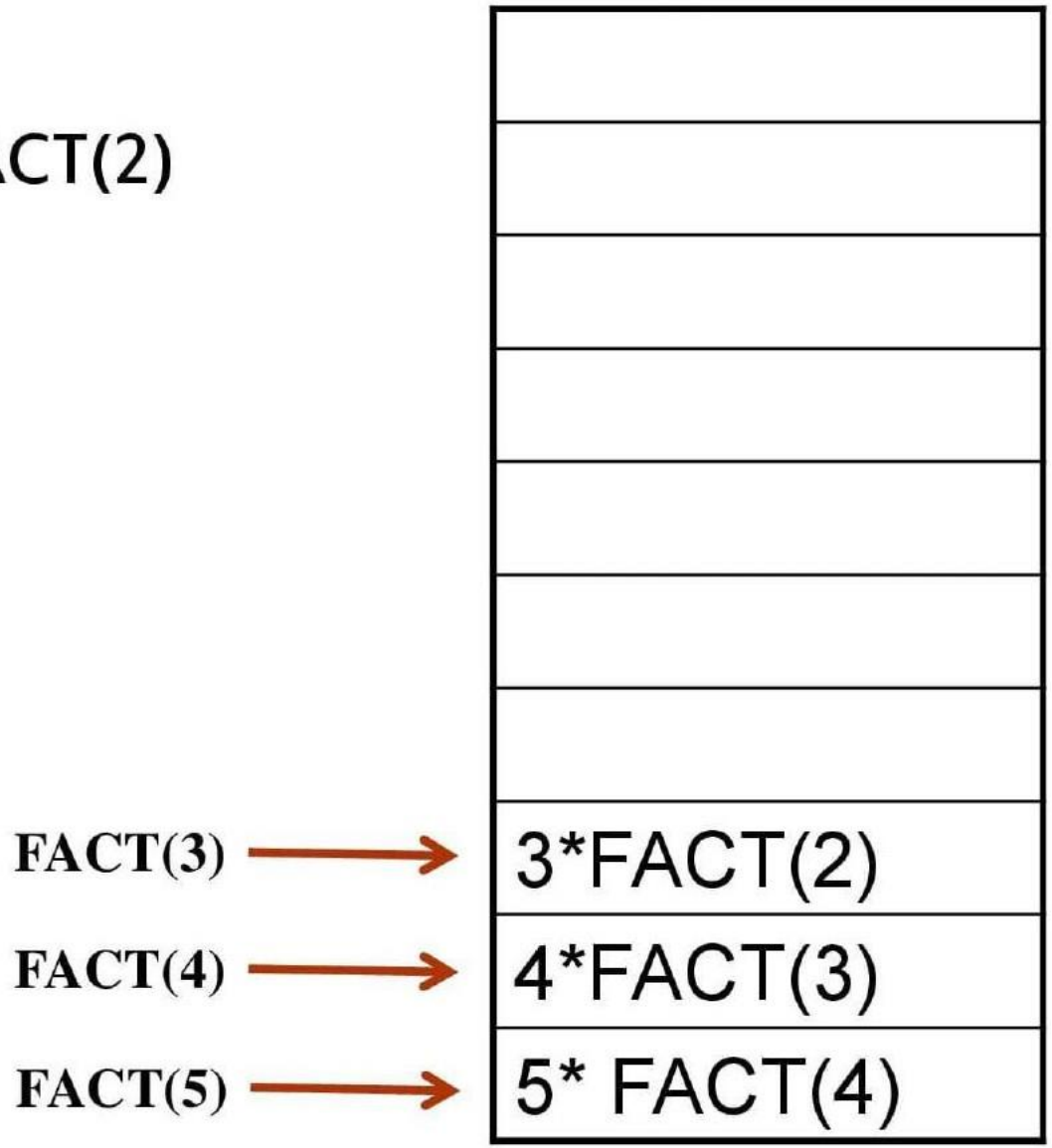
=3*FACT(2)



FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

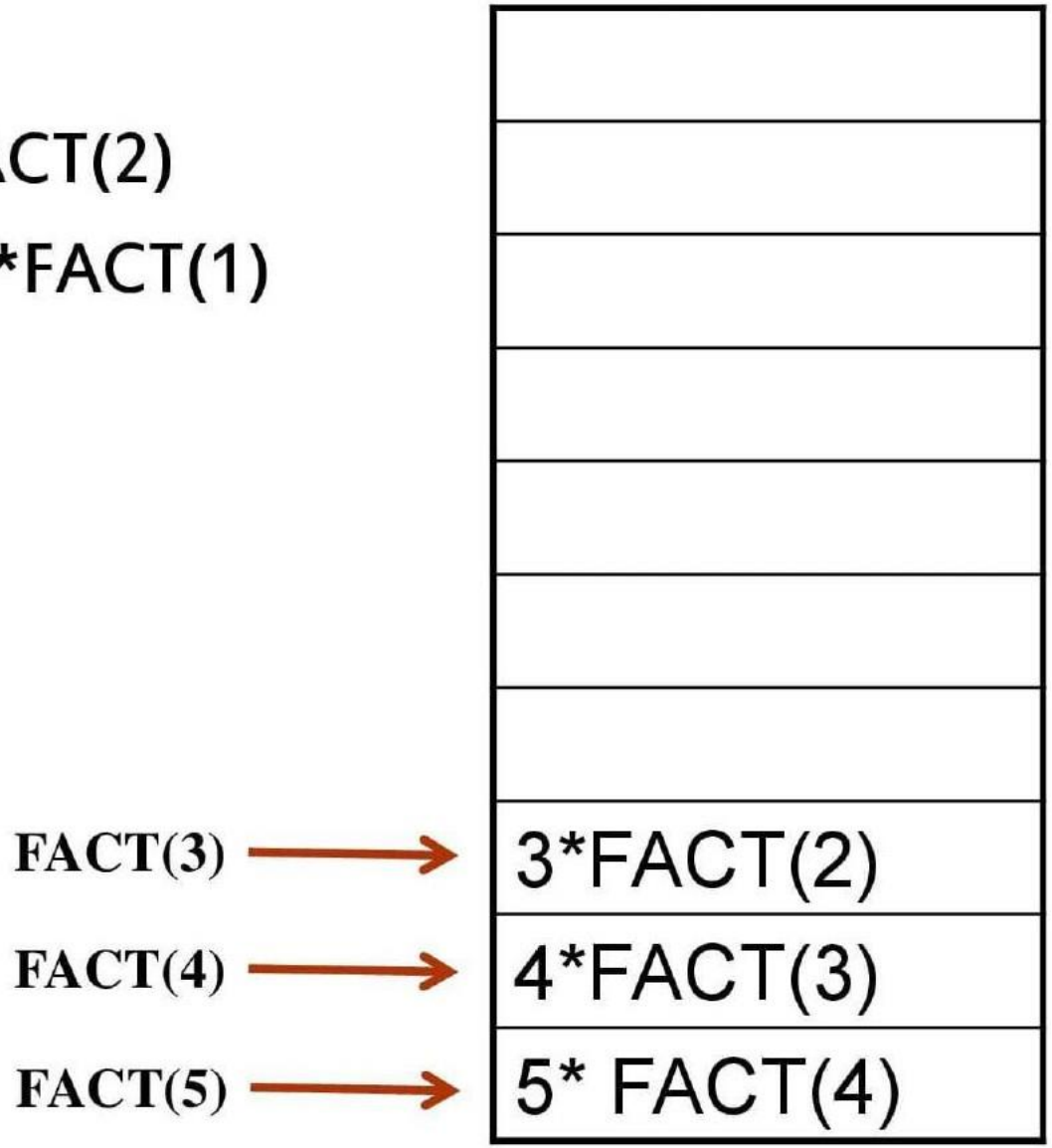


FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

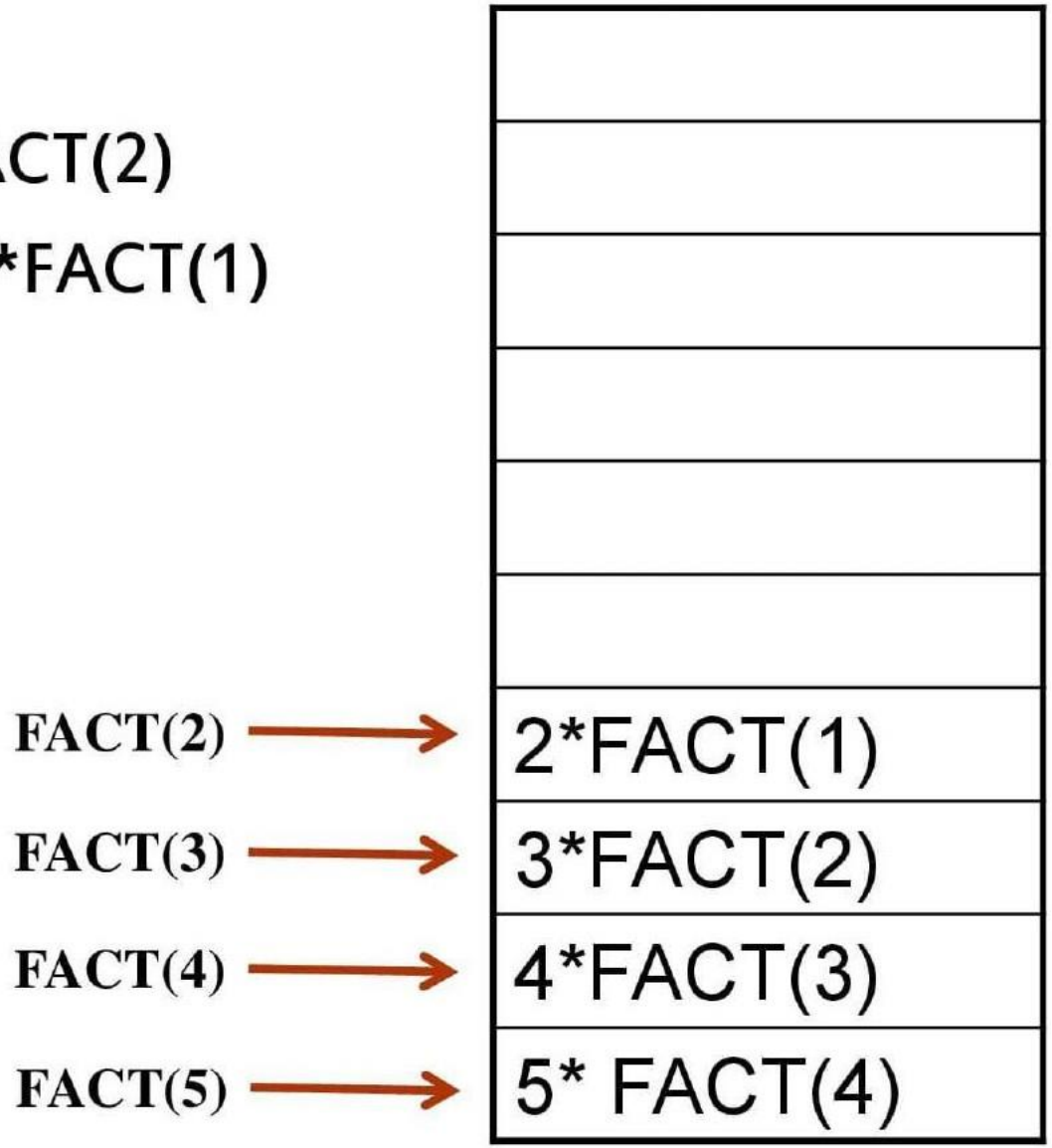


FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)



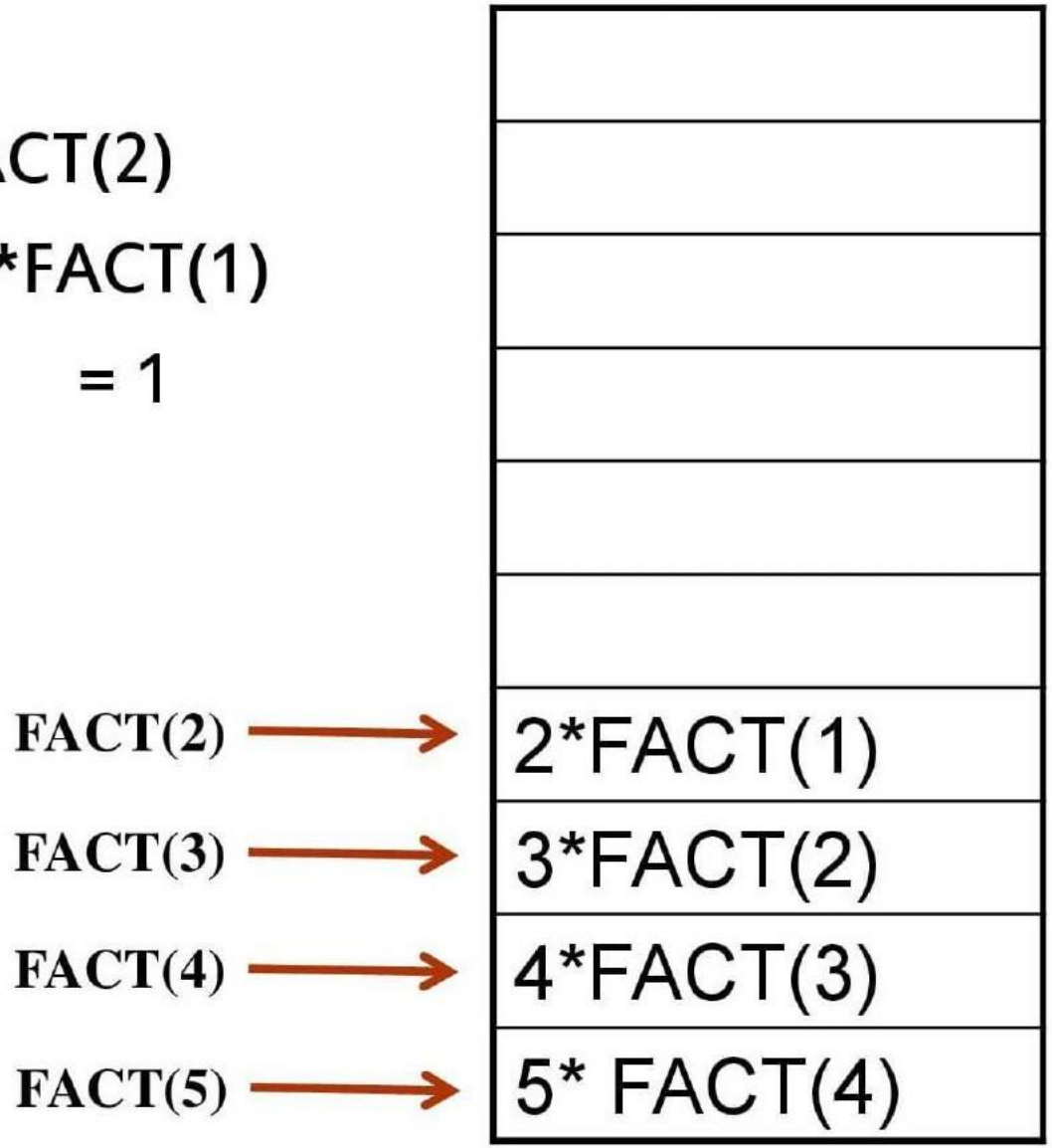
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



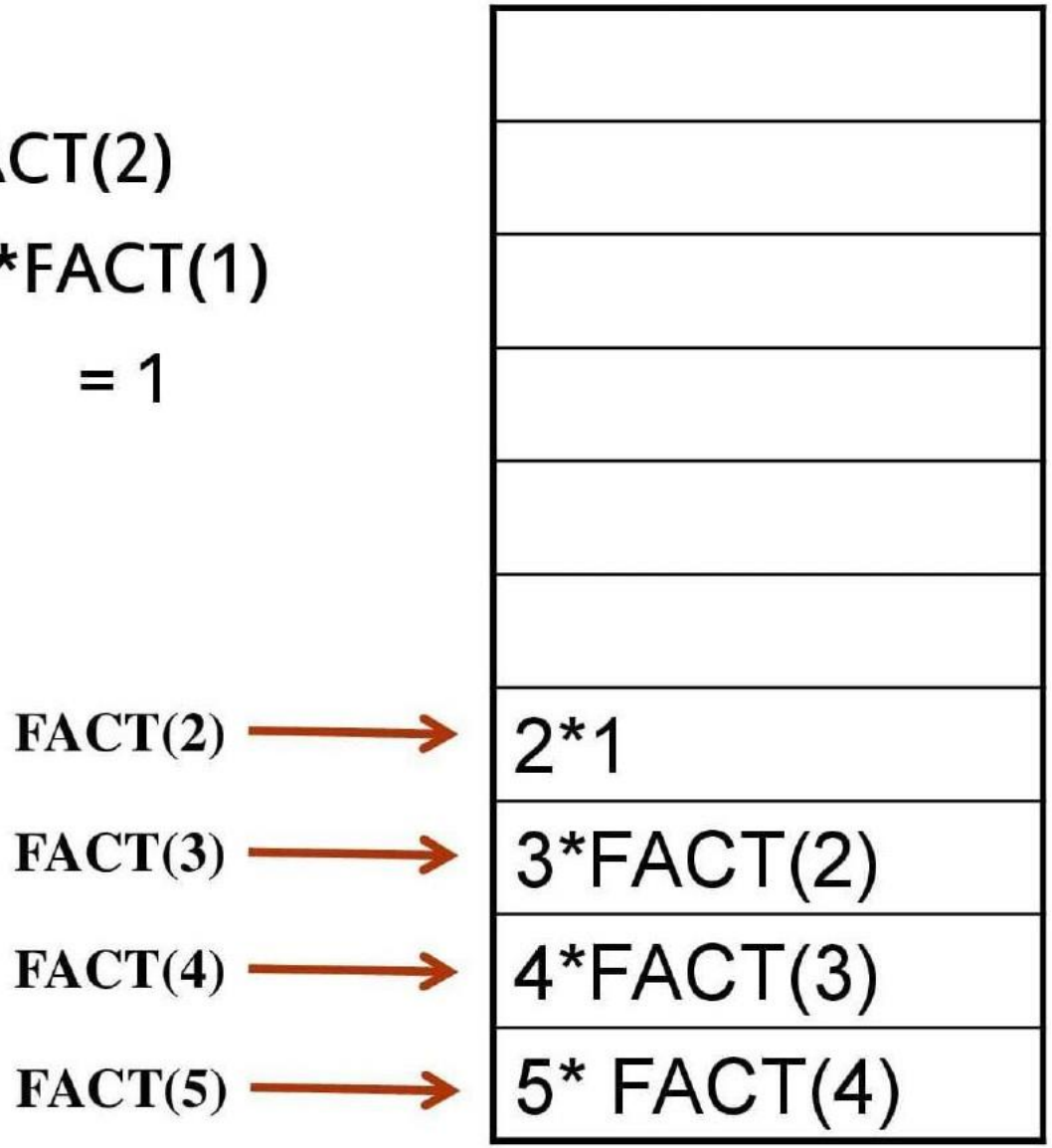
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



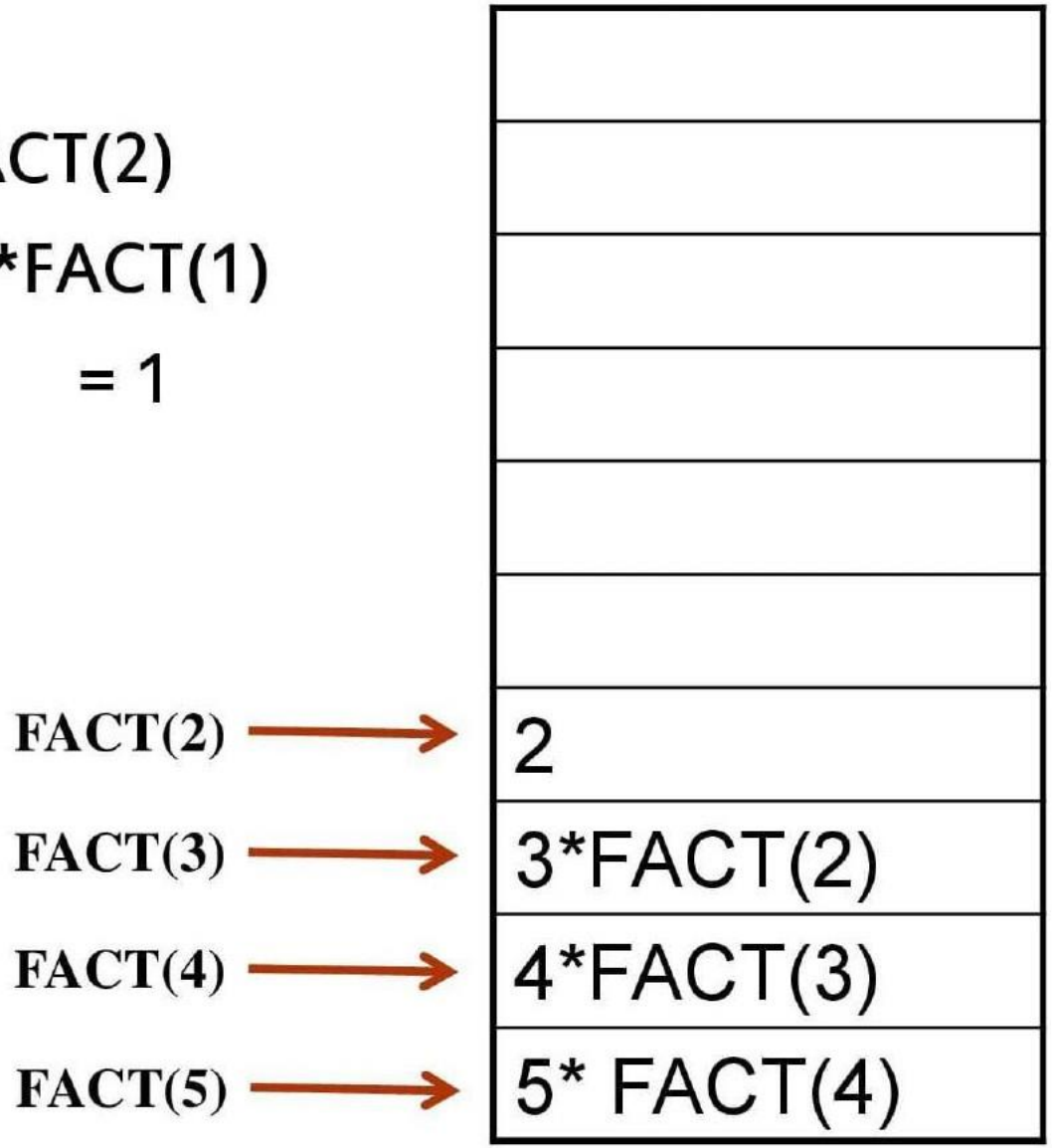
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



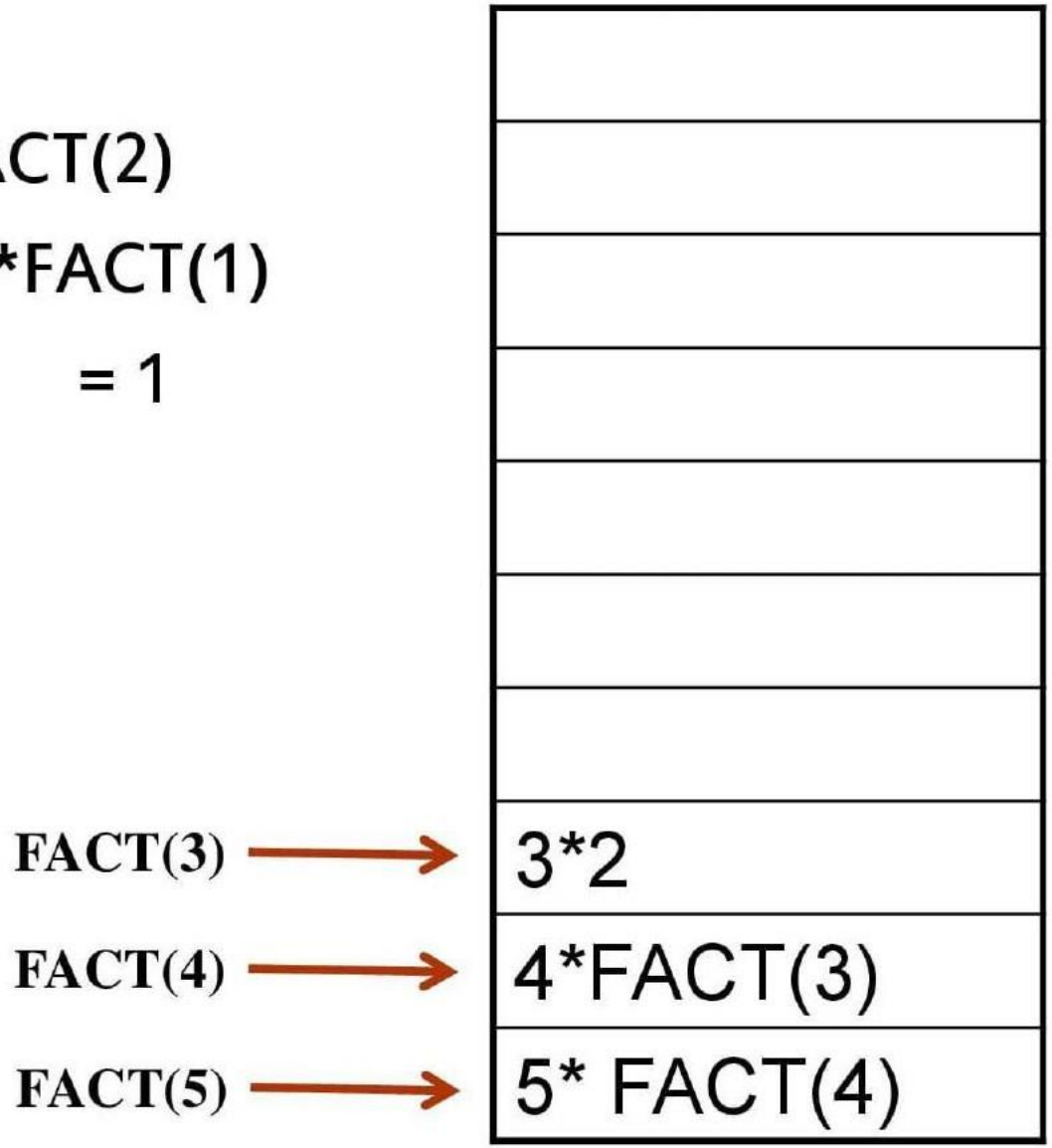
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



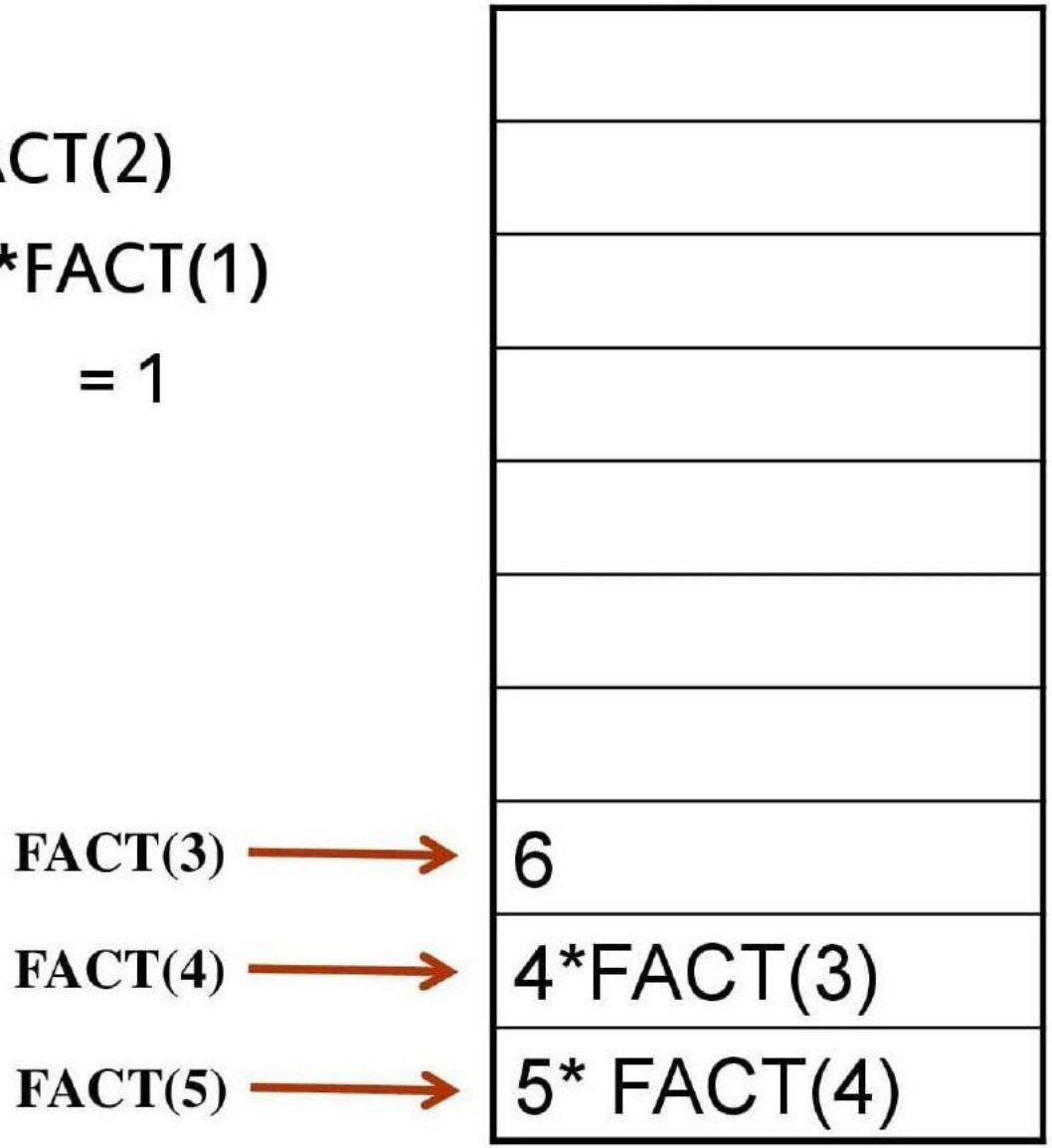
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



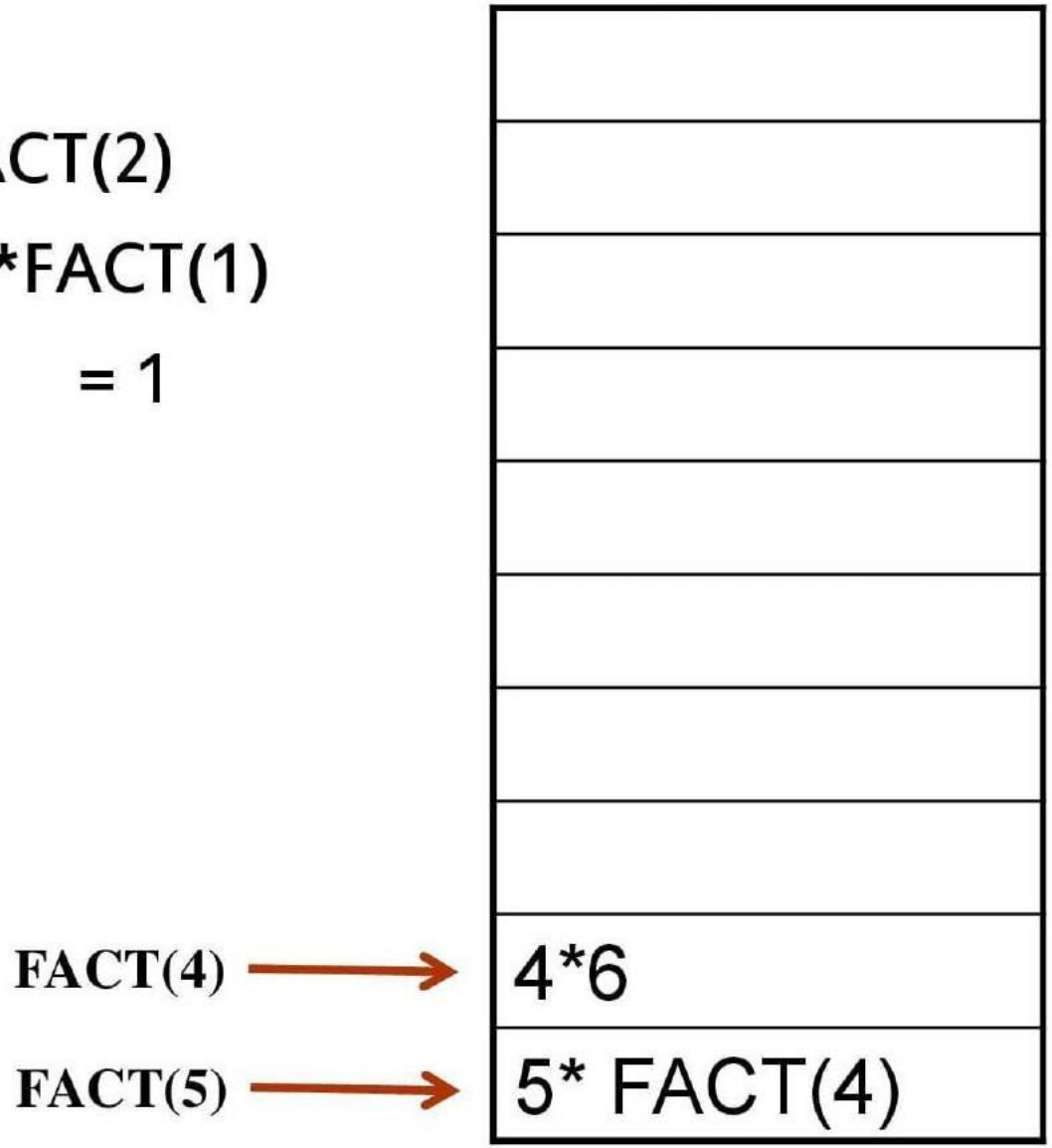
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



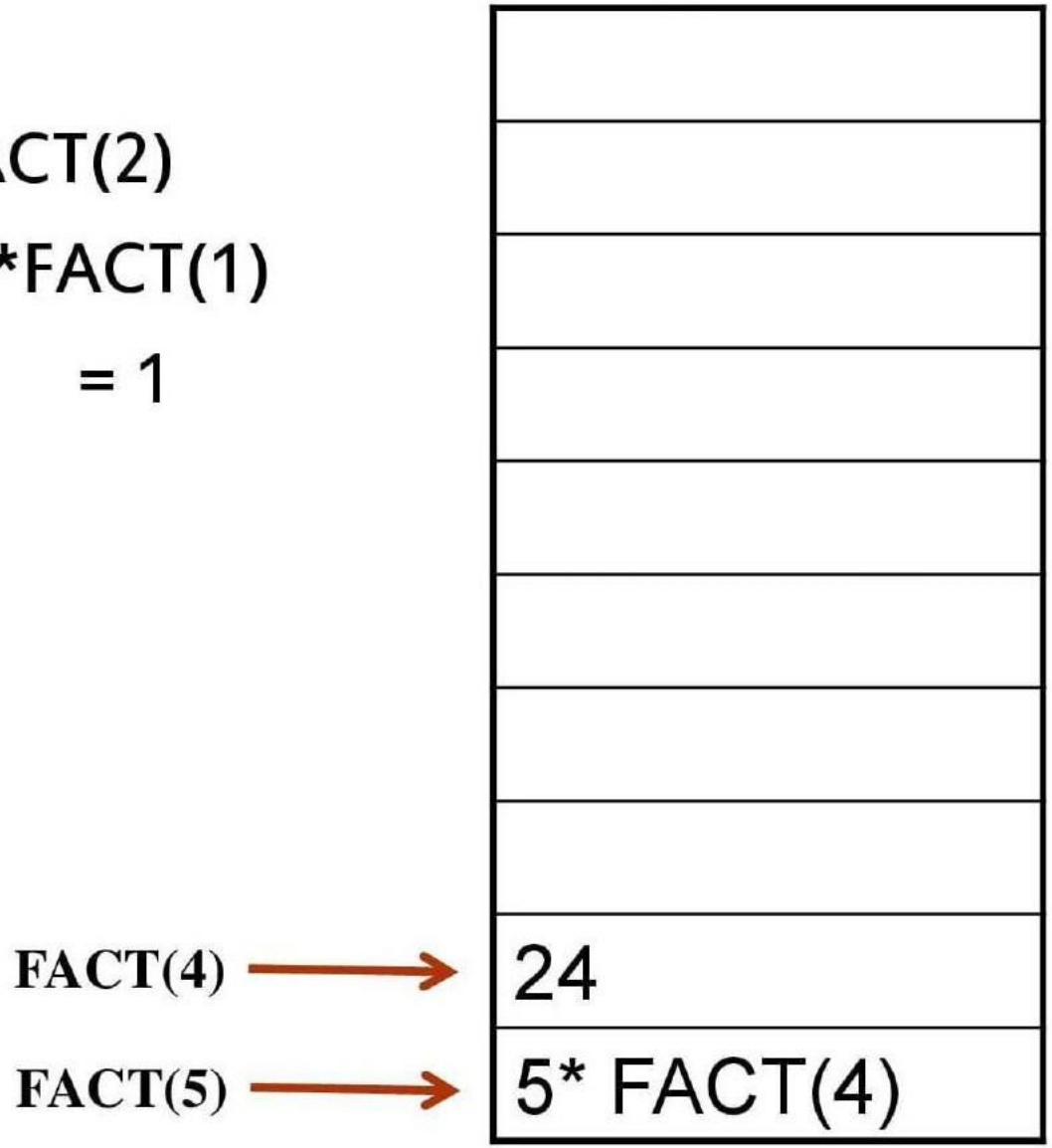
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



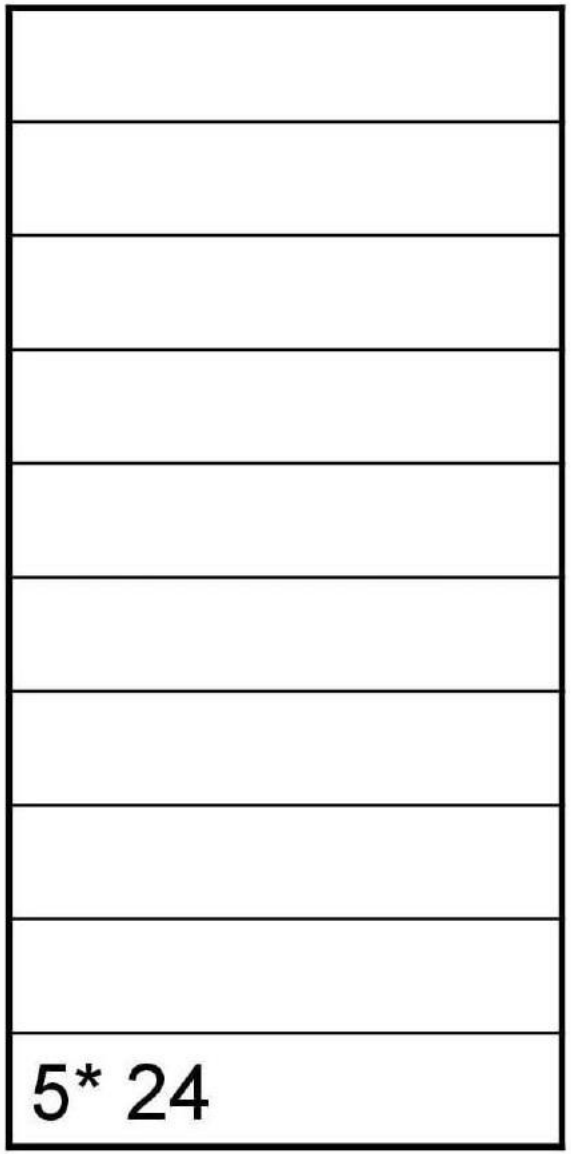
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

=2*FACT(1)

= 1



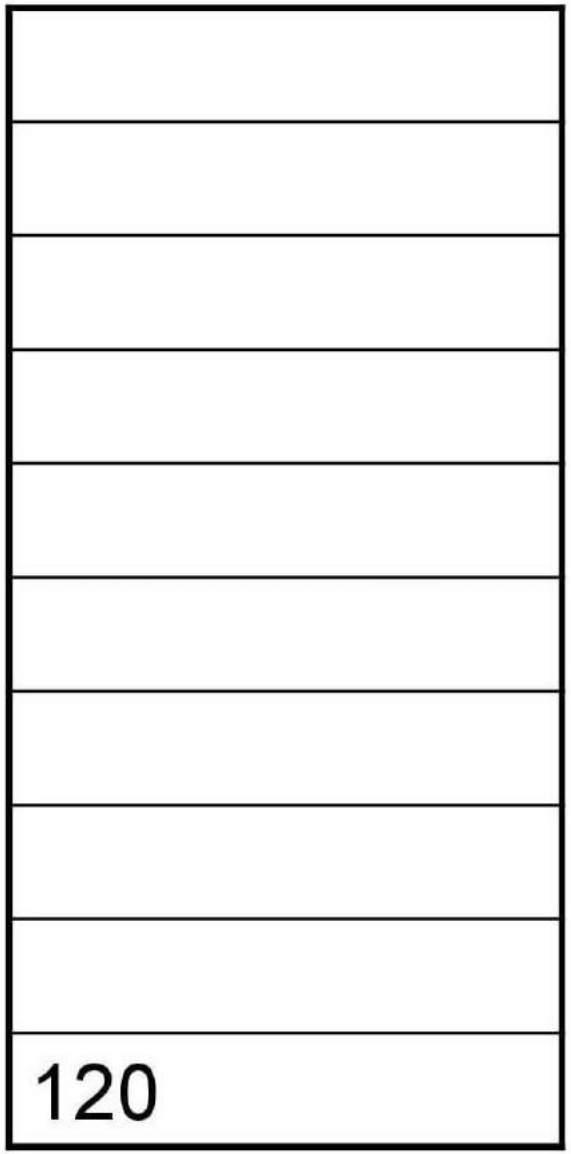
FACT(5)= 5*FACT (4)

=4*FACT(3)

=3*FACT(2)

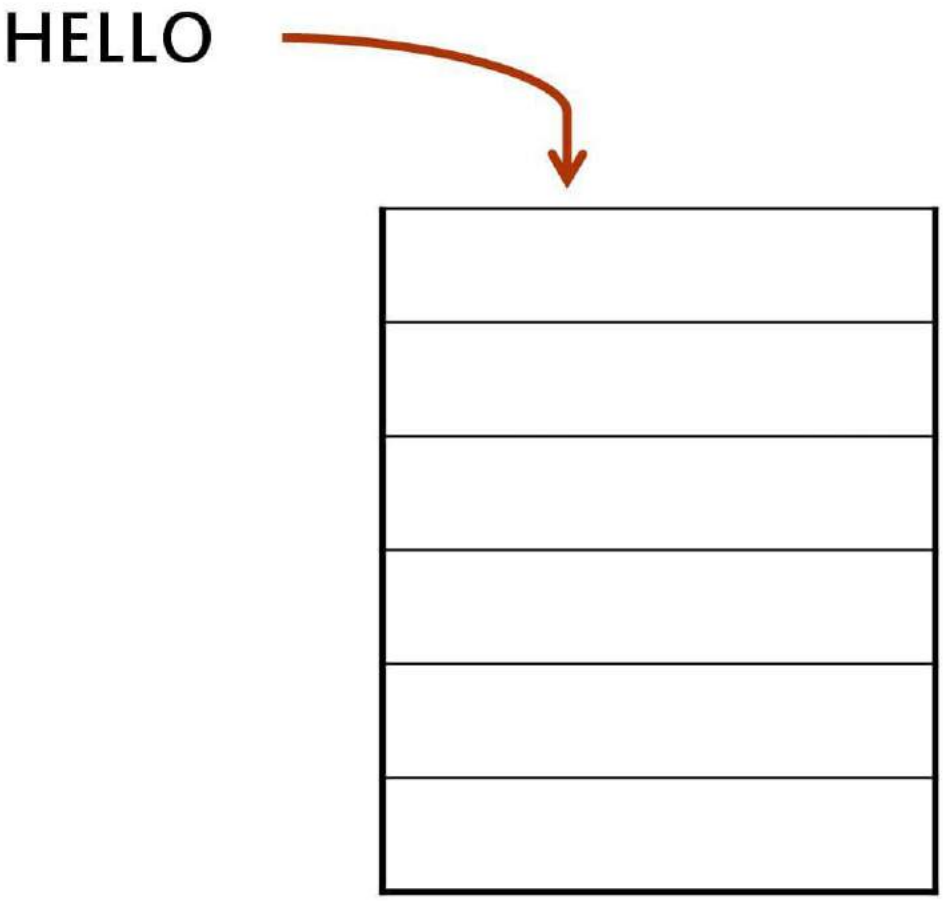
=2*FACT(1)

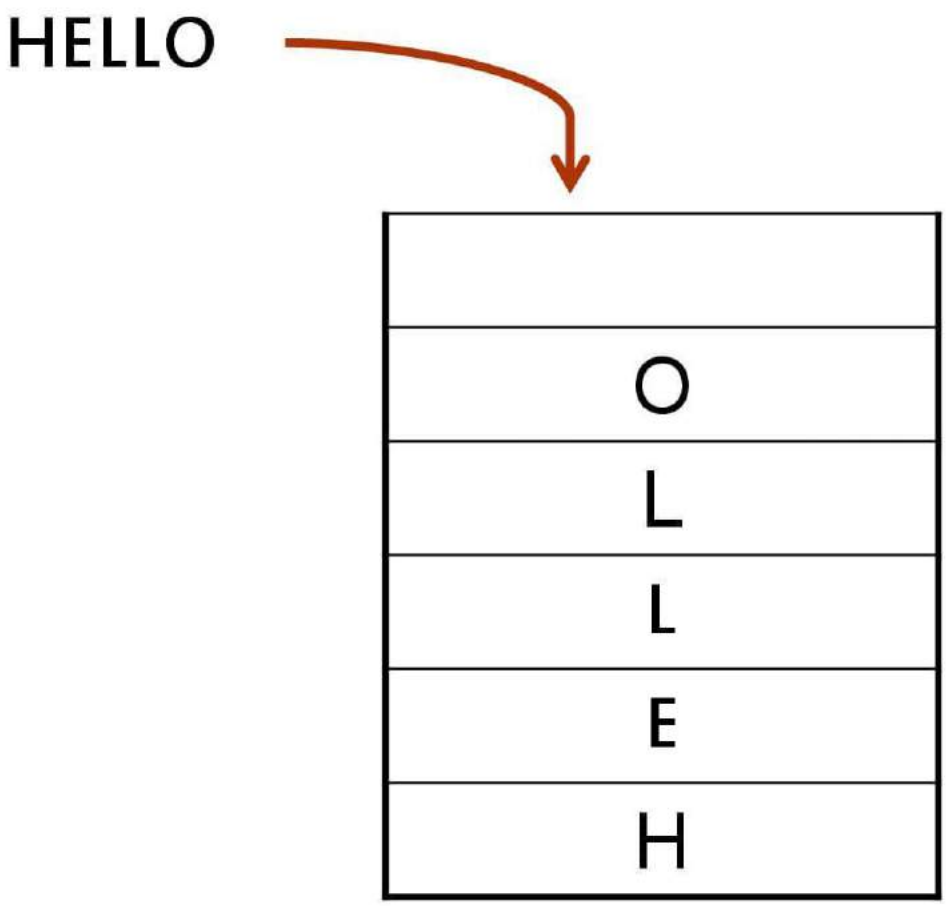
= 1

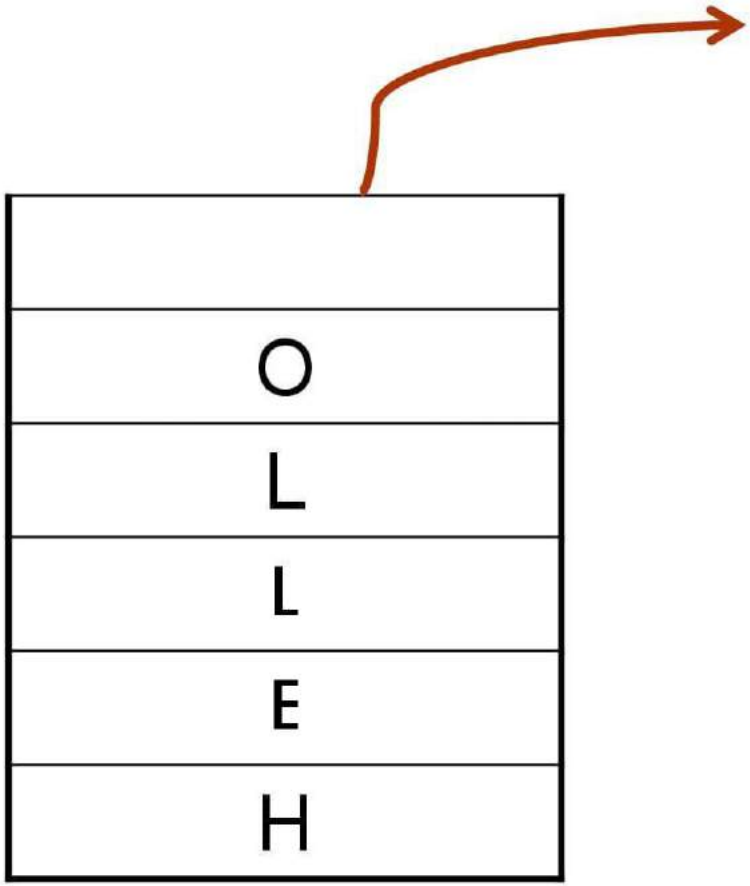


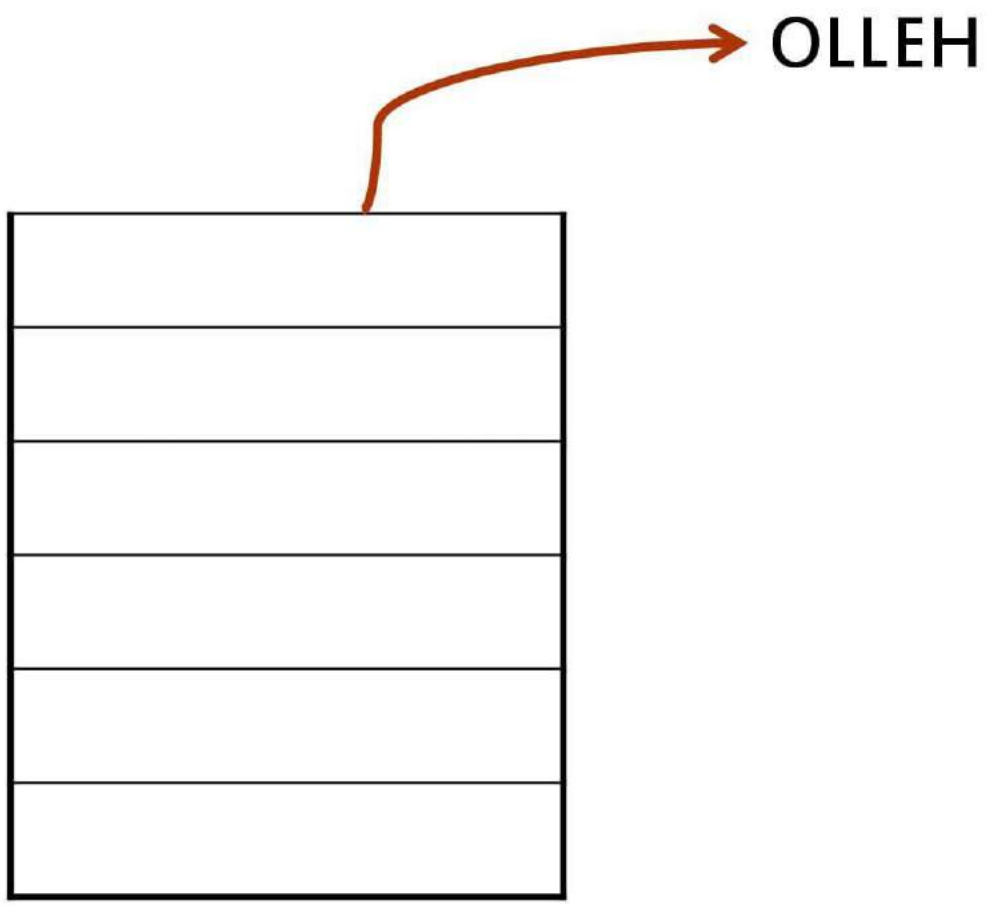
APPLICATIONS OF STACK

1. Evaluation of arithmetic expressions
2. Implementation of Recursion
3. **String reversal**


HELLO


HELLO
O
L
L
E
H


O
L
L
E
H


OLLEH