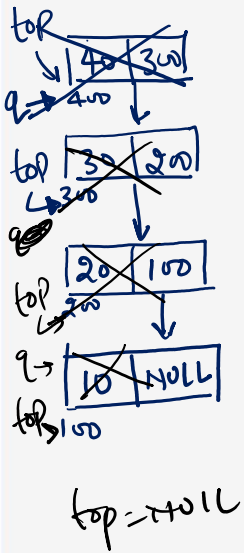
C-Programming and DS

Today Class Topics

-Linked list implementation of Stack
-Infix, Prefix and Postfix notation
-Prefix to postfix
-Postfix to prefix
-Postfix Expression evaluation
-Expression Tree
-Tower of Hanoi Recursion

Linked list Implementation of STACK :-

Push() :




```
Void Push (node *top, int item)
{ node *q;
  q = (node *) malloc (sizeof(node));
  q->data = item;
  q->link = top
  top = q;
}
```

TC = O(1)

Pop() :-

```
int pop (node *top)
{ int item;
  node *q;
  if(top == NULL)
    printf("STACK is empty");
  else
    { q = top;
      item = q->data;
      top = q->link;
      free(q);
      q = NULL;
      return item;
    }
}
```

TC = O(1)

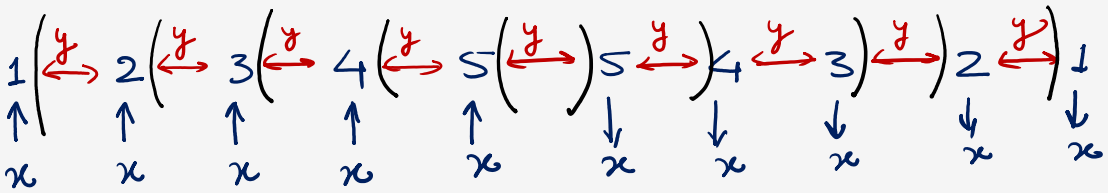
Display():

```
Void display(node *top)
{  node *q = top;
   while(q!=NULL)
   {  printf("%d", q->data);
      q=q->link;
   }
}
```

Finding average life time of an element in the Stack:

Let S be a stack of size n ≥ 1. Starting with the empty stack, suppose we push the first n natural numbers in sequence, and then perform n pop operations. Assume that Push and pop operation take X seconds each, and Y seconds elapse between the end of one such stack operation and the start of the next operation. The average stack-life of an element of this stack is

let $n = 5$



Stack life:

1 → $8x + 9y$
2 → $6x + 7y$
3 → $4x + 5y$
4 → $2x + 3y$
5 → y

$$\text{Avg life} = \frac{20x + 25y}{5} = 4x + 5y \Rightarrow 5x + 5y - x \Rightarrow 5(x+y) - x$$

$n = 10$
$x = 2$
$y = 3$

$10(2+3) - 2$
$50 - 2$
48

Avg
$n(x+y) - x$

Infix, Prefix & Postfix :-

Infix: a + b
(op1, operator, op2)

Prefix: + ab
(operator, op1, op2)

Postfix: ab +

Precedence	Associativity
↑	R to L
* /	L to R
+ -	L to R

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

Prefix :- ?

Postfix :- ?

Prefix:

A + B * +CD /F + D*E

A + *B+CD /F + D*E

A + / *B+CDF + *DE

+A /*B+CDF + *DE

++A/*B+CDF*DE

Postfix:

A+ B * CD+ /F + D*E

A+ BCD+* /F + D*E

ABCD+*F/+ + DE*

ABCD+*F/+DE*+

Prefix to Postfix Conversion:- To Convert from prefix to Postfix we need

1) Operator followed by
2) two immediate operands, Then Convert it into Postfix

Ex:- Prefix: * + cd − ba

Postfix: * cd+ ba−

cd+ba− *

Ex: Prefix: * ↑ / + ca e * db ↑ fg

Postfix: * ↑ / ca+ e * db ↑ fg

* ↑ ca+e/ db* fg↑

* ca+e/db*↑ fg↑

ca+e/db*↑fg↑*

Question:- match the following

Prefix	Postfix
1) *+ab-cd	a) ab*cd+-
2) +-ab*cd	b) ab*cd-+
3) +*ab-cd	c) ab+cd-*
4) -*ab+cd	d) ab-cd*+

Prefix to Infix Conversion:- To Convert from Prefix to Infix we need.

1) Operator followed by
2) Two immediate operands :- Then Convert it into infix expression

Ex: Prefix: $* + cd - ab$

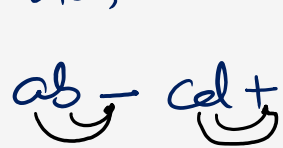
$* (c+d) (a-b)$

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

Postfix to Infix Conversion:- We need

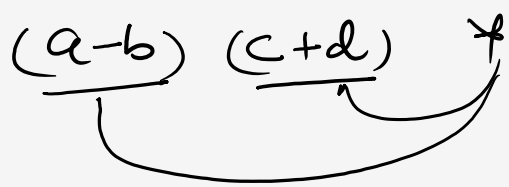
1) two operands followed by
2) immediate operator, Then convert it into infix & prefix

Postfix: ab- cd+ *



Infix:

(a-b) (c+d) *



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

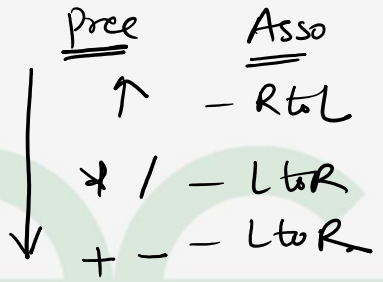
Prefix

- ab + cd *

* -ab + cd

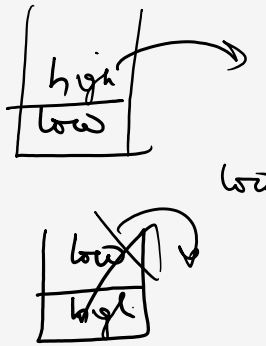
$(5 * 4) + (3 * 6) / 4 - 2 \uparrow 3 * 6 / 5 - 2 + 3 \uparrow 2 / 6$

① ⑨ ② ⑤ ⑩ ④ ⑥ ⑦ ⑪ ⑫ ③ ⑧



Infix to Postfix Conversion (using operator stack) :-

Top of STACK	Next operator	Operation
low	high	Push
high	low	Pop
Same	Same (L→R)	Pop
Same	Same (R→L)	Push

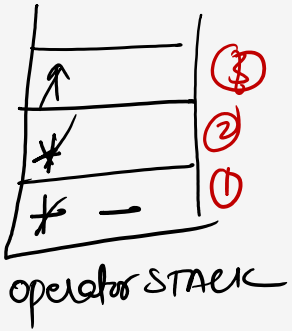


NOTE: If any open braces comes, treat it as new start or new stack, until the closing braces

Question: Infix : a + b * c ↑ d − e

Postfix : ? a b c d ↑ * + e −

Maximum stack size used (MSU) = 3 Ans

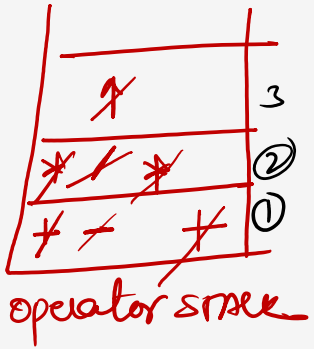


operator STACK

Question: Infix: a+b*c-d/e↑f*g+h

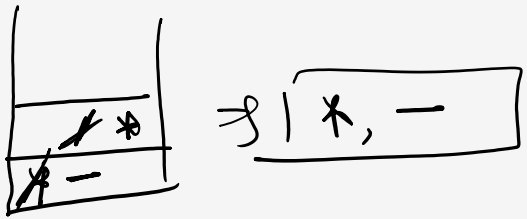
Postfix = ? abc*+def↑/g*-h+

MSU = ? (3) Ans



Question: When we will reach symbol 'f' while scanning during conversion from Infix to Postfix, what will be the content of stack

Infix: a * b − c / e * f + g



Question: Infix : ((a+b)*(c-d))

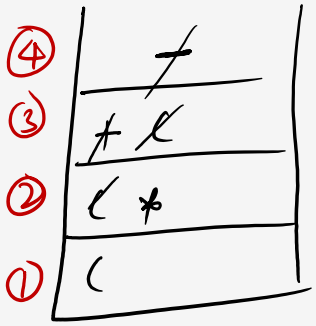
Postfix : ? ab+cd-*

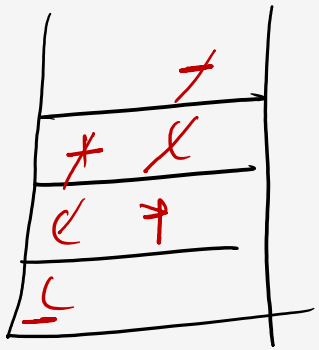
MSU = ?

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

(ab+ * cd-)

ab+cd-*





Postfix :- ab+cd-*

Convert Postfix to Infix and Prefix by STACK:

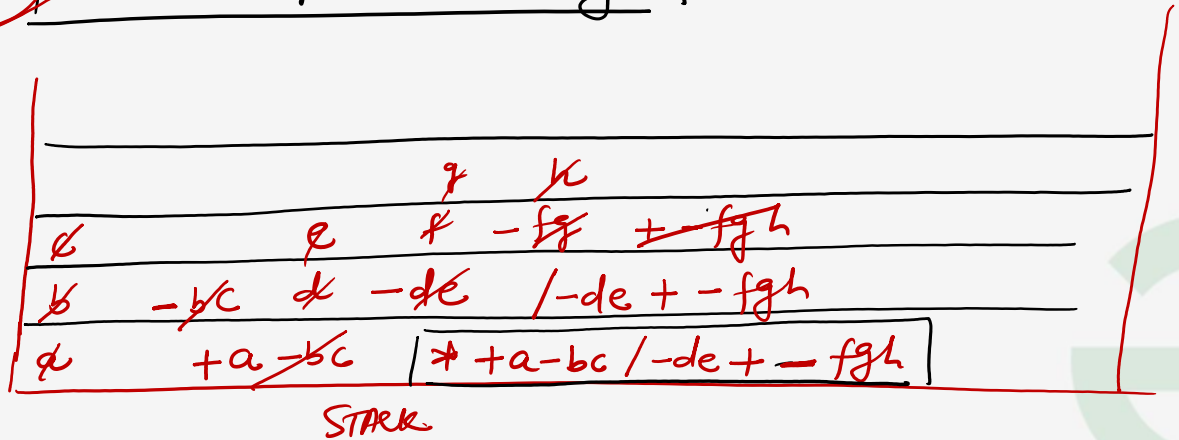
Postfix: abc − + de − fg − h + / *

Infix: ?

Prefix: ?

⇒ (a + (b − c)) * ((d − e) / ((f − g) + h))

*** + a − bc / − de + − fgh**



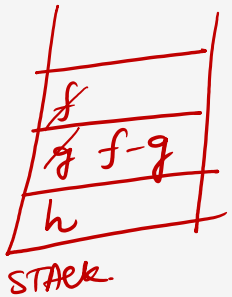
Convert Prefix to Infix and Postfix by STACK:-

Prefix: $*+a-bc/-de+-fgh$

Infix: ?

Postfix: ?

$\Rightarrow$



Infix: $f-g$

Postfix: $fg-$

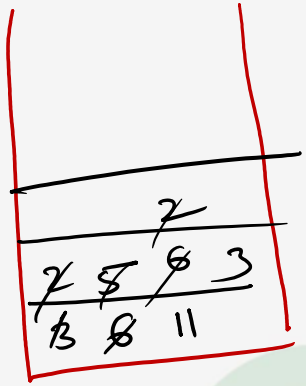
$\Rightarrow$

Postfix evaluation :- using operand STACK

1) if (Symbol == operand)
   push(Symbol);
2) if (Symbol == operator)
   OP_2 = pop();
   OP_1 = pop();
   push (OP_1 <operator> OP_2);
3) At the end, Result will be in the STACK

Ex:- 3 2 * 5 + 6 2 / -

⇒ 8 Ans



11 − 3 = 8

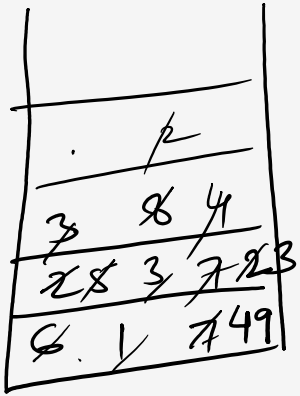
6 / 2

3 * 2 = 6

6 + 5 = 11

Question: Postfix: 6 2 3 + − 3 8 2 / + * 2 ↑ 3 + ?

MSU = ? (4)



(4)

⇒ 49 + 3 − 5 2

7 ↑ 2

2 ↑ 7

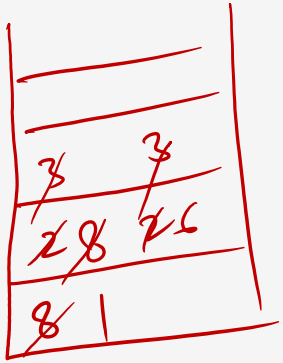
Question: Postfix: 8 2 3 ↑ / 2 3 * + 5 1 * − , top two elements of the stack after the first * is evaluated are

a) 6, 1

b) 5, 7

c) 3, 2

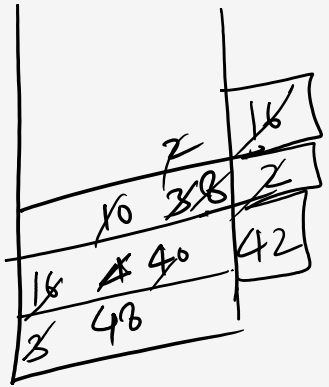
d) 1, 5



Question:- Evaluate Prefix expression using STACK for A=16, B=2, C=3, D=10, E=4

−+/A↑BC*DE*AC ? ⇒ (−6) Ans

− + / 16 ↑ 2 3 * 10 4 * 16 3



16/8

2↑3

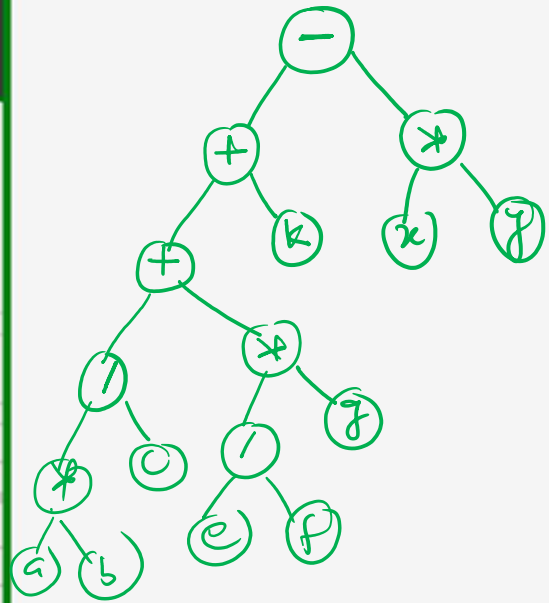
42 48 ⇒ (−6)

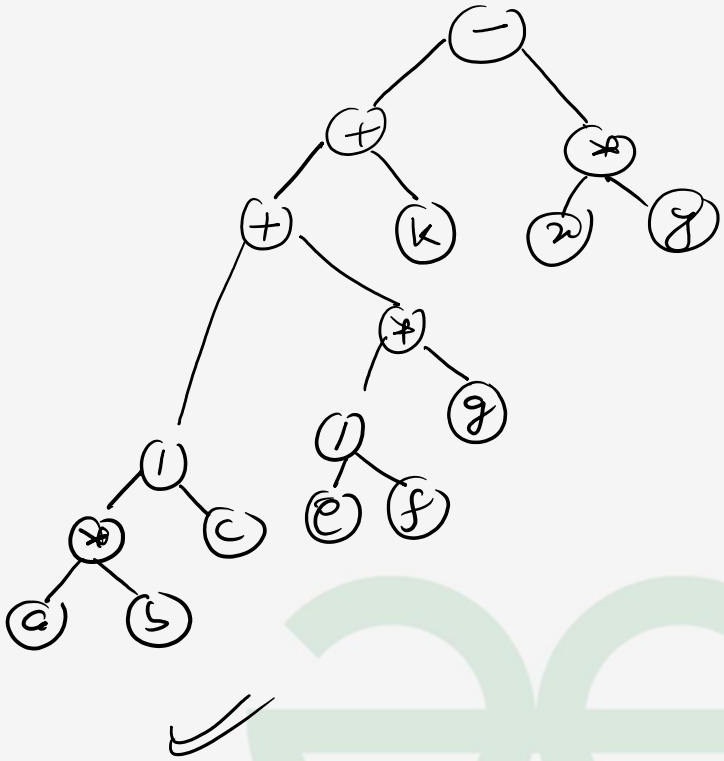
Binary Expression Tree:-

leaf nodes are operands
Internal nodes are operators

Infix: a*b/c+e/f*g+k-x*y

Postfix: ab*c/ef/g*+k+xy*-

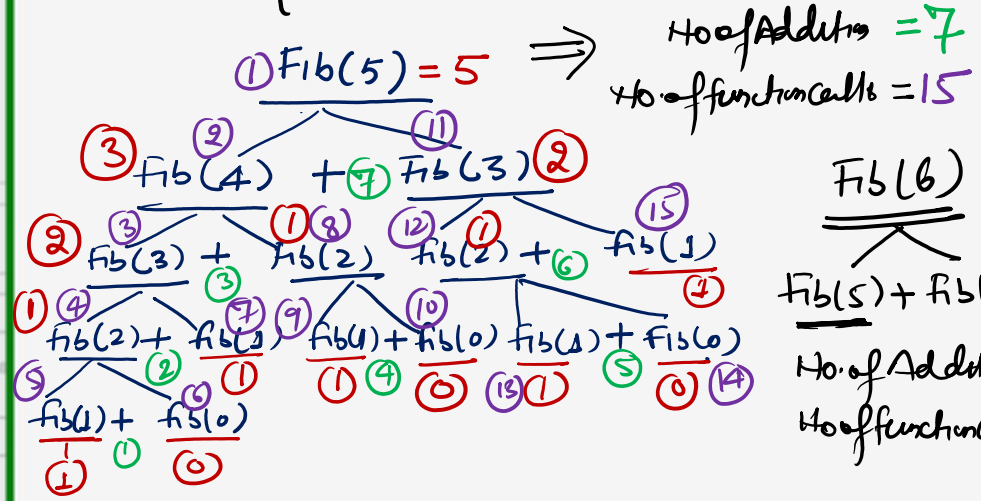




Fibonacci Series :-

n	0	1	2	3	4	5	6	7	8	9	10	----
fib(n)	0	1	1	2	3	5	8	13	21	34	55	=

$$Fib(n) = \begin{cases} n, & \text{if } n = 0 \text{ (or) } n = 1 \\ Fib(n-1) + Fib(n-2), & \text{otherwise} \end{cases}$$



⇒ No of Additions = 7
No of function calls = 15

Fib(6)

Fib(5) + Fib(4)

No. of Additions = 12
No of function call = 25

Fib(n)	No. of Addition	No of function calls
0	0	1
1	0	1
2	1	3
3	2	5
4	4	9
5	7	15
6	12	25
7	20	41
8	33	67
9	54	109
10	88	177

if $n > 1$

No of Additions in Fib(n) = No of Add fib(n-1) + No. of Add in Fib(n-2) + 1

if $n > 1$

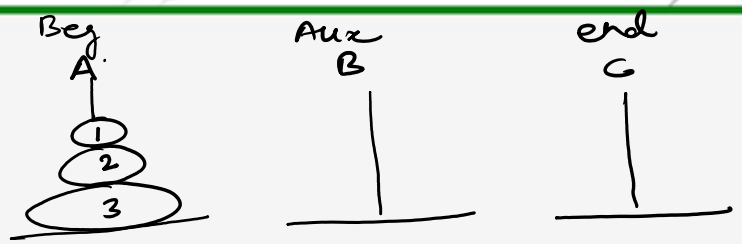
No of function calls in Fib(n) = No. of function calls in fib(n-1) + No. of function calls in fib(n-2) + 1

(

Tower of Hanoi :-

$n = 10$

$M(n)$ = No. of movements



Algo: TOH(n, A, B, C) ⇒ T(n)

{ if (n == 1) — 1

move A → C — 1

if (n > 1) — M(n-1)

{ TOH (n-1, A, C, B) — T(n-1)

Move A → C — 1 — 1

TOH (n-1, B, A, C) — T(n-1)

↳ M(n-1)

}

}

$T = O(2^n)$

Time complexity:

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T(n-1)+1, & \text{if } n > 1 \end{cases}$$

No. of movements = M(n)

$$M(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2M(n-1)+1, & \text{if } n > 1 \end{cases}$$

Recurrence Relation

$$M(n) = 2M(n-1)+1 \Rightarrow 2^n - 1$$

$$M(n) = 2^n - 1$$

$$T(n) = 2T(n-1)+1 = 2^n - 1$$

$$= O(2^n)$$

GATE:

Question (a)

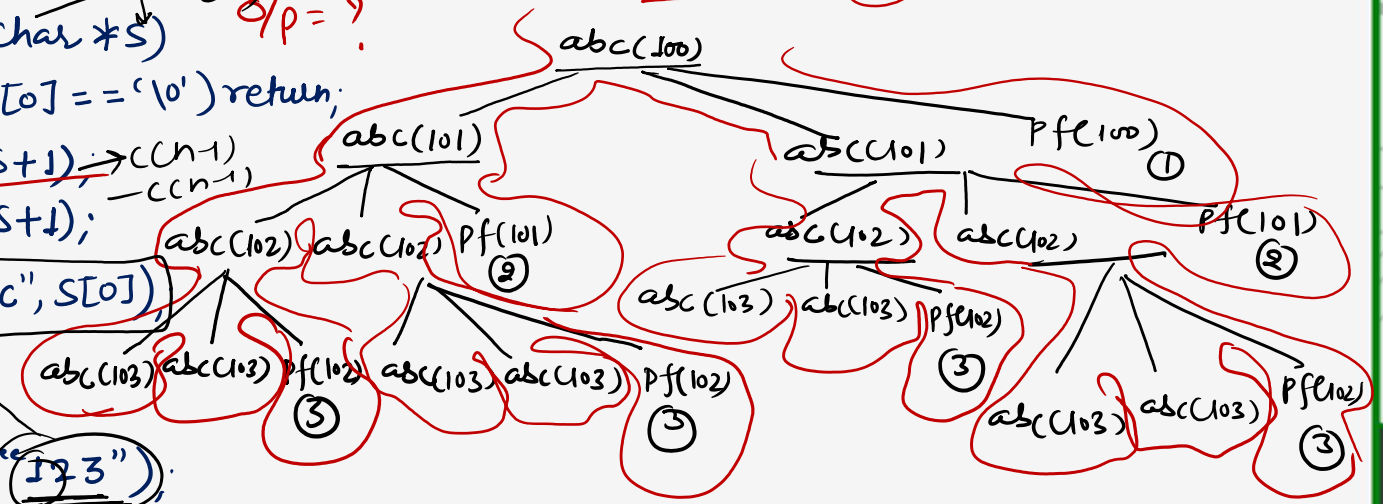
```
void abc (char *s)
{ if (s[0] == '\0') return;
  abc(s+1);   -> C(n-1)
  abc(s+1);   -> C(n-1)
  Pf("%c", s[0]);
}
main()
{ abc("123");
}
```

C(n)

C(n-1)

C(n-1)

o/p = ?



o/p = 3 3 2 3 3 2 1

GATE

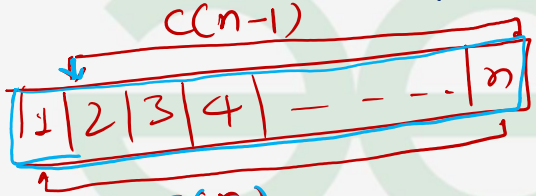
b) if abc(S) is called with NULL terminated string S of length n character (not counting '\0' character)

How many character will be printed by abc(S)

Recurrence Relation:

$$C(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2C(n-1)+1 & \text{if } n > 1 \end{cases}$$

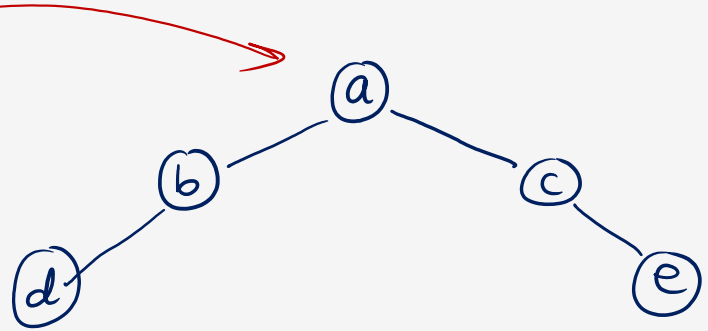
$C(n) = 2C(n-1)+1$



$C(n) = 2^n - 1 \Rightarrow 2^3 - 1 = 7$

Question:

```
void to(struct node *t)
{
  if(t)
  {
    Pf("t->data");
    to(t->Lc);
    Pf("t->data");
    to(t->Rc);
    Pf("t->data");
  }
}
```



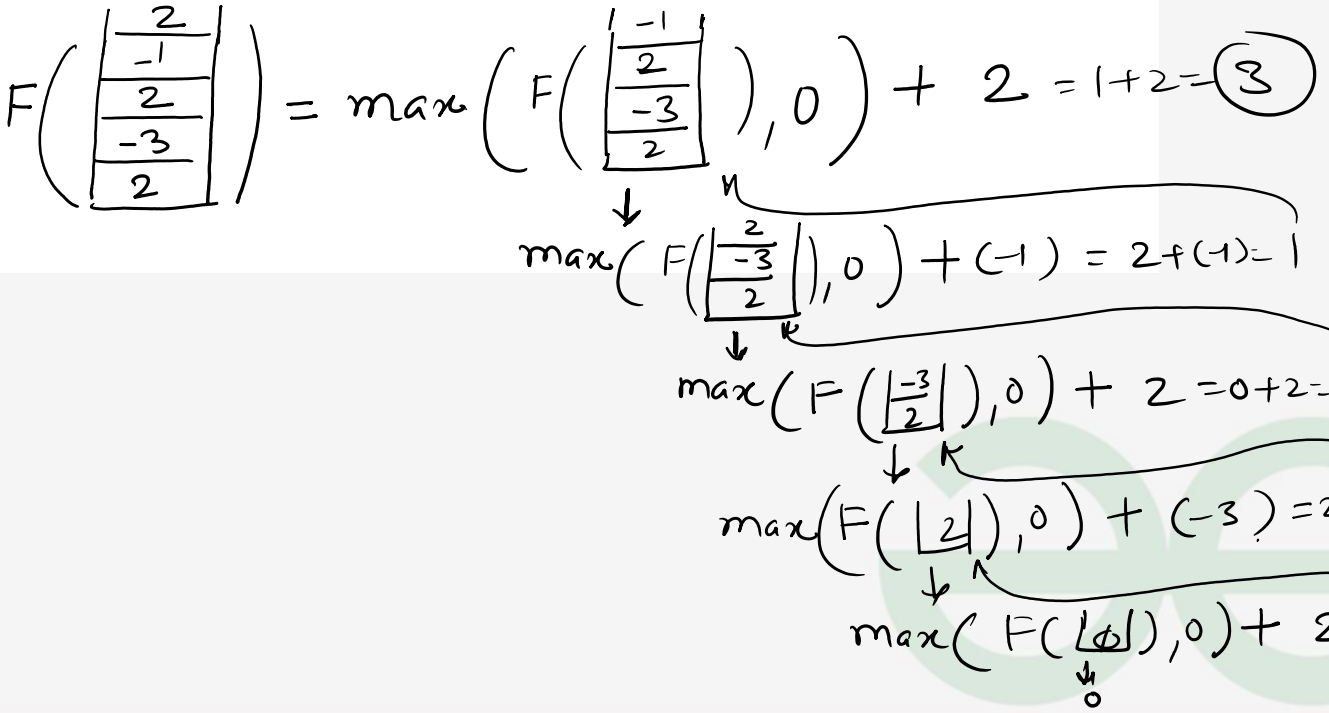
Print = ?

Qust. A function f defined on stacks of integers satisfies the following properties.

$f(\emptyset) = 0$ and $f(push(S, i)) = max(f(S), 0) + i$ for all stacks S and integers i.

If a stack S contains the integers $2, -3, 2, -1, 2$ in order from bottom to top, what is $f(S)$?

A. 6
B. 4
C. 3
D. 2



Thank You !