15/6/11 Data structure

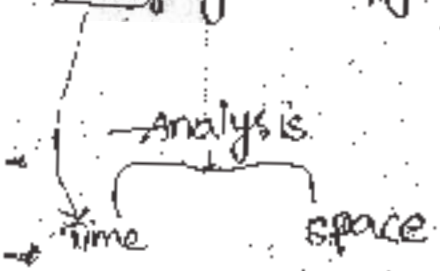
Data structure
- A logical or mathematical model

Eg: stack (or) LIFO (or) FILO

physical structure (4)
Implementation of DS (model) in physical memory

Eg: Arraystack, linkedlistStack

- why data structure is required?
  Solving a prodem fastly by occupying the optimum memory



1D-Arrays:

P) A is an array represented as follows
A: array [5---5] of elements. Each element occupies two memory cells. The starting location is 1999. Calculate the location of element A[0]

sol: A: array [lb---ub] of elements
lo: starting location = 1999
c: elements size = count = 2
lb: lower bandary = -5

loc A[i] = Lo + (i-lb) * C
loc A[0] = 1999 + (0-(-5)) * 2
= 1999 + 10
= 2009

Books

I level:
- sahani
- weiss
- kruse

II Level:
- coreman
- Goodrich & Tamzion
- (Interview) Drozdek

→

[ ]	*
Array	pointer
static	dynamic
early binding	late binding
compile time	runtime
user-friendly	machine friendly
a[i]	= *(a+i)
constant pointer	pointer variable

→ when calling by name control is inside
& " outside
* is used to get content

→

loc A[i] = Lo + (i - lb) * c

→ scalar arithmatic

&A[0] + (i - 0)

loc A[i] = A + i

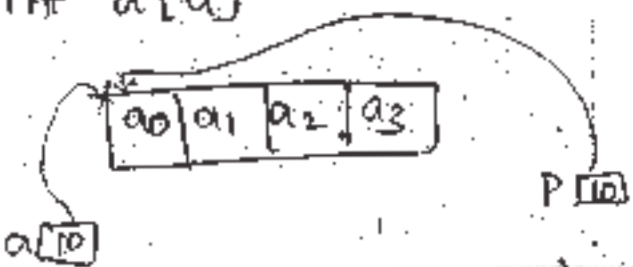
A[i] = *(A + i)

On c
char - 1 byte
int - 2 "
float - 4

char 10+1 = 11
int 10+1 = 12
float 10+1 = 14

Example:

int a[4]



int *P

a = &a[0]

10 = 10

a+1 ← valid

a = 10+1 = 12

invalid ← constant pointer

P = a

P+1 ← valid

10+1 = 12

P = P+1 → valid

P = 12

16/6

2D Array:

p) A: array [-2..2, 3..7] of elements. The starting location is 1000. each element occupies 2 memory cells. calculate the location of element A[0,5] using i) RMO ii) CMO

RMO

In simple

$$loc\ A[i,j] = Lo + (i-1)r_2 + j - 1$$

In detail

$$loc\ A[i,j] = Lo + [(i-b_1)(u_2-b_2+1) + (j-b_2)] * C$$

A: array $[b_1..u_1, b_2..u_2]$ of elements

$b_1..u_1$: row, r_1; $b_2..u_2$: col, r_2

$r_1 \times r_2$

$(u_1-b_1+1) \quad (u_2-b_2+1)$

$$loc\ A[0,5] = 1000 + [[0-(-2)] \times 5 + (5-3)] * 2$$

$$= 1000 + 24$$

$$= 1024$$

16/6/21

CMO: In simple

$$loc\ A[i,j] = Lo + (j-1)r_1 + (i-1)$$

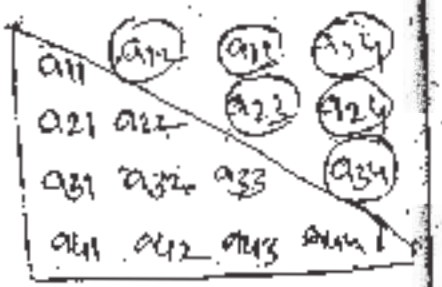
In detail

$$loc\ A[i,j] = Lo + [(j-b_2)(u_1-b_1+1) - 20 + (i-b_1)] \times C - 10$$

Loc A(6,5) = 1000 + [(6-3) × 5
+ 0-(-2)] × 2

= 1000 + 24 = 1024

Lower triangular matrix:



Def: - Is a square matrix

- A(i,j) = 0 , iff i<j

Size: 1+2+3+ - - - + N

$$\sum_{q=1}^{n} q = \frac{N(N+1)}{2}$$

Store:

RMO:

R: Lo + the no of elements present in (i-1) rows

+ j-1

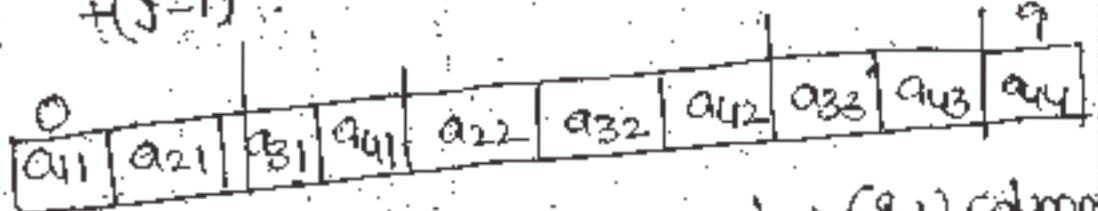
= Lo + (1+2+3+ - - - + i-1)

+ (j-1)

= Lo + $\frac{i(i-1)}{2}$

+ (j-1)

CMO:



= Lo + the no. of elements present in (j-1) columns

+ (i-j+1) → (i-j's tb)

+ n-(j-2)] + - - -

= Lo + [n + (n-1) + (n-2) + - - -

+ (i-j)

$$= L_0 + \frac{(i-1)}{2}\left[2n + (i-2)(-1)\right] + (j-i)$$

$$= L_0 + \frac{(i-1)}{2}\left[2n - i + 2\right] + (j-i)$$

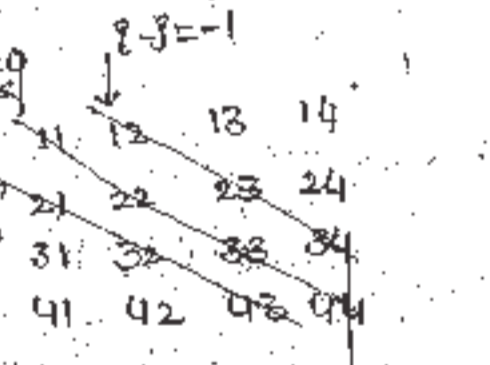
Check: a(4,3)

$$= 0 + \frac{(3-1)}{2}\left[2(4) + (3-2)(-1)\right] + (4-3)$$

$$= 0 + 7 + 1$$

$$= 8$$

Tridiagonal Matrix:



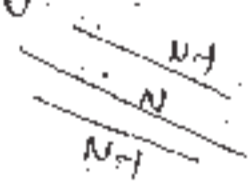
Def: – Is a square matrix

– A(i,j) = 0 iff |i-j| > 1

Eg:

11	12	0	0
21	22	23	0
0	32	33	34
0	0	43	44

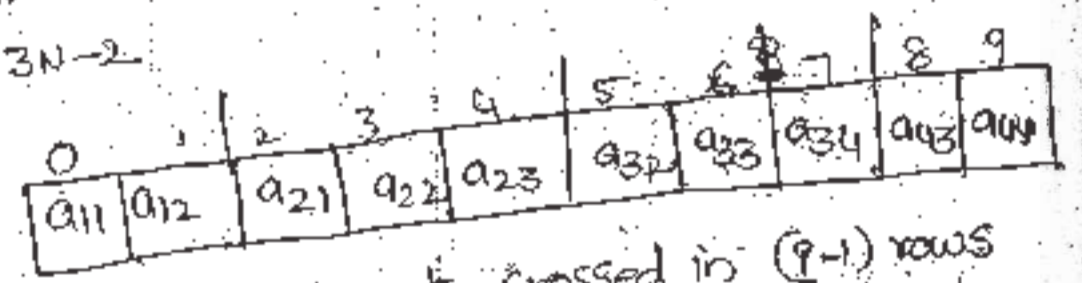
Size:



$$= N + N + N - 1$$

$$= 3N - 2$$

Store:



i) RMO:

R: Lo + No. of elements crossed in (i-1) rows + (j - j's lb)

$$= L_0 + \left[3(i-1) - 1 + j - (i-1)\right]$$

$$= L_0 + a + j - 1$$

i ≠ 1

i	j's lb
4	3
3	2
2	1

Except 1 row

i = 1

check:

$A(3,4) = 0 + 3(3-0-1 + 4-(3-1)$

$= 7$

CMO:

a_{11} a_{21}	a_{12} a_{22} a_{32}	a_{23} a_{33} a_{43}	a_{34} a_{44}

R: Lo + No. of elements present in $(j-1)$ columns

$+ \; q-(j-1)$

$= Lo + 3*(j-1) - 1 \qquad j \neq 1$

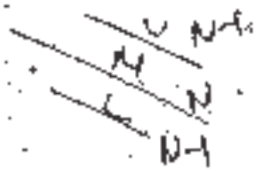
$+ \; q-(j-1)$

$= Lo + 0 \qquad j = 1$

$+ \; q-1$

diagonal by diagonal:

{L M U}



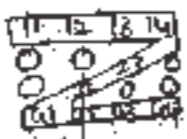
Lower	Main	Upper
a_{21} a_{32} a_{43}	a_{11} a_{22} a_{33} a_{44}	a_{12} a_{23} a_{34}

Condition: $q-j=+1$ | $q-j=0$ | $q-j=-1$

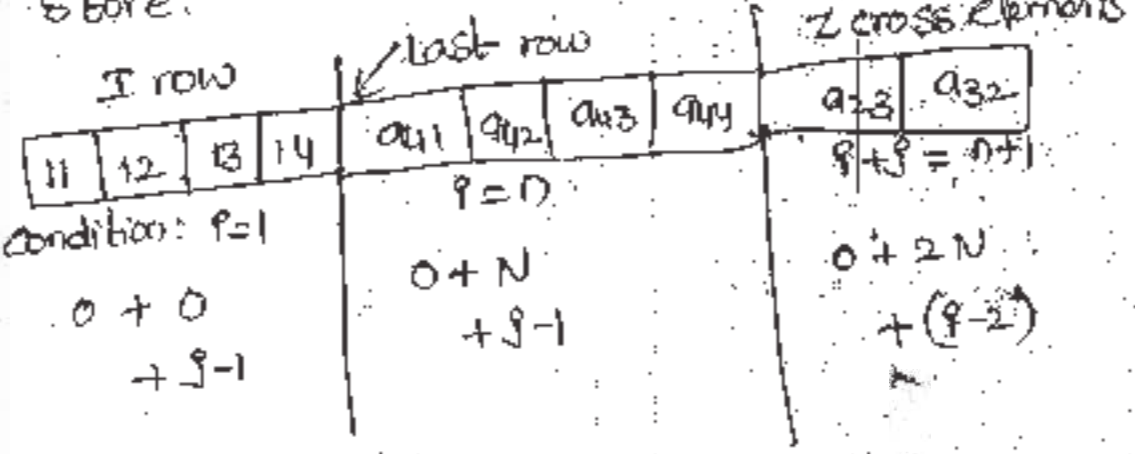
$Lo + 0 + q-2$	$Lo + (N-1) + q-1$	$Lo + N-1+N + q-h$
$Lo + 0 + q-1$	$Lo + (N-1) + q-1$	$Lo + N-1+N + q-2$

```
switch(i-j)
{
case 1: // Low diagonal
        return A[i-2];
        (or) A[j-1];
case 0: // Main diagonal
        return A[N+i-2];
        (or) A[N+j-2];
case -1: // Upper diagonal
        return A[2n+i-2];
        (or) A[2n+j-3];
Default: return 0;
}
```

Z-Matrix;



Store:



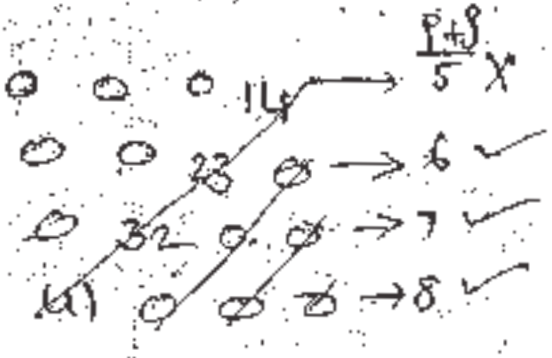
```
if (i==1)
    return A[j-1];
else if (i==N)
    return A[N+j-1];
else if (i+j==N+1)
    return A[2N+j-2];
else
    return 0;
```

b) what is the size of square matrix in which the elements are present at $i-j > n+1$

$$\begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix}_{4\times4}$$

not possible
$i-j > n+1$
∴ Zero size

~~if~~ $i+j > n+1$



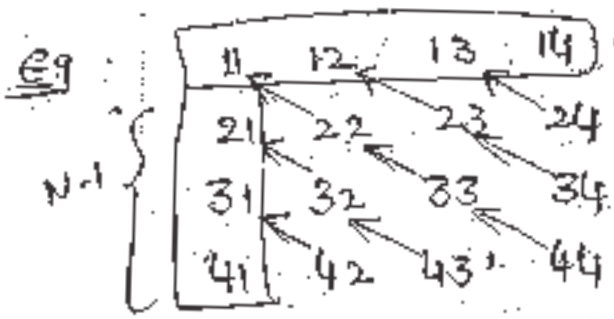
$N+1 + N-2 + \cdots + 1$

$= \frac{N(N+1)}{2}$

Toeplitz matrix

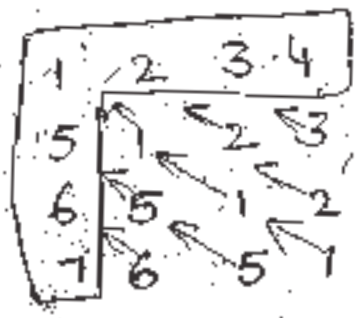
- Square matrix
- $A(i,j) = A(i-1, j-1)$ for all i, j

  where $i > 1$, $j > 1$

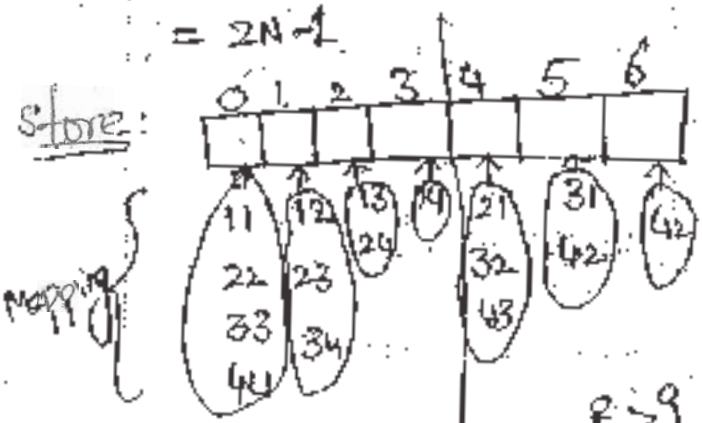


Size: $N + N - 1$

$= 2N - 1$



Store:



if $(i \le j)$

$= Lo + 0 + (j - i)$

$i > j$

$Lo + N + i - j - 1$

if $(i \le j)$

return $A[j-i]$;

else return $A[N + i - j - 1]$;

11/6/11

Square band matrix:

- denoted by | $D_{n,a}$

where 'n' means nxn matrix

'a' means (a-1) diagonals above and below the main diagonal

Size:

$= n + 2$ {no. of elements present in 'a-1' diagonals}

$= n + 2\{(n+1)+(n-2)+\cdots+n-(a-1)\}$

$= n + 2 \cdot \frac{a-1}{2}\{2(n-1)+(a-2)(-1)\}$

$= n + (a-1)\{2n-a\}$

Symmetric Square band matrix:

Store only the lower diagonals starting from the lowest diagonal in the style of diagonal by diagonals. Then calculate retrieve formula.

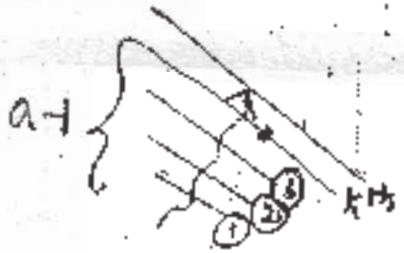
ex: Symmetric

$A(i,j) = A(j,i)$

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 6 & 8 & 9 \\ 4 & 7 & 9 & 10 \end{bmatrix}$$

(or)

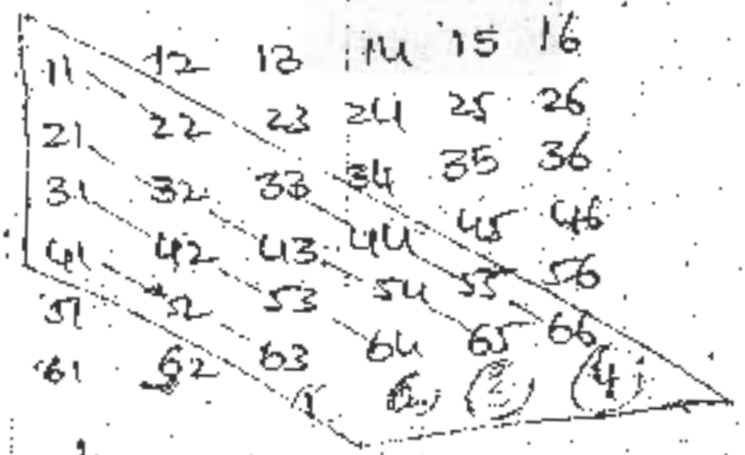
$1+2+\cdots+n = \frac{n(n+1)}{2}$



what is the value of k for a(i,j)?

K = ?

D6,4 –



element

a(5,4)

D6,4 ⇝ K = 3rd

D6,5 ⇝ K = 4th

Ob:

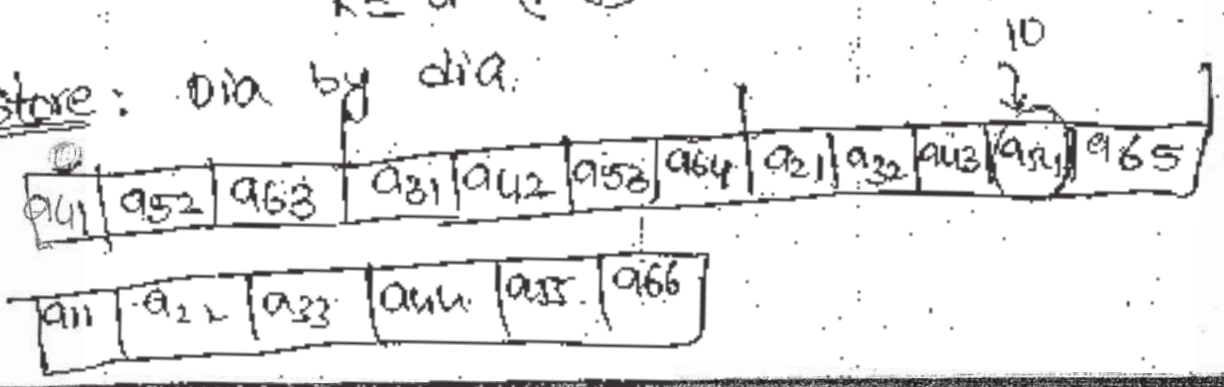
1. "i-j" value remains same/constant throughout diagonal

& As 'a' changes, accordingly 'K' value also getting changed

$\therefore K = f(a, i, j)$

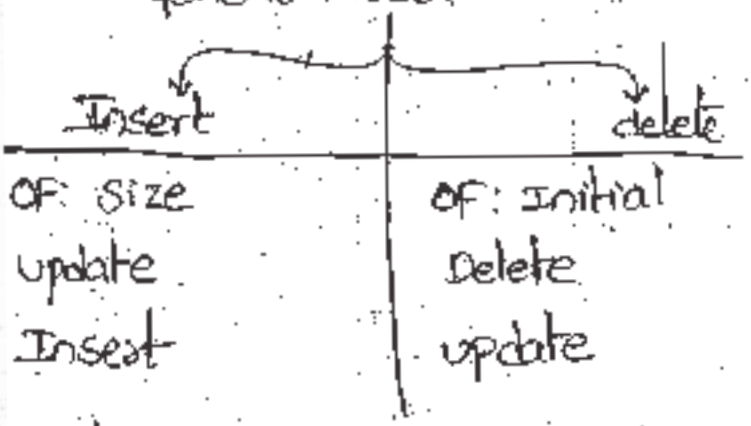
$K = a - (i - j)$

Store: Dia by dia



stack

Generic model



Initially TOP = 0

→ Push (S, N, TOP, X)

① if (TOP = N)

write : stack is overflow

exit()

② TOP = TOP + 1

③ S[TOP] = X

④ stop

→ POP (S, N, TOP, Y)

① if (TOP = 0)

write : stack is underflow

exit();

② Y = S[TOP]

③ TOP = TOP - 1

④ return Y;

ADT of stack

{

Def:- LIFO (or) FILO model

- one side is open and the other side is closed
- TOP pointer, pointing to topmost element among the available elements

operations: push(x) : Insert element 'x'
pop() : delete topmost element

}

Abstraction:

Hiding the internal implementation details

Multiple stacks:

Given an array of size 'm', there are 'n' stacks. The initial configuration is as follows:

for $0 \le i < n$

$B[i] = T[i] = i\left[\frac{m}{n}\right] - 1$

for $i = n$

$B[i] = m-1$ where B = bottom = fixed
T = TOP & move

fillup the blank

a) $T[i] = T[i+1]$

b) $T[i] = B[i]$

c) $T[i] = B[i+1]$

d) $T[i] = B[i-1]$

```
void push(int i, int x)   // i ← stack no
{ if (i == n)
      print: stack 'i' is over-flow
  else
      s[++TOP[i]] = x;
```

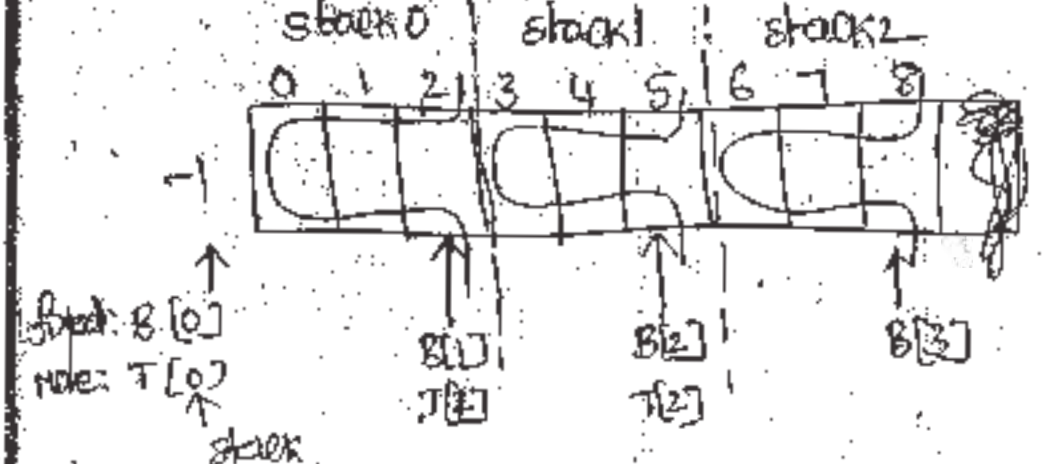
```
int pop(int p)
{
    if (p==8)
    {
        print: stack 'p' is underflow
        return 0;
    }
    else
        return S[TOP[p]--];
}
```

Sol: Consider m=9, n=3 stacks

Initial configuration:

$0 \le p < 3$

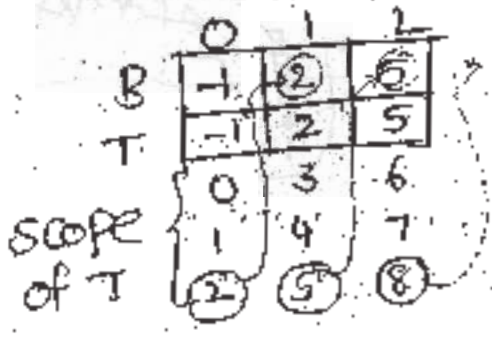
	0 stack	1 stack	2 stack	3
B[] =	-1	2	5	B[3]=8
T[] =	-1	2	5	



∴ push overflow: TOP[p] = Bottom[p+1]

T[p] = B[p+1]

Stack



pop underflow: Initial : TOP[i] = Bottom[i]
i.e. T[i] = B[i]

Stack applications

1. Permutations

Eg: for n, no of permutations = n!

stack is used.

Every element is pushed in and poped out

logic: Based on the desired o/p, the desired elements are popped out

p) which of following is invalid stack permutation when input sequence is 1,2,3,4,5

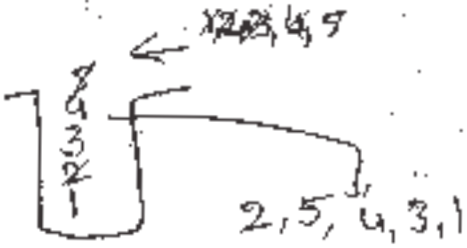
a) 4, 5, 3, 2, 1 - valid

b) 5, 3, 2, 4, 1 - Invalid

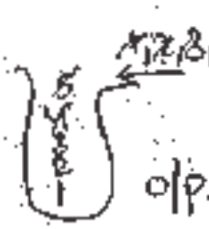
c) 3, 4, 5, 2, 1 - valid

d) 2, 5, 3, 4, 1 - Invalid

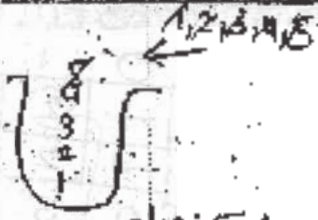
→ d)



c)

a) 

o/p: 4,5,3,2,1

b) 

o/p: 5,4

2. Polish Notation: (or) Lukasiwicz notation

- revised by Lukasiwicz, polish logician
- It avoids the repeated scanning of infix expression
- In one scan, we get result if the expression is either prefix (or) postfix
- Machines like compilers peer either prefix (or) postfix

Rule: The relative position of operands can't be disturbed

operators: disturbed according to the precedence and associativity rules

Evaluation → Precedence → same → Associativity → left / right

Eg:

(1) a + b*c

* is higher precedence than +

prefix:
a + b*c
(b*c)
a + *bc
+ a * bc

postfix:
a + b*c
(bc*)
a + bc*
a bc* +

(2) Left associative

a * b | c

prefix

a * b | c

(a * b)

* ab | c

| * ab c

postfix

a * b | c

(a * b)

ab * | c

ab * c |

(3) Left associative

a | b * c

prefix

a | b * c

(a | b)

| ab * c

* | ab c

postfix

a | b * c

(a | b)

$ab| * c

$ab| c *

(4) Right associative

prefix

a ↑ b ↑ c

(b ↑ c)

↑ bc

a ↑ ( )

↑ a ↑ bc

postfix

a ↑ b ↑ c

(b ↑ c)

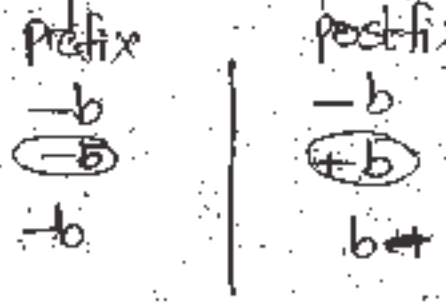
bc ↑

a ↑ ( )

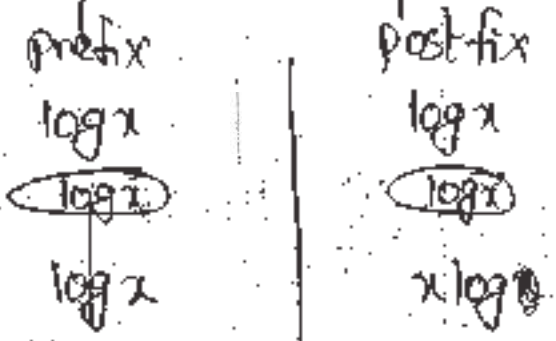
abc ↑ ↑

p) 1) $-b$

— unary operator



2) a) $\log x$



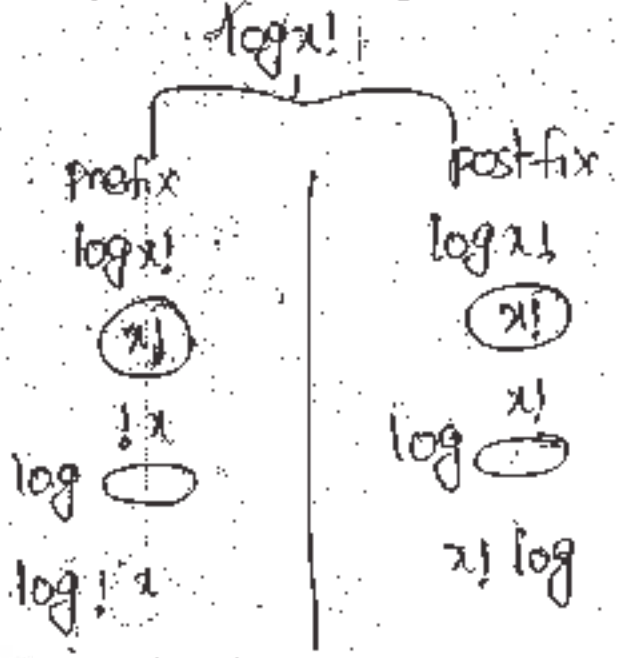
b) $x!$

prefix: $!x$

postfix: $x!$

3) $\log x!$

Note: ! is higher precedence than log

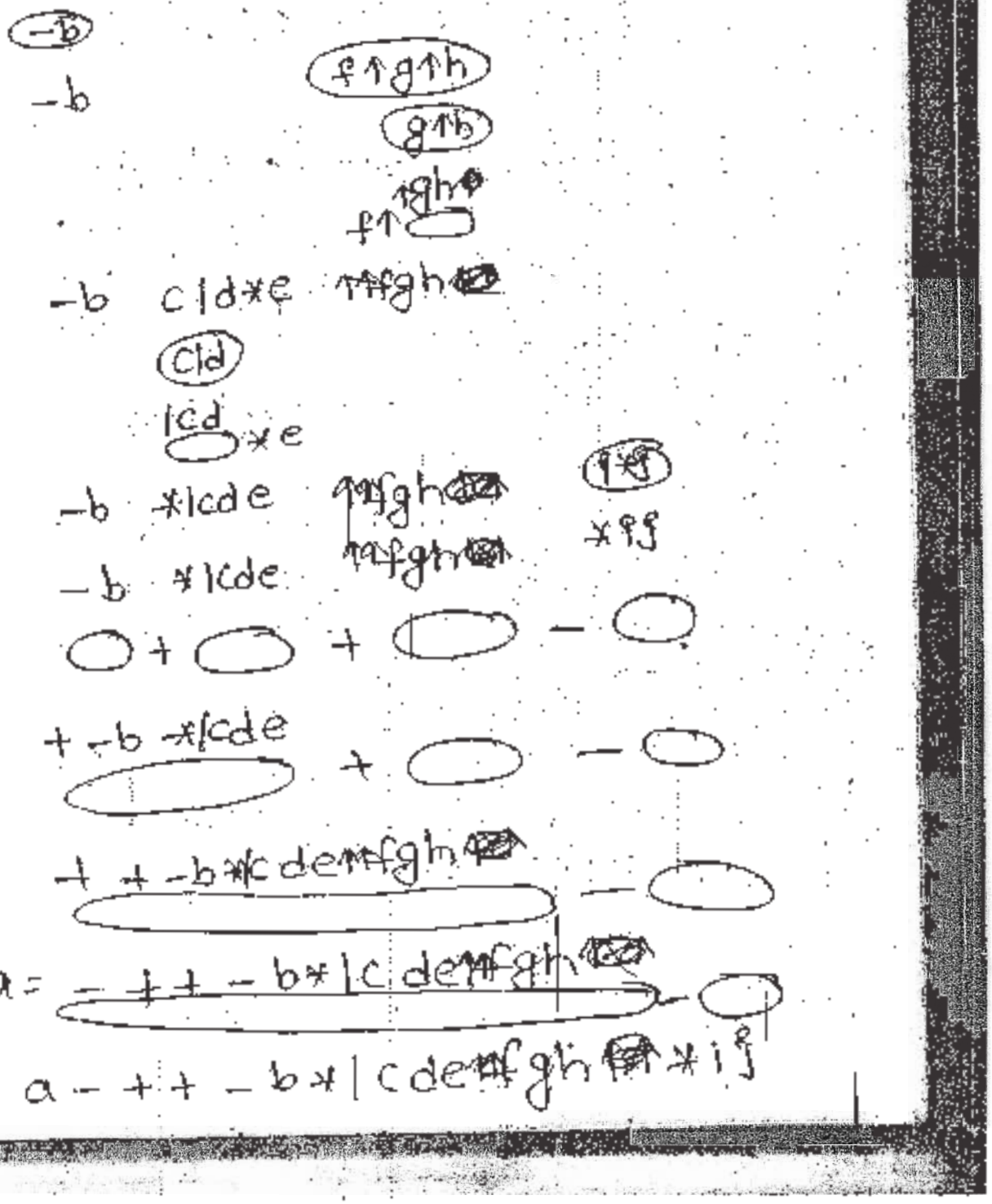


p) $a = -b+c/d*e+f\uparrow g\uparrow h-9*8$.

Note: precedence
- : unary
↑ : Right associative
*/ : left "
+- : left "
:= : assignment

Prefix:

$a = -b+c/d*e+f\uparrow g\uparrow h-9*8$



postfix

$a = -b + c/d*e + f \uparrow g \uparrow h - i * j$

$(g \uparrow h)$ $(i*j)$

$(-b)$ (c/d)

$b-$ $cd/$ $*e$ $f \uparrow gh\uparrow$ $ij*$

$(\quad) + (cd/e*) + (fgh\uparrow\uparrow) - (\quad)$

$b-cd/e*+ \quad + (fgh\uparrow\uparrow) - (ij*)$

$b-cd/e*+fgh\uparrow\uparrow+ \quad - (ij*)$

$a = b-cd/e*+fgh\uparrow\uparrow+ij*-$

$ab-cd/e*+fgh\uparrow\uparrow+ij*-=$

p) $a<b$ OR $c>d>e$ AND $f<g$

8/7/11

evaluation : infix 6-2=4

prefix: −, 6, 2 ←

postfix: 6, 2, − →

operand: push

operator: (i) pop
{never pushed in}
(ii) evaluate
(iii) push

Obs: 1) In postfix, topmost element is second operand and it is evaluated from left to right

2) In prefix, evaluation is done from right to left and topmost element is first operand.

Gate 08

3) 8, 2, 3, ^, /, 2, 3, *, +, 5, 1, *, + evaluate this postfix

1) which of following is not possible stack content

a) 8|2|3 b) 7|5|1 c) 1|2|3|* d) 1|6

operator is not pushed

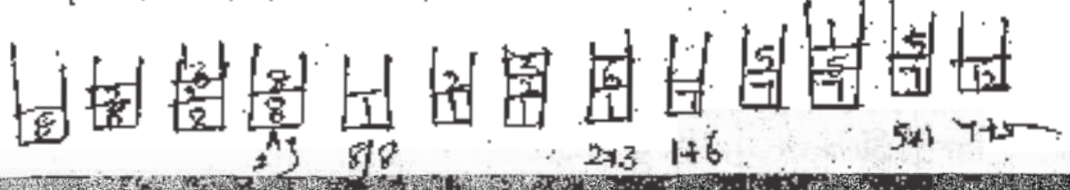
2) given the status of stack contents after evaluating first '*'

a) 6|1 b) 1|6 c) 5|1 d) 1|2|3

3) final result is

a) 12 b) −12 c) 13 d) −13

8, 2, 3, ^, /, 2, 3, *, +, 5, 1, *, +



1) Infix 2+3*4-1
convert into prefix and evaluate

-+2*3 4 1

3) Recursion:

Types:

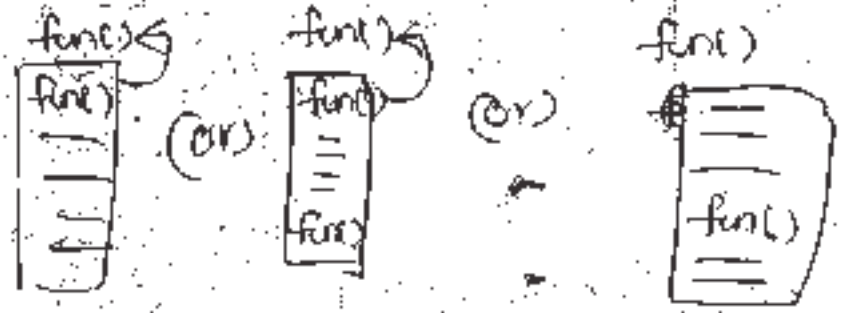
i) Direct recursion

Def: A function calling to itself

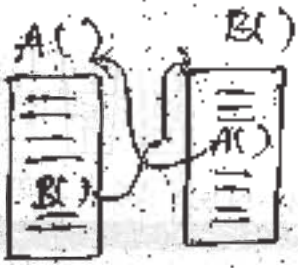
a) Tail recursion

eg: factorial

b) Non tail Recursion (or) head recursion



ii) Indirect recursion



eg: tree

iii) excessive recursion

By default: The nature of recursion is excessive, means does not remember the previously evaluated values

Eg: fibonacci

iv) Nested Recursion

Eg: Ackerman's function

Eg:

Tail Recursion:

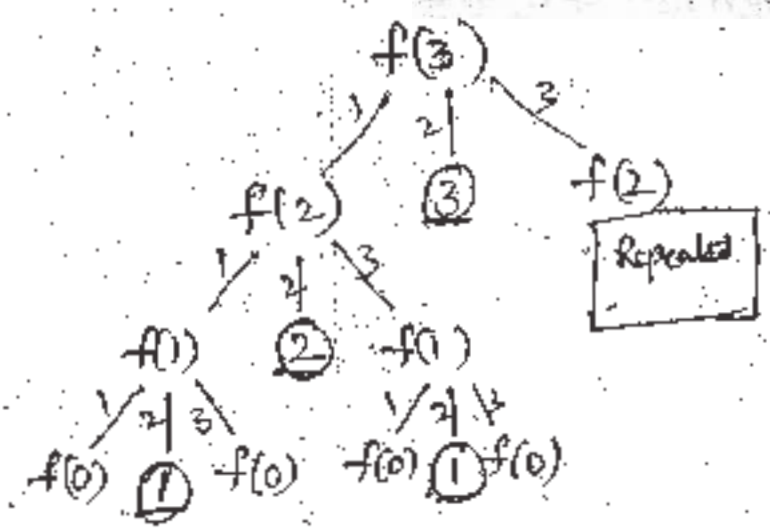
```
p)void func (int x)
  { if (x>0)
    {
      ①printf ("%d",x);
      ②func (x-1);
    }
  }
```

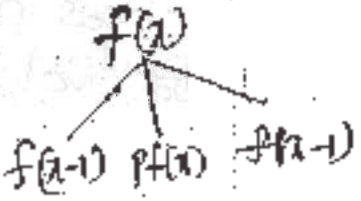
func(3) → what is o/p?

→ f(3) → 3
f(2) → 2
f(1) → 1

o/p: 321

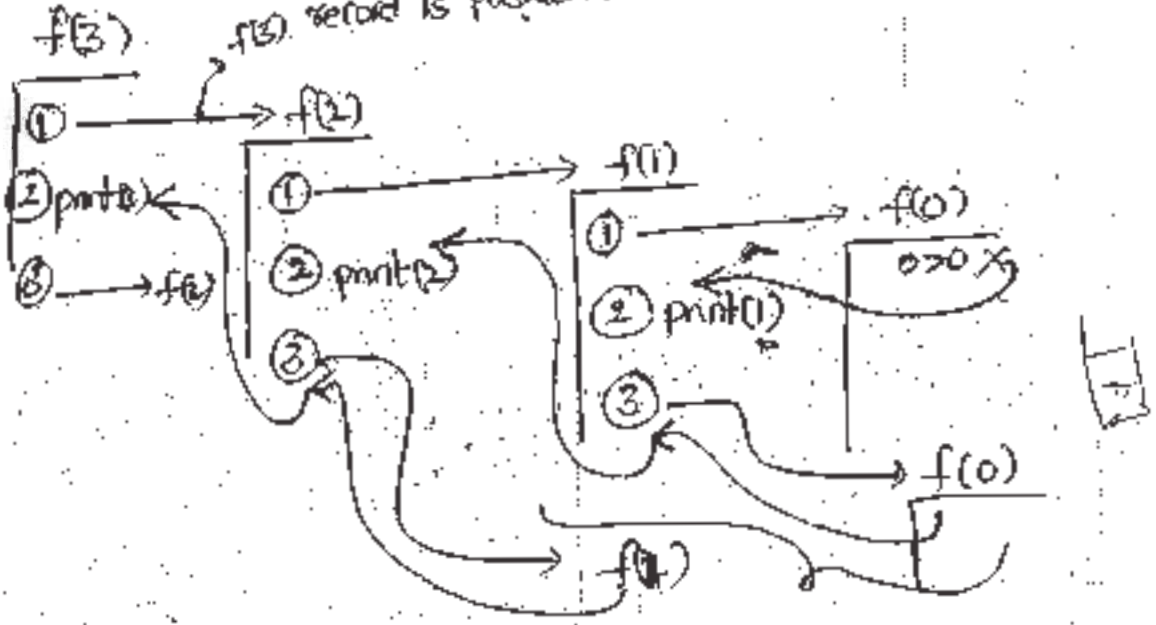
```
P) void func (int x)
   { if (x>0)
     {
       1) func(x-1);
       2) printf("%d", x);
       3) func (x-1);
     }
   }
```





o/p: 1 2 1 3 1 2 1

f(3) record is pushed in stack. this is inactive record.



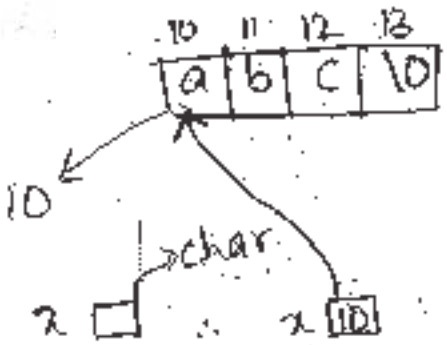
o/p: 1 2 1 3 1 2 1

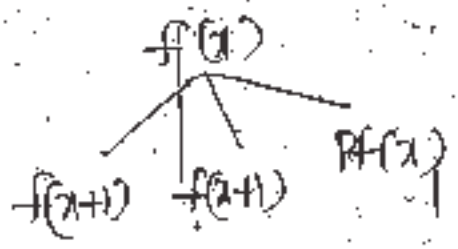
```
p) void func(char **x)
{   if (**x != NULL)
    { ① fun(x+1);
      ② fun(x+1);
    } ③ printf("%c", **x);
}
```

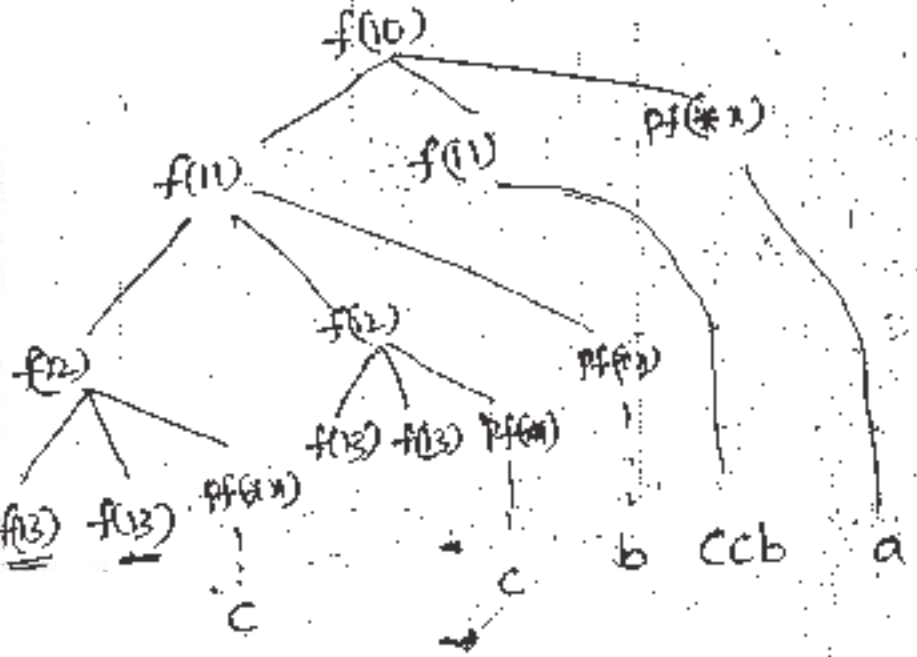
(or) x[0]

```
void main()
{
    func("abc");
}
```

Sol:



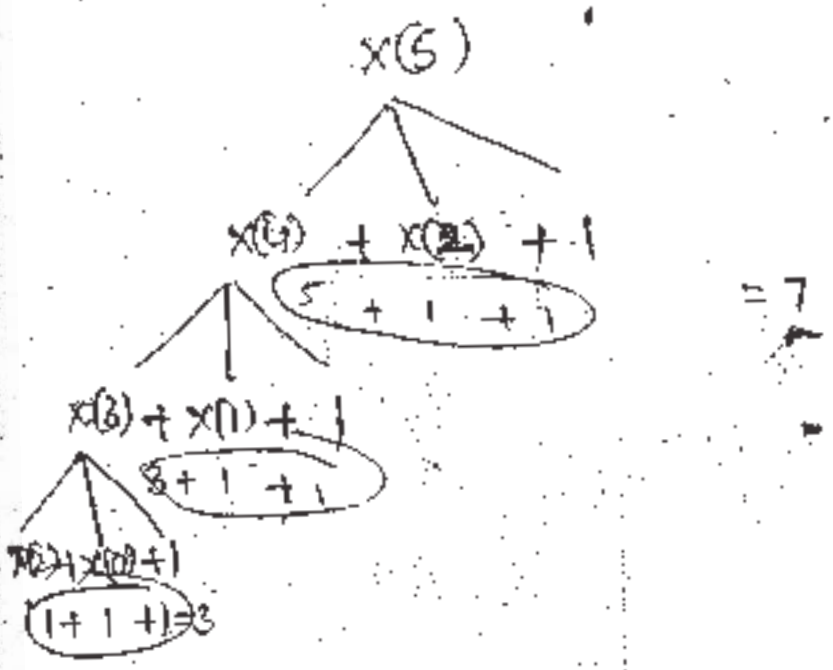


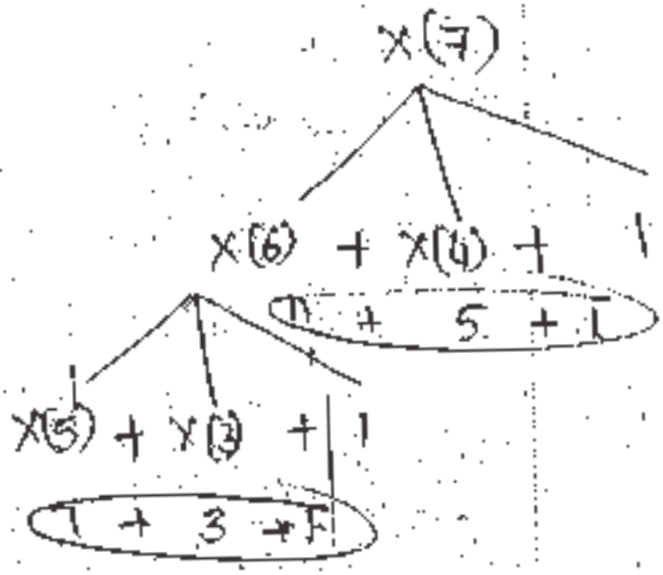


o/p: ccbccba

9)
```
int x(int N)
{   if (N<3)
        return 1;
    else
        return x(N-1) + x(N-3) + 1;
}
```

Q) How many total invocations for evaluating X(X(5))





total invocations = 24

for X(5) = 7

X(7) = 17

X(X(5)) = 24

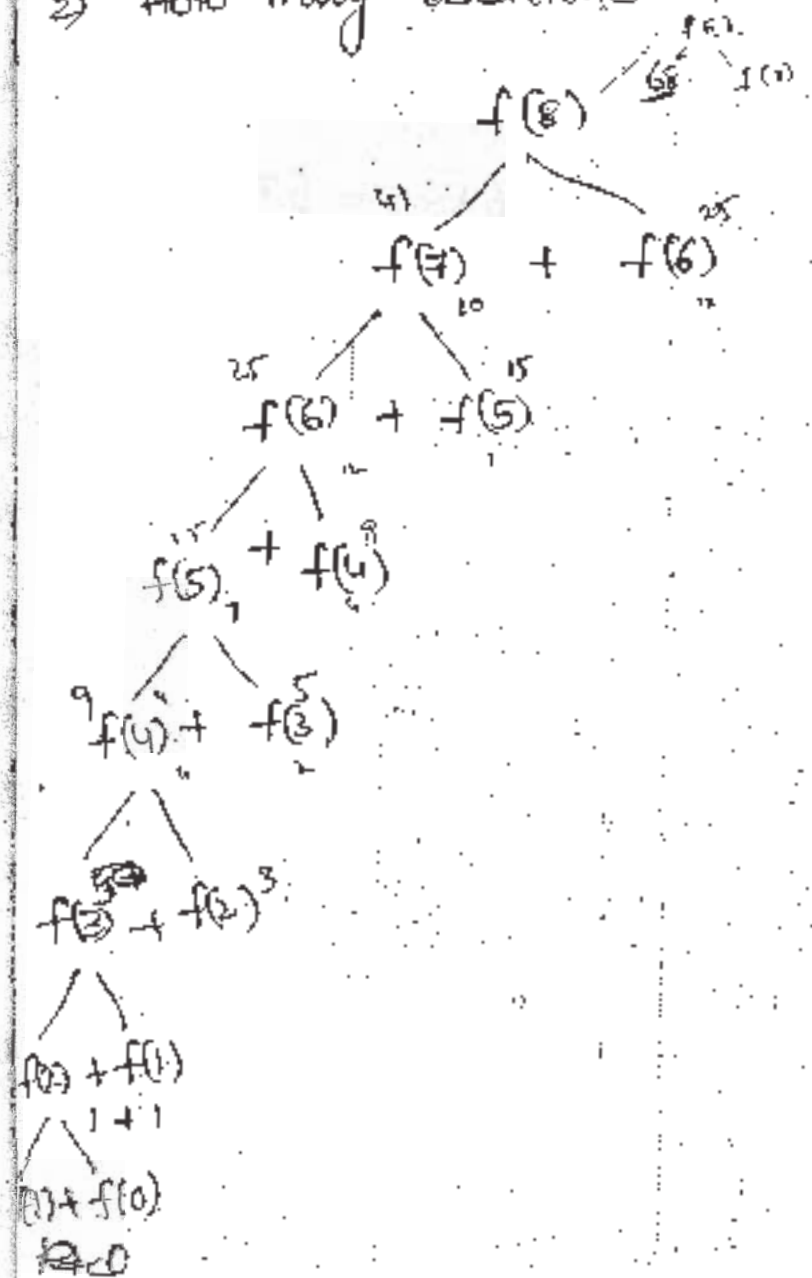
obs: return value = no of invocations

excessive recursion:

Eg: fibonacci series

$$fib(n) = 0, \quad n=0$$
$$= 1, \quad n=1$$
$$= fib(n-1) + fib(n-2), \quad n>1$$

1) How many invocations for evaluating fib(8) = ?
2) How many additions " " fib(9) = ?



f1-b sequence

n	0	1	2	3	4	5	6	7	8	9	10
f(n)	0	1	1	2	3	5	8	13	21	34	55
Invocations				5	9				67		
additions				2	4				33	54	

Invocations of $f(n) = 2*f(n+1) - 1$

$f(8) = 2*f(9) - 1$

$= 66 - 1 = 67$

Additions of $f(n) = f(n+1) - 1$

$f(9) = f(10) - 1$

$= 55 - 1 = 54$

24/7/11

(P) fillup the blank with options a) n
b) n-1
c) n+1
d) 0

$$pow(x,n) = \begin{cases} 1, & n=0 \\ 0, & x=0 \\ x*pow(x,n-1), & n>0 \\ \frac{1}{x} * pow(x,?), & n<0 \end{cases}$$

Soln $n=0 \Rightarrow x^n = x^0 = 1$

$x=0 \Rightarrow x^n = 0^n = 0$

$2^3 = 2 * 2^{3-1} \Rightarrow x^n = x * x^{n-1}$

$2^3 = 2^1 * 2^{n+1}$ $\Rightarrow x^n = \frac{1}{x} * x^{n+1}$; $n<0$

option ©

(P) Modify the above routine in such a way that the no. of multiplications will be less than 10 for evaluating 2^{37}

Hint: the function is not excessive.

Excessive:

Does not remember the previously evaluated values

Not excessive: Remembers previously evaluated values

Eg:

$2^3 = 2 * 2^2$

$2 * 2^1$

$2 * 2^0$

$2 * 2 * 2 * 2$ → 3 multiplications

If $x^n = x^1$ if $n=1$

$\Rightarrow$ $2^3 = 2 * 2^2$

$2 * 2^1$

2

$2 * 2 * 2$ → 2 multiplications

for 2^3 ⇒ 3(or) 2 multiplications

2^{37} ⇒ 31(or) 36 multiplications

⇓

10 multiplications

Drastically is to be decreased

Division operations

$2^4 = 2^{4/2} * 2^{4/2} = 2^2 * 2^2$

$2^5 = 2^{5/2} * 2^{5/2} = 2^2 * 2^2 * 2$

$$pow(x,n) = \begin{cases} x & \text{if } n=1 \\ 0 & \text{if } n=0 \\ pow(x,\frac{n}{2}) * pow(x,\frac{n}{2}) & \text{if } n \text{ is even} \\ x * pow(x,\frac{n}{2}) * pow(x,\frac{n}{2}) & \text{if } n \text{ is odd} \end{cases}$$

$2^{37} = 2 * 2^{18} * 2^{18}$

$2^9 * 2^9$

$2 * 2^4 * 2^4$

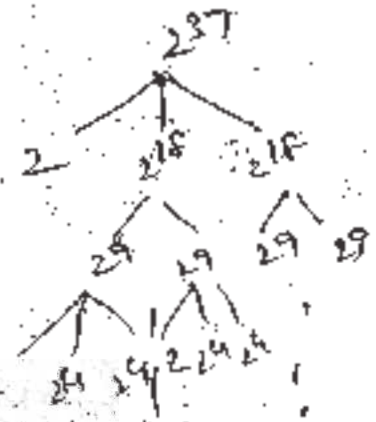
$2^2 * 2^2$

$2 * 2$

7 multiplications

Here if once 2^{18} is evaluated, other is no need to evaluate again as it is not excessive

If excessive then it will be 36 (or) 37 multiplications



Formulate Recursive procedure:

properties
- ① Base (or) Termination (or) anchor step
- ② Recursive (or) repeated (or) Inductive step