DATA STRUCTURE :- A Data structure is a particular way of storing and organising data in a computer, so that it can be used efficiently. The organised collection of data in Mathematical or logical way is called Data structure.

The study of diff Data structure includes :-

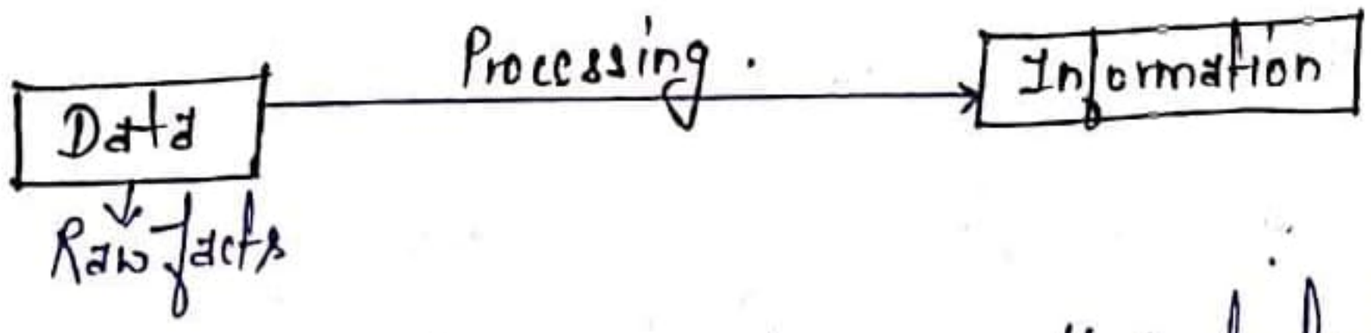
(i) logical description of Data ..

(ii) Implementation Detail of structure on a computer.

(iii) Quantative Analysis of Data structure which includes :- (a) Determing the Amount of Memory Needed to store the structure and

(b) time Required to process the structure.

DATA ORGANISATION :- Computer is an information processing system which is used to process Raw data to produce information.



Data and Data items :- ⊙ Datas are collec^n of facts and figure.

⊙ Data are simply values or set of values.

⊙ A Data item refers to a single unit of Values.

DATA ITEMS

Group items

Data items that are Divided into sub-items

(जिस Data item को हम split कर पाए. Eg. Name को हम split कर सकते हैं - first Name, Middle Name and sir Name.

Elementary item

Data items that are Not Divided into sub-items.

Eg. (i) Roll No. of a student

(ii) Phone No.

(iii) Age of someone.

Downloaded from TwoWaits

Scanned by CamScanner

Eg. (i) A student Name may be divided into 3 sub items - first Name; Middle Name & last Name.

(ii) Address is a group item, it includes Area, locality, street, city, state, country, etc.

These all items are not divided into sub-item or parts.

BASIC TERMINOLOGIES :-

(i) **ENTITY** :- An Entity has certain attributes or properties with Values. The Values May be Numeric or Non-Numeric.

Numeric (integer, float Values)

Non-Numeric (string, character)

Eg. student is an entity. It has properties, like :-

Roll No. (Numeric Value), Name (Non-Numeric), age (Numeric), Marks, etc...

(ii) **FIELD** :- An field is used to Represent an attribute of an Entity.

Eg. student's Roll No. is field or Name, age, Marks are diff field.

(iii) **RECORDS** :- It is a collection of field values of a given entity.

Eg. Record of one student in tabular form :-

Name	Roll No.	Marks	Age

(iv) **FILE** :- File is the collection of Records of the entity in the given Entity set. Eg. A file containing Records of students of a particular class. Eg. Record of Many students i.e. their information in a huge form.

(v) **PRIMARY KEY OR KEY** :- Each Record in a file May contain Many field items. In each Records there exists field, that uniquely determines the Records in the file. The unique field of a Record is called a primary key field.

(हमें ऐसा field identify करना है, जो दूसरे field से Unique है)

Eg. Name → दो बच्चों का same हो सकता है, तो ये key नहीं है)

D.O.B → Not key, Age → Not key, Address → key (H.No. अलग होगा)

Roll No. is a key क्योंकि सबका Roll. No. अलग है)

Marks → Not key (same हो सकते हैं)

etc...

Downloaded from TwoWaits

Scanned by CamScanner

DATA STRUCTURE OPERATIONS :-

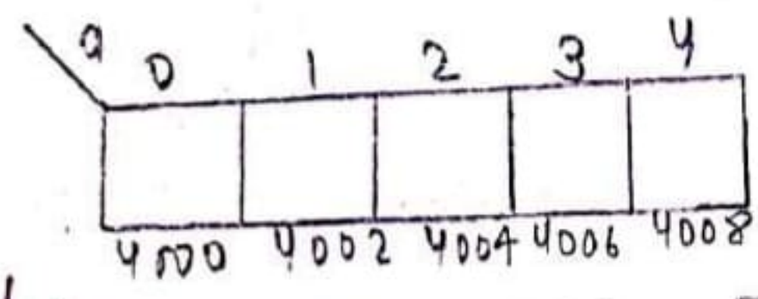
- Traversing Means One by One सारे Nodes को visit करना। Eg. To print Array elements one by one. हर एक

1. Traversing :- Accessing each Record exactly once (Data को एक बार seen करना)
2. Insertion :- Adding a New Record to the structure.
3. Deletion :- Removing a Record from the structure.
4. Searching :- Finding the location of the Record with a given key value.
5. Sorting :- Arranging the Records in some logical orders.
6. Merging :- Combining the Records in two diff sorted files into a single sorted file.

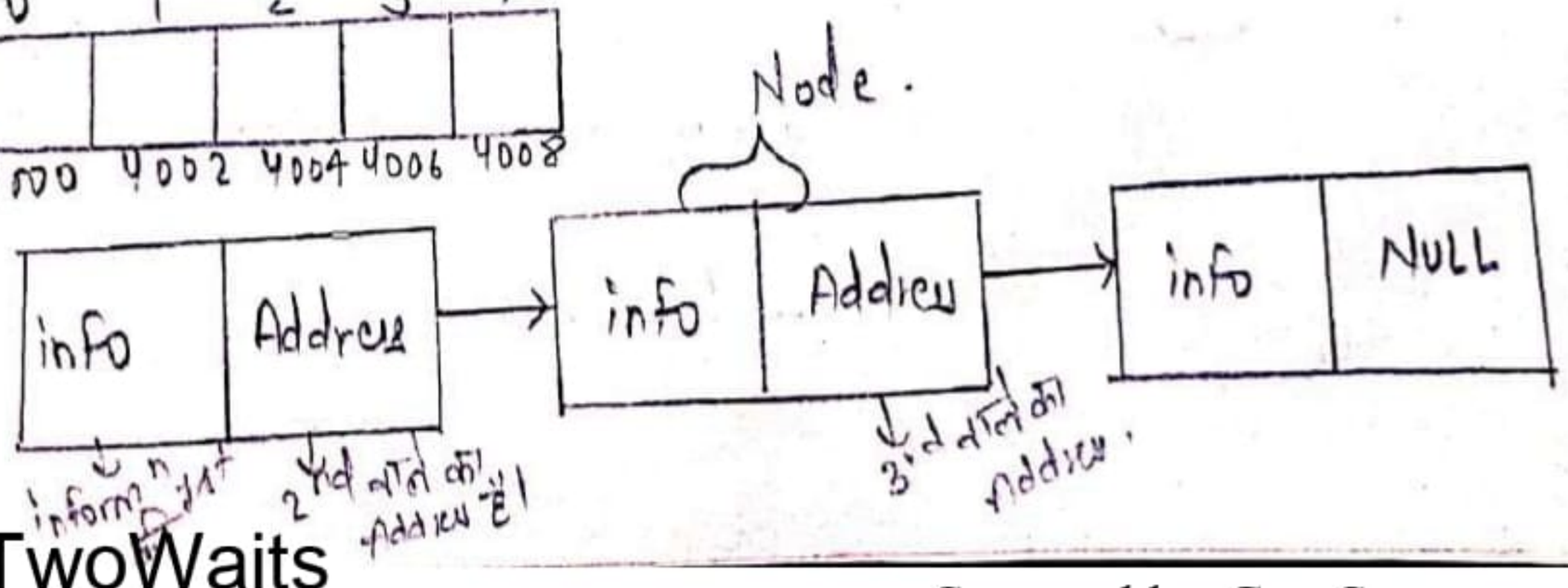
TYPES OF DATA STRUCTURE :-

Linear Data structure	Non-linear Data structure
Array, linked list, stack, Queue.	Tree, Graph.
(i) Every item is Related to it's previous and Next term.	(i) Every item is attached with Many other items.
(ii) Data is Arranged in linear sequence.	(ii) Data is Not Arranged in the sequence.
(iii) Atmost two ways of traversing at possible	(iii) More than two ways of traversing is possible.
(iv) Implementation is Easy.	(iv) Implementation is Difficult.
(v) Eg. Array, stack, etc.	(v) Eg. Tree or graph.

* Array :-



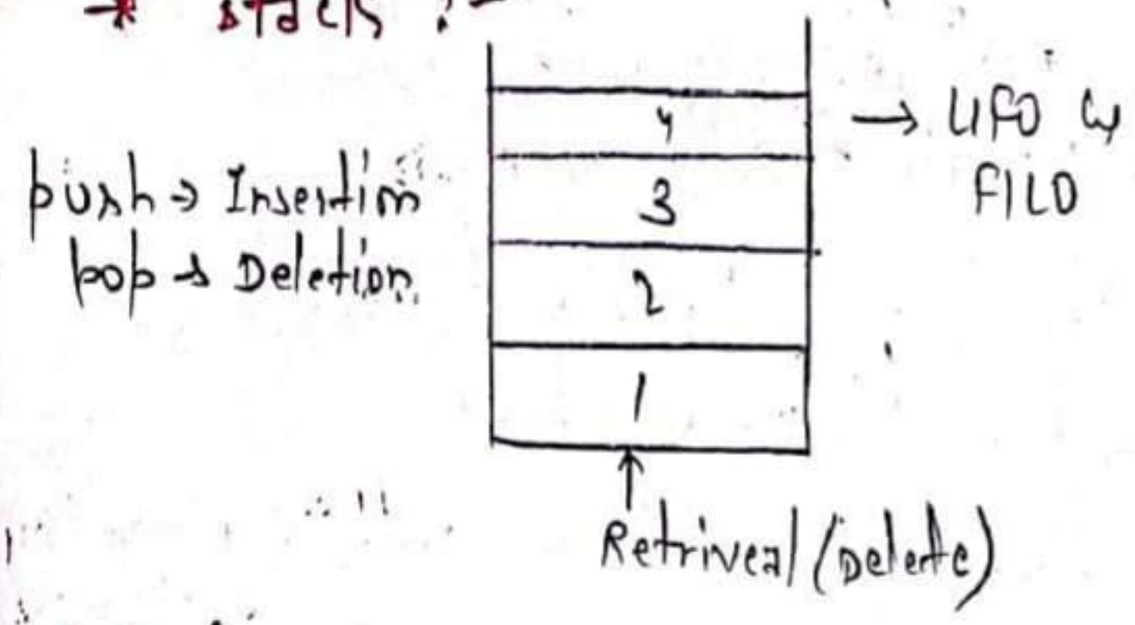
* linked list :-



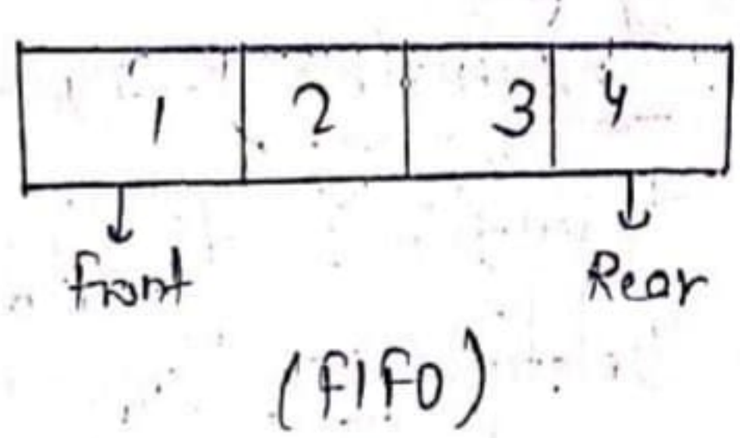
Downloaded from TwoWaits

Scanned by CamScanner

* stack :-



* Queue :-



(a) LINEAR DATA STRUCTURE :-

(i) Array :- An Array is a collec^n of data items having the same data type.

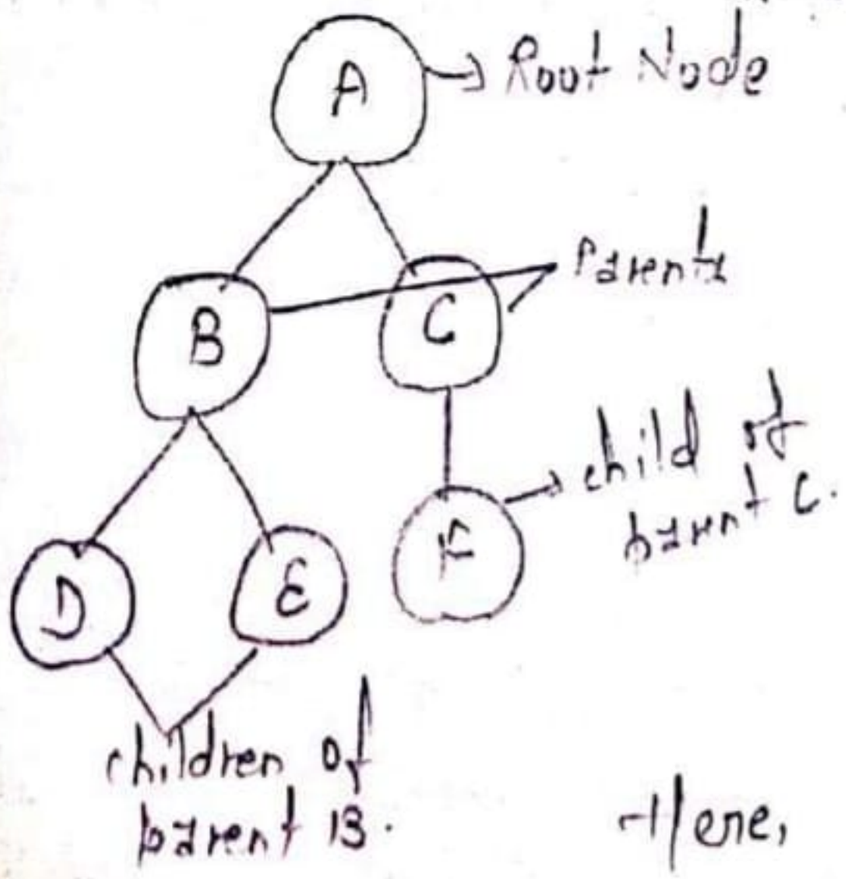
(ii) Linked list :- A linked list is a collec^n of Nodes, where each Node is Made up of a data element w a reference to the next Node in the sequence.

(iii) Stack :- Stack is a LIFO (Last in first out) data structure where elements that added at the end will be deleted first.
It has two operations :- push and pop.
It is also called FILO (First in last out).

(iv) Queue :- Queue is a FIFO (First in first out) Data structure where elements that Added first will be deleted first.

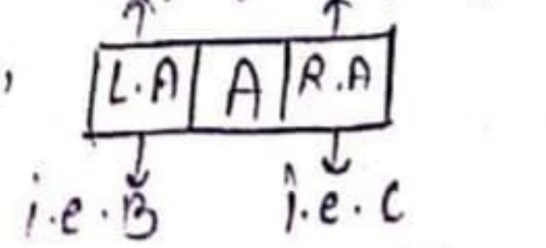
(b) NON-LINEAR DATA STRUCTURE :-

(i) Tree :- Tree is a collec^n of Nodes where Nodes are arranged in hierarchly and form a parent-child Relationship.



parent-child Relationship.

Here,



for traversing :-
* Pre-order में पहले Root Node (A) पर जायेगे फिर left के (B) and then Right के (C) पर जायेगा।
* In-order or Post-order देखे है।

for traversing, there that two Methods :-

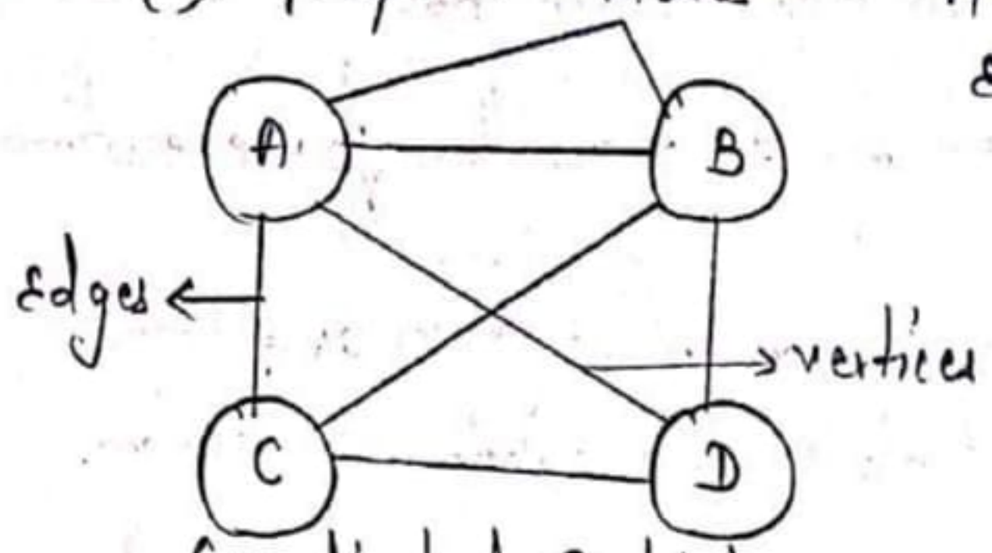
(i) Pre-order - Root L R
(ii) In-order - L Root R
(iii) Post order - L R Root

Downloaded from TwoWaits

Scanned by CamScanner

(ii) Graph :- Nodes

A Graph is a finite No. of vertices and an edges that connect these vertices.



(Undirected Graph)

ALGORITHM...

An algorithm is a finite sequence of instructions for solving a problem. We can design an algorithm to get a soln of a given problem. A problem can be solved in more than one ways.

* properties of an algorithm :-

(i) Input :- It may take zero or more no. of arguments.

(ii) output :- It processes input presented to it & produces at least one value as output.

(iii) finiteness :- Every algorithm must terminate after finite no. of steps

(iv) Definiteness (unambigious) :- Each instruc^n in an algorithm must be clear & unambigious.

(v) effectiveness (feasible) :- It is not enough that each operation be definite but it must be (feasible). → Algo^m Real life में Execute कर सकते हैं या नहीं।

practical implementation.

* Efficiency of an algorithm :- The efficiency of an algo^m can be computed by determining the amount of resources it consumes. factors that affect the efficiency of a program includes the speed of a Machine, compiler, operating system, the programming lang. and the size of the input.

* Why analyze algorithm :- While analyse algorithm, time and space are two important factors.

(a) Time :- The CPU time required to execute the algo^m.

Downloaded from TwoWaits

Scanned by CamScanner

It Means time required to produce an output, when input is given.

(ii) Space :- The amount of Memory used by an algorithm for execution.

* What is Analysis?:- It provides inform^n that gives us a General idea of how long an Algo^m will take for solving a problem.

* Complexity :- The complexity of an Algo^m is the func^n which gives the running time or space in terms of input size by it Measures the efficiency of an algo^m.

So, complexity of an algo^m 'm' is the func^n F(n), which gives the running time and storage space requirement of an algo^m.

Here, 'm' is the algo^m and 'n' is the size of input Data.

Complexity is of two types :- (i) time and (ii) space. } same as above.

* Cases considered during Analysis :- 1. Best case
2. Worst Case
3. Average Case

(i) Best Case :- With this input algo^m, takes shortest time to execute.
Eg. Searching algo^m. (Value found at 1st location)

(ii) Worst Case :- It takes highest time to execute.
Eg. Searching algo^m (Value found at last location).

(iii) Average case :- It gives average performance. Average case Analysis is a fourth step process :-

(i) Determing the No. of Diff groups into which All possible input sets can be divided.

(ii) Determine the probability that a input will come from each of these groups.

(iii) Determine the how long the algo^m will run from each of these groups.

(iv) Calculate, Average Case time using the formula :-

$$A(n) = \sum_{i=1}^{m} P_i * t_i$$

Downloaded from TwoWaits

Scanned by CamScanner

where, n is the size of input.
m is the No. of groups.
P_i stands for probability that input will be from group i.
and t_i stands for time that the algo^m will takes for the input from group i.

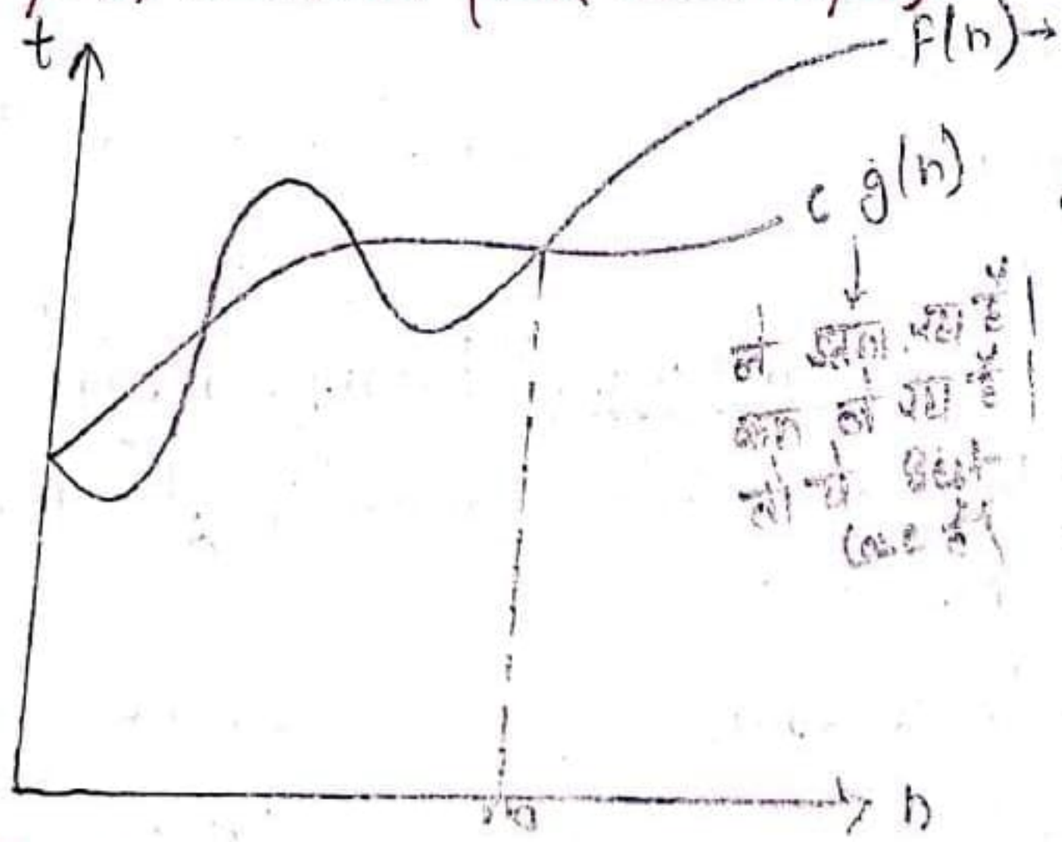
* ASYMPOTATIC NOTATION OR RATE OF GROWTH :- *

Asympotatic Notations are used to describes the Running time of an Algorithm. This shows the order of growth of function.

There are three types of Notations :-

(i) Omega Notation
(ii) Big O Notation
(iii) Theta Notation.

(i) OMEGA NOTATION (FOR BEST CASE):-



$f(n) \rightarrow$ हमने सोचा था की इतना time And space occupy होगा But

$c\,g(n)$ → ये इतना रहा है | कम से कम ये रहा है | तो ये Best case है

$$f(n) = \Omega\, g(n)$$

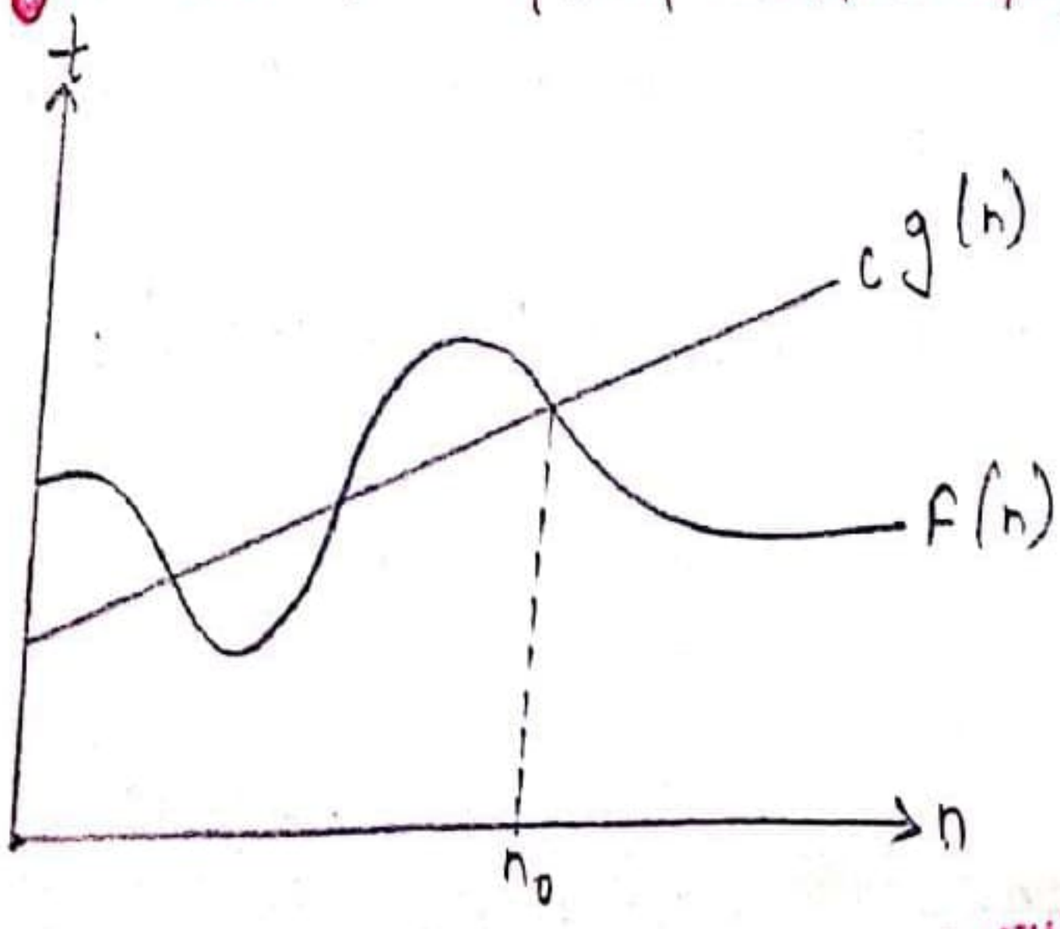
$$f(n) \geq c\,g(n)$$

$c \rightarrow$ constant.

$n \geq n_0$

$c > 0,\ n_0 \geq 1$.

(ii) Big O NOTATION (FOR WORST CASE):-



* $F(n)$ वो है जो हमने सोचा था, कि इतना time, या space लगा But $c\,g(n)$ actual है, जो ज्यादा time या space ले रहा है | तो ये worst case है |

$$f(n) = O\, g(n)$$

$$f(n) \leq c\, g(n)$$

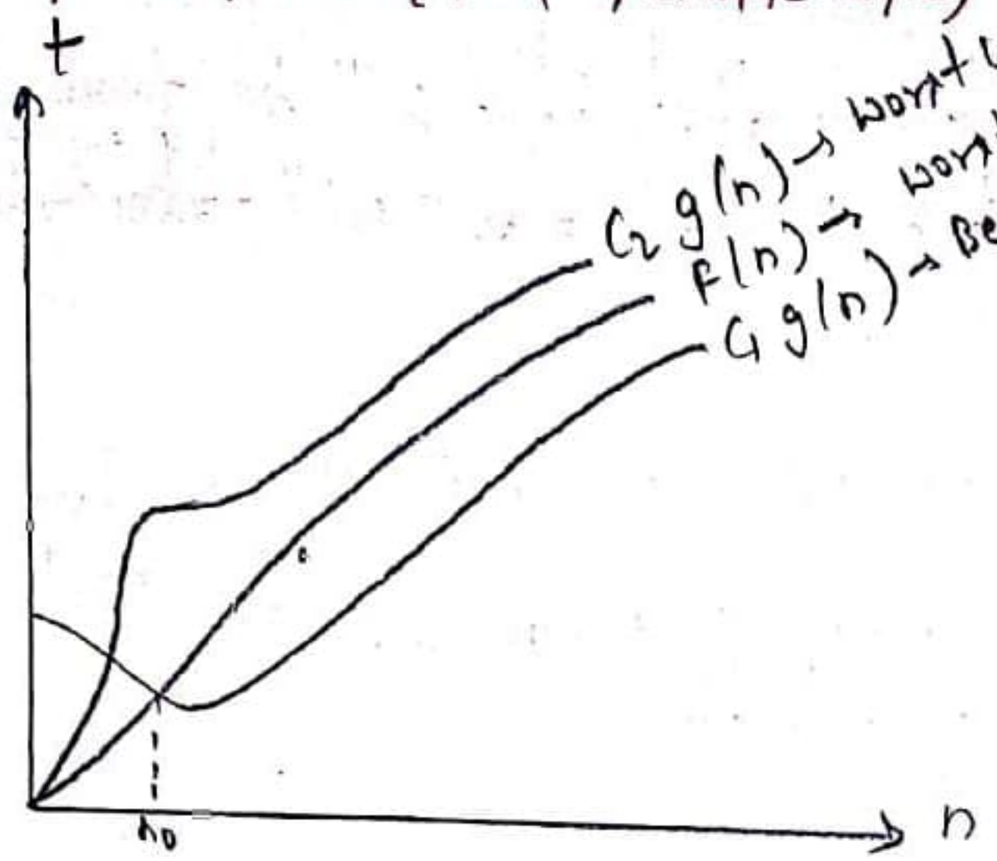
$n \geq n_0$

$c > 0,\ n_0 \geq 1$

Downloaded from TwoWaits

Scanned by CamScanner

(iii) THETA NOTATION (FOR AVERAGE CASE) :-



$f(n) = \Theta g(n)$.

ARRAY :- An Array is a collection of ordered set of Homogeneous Data. Array is used to store large amount of Data of same type.

Characteristics of an Array :-

(i) Homogeneous :- An array holds elements that have the same Data type.

(ii) Array elements are stored in Contigeous Memory location.

(iii) Array size should be Mentioned in the Declaration, it Must be a constant expression, and Not a variable.

(iv) Array Name represents the address of the starting element

(v) Array's Index starts with '0' and ends with '$N-1$'.

(vi) When an Array is Declared and Not initialized, it contains Garbage Values and if the Array is Declared at static, all elements are initialized to 'zero'.

(vii) Bound checking :- Array doesn't perform Bound checking.

ARRAY
- One-Dimensional Array
- 2-Dimensional Array.
- Multi-Dimensional Array.

Downloaded from TwoWaits

Scanned by CamScanner

(i) **One - Dimensional Array** :- It is a type of linear array, in 1-D array single subscript is present which is used to access the individual elements of an array.
(subscript [ ]) → It shows index value

Subscript, Ranges from 0 to N-1, where 'N' is the size of array.

for Eg. N = 5
Range is 0 to (5-1)
= 0 to 4 → upper bound.
↓ lower bound

* The lower bound of the index is 0 and the upper bound of index is N-1.

Declaration Syntax :-

< storage class > < Data type > < array Name > [< size of an array >];

Eg. int a[5] or static int a[5];
→ a[0] → lower bound
a[4] → upper bound.

→ Range is 0 - 4.
→ Index variable a[0] - a[4]
→ Max. elements :- 4+1 = 5.

Hence, $\boxed{\text{Maximum elements} = \text{upper bound} + 1.}$
OR
size of an Array = U.B + 1.

ADDRESS CALCULATION IN A 1-D ARRAY :- When the Array is Declared then the Memory is allocated to the Array by the compiler and operating system with the help of size of an Array and type of an Array.

$\boxed{\text{Memory allocation} = \text{Type of array} * \text{size of an Array}}$.

Eg. = int a[5];
= 2 * 5
= 10 bytes
(int has 2 Byte storage capacity).

→ All the Memory allocated to a particular array is consecutive / or contigenoous) Memory allocation.
→ Array Name is a pointer Variable, which stores the Base address (Address of starting element).

Downloaded from TwoWaits

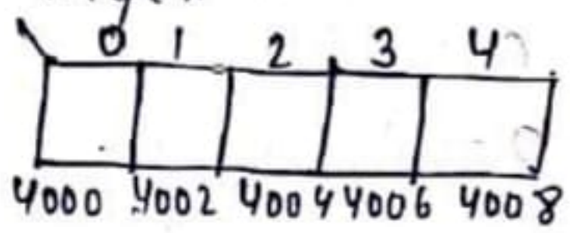
Scanned by CamScanner

ADDRESS OF INDEX 'i' ELEMENT :-

FORMULA :- $\boxed{B + W * i}$

Where, B = Base Address
W = size of Data type
i = index.

for Eg. a



Here, int a[5]

To find index '3' element's Address :-

$= 4000 + 2 \times 3$

$= 4006$.

NOTE :- a [L : U]
L → lower Bound, U → Upper Bound

Eg. a [5 : 15] → अगर L.B '0' से start नहीं है, तो ये formula use करेंगे।
5 → L.B., 15 → U.B.

Hence, for a[L:U] :-

FORMULA :- $\boxed{B + W * (i - L.B)}$

where L.B = Lower Bound.

for size of Array :- $\boxed{U.B - L.B + 1}$

Eg. a [5 : 15]

$= 15 - 5 + 1$

$= 11$.

Q:- Given an Array A [0 : 15], Base Address is 1000, size = 2, then Calculate the Address of A[6].

Ans :- Address of A[6] $= B + W * (i - LB)$

$= 1000 + 2 * (6 - 0)$

$= 1012$.

Q:- Given an Array with L.B = 0, address of A[10] = 2075 and Address of A[15] = 2090. Calculate the address of A[6]?

Downloaded from TwoWaits

Scanned by CamScanner

Ans :- $A[10] = 2075$ — (1)

$A[15] = 2090$ — (2)

By using (1) and (2), we create (3) eqn.

$$2075 = B + W * (1 - LB)$$

So, $2075 = B + W * (10 - 0)$

$$2075 = B + 10W \quad — (3)$$

By (2),

$$2090 = B + 15W \quad — (4)$$

By (3) and (4).

$$\begin{aligned} B + 10W &= 2075 \\ -\ B + 15W &= 2090 \\ \hline -5W &= -15 \end{aligned}$$

$$\boxed{W = 3}$$

So, $2090 = B + 15 \times W$ (By (4))

$$2090 = B + 15 \times 3$$

$$B = 2090 - 45$$

$$\boxed{B = 2045}$$

So, $A[6] = B + W * (i - L \cdot B)$

$$= 2045 + 3 * (6 - 0)$$

$$A[6] = 2045 + 18$$

$$A[6] = 2063$$

NOTE :- For Bound checking :-

For $a[-3]$ = Unpredictable index (if (-)ve index value)

For $a[5]$ = Value predictable (size की Range में ही आता है) and index valid

For $a[6]$ = Valid index but Value unpredictable → Index value (+)ve दोनों और वो size से बड़ा है।

Downloaded from TwoWaits

Scanned by CamScanner

Q:- Calculate the sum and Average of 10 no.s. No.s are input by keyboard.

Ans:-

```
// Calculate sum and Average.
# include <stdio.h>
# include <conio.h>
void main ()
{
(int or) float a[10], sum = 0;  int i;
printf (" Enter 10 No.s = ");
for (i= 0 ; i<10 ; i++)
{
scanf ("%f", &a[i]);
sum = sum + a[i];
}
printf ("sum of no.s are \t %f", sum);
printf (" average is = %f", sum/10);
getch();
}
```

Q:- print the array is Reverse order.

Ans:-

```
// Array in Reverse order
# include <stdio.h>
# include <conio.h>
void main ()
{
int a[10],
printf ("Enter 10 elements:");
for (i=0 ; i<=0 ; i++)
scanf ("%d", &a[i]);
for (i=9 ; i>=0 ; i--)
printf ("%d", a[i]);
```

Downloaded from TwoWaits

Scanned by CamScanner

```
getch();
}
```

OPERATIONS ON ARRAY :-

(i) INSERTION :-

Algorithm for Insertion (a[ ], len, pos, num)

Parameters of the Algo^m.

let 'a' is an Array,
'len' is the No. of elements present into the Array, 'pos' is index value at which data value is to be inserted, 'num' is data value to be inserted

step ① :- [initialize the value of i to len-1]
i = len-1

step ② :- [shift the element at the array to right direction by one position].
Repeat for i = len-1 to pos(index)
set a[i+1] = a[i]
[end loop]

Eg:-

0	1	2	3	4	5

step ③ :- [insert Data into the array]
set a[pos] = num

step ④ :- [update the length of array]
len = len+1

step ⑤ :- [Display the updated array]

step ⑥ :- stop.

program for Insertion (By using func^n)

```
// Insertion
# include <stdio.h>
# include <conio.h>
void insertion (int a[ ], int len, int pos, int num)
{ i = len-1;
  for ( ; i >= pos; i--)
```

Downloaded from TwoWaits

Scanned by CamScanner

```
    a [i+1] = a[i];
  }
  a [pos] = num;
  len = len+1;
  For (i=0; i<len; i++)→ loop to print update Array.
  {
    printf ("%d\t", a[i]);
  }
  getch();
}
```

(ii) DELETION :- Eg. int a[10].

① original array :-

	0	1	2	3	4	5	6	7	8	9
a	5	6	1	2	3	4	9	7		
	100	102	104	106	108	110	112	114	116	118

② To delete the value of index 4 :-

Deleted Element.

	0	1	2	3	~~4~~	5	6	7	8	9
a	5	6	1	2		4	9	7		

shifting elements towards left to occupy the space of the Deleted element.

③

	0	1	2	3	4	5	6	7	8	9
a	5	6	1	2	4	9	7			

→ then, we will ↓ the value of length by 1.
i.e. len = len-1.

Algorithm for Deletion :- (a[], len, pos, num)

Here, 'a[]' is an Array, len is the no. of elements present into Array, 'pos' is index at which delete operation will be performed, 'num' is the value to store the Deleted Data from Array.

step ① :- [store the Deleted element]
set, num = a [pos]

step ② :- [Initialise the Variable].
set, i = pos

step ③ :- [shift the elements of the Array to the left by 1 position]

Downloaded from TwoWaits

Scanned by CamScanner

Repeat for i = pos to len-1
a[i] = a[i+1]
[end of loop].

step ④ :- [update the length of Array]
set len = len-1

step ⑤ :- [Display the updated Array]

step ⑥ :- stop

* अगर 'Value' जो Delete हुई है, उसे Return करना है तो 'int' लिखेंगे नहीं तो Void लिखेंगे।

Program for Deletion ...

```
int dete (int a[], int len, int pos)
{
  int num, i ;
  num = a [pos] ;
  For ( i = pos ; i < len ; i++)
  {
    a[i] = a[i+1];
  }
  len = len-1 ;
  For (i=0 ; i<len ; i++)
  printf ( "%d\t ", a[i]);
  return(num) ;
}
```

→ ये Parameters Globally Declare करते हैं, क्योंकि ये throughout the program same रहते हैं, Becoz ये parameters user देता है।

→ i and num change होंगे। इसलिए ये locally declared parameters हैं।

→ loop To print updated Array.

→ value Return होगी।

(iii) SEARCHING :- There are two types of searching :- (i) linear Search
(ii) Binary Search.

LINEAR SEARCH :- Search Data is Matched with the index 'zero', if it is Matched, then we return the index value otherwise we Match the Search Data with the element present at Next index. This process will be repeated until :-

Downloaded from TwoWaits

Scanned by CamScanner

(i) Search Data found.
(ii) We Reached at the end of the Array.

Eg. int a[5];

	0	1	2	3	4
a	7	8	1	2	4

The item to be searched

item = 1

loc = -1

Algorithm for Binary Search :- (a[ ], len, item)

Here, len is a no. of item present in Array, a[ ] is an array, item is the data to be searched into Array. 'i' used for index value 'loc' is used for location that searched element is found or not.

step ① :- [Initialise the Variable]
set i = 0
set loc = -1

step ② :- [Repeat steps ③ and ④]
while (i < n && loc == -1)

step ③ :- if a[i] == item, then
set loc = i

step ④ :- set i = i+1
[End of loop]

step ⑤ :- If loc = -1
then print "search item is not present"
else "search item is present at location (index)

step ⑥ :- stop.

Program for Binary Search :-

```
void search( int a[ ], int len, int item)
{
  int i=0;
  int loc = -1;
 while ( i < len && loc == -1)
 {
```

Downloaded from TwoWalls

Scanned by CamScanner

```
if (a[i] == item)
  {loc = i;
  }
  i++;
}
  if (loc == -1)
printf(" element not found");
else
  printf(" element found at %d index", loc);
}
```

(iv) **TRAVERSING** :- हर एक No. of visit करेगा

Algorithm for traversing :-

let a[L : U] is 1-D Array in which 'a' is an array, 'L' is the lower bound & 'U' is the upper bound.

step ① :- [Initialization of variable]
set i = L

step ② :- Repeat for i = L to U.

step ③ :- Apply process to a[i] (उसको print करेगा सारी values Array की) (या scanf करेगा मतलब Address देगा)
Scanf / Printf

step ④ :- [increase counter]
set i = i+1
[End of loop]

step ⑤ :- stop.

function for traversing :-

```
void Traverse (int a[], int L, int U)
{
  int i = L;
  for ( ; i ≤ U; i++)
  {printf ("%d", a[i]);
  }
```

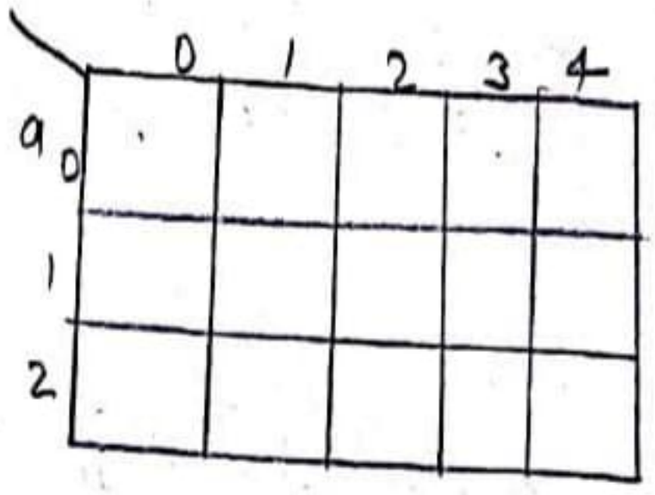
Downloaded from TwoWaits

Scanned by CamScanner

TWO-DIMENSIONAL ARRAY :-

int a [3] [5];

m → No. of Rows, n → No. of columns



Base Address = 100

No. of element = Row x Column
= 3 x 5
= 15

Size of Array = 15 x 2 (for int)
= 30.

* इस Address 2-D Array में Row Major, Column Major या दोनों से निकाला जा सकता है।

*, a [-2 : 20 , -5 : 17]

No. of Rows, No. of columns

No. of elements in Row = 23
No. of elements in Column = 23

Two-Dimensional Array is defined as an array of arrays or it is a collection of rows and columns.

Declaration of 2-D Array :-

< data type > < name of array > [size 1] [size 2].

Here, size 1 - represents the No. of Rows and
size 2 - represents the No. of columns.

Eg. int a [3] [5];

Total No. of elements in 2-Darray = size 1 x size 2.

Total Memory Required = No. of elements x size of Datatype.

Eg. shown above.

INPUT (In 2-D Array)

- Compile time
- Run time (scanf)

Downloaded from TwoWaits

Scanned by CamScanner

In compile time :- int a[2][4] = {{1, 3, 5, 6}, {4, 5, 7, 9}}

4 columns 4 columns

ROW 1 ROW 2

In Run time :-

Here, for (i=0; i<2; i++)

{ for (j=0; j<4; j++)

scanf ("%d", &a[i][j]);

}

ADRESS CALCULATION IN 2-D ARRAY :-

METHOD 1 :- ROW MAJOR IMPLEMENTATION...

In this Method, first of all, all the elements of 1st Row is stored and after that, all the elements of 2nd Row and so on....

* Base Address or starting address is stored in Index [0][0].

No. of elements in 2-D Array :- = $\underbrace{(U_1 - L_1 + 1)}_{\text{for Row}} * \underbrace{(U_2 - L_2 + 1)}_{\text{for column}}$.

For Eg. [-2 : 20, -3 : 20]

L_1 U_1 L_2 U_2

* int a[3][4];

m n

Base address = 100

Eg. Calculate the Address of a[2][1] ?

= 100 + 2 * (2 * 4 + 1)

= 118

$$\boxed{\text{Address} = B + W * ((i - L_1) * n + (j - L_2))}$$

And if index = 0.

$$\text{Address} = B + W * [(i * n) + j].$$

where, B = Base address, W = size of Datatype.

n = no. of columns in 2-D Array.

i = Row index of the element which address is to be calculated.

j = column index of Element which index is to be calculated.

Downloaded from TwoWalls

Scanned by CamScanner

L_1 = lower index of Row.
L_2 = lower index of column

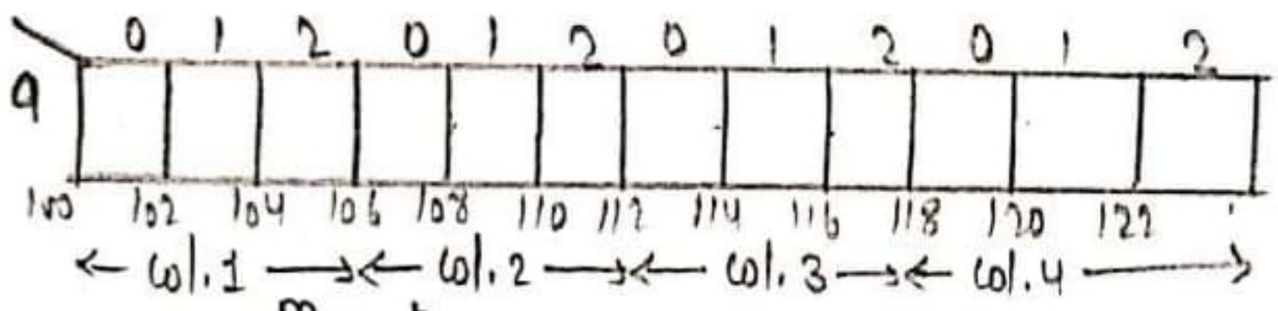
Q:- A 2-D Array is defined as $a[4---7, -1---3]$ by here
$W \rightarrow 2$ Bytes.
$B = 100$
Calculate Address $a[\overset{i}{6}][\overset{j}{2}]$, By Row Major?

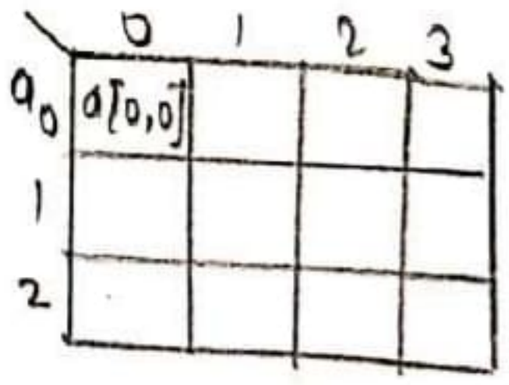
Ans:- Address $= B + W * [(i - L_1) * n) + (j - L_2)]$

$n = 5$ (No. of columns), $i = 6$
$L_1 = 4$ $\quad j = 2$.
$L_2 = -1$.

Address $= 100 + 2 * [((6-4) * 5) + (2+1)]$
$= 100 * 2 [10 + 3]$
$= 100 * 2 \times 13$
$= 196$.

METHOD:-2 COLUMN MAJOR IMPLEMENTATION...





int $a[\overset{m}{3}][\overset{n}{4}]$;

Eg. Calculate the Address of $a[\overset{i}{2}][\overset{j}{1}]$?

Ans:- $= 100 + 2 * ((1 * 3) + 2)$
$= 100 + 2 * (5)$
$= 100 + 10$
$= 110$

If, Index = 0
Address $= B + W * [(j * m) + i]$

and $\boxed{\text{Address} = B + W * [(j - L_2) * m + (i - L_1)]}$

Downloaded from TwoWaits

Scanned by CamScanner

First of all, all the elements of 1st column is stored and after that all the elements of 2nd column and so on.

Q:- Each element of an 2-D Array Requires 4-Bytes of storage. Base Address is 1000. Calculate Address using colum major implementation of A[0,30].

A [-20---20, 10---25]

given :- B = 1000
w = 4
i = 0
j = 30
m = 41
n = 16
L_1 = -20
L_2 = 10.

Ans :- Address = $B + w[(j - L_2) * m + (i - L_1)]$

$= 1000 + 4 * ((30 - 10) * 41 + (0 + 20))$

$= 1000 + 4 * (20 * 41 + 20)$

$= 1000 + 4 * (820 + 20)$

$= 1000 + 4 * 840$

$= 1000 + 3360$

Add. = 4360 Ans...

Ap,

$$\boxed{\begin{aligned} m &= (U_1 - L_1 + 1) \\ n &= (U_2 - L_2 + 1) \end{aligned}}$$

For array :- like...
A[20 - - 20, 10 - - - 25]
m n.

Given :-

Q :- A[-2 : 4, -6 : 10]
size :- 8 Byte.
Base Address = 400

. Calculate Address of A[3, -5] by Row and column implementation.

given :- i = 3
j = -5
m = 7
n = 17
L_1 = -2
L_2 = -6

Ans :- (a) By Row Major :- Add. = $B + w * ((i - L_1) * n + (j - L_2))$

$= 400 + 8 * ((3 + 2) * 17 + (-5 + 6))$

$= 400 + 8 * (5 * 17 + 1)$

$= 400 + 8 * (85 + 1)$

$= 400 + 8 * 86$

$= 1088$

(b) By column Major :- Add. = $B + w((j - L_2)) * m + (i - L_1))$

$= 400 + 8((-5 + 6) * 7 + (3 + 2))$

$= 400 + 8(7 + 5)$

$= 400 + 8(12)$

$= 400 + 96$

$= 496.$

Downloaded from TwoWaits

Scanned by CamScanner

SPARSE MATRIX ...

Sparse Matrix is a Matrix in which many or most elements having value zero. it means very low No. of Non-Zero value is present.

Problems of sparse Matrix :-

* We waste a lot of space to represent the sparse Matrix becoz the amount of Meaningful data to store into the Memory is very less.

Solution :- This problem can be removed by only storing Meaningful Data of sparse Matrix, with the help of Triplet.

Triplet :- A triplet can be used to store Meaningful data and it is a Array Representation of data. It contains 3 - columns. 1st column contains row index, 2nd contains column index 3rd contains Value. and 1st Row contains total No. of Rows, total No. of columns and total No. of Meaningful Data in sparse Matrix. In this way Memory utilization can be ↑ed.

Eg. int a[10][10];

Row \ Col	0	1	2	3	4	5	6	7	8	9
0	2									
1		9								
2			7							
3				6						
4					5					
5						4				
6							3			
7								2		
8									1	
9										9

* Blank boxes occupy '0' Element.
* ये sparse Matrix ज्यादा space का Memory लेता है।

Total Memory required = (10×10) × 2
= 200 Bytes

Information stored = 10×2
(Non-Zero Values) → 10.
= 20 Bytes.

Wastage of Memory = 200 − 20
= 180 Bytes

Downloaded from TwoWaits

Scanned by CamScanner

Triplet Representation :-

Row Index	Column Index	Value
10	10	10
0	0	2
1	1	9
2	2	7
3	3	6
4	4	5
5	5	4
6	6	3
7	7	2
8	8	1
9	9	9

→ This 1st Row shows the Total No. of Rows, Column and No. of Meaningful Data.

Total Memory Required = 11 × 3 = 33 elements
33 × 2 = 66 Bytes.
Information stored = 10 × 2 = 20 Bytes
Wastage of Memory = 66 − 20 = 46 Bytes.

STACK :-

It is a linear data structure. It is an ordered collection of items in which item may be deleted or inserted from the single end, which is called, top of the stack.

Stack can also be called LIFO (last in first out) or FILO (First in last out)

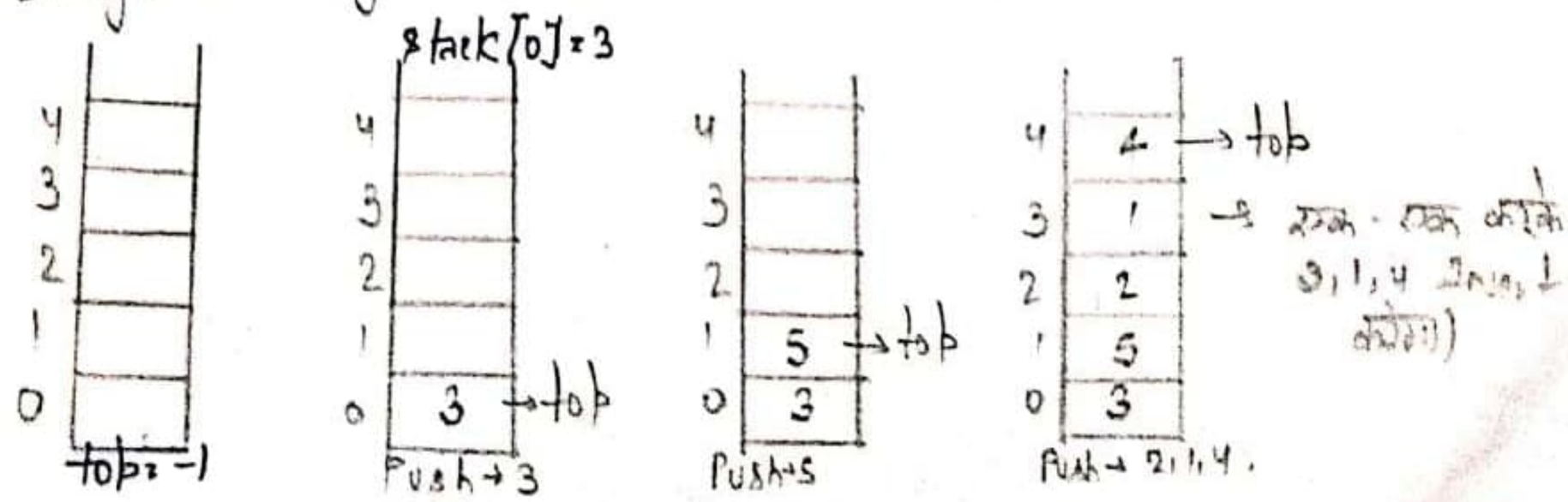
INSERTION :- Insertion operation is called Push operation.

DELETION :- Delete operation from the stack is called POP.

* कोई भी element अगर stack में Insert या Delete करना होता है, तो top of the stack से करते हैं starting

* stack में Integer या string कोई भी Values डाल सकते हैं।

PUSH Eg.



Downloaded from TwoWaits

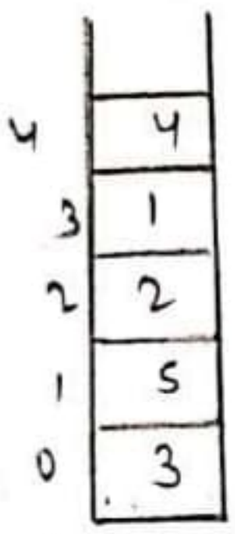
Scanned by CamScanner

In Diagram :- stack [maxsize]
top = -1

Here, int stack [s] → max size
top = -1

set top = top + 1
stack [top] = item } Eg. for 3.
top = -1
top = top + 1
= -1+1
top = 0.
stack [0] = 3.

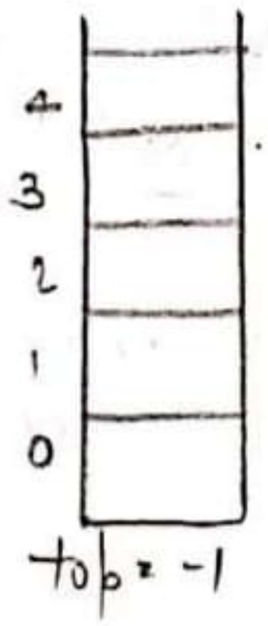
* अगर element Max. size से ज्यादा Insert हो रहे हैं, तो overflow of elements हो जाएगा
तो हम ये Msg print करेंगे :- if (top == Maxsize - 1)
printf ("overflow")
else
insert करेंगे elements. { top = top + 1
stack [top] = item.



Push → 10

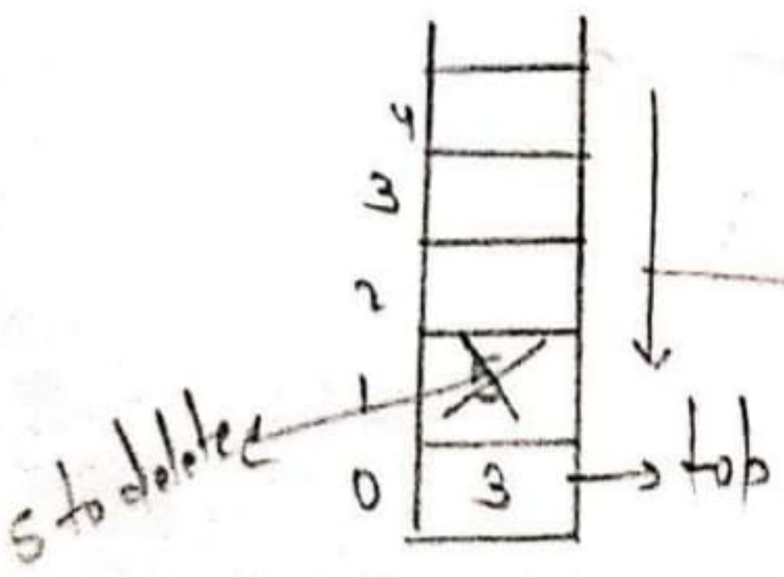
:- if (top == Maxsize - 1)
then, overflow...

POP Eg.



* अगर इस stack में कोई element नहीं है तो, Delete या pop नहीं कर सकते कुछ भी।
तो ये condition Underflow है।

→ (if top == -1)
then, underflow...



→ item = stack [top]
top = top - 1

→ ऊपर से नीचे Delete करते चलते हैं elements को तो, top की Value कम होती रहेगी एक-एक। इसलिए top = top - 1.

Downloaded from TwoWaits

Scanned by CamScanner

OPERATIONS ON STACK :-

① PUSH OPERATION :- In push operation, data is added to the stack at the top position. For the push operation :-

(i) Overflow condition is checked by (top == Maxsize - 1).

(ii) If the stack is Not Full, then top is increment by one and New Element is inserted at the New top position.

② POP OPERATION :- In the pop operation, top element of the stack is deleted. For the pop operation :-

(i) stack underflow condition is checked by (top == -1).

(ii) If the stack is Not Empty, then the top element of the stack is store in a variable and top is decrement by one.

③ TRAVERSE OPERATION :- In traversing, all the elements of the stack is displayed from top to bottom. For the traverse operation :-

(i) we check the underflow condition. (top == -1)

(ii) If the stack is not Empty then, loop is started from top to the zeroth index and element of the stack is Displayed.

① Algorithm for Push operation :- [stack, top] and [item].

Here, stack[] is an Array, item is which value we want to push, and top is a global variable. Here, int top = -1.

step ① :- [check for overflow condition]

```
if (top == Maxsize - 1)
print" stack is overflow and exit".
```

step ② :- [Increase the value of top by one]

```
set top = top+1
```

step ③ :- [Push item in the New position of top]

```
stack[top] = item
```

step ④ :- stop.

* stack ki ye restriction है, कि एक side से ही Data का input (push) या Retrieve (Pop / Delete) कर सकते है।

Downloaded from TwoWaits

Scanned by CamScanner

(ii) **Algorithm for pop operation :-** [stack[Maxsize], item].

Here, stack[Maxsize] is an Array (stack[ ]), item is the variable in which we want to store the pop item.

Step ① :- [Check the underflow condition]
if (top == -1)
print "stack is underflow and exit".

Step ② :- [Store the deleted item into an item variable]
item = stack [top]

Step ③ :- [Decrease the top by one]
set top = top - 1

Step ④ :- [Return the deleted element from the stack].

Step ⑤ :- Stop.

(iii) **Algorithm for traversing operation :-**

Step ① :- [check for Underflow]
if (top == -1)
then print " stack is underflow and exit".

Step ② :- [Initialise the Variable]
set i = top

Step ③ :- [Display all the elements of stack from top to Bottom]
Repeat for i = top to 0
Display stack [i].

Step ④ :- Stop.

PROGRAM ...

① program of structure :- (Avg. %age of 50 students)

```
# include <stdio.h>
struct student
{
  char name [20];
  int roll;
  int per;
```

Downloaded from TwoWaits

Scanned by CamScanner

* pointer value के लिए हमलोग structure में तो (&s→roll no.) Arrow लगायेंगे।
* simple value के लिए (.) Dot लगायेंगे (&s.roll no.)

```
};
typedef struct student stdt;
int main ()
{ stdt s[50];
  float sum = 0
  int i;
  for (i=0 ; i<50 ; i++)
  {
   printf (" Enter name, roll No., percentage = ");
  scanf (" %s %d %d ", s[i].name, &s[i].roll, &s[i].per);
  sum = sum + s[i].per ;
  }
  print ("Avg. percentage = %f ", sum/50.0);
  return 0;
}
```

* structure और Array दोनों में Contigeous Memory allocate होती है। But Array में हम Homogeneous elements store कर सकते हैं And structure में Heterogeneous elements. जैसे आगे Array में Name store करना है, और Marks भी तो दो अलग-अलग Data type के Array बनायेंगे, ex int a[10] और दूसरा char b[10], But structure लेकर उसमें string के साथ-साथ, int elements भी दे सकते हैं।

* In structure, we store the attributes of an Entity. * Here, Entity is "student" & it's Attributes are it's Roll No., %age, etc.

* हम हमेशा top की updated value पर work करेंगे।

(3) program of stack :- (For Pop, Push & traverse)...

```
#include <stdio.h>
#include <conio.h>
# define maxsize 10
typedef struct
{
 int stack [Maxsize]
  int top;
 } stack 1; → name of structure

// Funcn prototype ← void push (stack 1 *s, int item) ;
                    int pop (stack 1 *s) ;
                    void traverse (stack 1* s) ;
                    } funcn Declaration

void main()
{
 stack 1 s ; → Variable used for structure (struct 1)
 int item, choice ;
 char ch ;
 clrscr() ;
 do →
 {
```

→ हमने यहाँ Maxsize Define करा है। जहाँ-जहाँ Maxsize है, वहाँ '10' आ जायेगा। Agar हमें size change करना है, तो यहीं कर दो बस, उससे सारे में change हो जायेगा।

→ हमने Do while इसलिए लगाया है, ताकि एक बार तो Prog. Perform हो, then condition के according check करके Prog. perform हो।

Downloaded from TwoWaits

Scanned by CamScanner

```
clrscr();
printf("\n 1. push");
printf("\n 2. pop");
printf("\n 3. traverse");
printf("\n 4. Enter your choice");
scanf("%d", &choice);
switch(choice)
{
 case 1:
printf("\n enter the stack element");
 scanf("%d", &item);
  push(item, &s);  → funcn (goes to definition)
  break;
  case 2:  → यहाँ हमने element नहीं पूछा क्योंकि By default end value ही pop होगी।
  printf("\n the deleted item %d", pop(&s));  → funcn (goes to definn)
   break;
 case 3:
  traverse(&s);  → funcn (goes to definn)
   break;
   default;
printf("\n You have entered wrong choice");
 }
   fflush(stdin);
printf("\n do you wish to continue (Y/N)");
 scanf("%c", &ch);
}
while (ch==='y' || ch==='Y')  → हमें ये नहीं पता, कि User 'Y' या 'y' देगा, तो ये दोनों condn में अपने दोबारा work करेगा।
                              (Y या y)
}
Void push (int item, stack 1 *s)  → we define push funcn, here.
{
 If (s→top == Maxsize-1)
 {
 printf("stack overflow");
 }
 else
```

Downloaded from TwoWaits

Scanned by CamScanner

```
{
 s->top = s->top+1;
  s->stack[s->top] = item;
}
}
```

int pop(stack *s); → we define pop funcⁿ here.

```
{
 int m;
```
→ 'm' is the variable to store the deleted element in pop operation.

```
 if (s->top == -1)
 {
  printf("stack underflow");
 }
 else
  m = s->stack[s->top];
  s->top = s->top-1;
```

return(m); → कोई Deleted Value, pop से Return की है, ताकि Use करना हो तो आगे use कर सके। इसलिए 'm' से पहले ही store कर लिया Deleted Element.

```
}
```

void traverse(stack *s) → we define traverse funcⁿ here.

```
{
 int i;
 if (s->top == -1)
 printf("stack is Empty");
 else
 {
  printf("traverse the element");
  for(i = s->top; i >= 0; i--);
  printf("\n %d", s->stack[i]);
 }
}
```

NOTE... * We use the updated value of top always. We Declare "top" in struct Beoz main() में करेंगे तो फिर उसकी limit Main() तक ही रहेगी।

```
* void main()
  {
   int top = -1
   push(&top)
   top++
```
} top is Declared in Main()

* हमें top की value सिर्फ push & pop में use करनी है।
* हम top को Globally Declare नहीं कर सकते Beoz अगर Globally Declare करा तो वो push, pop or किसी भी funcⁿ में use किया जा सकता है। (जैसे password हमें ही पता होना चाहिए, just like that)

Isliye हम top को struct में Declare करेंगे ताकि वो hide रहे और हमें जहाँ use करना है, वहाँ use भी हो सके।

```
 top = -1
 {
  push(&top)
  pop(&top)
 }
```
} top is declared Globally

Downloaded from TwoWaits

Scanned by CamScanner

POINTER

Note :- (i) int * b, a ;
a = 5 ;
b = &a ;
So, printf ("%d", b) ; } they both have same Meaning.
or printf ("%d", &a) ;

(ii) *b = *(&a)
↓
Value at the address of a.

(iii) 'b++' होगा तो जिस type का Variable है (Datatype), उतना ही increment होगा।
like, b = 100
For int :- b++
b = 102.

* Addition of 2 No.s using pointers :-

```
#include <stdio.h>
# include <conio.h>
int main ()
{ clrscr();
  int *a, *b;
  int x, y;
  Printf ("Enter two No.s = ");
  scanf ("%d %d", &x, &y);
  a = &x;
  b = &y;
  Printf ("Sum = %d", *a + *b);
  getch();
}
```

FUNCTIONS ...

* func^n are used to organise the specific task

1. func^n Prototype → Also called func^n Declaration.
   ↓
   * Name of func^n
   * Return type
   (* Argument → Not Necessary)
2. func^n Calling.
3. func^n Defination.

Downloaded from TwoWaits

Scanned by CamScanner

```
#include <stdio.h>
 void inc (int *);
#include <conio.h>
void main ()
{
 int a = 10;
 clrscr();
 inc(&a); → // func^n calling
 printf ("Value of a is \t %d", a);
 getch()
}
void inc(int *d)
{
 *d = *d + 10;
}
```

→ func^n calling में हम func^n की Return type नहीं लिखते।

Q: swapping of two floating point No.s using call by reference.

Ans:

```
#include <stdio.h>
void swap (Float*, Float*);
int main ( )
{
 Float a, b;
 printf ("Enter 2 No.s to be swapped");
 scanf ("%f %f", &a, &b);
 swap ( &a, &b);
 printf ("%f %f", a, b);
 return 0;
}
Void swap (Float* x, Float* y)
{
 Float z;
 z = *x;
 *x = *y;
 *y = z;
 printf ("%f %f", *x, *y);
}
```

Downloaded from TwoWaits

Scanned by CamScanner

Q:- Swapping of 2 No.s using call by value.

Ans:-
```
#include <stdio.h>
void swap (int, int)
int main ()
{
  int a, b;
  Printf (" Enter 2 No.s to be swapped = ");
  scanf ("%d%d", &a, &b);
  swap (a, b);
  return 0;
}
void swap (int x, int y)
{
  int z;
  z = x;
  x = y;
  y = z;
  printf ("%d-%d", x, y);
}
```

* For compile time: In General way	In Array	In structure

int a;
a = 10;
 	
Memory occupied = 2 Byte
	int a[5] = {1,2,4,5,6}
	
	Memory occupied = 5 X 2 = 10 Bytes
		struct student
		{
		char name [15];
		int roll no;
		Float Marks;
		};
		struct student s = {'ABC', 10, 10.5}
		Memory occupied = 15X1 + 2 + 4 = 21 Bytes.

For run time: int a; printf ("Enter the value of a"); scanf ("%d", &a);	int i; int a[5]; printf ("Enter 5 No"); for (i=0; i<5; i++) scanf ("%d", &a[i]);	struct student s; printf ("Enter name, rollno, and Marks"); scanf ("%s%d%f", s.name, &s.rollno, &s.marks);

Downloaded from TwoWaits

Scanned by CamScanner

To create array in structure :- struct student s[10];
int i;
printf (" Enter name, roll no. and marks of 10 students");
for (i=0; i<10; i++)
scanf ("%s%d%f", s[i].name, &s[i].rollno, &s[i].marks);

PROGRAM :- (1) // Input and print the elements of the structure
(Run time)

```
#include <stdio.h>
#include <conio.h>
struct student
{
  char name [10];
  int roll no;
  float Marks;
};
struct student s;
void main ( )
{
  printf (" Enter the Name, roll no and marks");
  scanf ("%s%d%f", s.name, &s.rollno, &s.marks);
  printf ("\n Name is : \t%s", s.name);
  printf ("\n rollno is:\t%d", s.rollno);
  print ("\n marks is \t%f", s.marks);
  getch ( );
}
```

To Input By user.

For display or print on output screen.

(2) // Input and print the elements of the structure using Array.

```
#include <stdio.h>
#include <conio.h>
struct student
{ char name [10];
  int roll no;
  float Marks;
};
```

Downloaded from TwoWaits

Scanned by CamScanner

```
struct student s[10];
void main()
{
  int i;
 printf(" Enter Name, RollNo., %age");
  for(i=0; i<10; i++)
 scanf("%s%d%f", s[i].name, &s[i].rollno, &s[i].marks);
  printf("\n Name is : \t %s", s.name[i])
  printf("\n rollno is : \t %d", s.rollno[i]);
 printf("\n marks is : \t %f", s.marks[i]);
  getch();
}
```

(3) //Call by Value

```
typedef struct
{
 char name[10];
 int rollno.;
 Float Marks;
} student;
void display(student);
void main()
{ student s;
  clrscr();
 printf("Enter name, rollno. and Marks");
 scanf("%s%d%f", s.name, &s.rollno, &s.marks);
 display(s);
 getch();
}
void display(student s1)
{
 printf("%s %d %f", s1.name, s1.rollno, s1.marks);
}
```

Downloaded from TwoWaits

Scanned by CamScanner

(4) // call by reference. (we use pointer for address)

```
typedef struct
{
  char name [10];
  int roll no;
  float Marks;
} student;
void display (student *);
Void Main()
{
  student s;
  clrscr();
printf ("Enter name, roll no and Marks");
scanf ("%s%d%f", s.name, &s.roll No., & s.marks);
 display ( &s)
 getch();
}
void display (student *s1)
{ printf ("%s%d%f", s1->name, s1->roll no, s1->marks);
}
```

IMPLEMENTATION OF STACK...

stack can be implemented in two ways :-

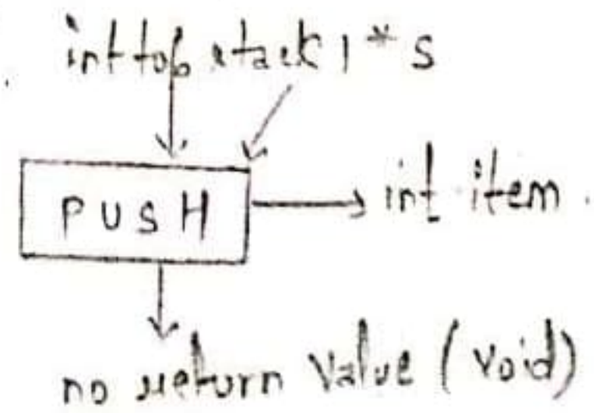
(i) static implementation → Array implementation.

(ii) Dynamic implementation → linked list

(i) func^n for PUSH operation :- # define maxsize 10.

```
typedef struct
{
  int stack [maxsize];
  int top;
} stack 1;
```

this all will be written in POP oper^n.



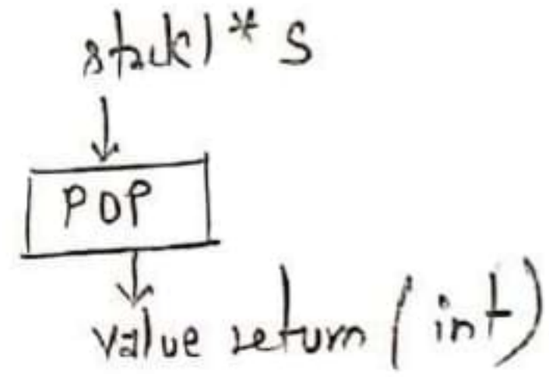
let 's' is a pointer variable of type stack 1 & item is a type of integer.

Downloaded from TwoWaits

Scanned by CamScanner

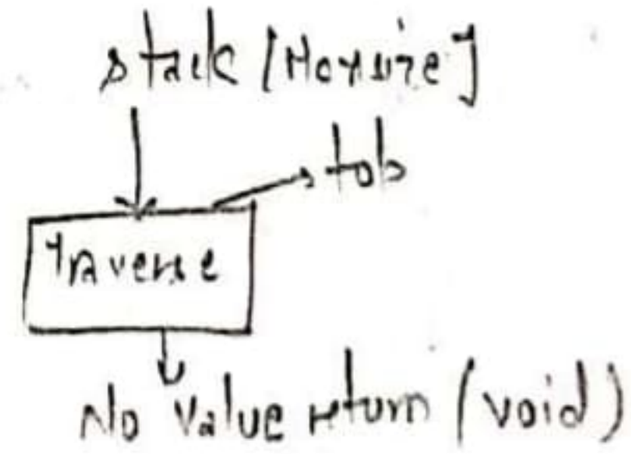
```
void Push (stack1 *s, int item)
{
  if (s->top == maxsize-1)
  {
    printf ("stack overflow");
      exit ( );
  }
  else
  {s -> top = s -> top+1;
    s -> stack [s -> top] = item;
  }
}
```

(ii) <u>function for POP operation :-</u> (- - - - -) same as written in push.




```
int pop (stack1 * s)
{ int item;
  if (s -> top == -1)
  {printf ("stack underflow");
   exit ( )}
  else
  {item = s -> stack [s -> top];
    s -> top = s -> top-1;
    return (item);
  }
}
```

(iii) <u>function for traversing operation :-</u>




```
# define maxsize 10
typedef struct
{
  int stack [Maxsize];
  int top;
} stack 1;
```

let 's' is a pointer variable of type stack 1 (y 'i' is a type of integer.

Downloaded from TwoWaits

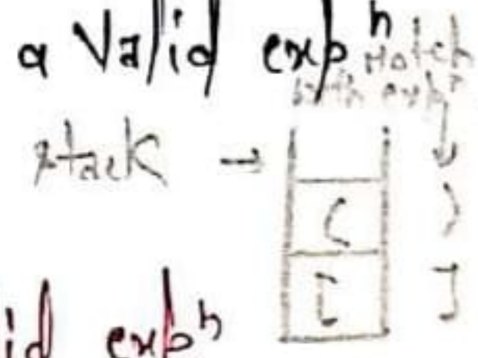
Scanned by CamScanner

```
void traverse (stack1 *s)
{
    if(s->top == -1)
    { print ("stack is underflow");
      exit();
    }
    else
    {
     for (i=s->top, i>=0; i--)
     {
      printf ("%d", s->stack[i]);
     }
    }
}
```

APPLICATIONS OF STACKS...

1. **Checking the Validity of an Arithmetic expression** :- In a Valid expⁿ

[ ] , { }, ( ) must occur in pair.
(parthesis) (curly braces) (small Brackets)



Eg. [(A+B)*C] → Valid expⁿ [(A+B * C] → Not valid expⁿ

(a) Whenever an opening parantheis/ Bracket is encountered, it is pushed onto the stack.

(b) Whenever a closing parantheis is Encounter, the stack is examined :-

* If the stack is empty & the closing parantheis does have the opening parantheis then the expⁿ is invalid.
* If the stack is Not empty we pop the stack and check whether the popped item corresponds to the closing parantheis
  → If the Match occurs, we continue otherwise expⁿ is invalid
* When the end of the expⁿ is reached the stack Must be Empty.
  If the stack is Empty, then the expⁿ is Valid.
* If one or more opening parantheis doesn't have corresponding closing parantheis, then the expⁿ is Not valid.

Eg. check whether it is valid or invalid expⁿ :-

[(A+B) - {C+D}] - [F+G] .

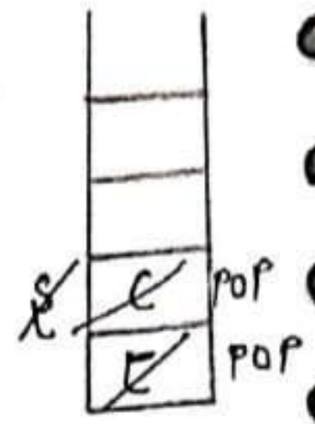
Downloaded from TwoWaits

Scanned by CamScanner

Ans :-

* stack में सिर्फ Brackets store होता है। और According to Expn. जब symbol scanned होता है, तो stack में से Bracket pop हो जाता है।
* last वाला Bracket ही Match करते है पहले, क्योंकि वही pop होता है stack से पहले।

S.No.	Symbol scanned	stack
01.	[	[
02.	(	[, (
03.	A	[, (
04.	+	[, (
05.	B	[, (
06.	)	[
07.	-	[
08.	{	[, {
09.	C	[, {
10.	+	[, {
11.	D	[, {
12.	}	[
13.	]	
14.	-	
15.	[	[
16.	F	[
17.	+	[
18.	G	[
19.	]	stack is empty



* Here, stack is empty at the end, so the Expn is Valid.

Q check whether the Expn is Valid or Not.

[(A+B*C])

Ans :-

S.No.	Symbol scanned	stack
01.	[	[
02.	(	[, (
03.	A	[, (
04.	+	[, (
05.	B	[, (
06.	*	[, (
07.	C	[, (
08.	]	STOP
09.	)	

ये Bracket 'C' Match नहीं हुआ, तो यहीं पर ये invalid हो गई है।

* Here, stack is Not Empty, so the expn is Not Valid.

→ इसके आगे नहीं करेंगे।

→ यहीं पर invalid हो गया

→ stack is Not empty.

Downloaded from TwoWaits

Scanned by CamScanner

2. Representation of Arithmetic expression :-

(a) INFIX NOTATION :- In Infix Notation operator is placed b/w the two operands. In the infix Notation, Brackets are Required to know the correct order of evaluation of infix expⁿ.

Eg. (i) A + B * C
(ii) (A + B) * C

operands.
A ⊕ B
operator

(b) PREFIX NOTATION :- It is also k/a POLISH NOTATION, it is Name after polish Mathematician Jan Lukasiewicz. This Notation refers in which the operator symbol is placed before it's operands.

→ The fundamental property of polish Notation is that the order in which the operations are to be performed, Completely determined by the positions of the operators and operands in the expⁿ.

→ So, Brackets are not Required while writing expⁿ in polish Notation.

(c) POSTFIX NOTATION :- It is also k/a REVERSE POLISH NOTATION, in which the operator symbol is placed after it's operands. In Reverse polish Notation, No Need to put brackets to know the correct order of Evaluation of expⁿ.

(* Infix Notation Mein Hum precendence rule follow Karte Hai, but prefix ya postfix Hein Nahi Karte.) → इसलिए Brackets use नहीं करते prefix and postfix में।

Algorithm for the conversion of Infix to postfix :-

Let, 'Q' be an infix expression and this algoᵐ gives postfix expⁿ 'P'.

step ① Add opening Bracket to the stack and closing bracket to the infix expⁿ, 'Q'.

step ② Scan all the symbols of expⁿ 'Q', left to Right and Repeat steps '3' to step '6', for each symbol present in 'Q', until stack is empty.

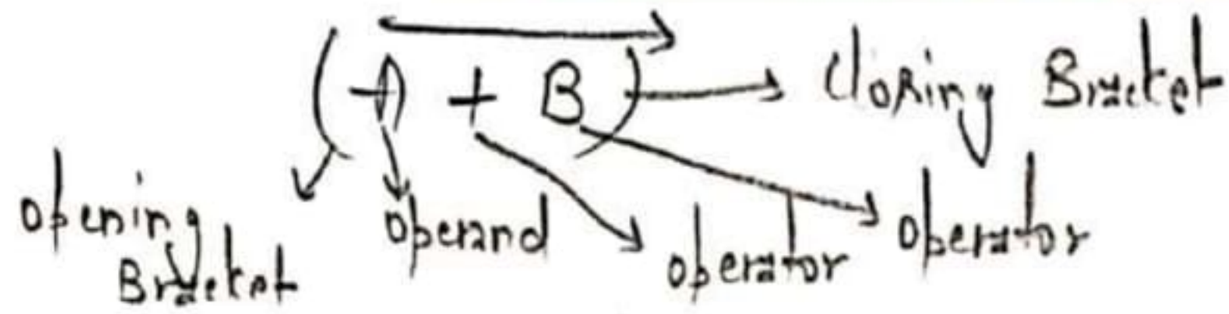
step ③ If an operand is encountered, then add it to 'P'.

step ④ If an left parantheis (opening Bracket) encountered then, push the stack.

step ⑤ If an operator 'X' is encountered, then :-
(i) Apply the pop operⁿ to the stack and add it to 'P', until top of the stack contain a operator which has precedence greater than or equal to operator 'X'.

Downloaded from TwoWaits

Scanned by CamScanner



" जिसकी precedence high है, उसके ऊपर lower precedence वाला नहीं आयेगा But lower के ऊपर high रख सकते हैं "

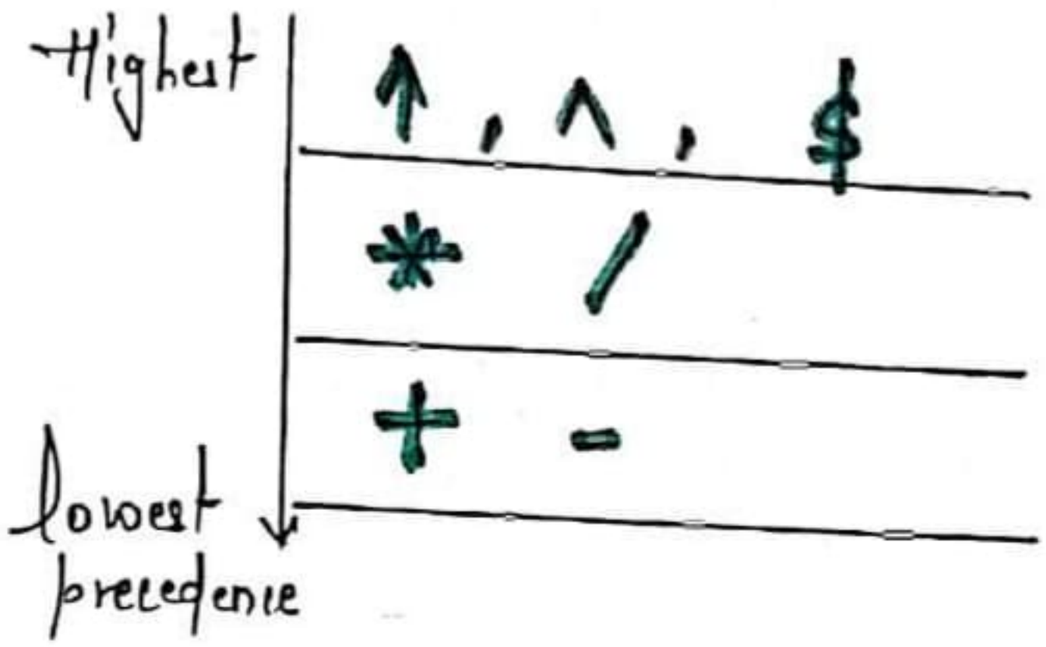
(ii) PUSH operator 'X' onto the stack.

<u>Step</u> (6) - If a Right Parantheis (closing Bracket) is encountered :-

(i) Apply (Remove) POP operation to the stack and add it to 'P' until top of the stack contains left parantheis.

(ii) (Remove) POP the left parantheis from the stack.

<u>Step</u> (7) - <u>STOP</u>.



* '(' की precedence '/' High है।

Q :- Convert infix into postfix :-

(i) ((A+B)/(D-E))

Ans :-

S. No.	Symbol scanned.	Stack	Postfix.
01.	(	(	–
2.	(	((	–
3.	A	((	A
4.	+	((+	– A
5.	B	((+	A B
6.	)	(	A B +
7.	/	(/	A B +
8.	(	(/(	A B +
9.	D	(/(	A B + D
10.	–	(/(–	A B + D
11.	E	(/(–	A B + D E
12.	)	(/	A B + D E –
13.	)	–	$\boxed{A B + D E - /}$ (Answer)

stack is finally

Downloaded from TwoWaits

Scanned by CamScanner

(ii) (A * B + C) :-

Ans :-

S.No.	Symbol scanned	Stack	Postfix
1.	(	(	
2.	A		A
3.	*	(*	A
4.	B	(*	AB
5.	+	(+	AB*
6.	C	(+	AB*C
7.	)		AB*C+ (Ans.)

(iii) ((A+B) * C - (D-E)) $ (F+G)) :-

Ans :-

S.No.	Symbol scanned	stack	postfix
1.	(	(	
2.	(	((	
3.	(	(((	
4.	A	(((	A
5.	+	(((+	A
6.	B	(((+	AB
7.	)	((	AB+
8.	*	((*	AB+
9.	C	((*	AB+C
10.	-	((-	AB+C*
11.	(	((-(	AB+C*
12.	D	((-(	AB+C* D
13.	-	((-(-	AB+C* D
14.	E	((-(-	AB+C* DE
15.	)	((-	AB+C* DE-
16.	)	(	AB+C* DE--
17.	$	($	AB+C*DE--
18.	(	($(	AB+C*DE--
19.	F	($(	AB+C*DE--F
20.	+	($(+	AB+C*DE--F
21.	G	($(+	AB+C*DE--FG
22.	)	($	AB+C*DE--FG+
23.	)	stack is Empty	AB+C*DE--FG+$

Downloaded from TwoWaits

Scanned by CamScanner

(iv) $(A+(B*C-(D/E\uparrow F)*G)*H)$:-

Ans :-

S. No.	Symbol scanned	Stack	Postfix.
1.	(	(	
2.	A	(	A
3.	+	(+	A
4.	(	(+(	A
5.	B	(+(	AB
6.	*	(+(*	AB
7.	C	(+(*	ABC
8.	-	(+(-	ABC*
9.	(	(+(-(	ABC*
10.	D	(+(-(	ABC*D
11.	/	(+(-(/	ABC*D
12.	E	(+(-(/	ABC*DE
13.	↑	(+(-(/↑	ABC*DE
14.	F	(+(-(/↑	ABC*DEF
15.	)	(+(-	ABC*DEF↑/
16.	*	(+(-*	ABC*DEF↑/
17.	G	(+(-*	ABC*DEF↑/G
18.	)	(+	ABC*DEF↑/G*-
19.	*	(+*	ABC*DEF↑/G*-
20.	H	(+*	ABC*DEF↑/G*-H
21.	)	stack is Empty.	ABC*DEF↑/G*-H*+

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

Ans :-

S.No.	Symbol scanned	Stack	Postfix
1.	(	(	
2.	A	(	A
3.	-	(-	A
4.	B	(-	AB
5.	/	(-/	
6.	C	(-/	ABC

Downloaded from TwoWaits

Scanned by CamScanner

7.	+	(+	ABC/-
8.	D	(+	ABC/-D
9.	*	(+*	ABC/-D
10.	E	(+*	ABC/-DE
11.	+	(+	ABC/-DE*+
12.	F	(+	ABC/-DE*+F
13.	)	stack is Empty	ABC/-DE*+F+

→ Less precedency है तो, पहले वाले को POP कर लो।

(vi) $(a+b*c-d\wedge e\wedge f)$:- abc*+dc∧f∧-

(vii) $((A-B+C)*D)(E+F))$:- AB-C+D*EF+

(viii) $A*(B+D)/E-F*(G+H/K)$:- ABD+*E/FGHK/+*-

Ans :- (vi)

S.No.	Symbol scanned	Stack	Postfix.
1.	(	(	
2.	a	(	a
3.	+	(+	a
4.	b	(+	ab
5.	*	(+*	ab
6.	c	(+*	abc
7.	-	(-	abc*+
8.	d	(-	abc*+d
9.	∧	(-∧	abc*+d
10.	e	(-∧	abc*+de
11.	∧	(-∧	abc*+de∧
12.	f	(-∧	abc*+de∧f
13.	)	Empty.	abc*+de∧f∧-

(vii)

S.No.	Symbol scanned	Stack	Postfix.
1.	(	(	
2.	(	((	
3.	A	((	A
4.	-	((-	A
5.	B	((-	AB
6.	+	((+	AB-
7.	C	((+	AB-C
8.	)	(	AB-C+
9.	*	(*	AB-C+
10.	D	(*	AB-C+D
11.	)		AB-C+D*
12.	(	(	AB-C+D*
13.	E	(	AB-C+D*E
14.	+	(+	AB-C+D*E
15.	F	(+	AB-C+D*EF

Downloaded from TwoWaits

Scanned by CamScanner

16.	)		AB - C + D * E F +
17.	)	)	AB - C + D * E F +

(viii) अगर starting से '(' आया तो भी valid है। → Here, stack is not Empty. So, it is invalid expⁿ.

S.No.	Symbol scanned	stack	Postfix
0.	(	(	
1.	A	(	A
2.	*	(*	A
3.	(	(*(	A
4.	B	(*(	AB
5.	+	(*(+	AB
6.	D	(*(+	ABD
7.	)	(*	ABD+
8.	/	(/	ABD+*
9.	E	(/	ABD+*E
10.	-	(-	ABD+*E/
11.	F	(-	ABD+*E/F
12.	*	(-*	ABD+*E/F
13.	(	(-*(	ABD+*E/F
14.	G	(-*(	ABD+*E/FG
15.	+	(-*(+	ABD+*E/FG
16.	H	(-*(+	ABD+*E/FGH
17.	/	(-*(+/	ABD+*E/FGH
18.	K	(-*(+/	ABD+*E/FGHK
19.	)	(-*	ABD+*E/FGHK/+
20.	)	stack is empty.	ABD+*E/FGHK/+*-

3. Evaluation of postfix Expression :- This algoⁿ will evaluate the value of postfix expression.

(i) Add a closing bracket to the end of the expⁿ. (opening की beginning की)

(ii) Scan the postfix expⁿ from left to Right, and repeat steps (3) & (4) until closing Bracket is scanned in the expⁿ.

(iii) If an operand is scanned then push into stack.

(iv) If an operator is scanned, then :- (a) pop the 2 elements of stack. In Expⁿ 'A' is the top element & 'B' is Next to top element.

(b) Evaluate B (*) A (B operator A). (any operator)

(c) Push the result of 'B' i.e. $B_1 = B * A$ into stack.

(v) get the value Equal to top Element of stack.

(vi) STOP.

Downloaded from TwoWaits

Scanned by CamScanner

Q1:- 5 6 2 + * 12 4 / - :-

Ans:-

S.No	symbol scanned	stack	
1.	5	5	
2.	6	5, 6	
3.	2	5, 6, 2	6+2 = 8
4.	+	5, 8	5*8 = 40
5.	*	40	
6.	12	40, 12	
7.	4	40, 12, 4	12/4 = 3
8.	/	40, 3	40-3 = 37
9.	-	37 (Ans.)	

Q2:- 6 2 3 + - 3 8 2 / + * 2 ^ 3 + :- 52

Ans:-

S.No.	symbol scanned	stack.	
1.	6	6	
2.	2	6, 2	
3.	3	6, 2, 3	
4.	+	6, 5	
5.	-	1	
6.	3	1, 3	
7.	8	1, 3, 8	
8.	2	1, 3, 8, 2	
9.	/	1, 3, 4	
10.	+	1, 7	
11.	*	7	
12.	2	7, 2	
13.	^	49	$7^2 = 49$
14.	3	49, 3	
15.	+	52 (Ans.)	

4. Conversion of infix Notation into prefix :- let 'Q' in an infix expⁿ by this algoᵐ gives the prefix expⁿ 'P'.

Step(i) Push closing Bracket to the stack and add opening Bracket to the left of infix expⁿ 'Q'.

Step(ii) Scan the 'Q' from Right to left and Repeat steps ③ to ⑥ until stack is Empty.

Downloaded from TwoWaits

Scanned by CamScanner

step (iii) If an operand is scanned add it to prefix expn 'P'.

step (iv) If Right parantheis (')') is scanned, then push it in stack.

step (v) If an operator 'x' is scanned :-

(i) Repeatedly pop the top element of the stack and add it to 'P', if top has operator of higher precedence.

(ii) Push operator 'x' to stack.

step (vi)(i) If left parantheis ('(') is scanned, then repeatedly pop the stack and add it to 'P', until top contains Right parantheis.

(ii) Remove the closing bracket from stack.

step (vii) STOP.

Q1:- (A+B) :- (Right to left scan)

Ans :-

Symbol scanned	stack	Prefix
	)	
B	)	B
+	)+	B
A	)+	AB
(	Empty	+AB

पहले Bracket को stack में डाल देंगे।

'A' पहले आता है, बाद में नहीं।

जैसे ही (+), AB के पहले आयेगा क्योंकि prefix expn में convert करना है।

Q2 :- (A+B-C) :-

Ans :-

Symbol scanned	stack	prefix
	)	
C	)	C
-	)-	C
B	)-	BC
+	)-+	BC
A	)-+	ABC
(	Empty	-+ABC

→ Equal precedence पर नहीं pop करेंगे, सिर्फ Higher पर करेंगे।

Q3 :- (A*B-C) :-

Ans :-

Symbol scanned	stack	prefix
	)	
C	)	C

Downloaded from TwoWaits

Scanned by CamScanner

-	)-	C
B	)-	BC
*	)-*	BC
A	)-*	ABC
(	Empty.	-*ABC

Q4 :- (A * B $ C) :-

Symbol scanned.	stack	prefix
	)	
C	)	C
$	)$	C
B	)$	BC
*	)*	$BC
A	)*	A$BC
(	Empty.	*A$BC

Q5 :- (A - B / (C * D $ E)) :- -A/B*C$DE

Q6 :- ((A+B) * (C-D)) :- *+AB-CD

Q7 :- (A $ B * C - D + E / F / (G+H)) :- +-*$ABCD//EF+GH

Ans (5)

Symbol scanned	stack	prefix.
	)	
)	))	
E	))	E
$	))$	E
D	))$	DE
*	))*	$DE
C	))*	C$DE
(	)	*C$DE
/	)/	*C$DE
B	)/	B*C$DE
-	)-	/B*C$DE
A	)-	A/B*C$DE
)	Empty	-A/B*C$DE

(6)

symbol scanned	stack	prefix
	)	
)	))	
D	))	D

Downloaded from TwoWaits

Scanned by CamScanner

−	)) −	D
C	)) −	CD
(	)	−CD
*	)*	−CD
)	)*)	−CD
B	)*)	B−CD
+	)*)+	B−CD
A	)*)+	AB−CD
(	)*	+AB−CD
(	Empty	$\boxed{*+AB-CD}$

(7)

Symbol Scanned	Stack	Prefix
)	)	
)	))	H
H	))	H
+	))+	GH
G	))+	+GH
(	)	+GH
/	)/	F+GH
F	)/	F+GH
/	)//	EF+GH
E	)//	//EF+GH
+	)+	D//EF+GH
D	)+	D//EF+GH
−	)+−	CD//EF+GH
C	)+−	CD//EF+GH
*	)+−*	BCD//EF+GH
B	)+−*	BCD//EF+GH
$	)+−*$	ABCD//EF+GH
A	)+−*$	ABCD//EF+GH
(	Empty	$\boxed{+-*\$ABCD//EF+GH}$

Q8:- $((A+B)*C-(D-E))\$(F+G))$:- $\$-*+ABC-DE+FG$

Symbol Scanned	Stack	Prefix
)	)	
)	))	
G	))	G
+	))+	G
F	))+	FG

Downloaded from TwoWaits

Scanned by CamScanner

(	)	+FG
$	)$	+FG
)	)$)	+FG
)	)$))	+FG
E	)$))	E+FG
-	)$))-	E+FG
D	)$))-	DE+FG
(	)$)	-DE+FG
-	)$)-	-DE+FG
C	)$)-	C-DE+FG
*	)$)-*	C-DE+FG
)	)$)-*)	C-DE+FG
B	)$)-*)	BC-DE+FG
+	)$)-*)+	BC-DE+FG
A	)$)-*)+	ABC-DE+FG
(	)$)-*	+ABC-DE+FG
(	)$	-*+ABC-DE+FG
(	Empty.	$-*+ABC-DE+FG

RECURSION...

When a func^n Calls itself.

A func^n is said to be Recursively defined if the func^n defination Refers to itself, in order for the defin^n not to be circular, it Must have two properties :-

(i) there Must be certain Argument^n called base Value for which the func^n does n't refers to itself.

(ii) Each time the func^n does refers to itself, the argument of the func^n Must be closer to a base Value.

A Recursive func^n with these 2 properties is also said to be well-defined.

Principle of Recursion :- (i) Original problem can be decompose into small size of same type of problem.

(ii) Recursive Call Must Reduce the size of problem.

(iii) Base Condition Must be present.

(iv) Base Condition Must be Reachable.

```
void display ()
{
  ....;
  display ();
}
```

→ ये number है। अगला value को बार-बार call कर रहा है।

* Recursion is done in stack too by push & pop oper^n.

Downloaded from TwoWaits

Scanned by CamScanner

PROGRAM :- // factorial of a number using Recursion.

```
#include <stdio.h>
#include <conio.h>
long int fact (int);
void main()
{
  long int f;
  int n;
  clrscr();
printf (" Enter no. =");
 scanf (" %d", &n);
f = fact (n);
 printf (" factorial is \t %ld ", f);
 getch();
}
long int fact (int num)
{

 if (num < 0)
{printf (" Negative no. Not allow");
 getch() ; exit ();
}
elseif (num == 0)
return (1);
else
return ( num * fact(num-1)) ;
}
```

→ Argument में वो variable होते हैं, जो user से लेते हैं।

$n \to (-)ve$	(Factorial not possible)
$n \to 0$	1
$n > 0$	$n*(n-1)!$

* यहाँ Base condition '0' तक पहुँचना है।

IN STACK :-

		Result
fact(0)	1	↓
fact(1)	1* fact(0)	→ 1*1 = 1
fact(2)	2* fact(1)	→ 2*1 = 2
fact(3)	3* fact(2)	→ 3*2 = 6

for n = 3

→ जब final value stack में मिलती i.e. '6' तो ही POP होगी।

exit () ;→ Garbage value print नहीं होगी।

ALGORITHM FOR FACTORIAL :- (N, fact)

This Algo^m calculate factorial of N i.e. N! and returns the value in the variable 'fact'

(i) IF N == 0, then
set, fact = 1 and return.

(ii) Call factorial (fact, N-1)
set, fact = fact * N

(iii) return

Downloaded from TwoWaits

Scanned by CamScanner

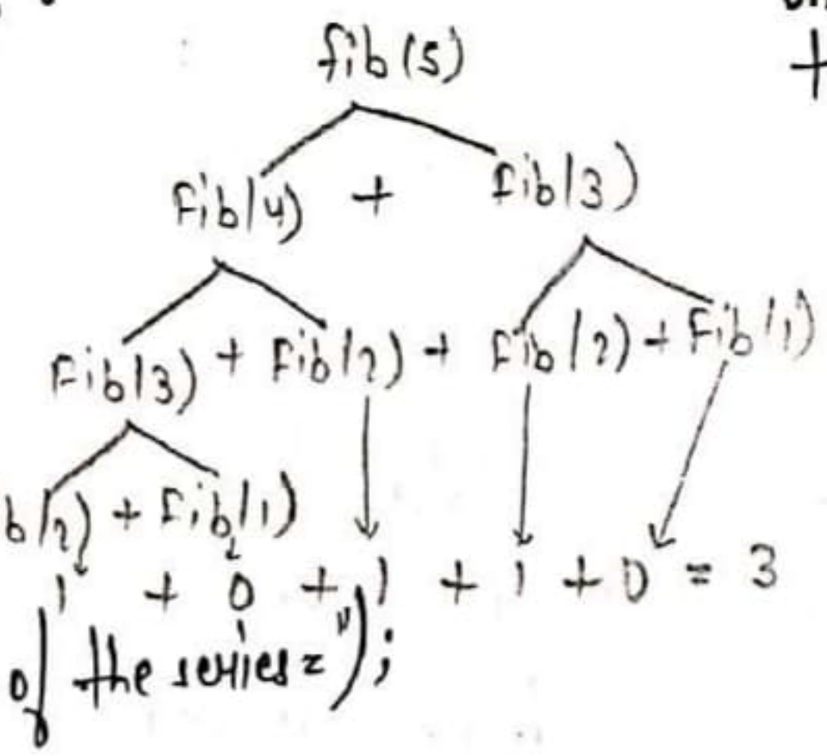
PROGRAM FOR MULTIPLICATION OF TWO NO.s USING RECURSION :-

```
# include <stdio.h>
# include <conio.h>
long int Multi (int, int);
void main ()
{ long int result;
  int a, b;
  clrscr ();
  printf (" Enter 2 No.s =");
  scanf ("%d %d", &a, &b);
  result = multi (a,b);
  printf (" Result is \t %ld", result);
  getch ();
}
long int multi (int a, int b)
{
  if (b == 0)
  return (0);
else
  return (a + multi (a, b-1)); ⟶ (3 + multi (3, 3))
}.
```

Concept :- By Add^n
if a = 3
b = 4
so, a+a+a+a = 12

PROGRAM FOR FIBONACCI SERIES :-

0 1 1 2 3 ⇒ only print one element, when the 'n' term is inputed.




```
# include <stdio.h>
# include <conio.h>
long int fib (int)
{ long int result; int n;
  clrscr ();
  printf (" Enter the no. of terms of the series =");
  scanf (" %d", &n);
  result = fib (n);
  printf (" Result is %ld", result);
  getch ();
}
long int fib (int n)
{
  if (n == 1)
  return (0);
  else if (n == 2)
  return (1);
  else
  return (fib (n-1) + fib (n-2));
}
```

Downloaded from TwoWaits

Scanned by CamScanner

ALGORITHM FOR FIBONACCI :- (fib, N)

Here, the algo^m Calculate Nth term and return the value in the fib.

(i) If N==1, then
set fib==0 and return.

(ii) else if N==2, then
set fib==1. and

(iii) else call fibonacci (fibA, (n-2))

(iv) Call fibonacci (fibB, (n-1))

(v) Then, set fib = fibA + fibB. and return.

OR fib = fib($\underbrace{n-2}_{A}$) + fib($\underbrace{n-1}_{B}$)

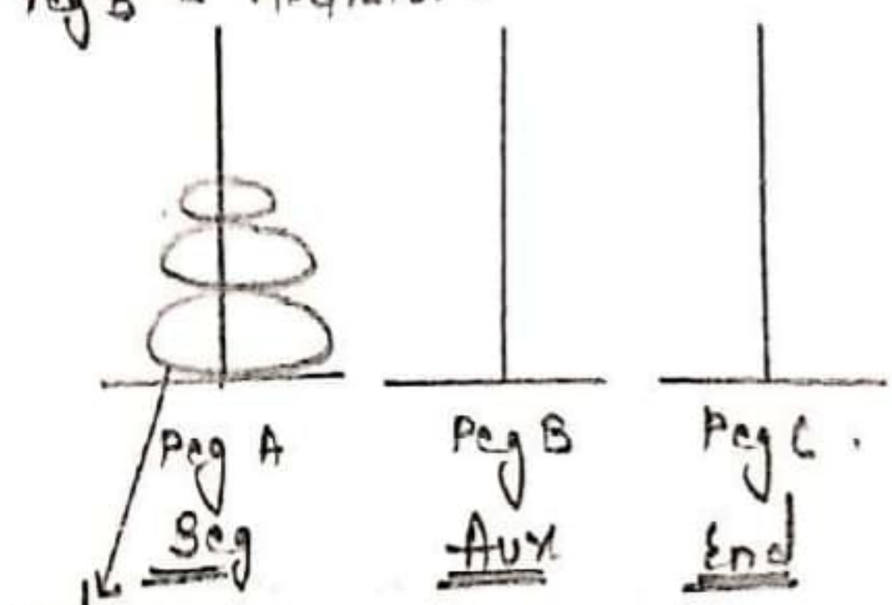
(vi) STOP.

0	→ fib(1)
1	→ fib(2)
fib(2) + fib(1)	→ fib(3)
fib(3) + fib(2)	→ fib(4)
fib(4) + fib(3)	→ fib(5)

TOWER OF HANOI

Value of n.	Pass	steps
n=1	1	A → C
n=2	3	A → B A → C B → C
n=3	7	A → C A → B C → B A → C B → A B → C A → C

For n=3 (n is No. of Data).
Peg 'B' is Mediator.



बर्तन के ऊपर भी रख सकते है। कोई के ऊपर नहीं।

Generalised formula to calculate steps or Pass = $\boxed{2^n - 1}$

Suppose, 3 Pegs Labelled A, B and C. on Peg A finite no. of 'n' Disk are placed in ring Order

Object of the Game / problem :- the object of the Game is to Move a disk from Peg 'A' to Peg 'C' using Peg B as an Auxillary Peg (or secondary Peg)

Rules of the Game :- (i) Only one disk is moved at a time from the top and can be placed to any other peg.

(ii) At No Time larger disk can be placed on smaller disk.

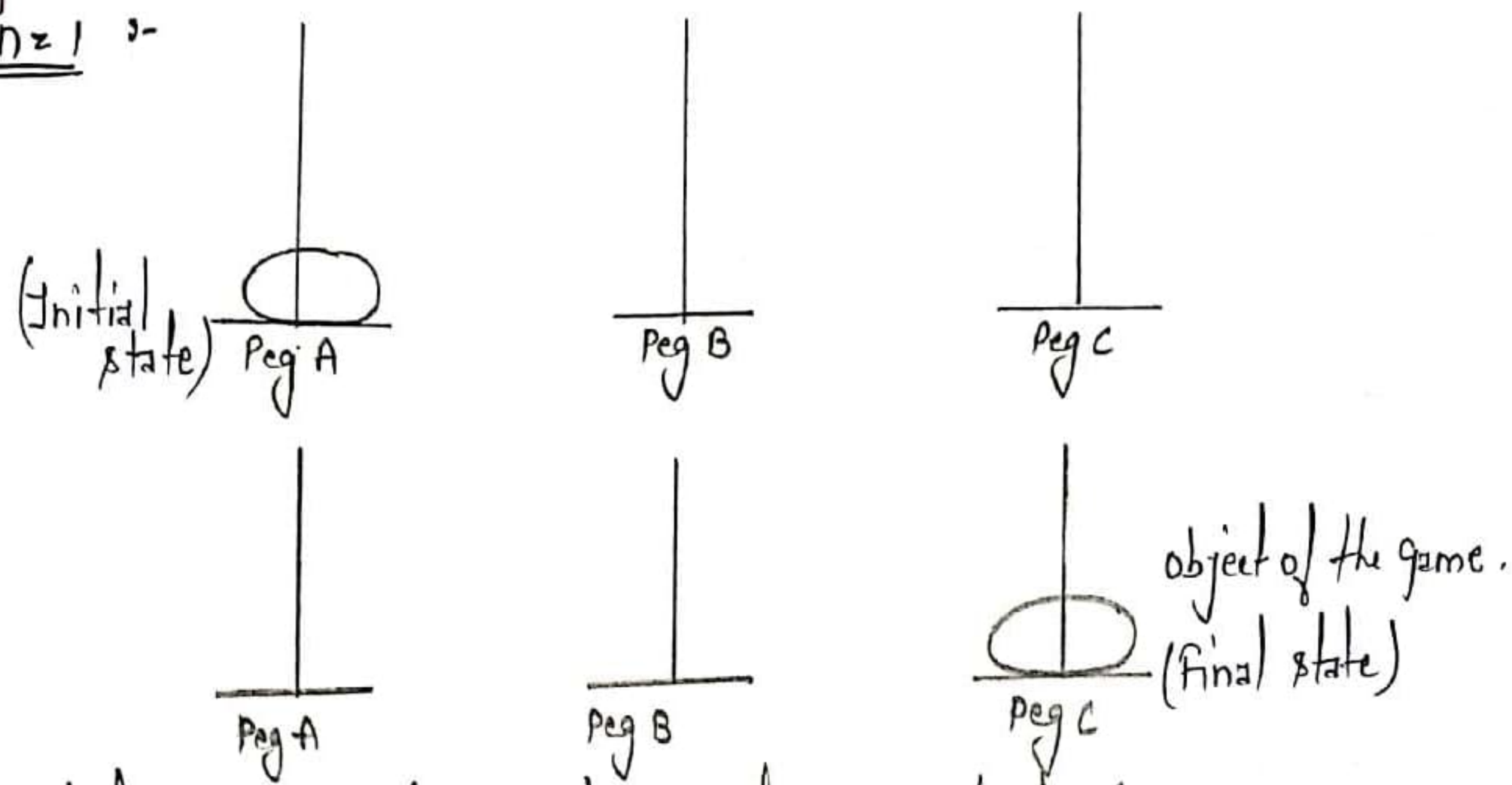
(iii) One and only one Extra peg could be used for intermediate storage of Disk.

Downloaded from TwoWaits

Scanned by CamScanner

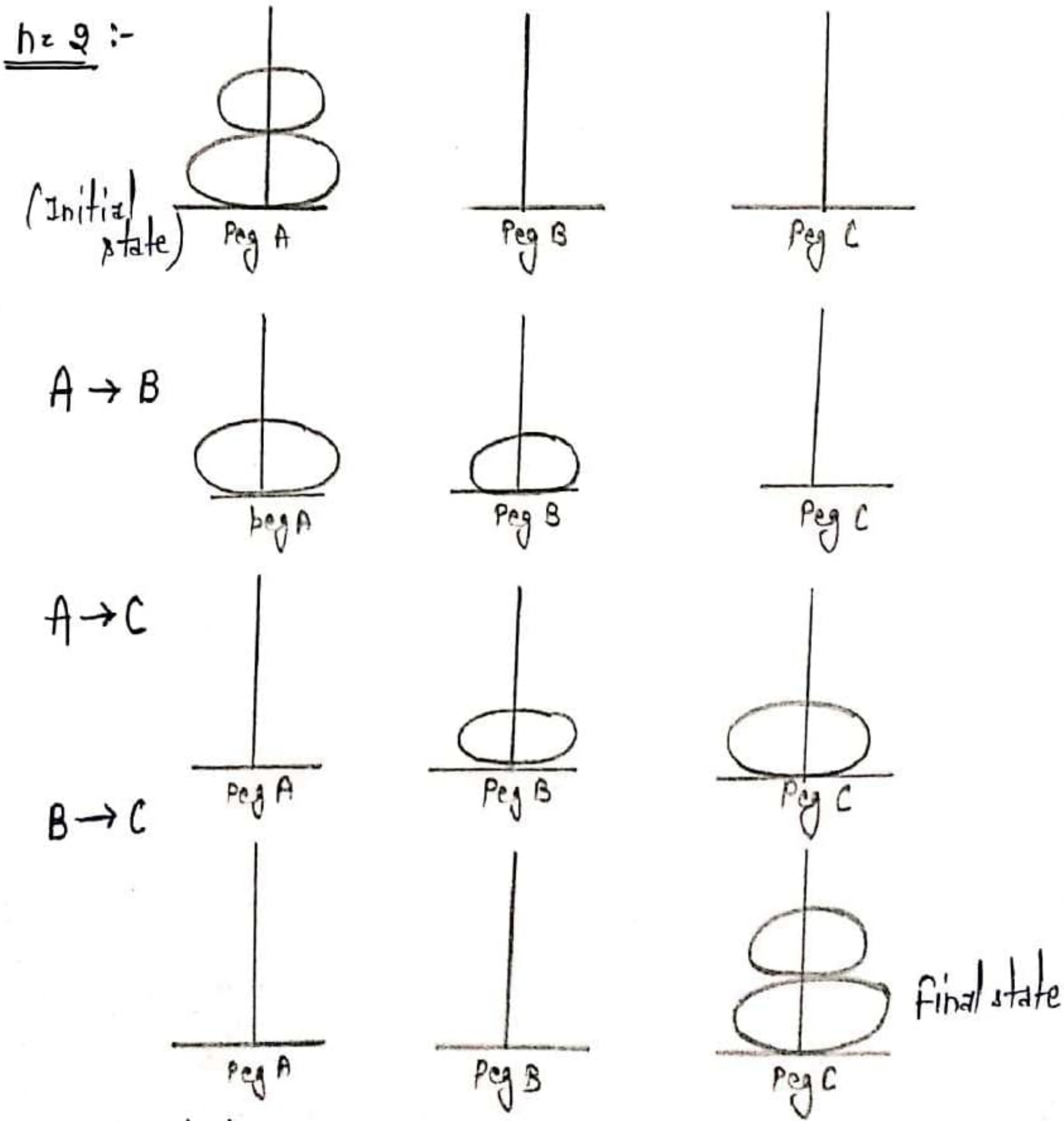
Diagrams :-

(i) $n = 1$:-



No. of passes = 1 $(A \rightarrow C)$ → Move from Peg A to Peg C.

(ii) $n = 2$:-

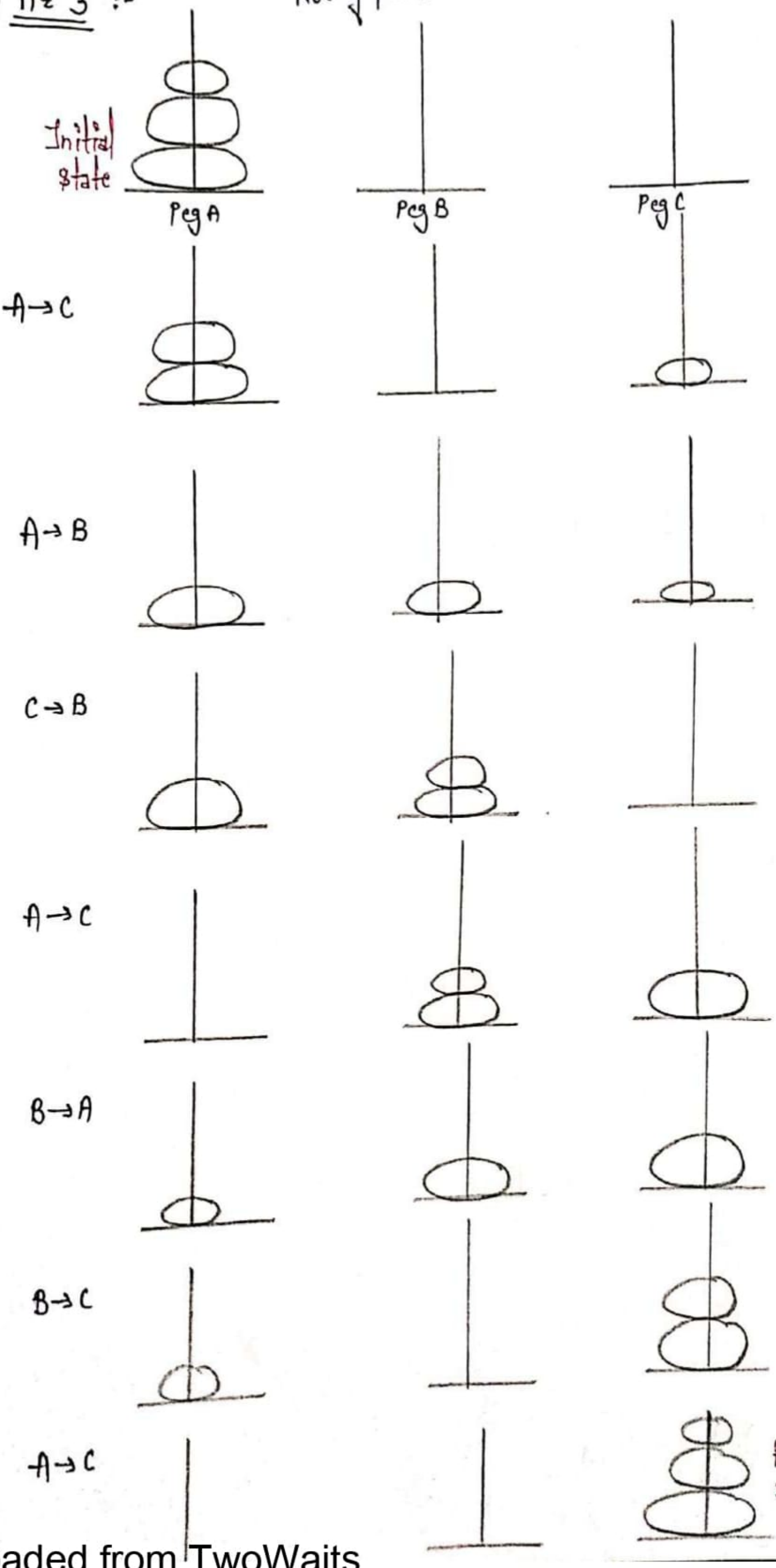


No. of passes :- 3

Downloaded from TwoWaits

Scanned by CamScanner

(iii) $\underline{\underline{n = 3}}$:- No. of passes :- 7



Downloaded from TwoWaits

Scanned by CamScanner

Recursive Solution for Tower of Hanoi :-

Here, Base condition is $n=1$

3 steps को एक से लिख लिया

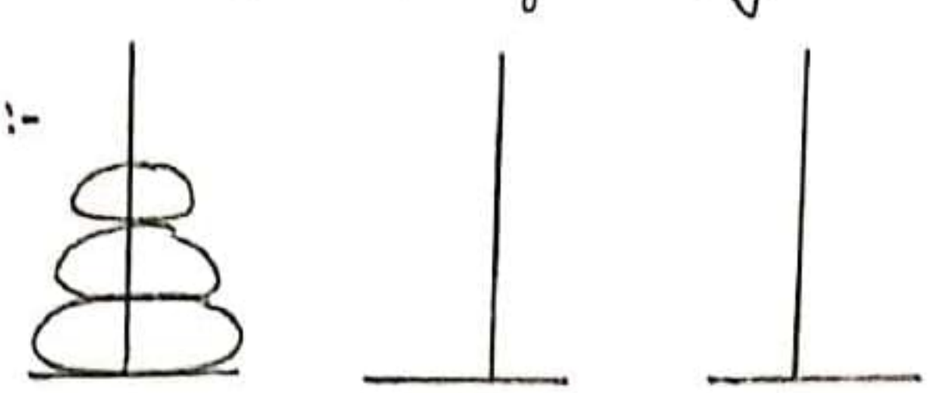
① $(n-1, A, C, B)$ → C का use करके A में से B में transfer किया है Disk.

② $A \rightarrow C$ (फिर 'A' से 'C' में डाल दिया Disk)

③ $(n-1, B, A, C)$ → A का use करके B में से 'C' में finally disk transfer कर दिया।

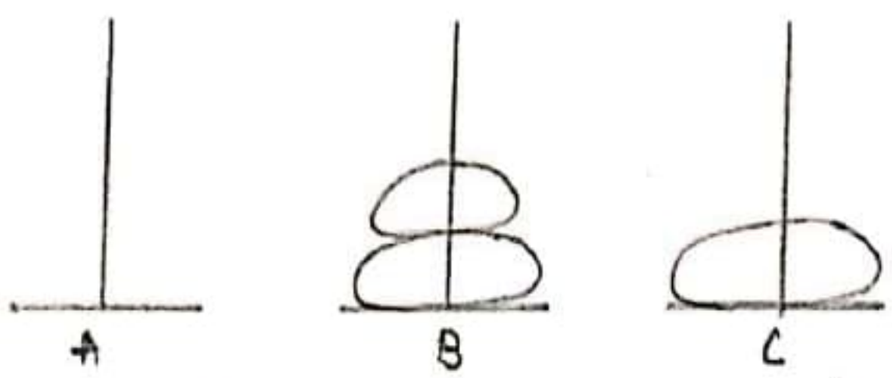
step ① Move the top (n-1) disk from peg 'A' to peg 'B'.
step ② Move the top disk from peg 'A' to peg 'C'.
step ③ Move the (n-1) disk from peg 'B' to peg 'C'.

① Initial :-

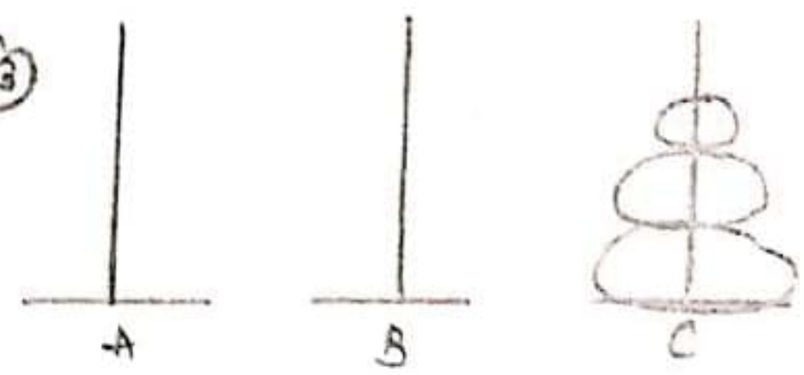


② step ①

③ step ②



④ step ③



Algorithm for Tower of Hanoi problem :-

This algo^m will move 'n' disk from tower 'Beg' (Beg stands for Begining) to tower 'End' (Ending) Using tower 'Aux' as a Auxiliary tower.

The prototype of the func^n :- It has 4 parameters :-

(i) No. of Disks.
(ii) Tower from which Disk is to be move.
(iii) Tower which is used as a Auxiliary tower.
(iv) Tower to which Disk will be move.

Tower (N, Beg, Aux, End) :-

Here, This procedure gives a Recursive sol^n to the towers of Hanoi problem for N-Disks.

step ① :- If N=1, then
write Beg → End
[End of if structure].

step ② :- [Move N-1 disks from peg 'Beg' to peg 'Aux']
Call tower (N-1, Beg, End, Aux)

step ③ :- write Beg → End

Downloaded from TwoWaits

Scanned by CamScanner

Step (4) :- [Move 'N-1' disk from peg 'Aux' to peg 'End'.]

Call Tower (N-1, Aux, Beg, End)

General Notation...

Tower (N, Beg, Aux, End)

- Call tower (n-1, beg, end, Aux)
- Beg → end
- Call Tower (n-1, Aux, Beg, end)

RECURSION TREE...

for N = 3 :-

Tower (3, A, B, C) (n, Beg, Aux, End)

- Tower (2, A, C, B) (Now, n, Beg, Aux, End)
  - Tower (1, A, B, C) = A → C
  - A → B
  - Tower (1, C, A, B) = C → B
- A → C
- Tower (2, B, A, C) (Now, n, Beg, Aux, End)
  - Tower (1, B, C, A) = B → A
  - B → C
  - Tower (1, A, B, C) = A → C

for N = 6 :- No. of moves :- $2^6 - 1 = 63$

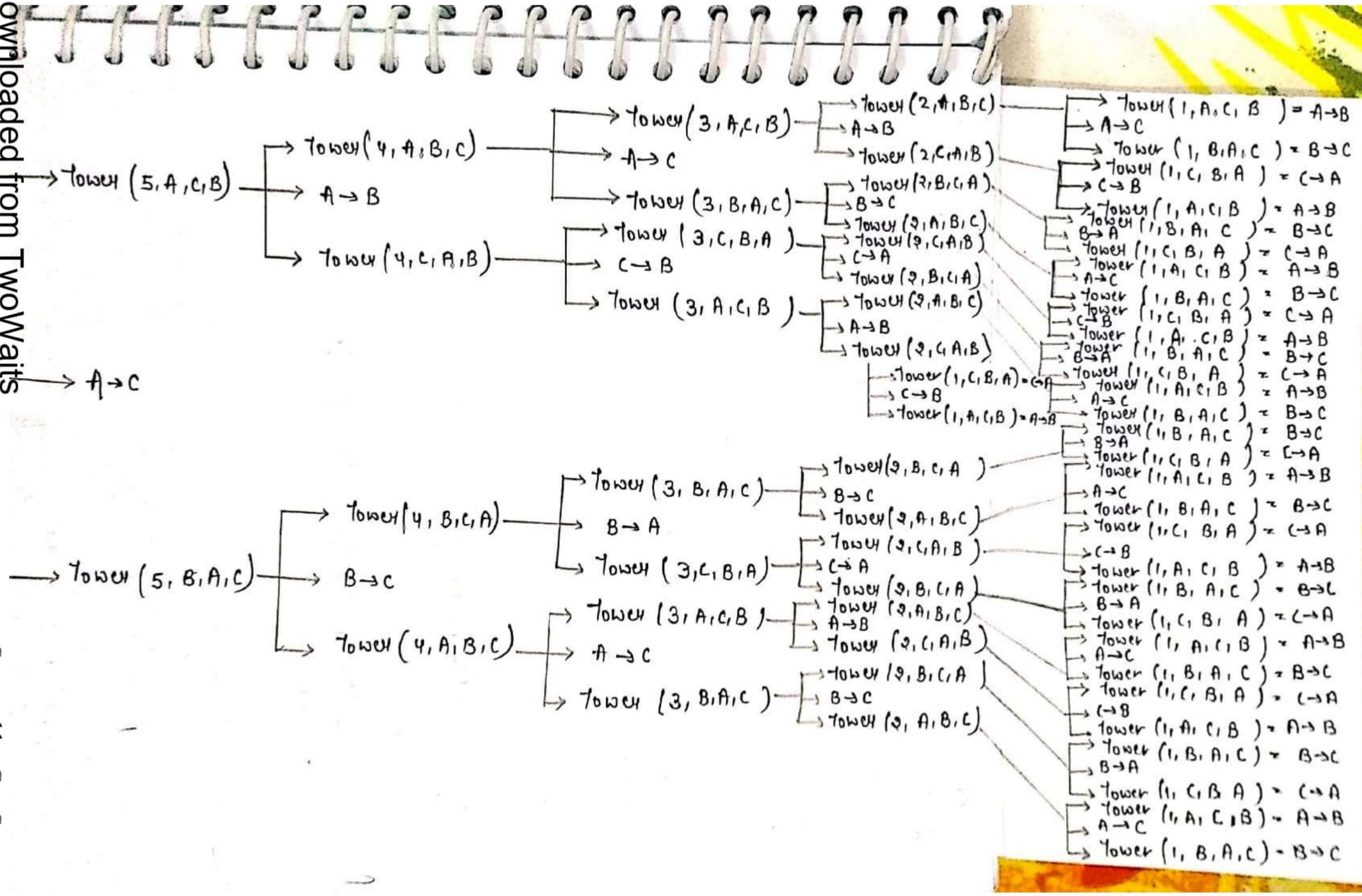
Steps :-

1. A → B	16. A → B	31. A → B	46. B → A
2. A → C	17. C → A	32. A → C	47. C → A
3. B → C	18. C → B	33. B → C	48. B → C
4. A → B	19. A → B	34. B → A	49. A → B
5. C → A	20. C → A	35. C → A	50. A → C
6. C → B	21. B → C	36. B → C	51. B → C
7. A → B	22. B → A	37. A → B	52. A → B
8. A → C	23. C → A	38. A → C	53. C → A
9. B → C	24. C → B	39. B → C	54. C → B
10. B → A	25. A → B	40. B → A	55. A → B
11. C → A	26. A → C	41. C → A	56. A → C
12. B → C	27. B → C	42. C → B	57. B → C
13. A → B	28. A → B	43. A → B	58. B → A
14. A → C	29. C → A	44. C → A	59. C → A
15. B → C	30. C → B	45. B → C	60. B → C
			61. A → B
			62. A → C
			63. B → C

Downloaded from TwoWaits

Scanned by CamScanner

Downloaded from TwoWaits



Scanned by CamScanner

PROGRAM FOR TOWER OF HANOI :- (USING RECURSION)

```
#include <stdio.h>
# include < conio.h>
  void tower ( int, char, char, char);
  void main ( )
  {
    int n;
    clrscr ( );
  printf ("Enter no. of disks ");
   scanf ("%d", &n);
  Tower ( n, 'A', 'B', 'C');
   getch ( );
  }
   void tower (int n, char beg, char aux, char end)
   {
     if (n==1)
      {
       printf (" move peg %c to %c pegn", beg, end);
       return ;
      }
     else
        tower (n-1, beg, end, aux);
        printf (" move peg %c to peg %c", beg, end);
        tower (n-1, aux, beg, end);
   }
```

* REMOVAL OF RECURSION :- Every Recursive problem can be implemented as Iterative (Looping) problem but all the iterative problem cannot be implemented as a Recursive problem.
Only those problems which will satisfy the 4-principles of Recursion can be implemented as a Recursion.

Iterative solution for Calculation of factorial :-

```
# include <stdio.h>
# include < conio.h> .
```

Downloaded from TwoWaits

Scanned by CamScanner

```
void fact(int); or int fact (int);
void main ()
{
 int n;
 clrscr();
 printf ("Enter Value to find factorial\n");
 scanf ("%d", &n);
 fact (n); ——— or printf ("factoral = %d", f) (if we use int)
 getch();
}
 void fact(int n) or int fact (int n) (if we use int)
 {
    int f=1, i;
    for (i=1; i≤n; i++)
    f = f*i;
   Printf ("factorial is %d ", F);   OR return (F); (if we use int)
 }
```

Iterative solution for calculation of fibonacci series :-

```
#include <stdio.h>
# include <conio.h>
 void fib (int);
 void main ( )
{
   int n;
   clrscr();
   printf (" Enter the term \n");
   scanf ("%d", &n);
      fib(n);

   getch();
}
void fib (int n)
{
   int f1=0, f2=1, f3, i;
   switch (n)
```

Downloaded from TwoWaits

Scanned by CamScanner

```
{
  case 0:
  printf (" not Valid ");
   getch ();
   exit ();
  case 1:
  printf (" 0 ");
  break;
  case 2:
  printf (" 1 ");
  break;
  default:
```

```
{ for (i=3; i<=n; i++)
  {
    f3 = f1 + f2;
    f1 = f2;
    f2 = f3;
  }
  printf (" the value at given
              term is = %d ", f3);
 }
}
}
```

TAIL RECURSION...

- A Recursive func^n is tail recursive, when recursive call is the last statement executed by the func^n. It Means there is Nothing to do after the func^n Returns Except Return it's Value.

In tail Recursion we perform calculations first, then we execute the Recursive Call and passing the Results of current steps to the Next Recursive step. It helps in improving the time and space efficiency of Recursive func^n. Whereas, in traditional Recursion perform Recursive Calls first and then take the Return Value of the Recursive call and calculate the Result. So, we have to wait till we Return every recursive call for result.

Advantages of Tail Recursion:-

(i) optimising the task of compiler for executing the Recursive func^n.
(ii) Improve the space and time efficiency of a recursive code.
(iii) Since, the Recursive call is the last stmnt. So, saving the current func^n's stack frame is of no use.

Eg. of Tail Recursion:-

```
void fun (int);
# include <stdio.h>
# include <conio.h>
void main ()
```

Downloaded from TwoWaits

Scanned by CamScanner

```
{
  int n;
  clrscr();
  printf(" enter n");
  scanf("%d", &n);
  fun(n);
  getch();
}
```

```
void fun(int n)
{
  if (n<0)
  return;
  printf("%d", n);
  fun(n--);
}
```

Difference b/w Recursion and Iteration :-

RECURSION	ITERATION
(i) A program is called Recursive, when a func^n calls itself Directly or indirectly.	(i) A program is called Iterative, when there is a repetition / looping.
(ii) Recursion is a process applied to a func^n.	(ii) Iteration is applied to a set of instruc^ns which we want to get repeatedly executed.
(iii) In Recursive func^n only Base Cond^n is specified.	(iii) Iteration includes initialization, condition, execution of stmnt within loop & update the control variable.
(iv) A conditional stmnt is included in the body of the func^n to force the func^n to return without Recursion call being executed.	(iv) The Iterative stmnt is repeatedly executed until the certain condition is reached.
(v) If the func^n doesn't goes to some Condition (Base condition), it leads to infinite Recursion and it can crash the system	(v) If the condition in the Iteration Never becomes false, it leads to infinite loop/ Iteration and it uses CPU cycles Repeatedly.
(vi) The stack is used to store the set of New local Variables & parameters Each time, when the func^n is Called.	(vi) Doesn't uses stack.
(vii) Recursion Reduces the size of Code	(vii) Iteration makes the code longer.
(viii) It is slow in Execution & finding the time complexity is more difficult than Iteration	(viii) Fast in Execution and finding the time complexity is less Difficult than Recursion.

Downloaded from TwoWalts

Scanned by CamScanner

(viii) Recursion forces the overhead of repeated funcⁿ calls.

(ix) No overhead of repeated funcⁿ calls.

Queue...

Queue is an ordered collecⁿ of items, in which items may be deleted at one end called the front & into which item may be inserted at the other end called the rear.

Every Queue has two ends,

(i) first end is called front end, from which data will be deleted.

(ii) Second end is called Rear end, from which data will be inserted.

Queue, works on the principle of FIFO (First in, first out).

Since, the first element in the Queue, will be the 1st element out of the Queue. In other words, the order in which elements Enter a Queue will be the order, in which they leave.

Eg (i) for insertion :-

queue [Maxsize]

0	1	2	3	4

Maxsize :- 5

Front = -1 (for Queue Empty)

Rear = -1 (")

(ii) Insert 20 ...

Queue

0	1	2	3	4
20				

front & Rear.

if (front == -1)

so, front = front + 1 (for insertion)

or set, front = 0

rear = rear + 1

or set, rear = 0

queue [rear] = 20

(iii) Insert 40 ...

Queue

0	1	2	3	4
20	40			

Front Rear

front = 0

rear = 0 + 1 = 1

Ab, (rear = rear + 1)

queue [rear] = 40.

* अगर front की Value '-1' नहीं है, तो हम front को update नहीं करेंगे। rear में '+1' करेंगे।

Downloaded from TwoWaits

Scanned by CamScanner

(iv) Insert 50, 60, 70 ...

Queue

0	1	2	3	4
20	40	50	60	70

front Rear.

front = 0 .
Rear = 4 .

(v) Insert 90 ...

Queue

0	1	2	3	4
20	40	50	60	70

Front Rear

Queue, is overflow.
We can't insert '90'.

Rear = 4 .
Maxsize = 5 .

If Rear == maxsize-1
Print("Queue overflow");
and exit.

OPERATIONS ON QUEUE ...

(i) Insertion :- Insertⁿ on the Queue takes place from the rear end. Before inserting any element to a queue on overflow condⁿ must be check and the overflow condⁿ is, Rear == maxsize-1 . When insert operⁿ is performed on Queue, rear is increment by 1 & then insertion takes place.

ALGORITHM FOR INSERTION ...

Let 'Queue[maxsize]' is an Array.
front and Rear are two integer variables, which is initialised by -1, i.e. front = -1 and Rear = -1 .

step ① :- [Check "Queue" for overflow Condition :-]

```
if (rear == maxsize-1)
then print ("Queue is overflow")
and return .
```

step ② :- [Read Data or item]

step ③ :- [Update the Rear and front by 1]

If front == -1, then set front = front+1 .
and Rear = Rear+1 .
or set front = 0
set Rear = 0

Downloaded from TwoWaits

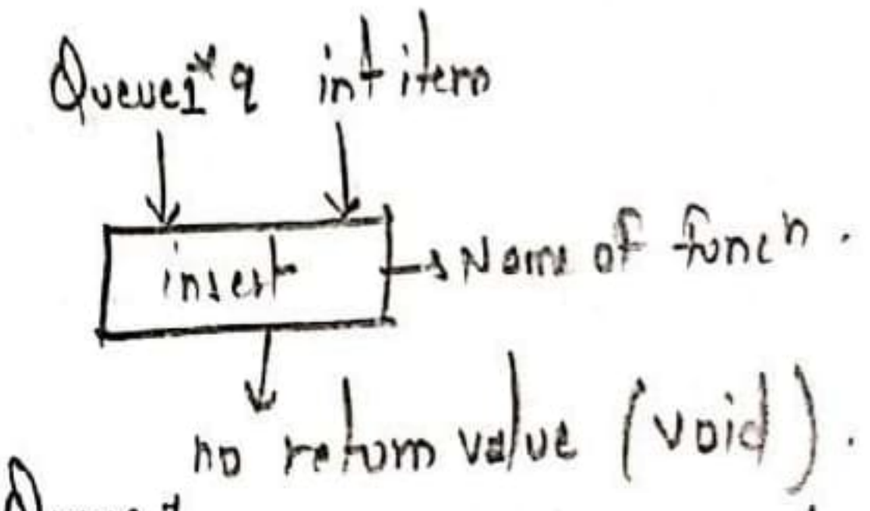
Scanned by CamScanner

step (4) else
[update the Rear]
set rear = rear + 1
step (5) insert the element / Data / item.
queue [rear] = item
step (6) STOP.

IMPLEMENTATION OF QUEUE :- Queue can be implemented in 2 ways :-
(i) static and → Array
(ii) Dynamic Implementation → link list.

following structure can be used for implementing Queue :- (ARRAY IMPLEMENTATION).

```
# define maxsize 5
typedef struct
{
    int queue [maxsize];
    int front;
    int rear;
} queue 1;
```



let 'q' is a pointer variable of type Queue 1
q item is a variable of type integer and func^n Doesn't Return a value.

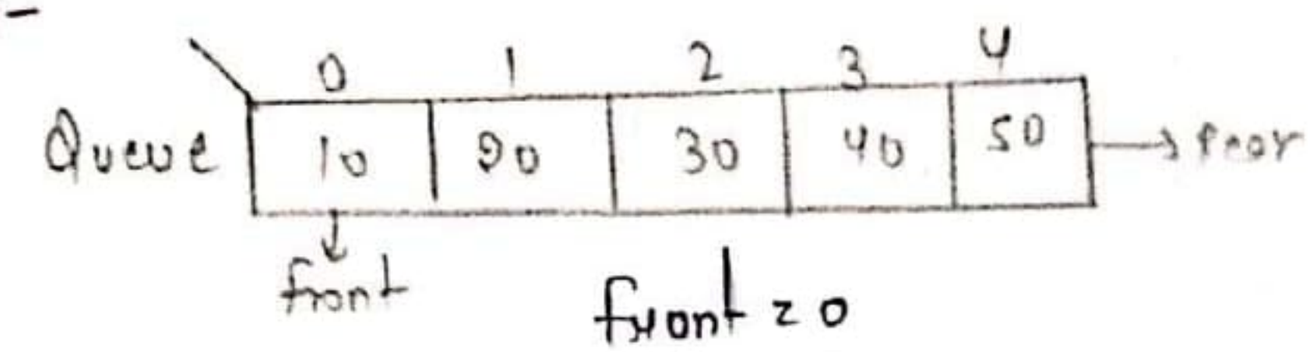
```
void insert (queue 1 * q, int item) → By Diagram.
{
    if (q → rear == maxsize - 1)
    {
        printf (" queue overflow");
        return;
    }
    else if (q → front == -1)
    {
        q → front = q → front + 1;
        q → rear = q → rear + 1;
    }
    else
```

Downloaded from TwoWaits

Scanned by CamScanner

q → rear = q → rear + 1 ;
q → queue [q → rear] = item;

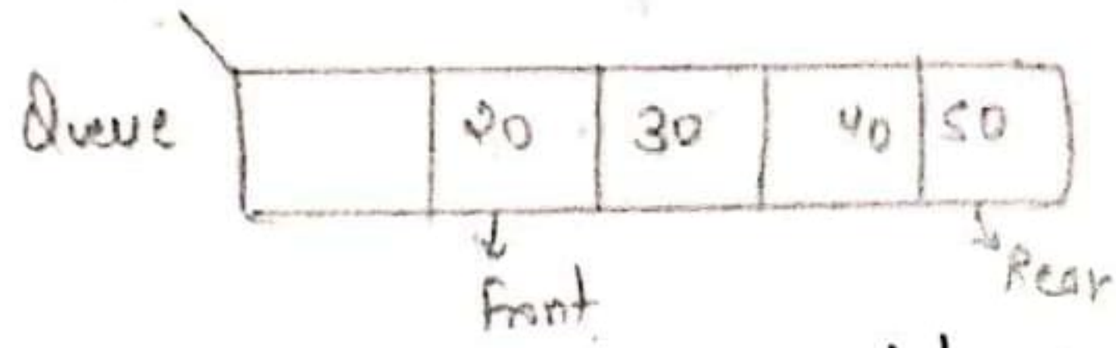
(ii) Deletion :-



maxsize → 5

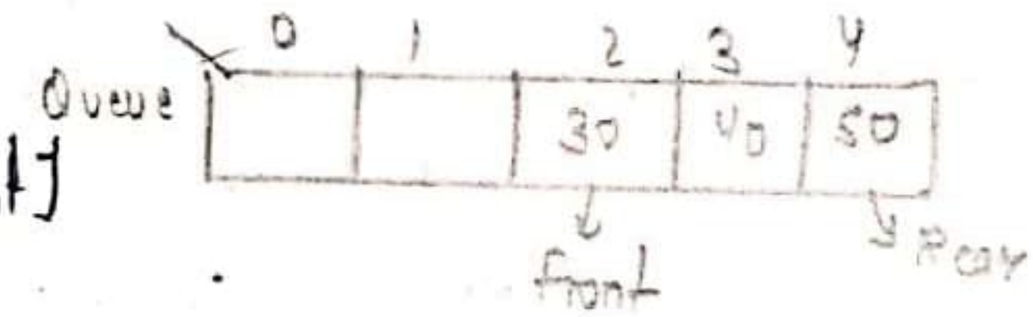
front = 0
Rear = 4

(i) Delete item = 10 ...
item = queue [front]
item = 10 .
Front = front + 1 .
so , front = 1 .
(front is increased by 1, But rear will be same as before) .
rear = 4 .



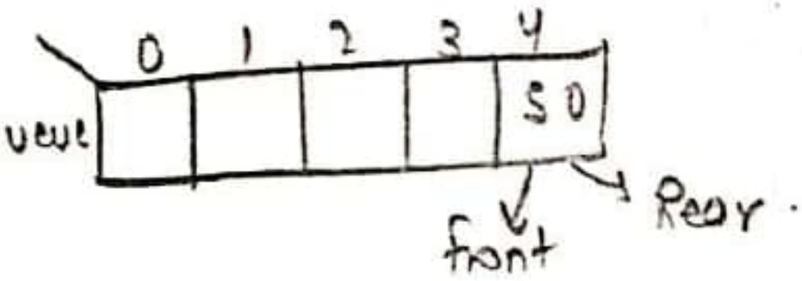
(ii) Delete :-
item again...

front = 1
rear = 4 .



item = queue [front]
item = 20
Front = front + 1 .
front = 2
rear = 4 .

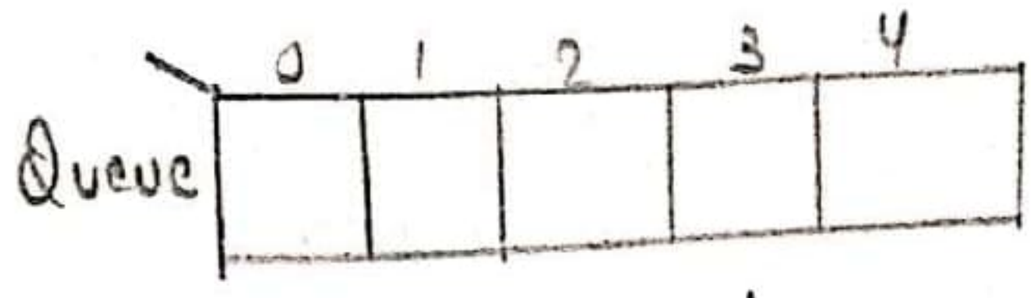
(iii) Now, Queue



item = Queue [front]
item = 40 .
front = 4
rear = 4 .

To delete 50 :-
front == rear
so, set , front = -1 .
and rear = -1 .

(iv) When Queue is underflow :-



if, front = -1
rear = -1
then Queue is underflow because front = -1 .

Downloaded from TwoWaits

Scanned by CamScanner

Delete operⁿ on the Queue takes place from front before deleting any element from a queue flow condⁿ must be check i.e. front == -1.

When any element is deleted from the queue, first it is stored in a variable after that front is increment by 1.

ALGORITHM FOR DELETE OPERATION :-

let queue [maxsize] is an Array, front and rear are two integers variable.

Step ① :- [Check for underflow condition :-]

if (front == -1)

then, print ("Queue underflow")

and return ;

Step ② :- [store the Element to be deleted into item variable]

item = queue [front]

Step ③ :- [Update the front and rear]

if front == rear,

then set front = -1 and

set Rear = -1 .

Step ④ :- else [update the front]

then, set front = front + 1 .

Step ⑤ :- STOP (or Return item) .

FUNCTION OF DELETION :-

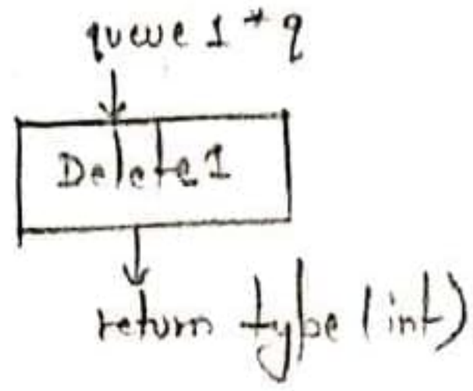
```
# define maxsize 5
typedef struct
{ int queue [Maxsize];
  int front;
  int rear;
} queue 1;
```

let, 'q' is a pointer variable of type "Queue1", and item is a variable of integer type and funcⁿ returns a value.

Downloaded from TwoWaits

Scanned by CamScanner

```
In Void Main ( ) → q. front = -1
initializ^n.      q. rear = -1.
                  queue 1 q ;
```




```
int delete 1 (queue 1 * q)
{ int item;
  if (q → front = = -1)
  {
     printf (" queue is underflow");
     return ;
  }
  item = q → queue [q → front];
  if (q → front = = q → rear)
  {
    q → front = -1 ;
    q → rear = -1 ;
  }
  else
     q → front = q → front +1 ;
     return (item) ;
}
```

(iii) Display or traversing :-
This oper^n can be performed to display the elements of the Queue starting from front to rear. In display oper^n first we check the under-flow condition i.e. front = = -1.

ALGORITHM FOR TRAVERSING :-
let, Queue [Maxsize] in an array, front & Rear are 2 integers variable.

step ① :- [check for underflow condition]
if (front = = -1)
then print (" Queue is underflow")
and return ;

step ② :- [Initialize the variable]
set i = front

step ③ :- [Display all the elements of Queue from front to Rear]
Repeat i = front to Rear
Display Queue [i]

Downloaded from TwoWaits

Scanned by CamScanner

Step (4) :- STOP.

FUNCTION OF TRAVERSING :-

```
# define maxsize 5
typedef struct.
{
    int queue[maxsize];
    int front;
    int rear;
} queue 1;
```

Queue1* q
↓
Display
↓
no return type (void)

let 'q' is a pointer variable of type queue 1 & item is a variable of type integer & func^n Doesn't Return a value.

```
{ In void main() → q. front = -1;
  initialization [ q. rear = -1;
                   queue 1 q; }
```

```
void Display (queue *q)
{
  if (q -> front == -1)
  {
    printf (" Queue is underflow");
    return;
  }
  for (i = q -> front ; i <= q -> rear ; i++).
  {
    printf (" %d\t", q -> queue[i]);
  }
}
```

* Limitations of linear Queue :-

(i) In the linear Queue overflow condition may be occur even if the array which is used to implement a Queue is not fill.

(ii) The limitation of linear Queue can be Removed by considering the index of an array in a circular way which gives the implementation of circular Queue.

Downloaded from TwoWaits

Scanned by CamScanner

CIRCULAR QUEUE...

A circular Queue is the one in which the insert^n of a new element is insert at the very first location of the Queue. If the last location of the Queue is full, the calculation of front and rear into the circular Queue is,

and front = (front +1) % maxsize.

Rear = (rear +1) % maxsize.

where, maxsize is the total No. of elements in the Queue.

(i) Queue is full and we have insert F :-

Maxsize = 5

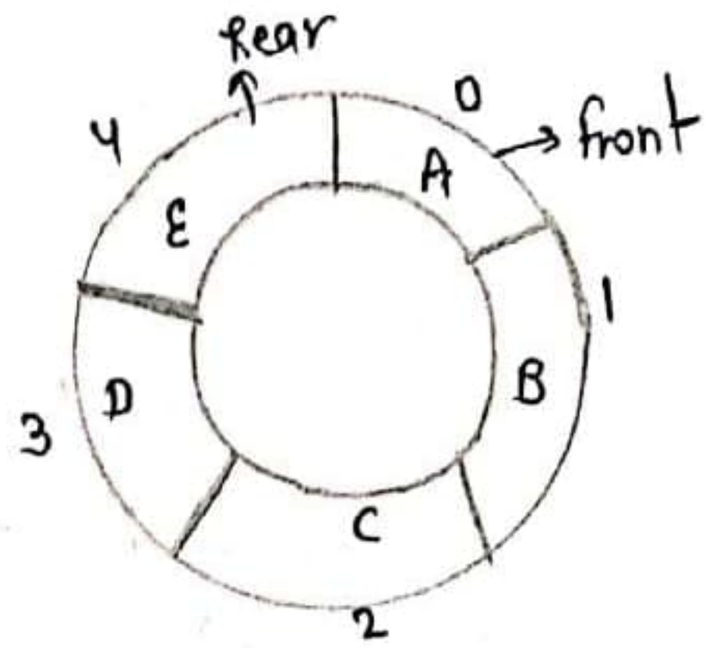
front = 0 && rear = 4

front == (rear +1) % maxsize

Now, front = 0 & rear = 0.

∴ front = rear

Hence, Queue is overflow.

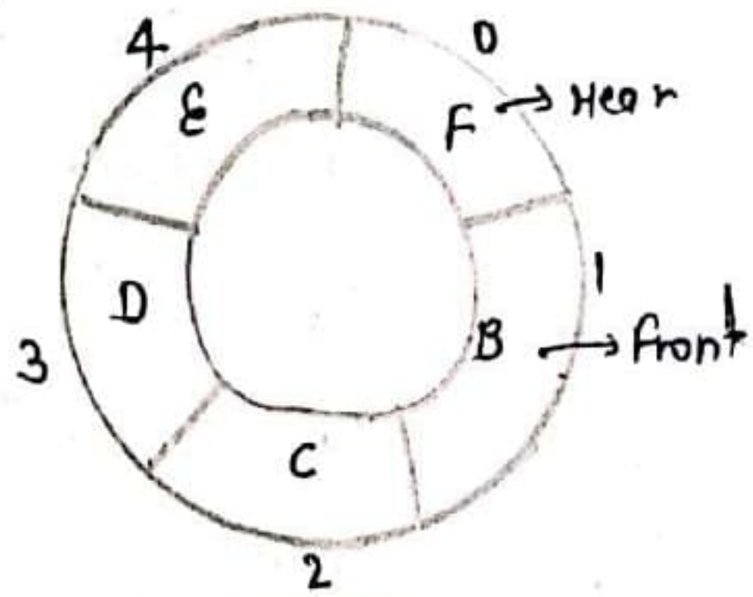


INSERT → F

front = 1 && Rear = 0

1 = (0+1) % 5

1 == 1. Hence, Queue is overflow.



(ii) Insert A :- if (front == -1)

for, front = -1, Rear = -1

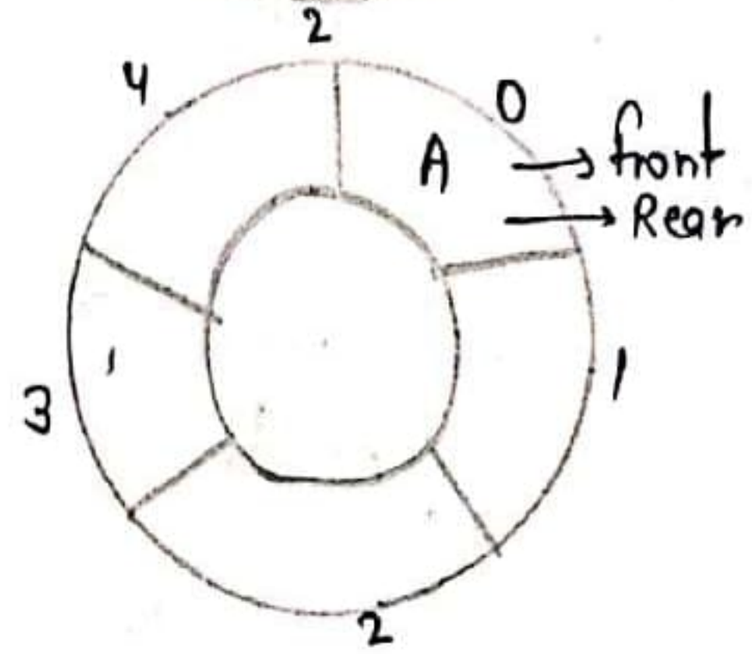
front = (front +1) % Maxsize.

rear = (rear +1) % Maxsize

OR

front = 0

rear = 0



(iii) If Elements are present in Queue, and Insert B :-

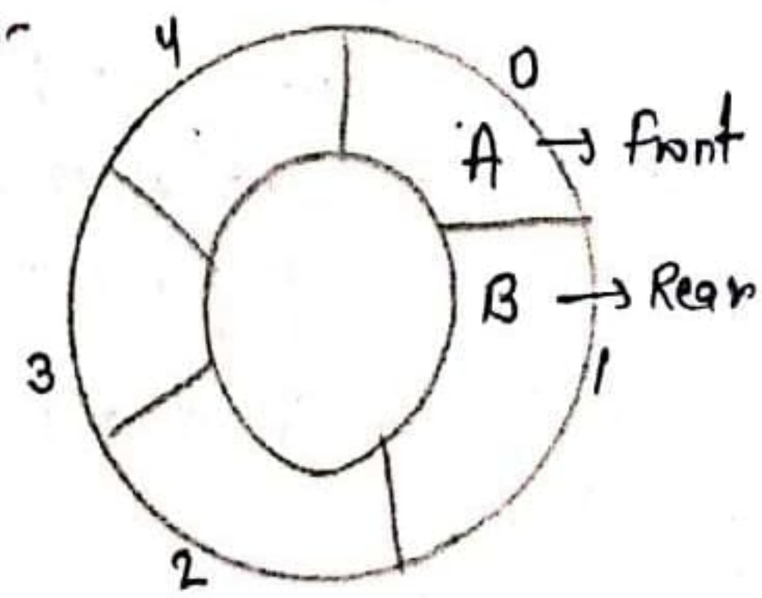
Here, front = 0, Rear = 0

front = 0

Rear = 1

OR

rear = (rear +1) % maxsize.



Downloaded from TwoWaits

Scanned by CamScanner

ALGORITHM FOR INSERT IN CIRCULAR QUEUE :-

let Queue [Maxsize] is an Array. Front & Rear are 2 integer Variables, which is initialised by -1

step ① :- [check for overflow condition]
if front == (rear+1) % Maxsize
then, print (Circular Queue is Overflow)
and exit.

step ② :- [Read item]

step ③ :- [update the front and Rear]
if front == -1
then, set front = (front+1) % Maxsize
& set Rear = (rear+1) % Maxsize.
[End of if structure]
otherwise
[update the rear]
rear = (rear+1) % Maxsize.

step ④ :- [insert the value into Queue [rear]]
set Queue [rear] = item

step ⑤ :- STOP.

FUNCn FOR INSERT DATA IN CIRCULAR QUEUE :-

similar to simple Queue

```
# define Maxsize 5
typedef struct
{
    int queue [Maxsize];
    int front;
    int rear;
} queue 1;
```

queue 1 *q int item
↓ ↓
insert C.Q.
↓
no return value (void)

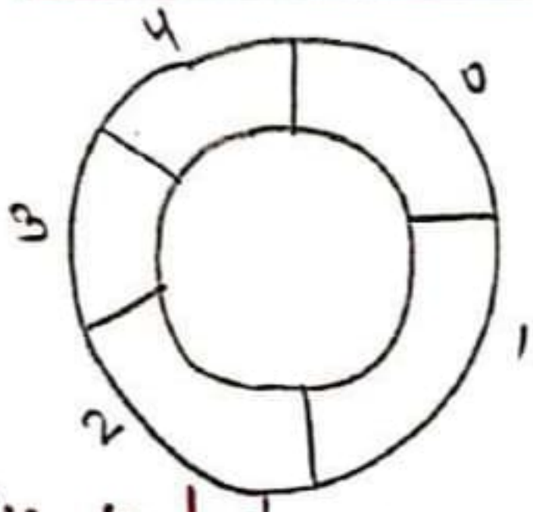
```
In void main () → queue 1 q;
                  q.front = -1;
                  q.rear = -1;
```

Downloaded from TwoWaits

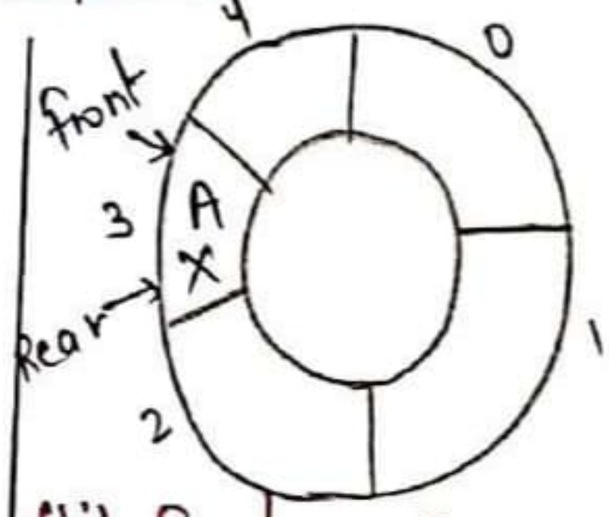
Scanned by CamScanner

```
void insertcq ( queue 1* q, int item)
{
 if ( q -> front == (q -> rear+1) % Maxsize)
 {
    printf (" queue is overflow");
    return ;
 }
 else if ( q -> front == -1)
 {
    q -> front = (q -> front+1) % Maxsize ;
 q -> rear = ( q -> rear+1) % Maxsize ;
 }
 else
    q -> rear = ( q -> rear +1) % maxsize ;
 q -> queue [q -> rear] = item;
}
```

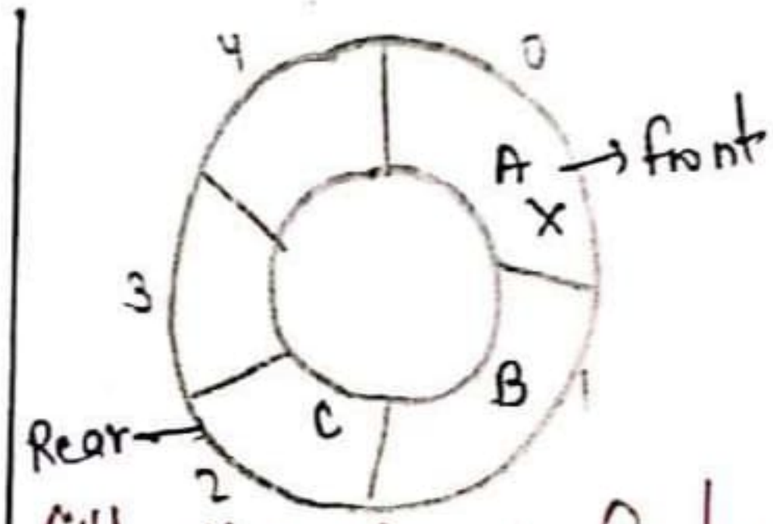
Deletion in Circular Queue... * item 'front' ⇒ Delete



(i) Underflow :-
front = -1
Rear = -1
Hence, Queue will be Underflow. No, Deletion will be performed.



(ii) front == Rear
Here, front = 3
Rear = 3
Delete A.
Then, front = -1
Rear = -1
Because stack get Empty.



(iii) When Rear & front are d/f.
Here, front = 0.
and rear = 2
front = (front+1) % maxsize.
Then,
front = 1
and rear = 2 (No change in Rear).

ALGORITHM FORDELETION IN CIRCULAR QUEUE...

Let, Queue [Maxsize] is an Array. front and Rear are 2 integers variable which are initized by -1.

step ①:- [check the underflow condition]

Downloaded from TwoWaits

Scanned by CamScanner

If, front == -1
then, print "Queue is underflow"
and exit.

Step ② [store the deleted item into a variable]
item = queue [front]

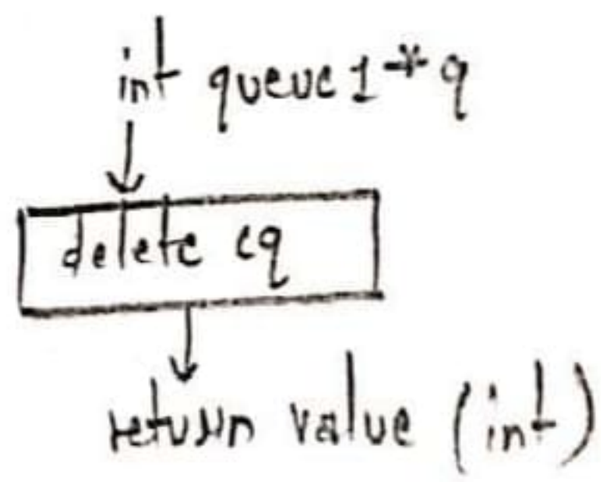
Step ③ :- [update the front and rear]
if front == rear
then, set front = -1
and set rear = -1
otherwise
[update the front]
front = (front + 1) % maxsize

Step ④ :- [Return the deleted item]

Step ⑤ :- STOP.

FUNC^n FOR DELETION IN CIRCULAR QUEUE :-

```
# define Maxsize 5
typedef struct
{
  int queue [maxsize];
  int front;
  int rear;
} queue 1;
int deletecq (queue1 *q)
{ int item;
  if (q -> front == -1)
  {
   printf ("Queue is Underflow");
   return;
  }
  item = q -> queue [q -> front];
  if (q -> front == q -> rear)
  {
   q -> front = -1;
   q -> rear = -1;
  }
  else
```



Downloaded from TwoWaits

Scanned by CamScanner

```
    q -> front = (q -> front+1) % maxsize;
    return (item);
  }
}
```

ALGORITHM FOR TRAVERSING IN CIRCULAR QUEUE :-

let queue [Maxsize] is an Array. front and Rear are 2 integer variable initialized by -1.

step ①- [check for underflow condition].
(front == -1) then
print "Underflow"
and return.

step ② - [intialize the variable]
set i = front

step ③ - [Display all the elements of Queue from front to rear-1]

step ④ - [Repeat i = front to not equal to rear]
[Display queue[i]]
[Calculate i]
i = (i+1) % maxsize
[Display the Rear index]
[Display q[i]]

step ⑤ STOP.

FUNC^n FOR DISPLAYING IN CIRCULAR QUEUE :-

int Queue1 *q
↓
displayq
↓
no return type (void)

```
#define MAXSIZE 5
typedef struct
{
    int queue [Maxsize];
    int front;
    int rear;
} queue1;
void displayq (queue1 *q)
{
    if (q -> front == -1)
{
```

Downloaded from TwoWaits

Scanned by CamScanner

```
printf ("queue is underflow");
return ;
}
for ( i = q -> front ; i != q -> rear ; i = (i+1) % maxsize)
{
  printf ("%d\n", q -> queue [i]);
}
printf ("%d\n", q -> queue [i]);
}
```

→ ये less than rear वाले element के लिए।

→ ये rear वाले element के लिए।

PRIORITY QUEUE :-

A priority Queue is a collech of elements where the elements are stored according to their priority levels. The order in which the item deleted or inserted is decided by the priority of the Queue.

Rules applied to priority Queue :- (Rules of BODMAS)

(i) Elements with Higher priority is processed before any element of lower priority

Eg. 2 + 3 * 4 → ये पहले execute होगा (Higher priority)

(ii) If two elements with the same priority, then the elements added 1st in the Queue could be get processed first.

⇒ Eg. 2 + 3 - 4 → ये पहले आया तो, पहले Execute होगा।

Types of priority Queue :-

(i) Ascending Priority Queue :- It is a collech of elements in which elements can be inserted randomly and from which only the smallest Element (priority) can be Removed any time.

(ii) Descending priority Queue :- It is the collech of elements into which Elements can be inserted Randomly but every time in the deletion largest element will be deleted first.

DOUBLE ENDED QUEUE (DEQUE) :-

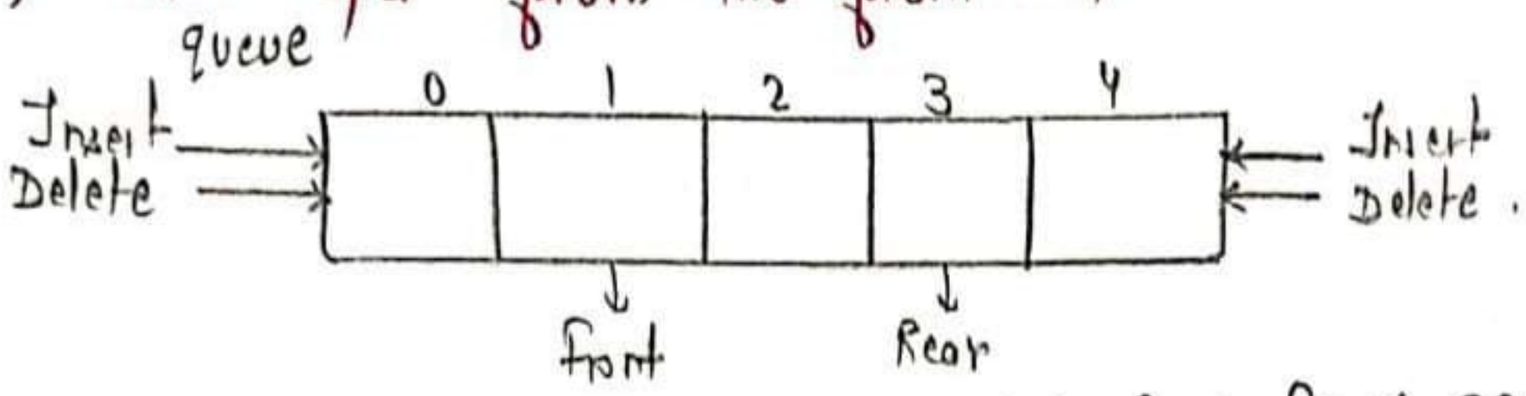
The Double Ended Queue we can perform insert and Delete operh from the both End i.e. front & Rear.

The following operh can be perform on Deque :-

Downloaded from TwoWaits

Scanned by CamScanner

(i) Insert operⁿ from the rear end.
(ii) Insert operⁿ from the front end.
(iii) Delete operⁿ from the rear end.
(iv) Delete operⁿ from the front end.

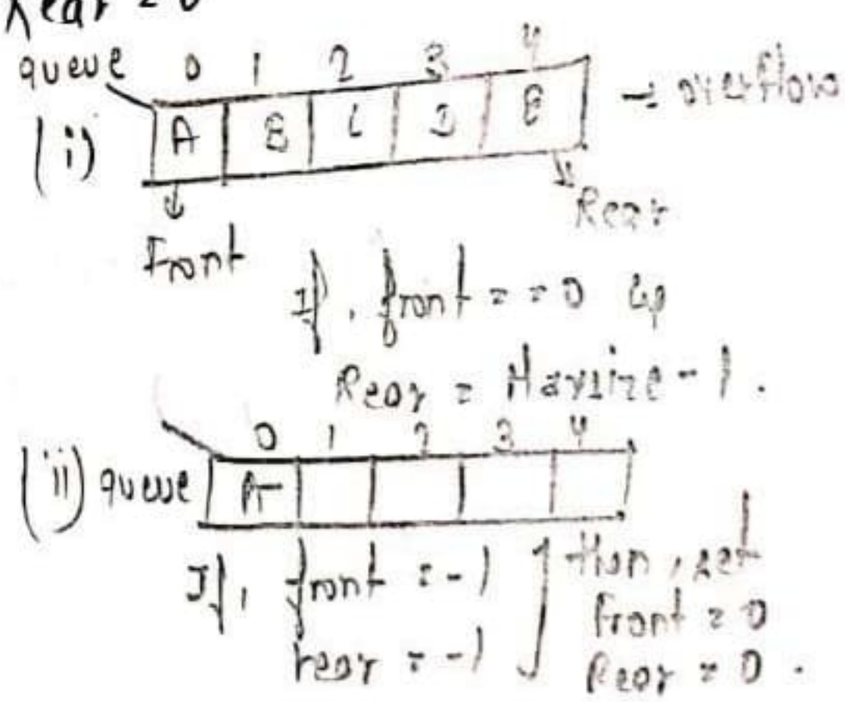


ALGORITHM FOR INSERT OPERⁿ IN DEQUE FROM REAR END :-
(same as linear Queue) ...

ALGORITHM FOR INSERT OPERⁿ IN DEQUE FROM FRONT END :-
let queue [maxize] is an Array. front and Rear are 2 integer variables initialised by -1.

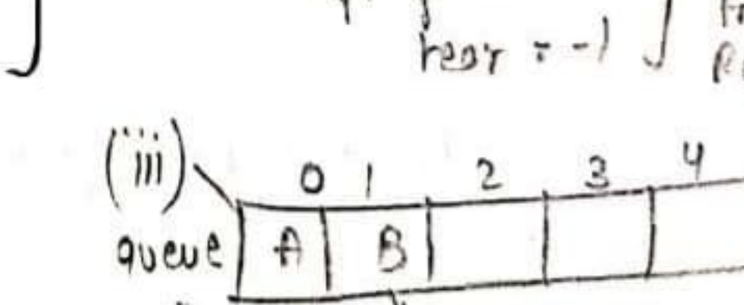
step(i) [check for overflow condition]
if front == 0 and Rear == maxize-1.
Then, print "Queue is Overflow"
and return.

step (ii) [update the front and rear]
if, front == -1, then set front = 0
and set Rear = 0
else if, Front == 0
repeat i = rear to front
queue [i+1] = queue [i]
[end of loop]
rear = rear + 1
[end of if]
else, [update the value of front]
front = front - 1.



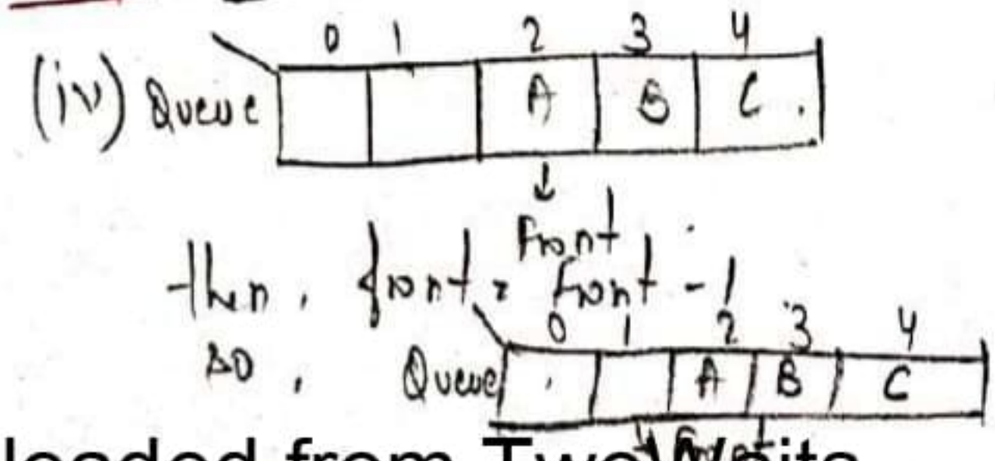
step (iii) [Insert the element].
queue [front] = item.

if, front == 0
* Rear को Right side shift करो।
* front को अगली छोड़ दो।
i = Rear to front.



Here, rear = rear + 1.

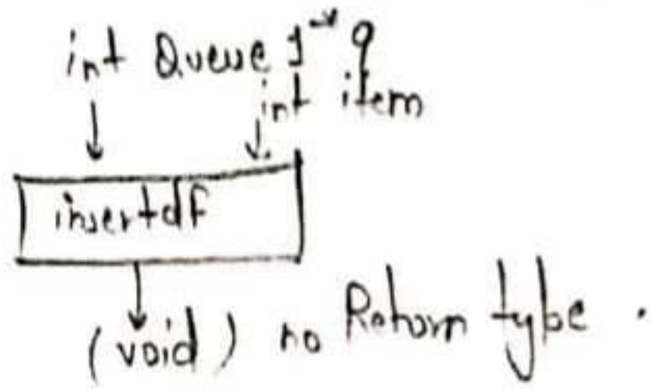
step (iv) STOP



Downloaded from TwoWaits

Scanned by CamScanner

FUNCTION FOR INSERTⁿ OPERⁿ IN DEQUE FROM FRONT END :-




```
# define maxsize 10 .
typedef struct
{
   int queue [Maxsize];
   int front;
   int rear;
} queue 1;
void insertdf ( queue 1 *q, int item)
{int i;
{
  if (q -> front == 0 && q -> rear == maxsize -1)
{
   printf (" Overflow");
   return;
}
elseif (q -> front == -1)
{ q -> front = 0;
  q -> rear = 0;
}
elseif (q -> front == 0)
{
   for( i = q -> rear ; i >= q -> front ; i = i-1)
{
   q -> queue[i+1] = q -> queue [i] ;
}
   q -> rear = q -> rear +1 ;
}
else q -> front = q -> front-1 ;
  q -> queue [q -> front] = item ;
}
```

ALGORITHM FOR DELETE OPERⁿ IN DEQUE FROM FRONT END :-
(same as linear Queue ...)

ALGORITHM FOR DELETE OPERⁿ IN DEQUE FROM REAR END :-
let, queue [maxsize] is an Array. front & Rear the two integers variable initialized by -1.

Downloaded from TwoWaits

Scanned by CamScanner

Step (i) :- [check for underflow condition].
If, front == -1, then
print "underflow" and return.

Step (ii) :- [store the deleted element into a variable]
item = queue [rear]

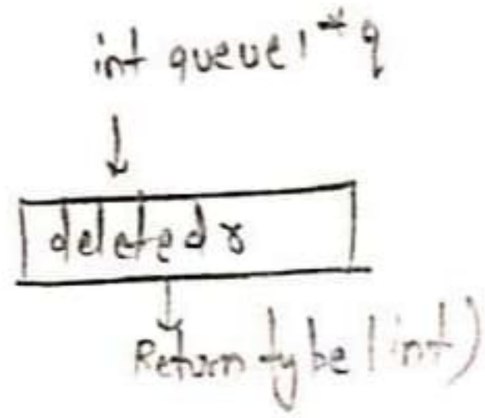
Step (iii) :- [update Rear and front].
if, front == Rear
then, set front = -1 and Rear = -1.
else, [update the Rear]
set, rear = rear - 1

Step (iv) [Return the deleted item]

Step (v) STOP.

FUNC^n FOR DELETE OPER^n IN DEQUE FROM REAR END :-

```
#define maxsize 10
typedef struct
{
    int queue [maxsize];
    int front;
    int rear;
} queue1;
int deleteder (queue1* q)
{
  int item;
  if (q->front == -1)
  {
    printf ("underflow");
    return;
  }
  item = q->queue [q->rear];
  if (q->front == q->rear)
  {
    q->front = -1;
    q->rear = -1;
  }
  else
    q->rear = q->rear - 1
    return (item);
}
```



Downloaded from TwoWaits

Scanned by CamScanner

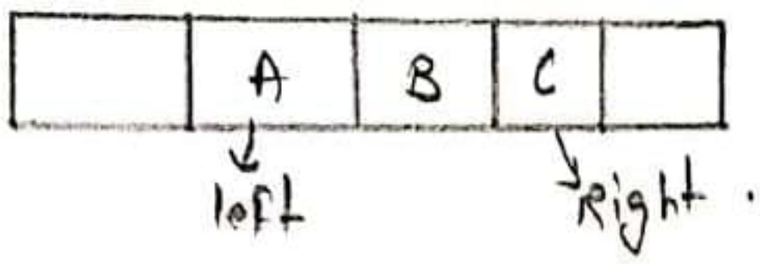
Types of Deque :-

(i) Input Restricted Deque :- Element can be added at only one end but we can delete the element from both ends.

(ii) Output Restricted Deque :- It is that Deque, where deletion takes place at only one end but allow insertion at both ends.

* In some Books, front and Rear are also specified as Left & Right.

like.



Downloaded from TwoWaits

Scanned by CamScanner

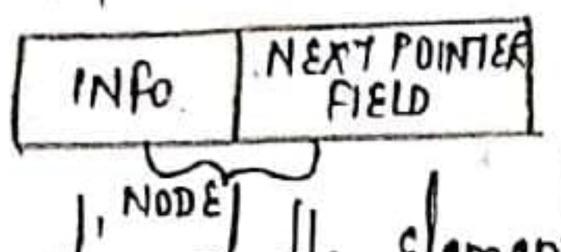
LINKED LIST...

* अगर program में हम getnode() use करते हैं तो, उसकी Def^n लिखनी पड़ेगी बाद में।

A linked list is a linear collection of Data elements known as Nodes where linear order is given by pointers. Each element in the linked list is called Node and each Node is Divided into two parts :-

(i) information (Info)

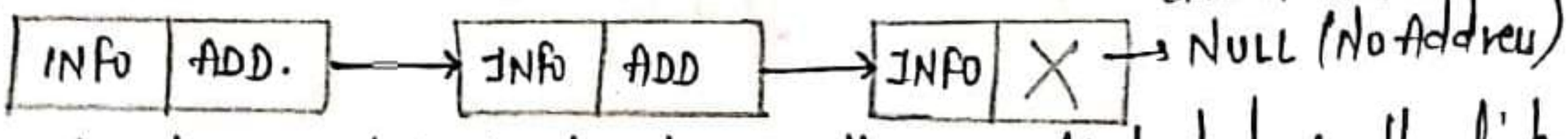
(ii) Address



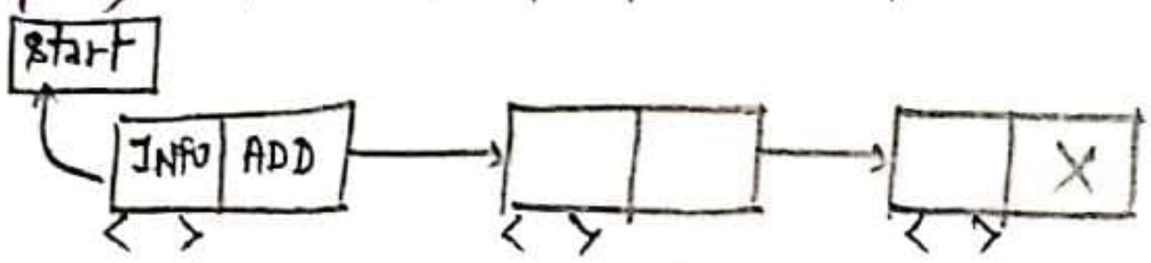
(i) Information :- It contains the information of the element.

(ii) Next pointer field / Link field / Next Address field :- It contains the Address of the Next Node in the list. This Address is used to access the Next Node in the list, and link field is a type of pointer Variable.

NULL pointer :- The link field of the last node contains NULL and it is a Null pointer and it indicates the End of the list.



EXTERNAL pointer :- It is the pointer to the very first Node in the linked list (START) It enables us to access the entire link list.



EMPTY LIST :- If the Nodes are not present in the list then it is called Empty linked list and is also called NULL list.

The Value of the external pointer Variable will be NULL for an Empty list.

GET NODE ( ) FUNCTION :- This funcⁿ will Generate a Node and Return the Address of that Node. * ये funcⁿ Node को create करेगा जितना User को चाहिए।

* अगर Node create करने के लिए Memory available नहीं है, तो Get Node ( ) ये error बता देगा।

* ये bye-defined funcⁿ नहीं है, User defined है।

```
p = getNODE( );
```

'p' में Address store होगा Node का तो, ये pointer Variable है।

, which is stored in a pointer variable 'p'.

NODE (p) :- Node (p) will represent the NODE, which address is stored in pointer Variable p.

INFO (p) :- Info (p) will represent the informⁿ part of a node which address is stored in pointer Variable p.

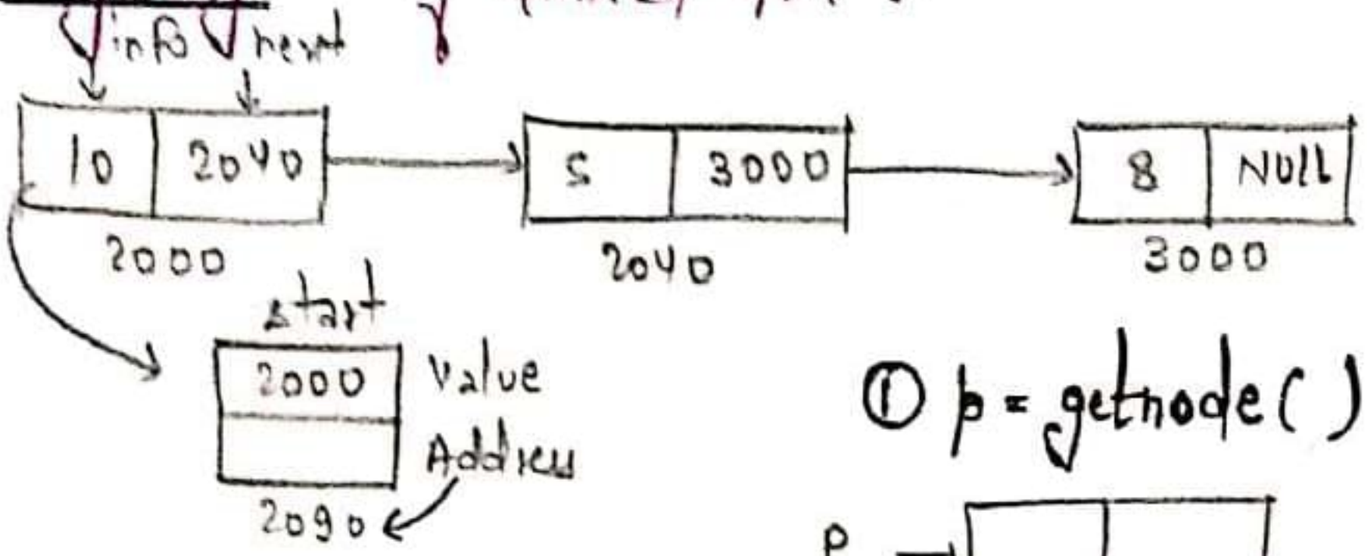
Downloaded from TwoWaits

Scanned by CamScanner

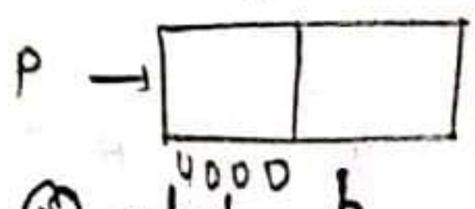
NEXT(p) :- It is a pointer variable which stores the Address of Next Node

FREE(p) or freenode(p) :- It will deallocate the Memory, allocated to a deleted node which address is stored into pointer variable p.

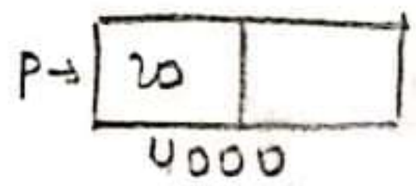
* Write an Algorithm for an insert operation of a NODE at a Begining of linked list :-



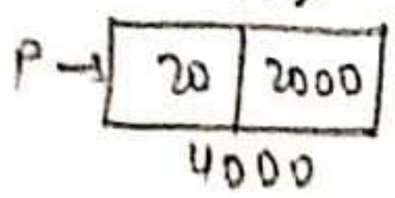
① p = getnode( )



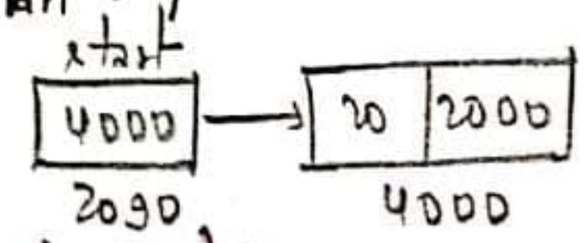
② info(p) = item



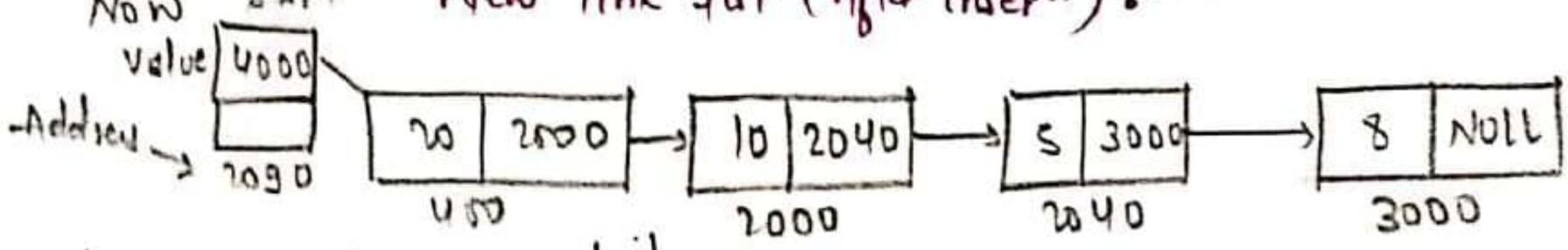
③ next(p) = start



④ start = p



Now New link list (After insertn) :-



Step ① :- [Get a New Node]

p = getnode( )

if, p == NULL

then print "Memory allocation Unsuccessful" and.

return.

Step ② :- [Write informn in New node]

info(p) = item

Step ③ :- [Insert New Node at the front of the link list, by adjusting start and Next pointer field of p]

next(p) = start

start = p

Step ④ :- STOP...

Downloaded from TwoWaits

Scanned by CamScanner

// Double pointer

```
void main ()
{
  int x, *start, **q;
   x = 10;
   start = &x;
   q = &start;
   printf ("\n Value of x is \t %d ", x);
or printf ("\n value of x is \t %d", *start);
or printf ("\n value of x is \t %d", **q);
   printf ("\n address of x is \t %u ", &x);
or printf ("\n address of x is \t %u", *q);
or printf ("\n address of x is \t %u", start);
   printf ("\n address of start is \t %d", q );
}
```

x	start	q
10	1000	2000
1000	2000	2010

} used to print value of x

} used to print address of x

→ used to print address of start.

* Write a func^n to create a node and Return its address :-

```
struct node
{
  int info;
  struct node * next;
}
struct node *getnode();
void main()
{
  struct node *p;
   p = getnode ();
}
struct node *getnode ()
{
  struct node *p;
  p = (struct node *) malloc (sizeof (struct node));
  If (p == NULL)
  printf ("Memory allocating unsuccessful");
   return;
  else
   return (p);
}
```

self referential structure.

ये यहाँ एक Node create करके देगा।

*getnode() func^n की Return type है "struct node".

→ ये pointer variable है 'struct node' type का होगा।

→ written for type casting.

→ Used for Dynamic Memory allocation.

Downloaded from TwoWaits

Scanned by CamScanner

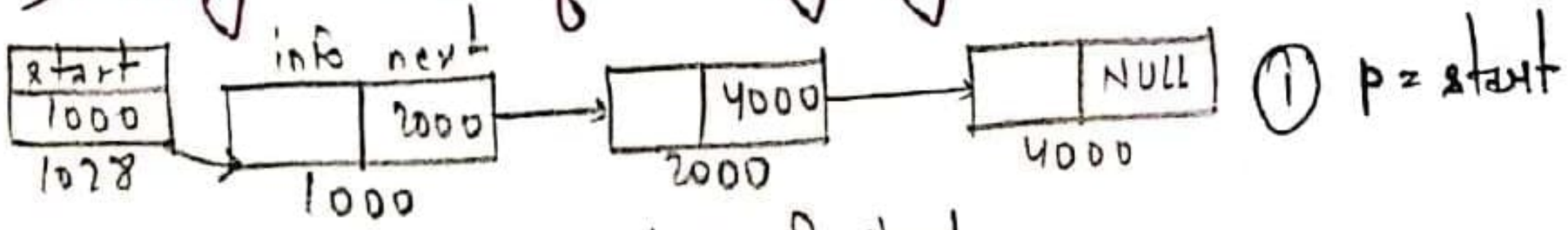
* Write a funcn for inserting a node at the front of the linked list :-

```
struct node
{
 int info;
 struct node* next;
};
 // initialization
 struct node* start;
 start = NULL;
 // function prototype struct node* getnode();
 void insert_front(struct node **);
 // calling
 insert_front(& start);
 // function Definition
 void insert_front(struct node** q)
{
 struct node* p;
 p=getnode();
 if (p != NULL)
 {printf("enter information");
  scanf("%d", &p->info);
  p->next = *q;
  *q = p;
 }
}
```

Here, q -> & start
*q -> start (i.e. value of start)

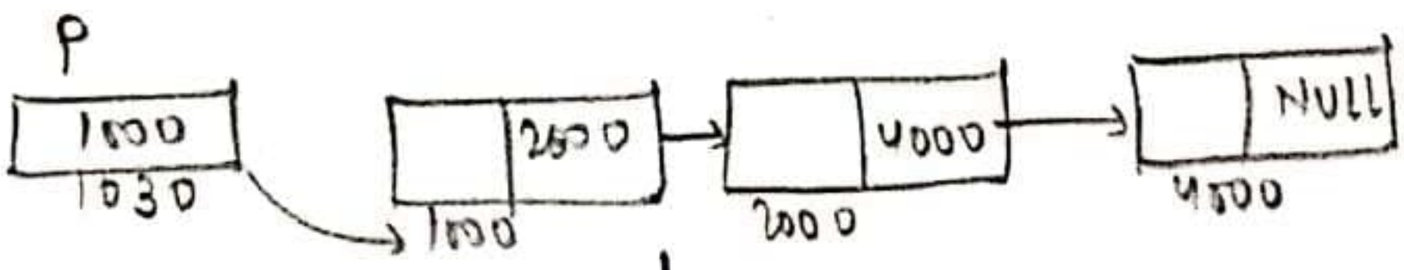
getnode() -> This funcn will Generate a Node and Return the Address of that node.

* Deleting a node from Begining :-

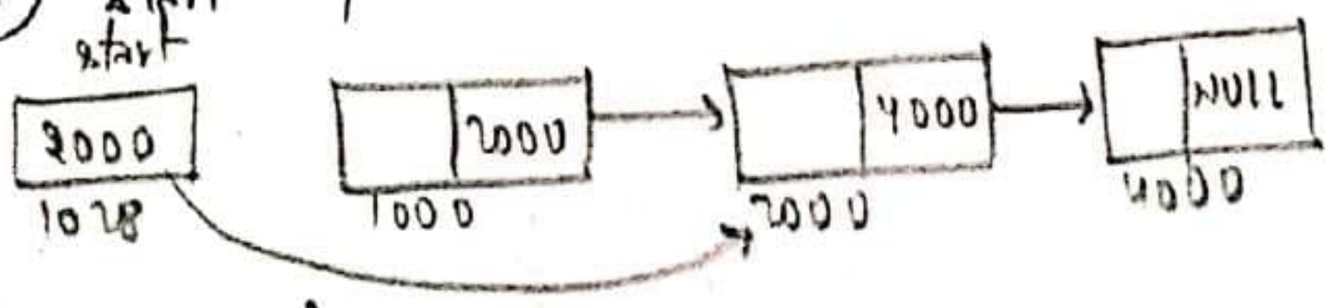


① p = start

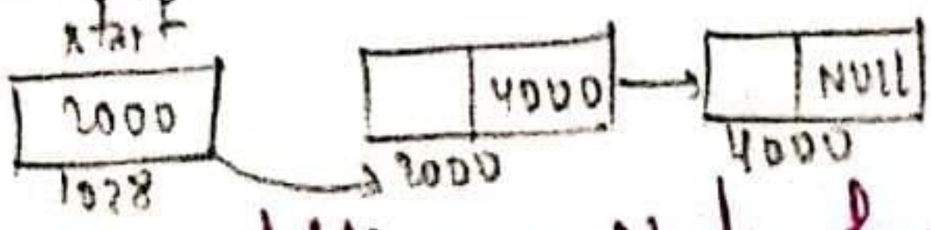
* node को create करने का task नहीं होगा।



② start = p -> next



③ Free(p)



* Algorithm for deleting a Node from Begining :-

Step ① :- [store the start address into another variable]
Set p = start.

Downloaded from TwoWaits

Scanned by CamScanner

step ② :- [check that link list is empty or not]
if p == NULL
then link list is empty and
return.

step ③ :- [store the inform^n Data of deletedNode]
item = info(p)

step ④ :- [delete the front node by adjusting the start and pointer variable 'p']
start = next(p)

step ⑤ :- [Deallocate the Memory from the deleted node]
free(p).

step ⑥ :- [Return the item]
return (item)

step ⑦ :- STOP

* write a func^n to delete a node from the begining :-

```
struct node
{
  int info;
  struct node* next;
};
int delete_front (struct node**q).
int x;
struct node *start;
start = NULL;
x = delete front (& start);
```

→ Delete में Deleted value Return होगी इसलिए 'int' लेंगे।
→ Linked list is empty in starting.
→ Calling the func^n. 'x' में Delete value store कराकर Return कराएंगे।

```
int delete front (struct node **q)
{
  struct node *p;
  int k;  → q की जगह item भी लिख सकते हैं
  p = *q;
  if (p == NULL)
  {
    printf("linked list is empty");
  }
  else
  k = p→info;
  print("\n");
  *q = p→next;
  free (p)
  return (k);
}
```

→ q → start
→ *q → & start

* TRAVERSING OF LINK LIST :- In the traversing of a link list, we print or traverse the content of the link list from 1st to last Node. The 1st Node is accessed by the pointer variable start and next Node are accessed by the next field of previous Node. End of the list is determined by the 'NULL' value which is present into the Next field of last node.
→ During display oper^n the list is not updated, so we only pass the variable value not address.

Downloaded from TwoWaits

Scanned by CamScanner

So, we take '*q' instead of '**q' in parameter.

* Algorithm for Displaying the contents of a linear link list :-

step ① :- [Assign the value of start into a variable].

p = start

step ② :- [check that linked list is empty or not]

If, p == NULL
then print "link list is empty".
and return.

step ③ :- [start loop for displaying the content of link list from first node to last node].

[Repeat steps ⑤ and ④ until p is not equal to NULL]

```
while (p != NULL)
{
p = p->next
}
```

p != NULL

step ④ :- [Display the inform^n Data]

print info (p)

step ⑤ :- [Update the pointer to point the next node in the list]

p = next (p)

step ⑥ :- stop.

* func^n of traversing in a link list :-

```
struct node
{
  int info;
  struct node* next;
};
void display (struct node *);
struct node* start;
 start = NULL;
display (start);
void Display (struct node *p)
{
  if (p == NULL)
  {
    printf ("linked list is empty");
    return;
  }
  else
    while (q != NULL)
    {
      printf ("%d", p->info);
      p = p->next;
    }
}
```

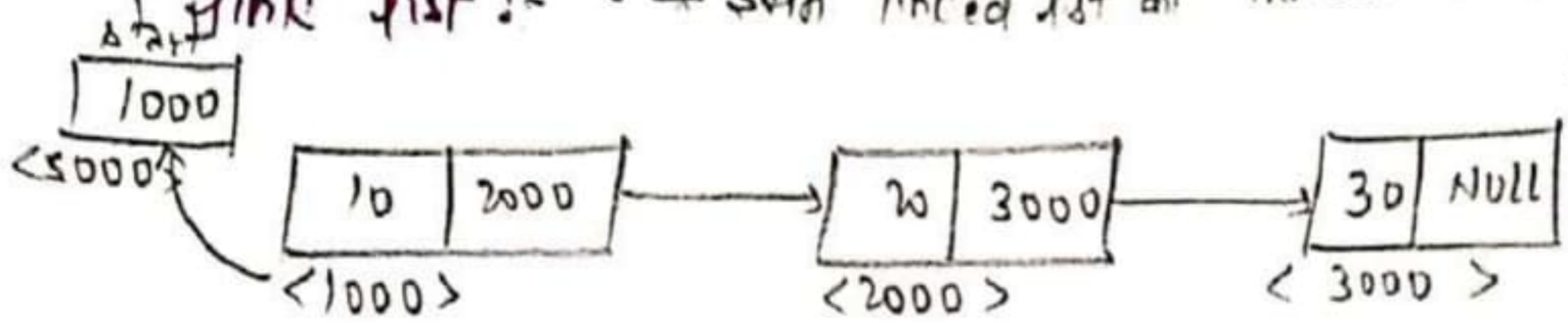
'p' का Information Part print करा रहे है।

Downloaded from TwoWaits

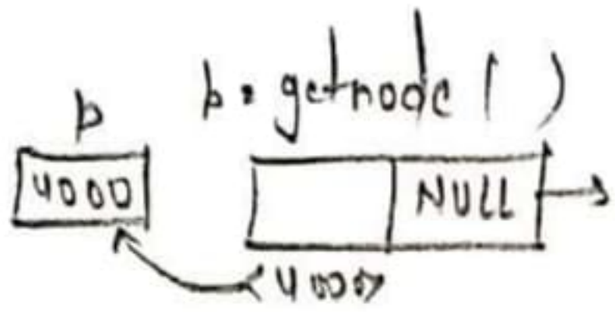
Scanned by CamScanner

* Write an Algo^m to insert a Node at the end of the link list :-

* इसमें linked list को traverse करना पड़ेगा, तभी



* यहाँ b → New node में जायेगा क्योंकि b = getnode()
* और 'temp' b का नया pointer variable बनेगा।



Here, temp = start

If linked list is empty → (temp != NULL)

To traverse the list → (temp → next != NULL)

temp → next = b

b → next = NULL

Here let "start" is the pointer variable which points the 1st node in the linked list.

step ① - [Initialize the variable]
temp = start

step ② - [Find the Address of last node in the linked list]
[Repeat step ③ until next of temp is not equal to NULL]
next (temp) != NULL

step ③ - [Find the Address of Next node]
temp = next (temp)

step ④ - [get a node]
b = getnode ()
If (b == NULL)
then Print "Memory allocation unsuccesful".
and Return

step ⑤ - [write inform^n into the New node]
into(b) = item.

step ⑥ - [Insert the node at the end of the link list].
next (temp) = b
next (b) = NULL

step ⑦ :- STOP

* func^n to insert a node at the end of link list :-

```
struct node
{
  int info;
  struct node* next;
};
void insert_last (struct node **);
struct node * start;
```

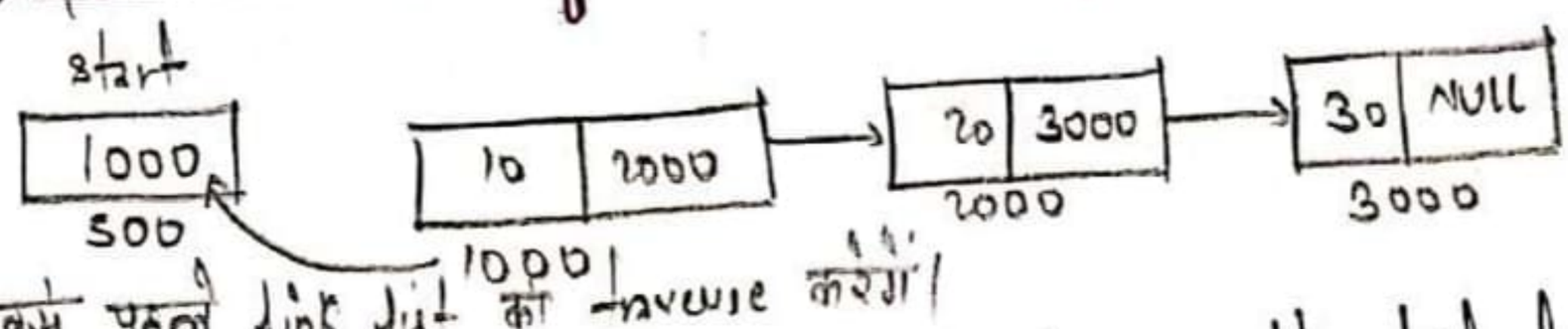
Downloaded from TwoWaits

Scanned by CamScanner

```
start = NULL;
        insert_last (& start);
void insert_last (struct node **q)
{
  struct node * temp, *p;
  temp = *q ;
  while (temp -> next != NULL)
  {
    temp = temp -> next;
  }
  p = getnode() ;
  if (p != NULL)
  {
    printf ("enter the information of
                new node=");
    scanf ("%d", & p -> info);
    temp -> next = p;
    p -> next = NULL;
  }
}
```

getnode() → this func^n will generate a node & Return the address of that node.

* Delete a Node from the end of the link list.



सबसे पहले link list को traverse करेंगे।

Algorithm :- let 'start' is the pointer variable which points the 1st NODE in the linked list, 'prev' and 'temp' are two structure pointer variable and item is a integer variable.

step ① :- [Initialize the variable].
temp = start

step ② :- [check that link list is empty or not].
If temp == NULL
then print " linked list is Empty"
and Return

step ③ :- [Find the address of the last node which is to be deleted and previous Node address]
[Repeat step ④ until next of temp is Not Equal to NULL].
next (temp) != NULL

step ④ :- [find the address of last node & previous node]
prev = temp
temp = next (temp)

step ⑤ :- [store the inform^n of the data of the last node]

Downloaded from TwoWaits

Scanned by CamScanner

item = info (temp)

step ⑥ :- [Delete the last node by adjusting the previous pointer].
next (prev) = NULL or next (prev) = next (temp)
By adjusting the previous to temp pointer.

step ⑦ :- [Deallocate the memory from the Deleted node]
free (temp).

step ⑧ :- [Return the deleted item]
return (item)

step ⑨ :- STOP

* Func^n for Deleting a Node from end of list :-

```
struct node
{
 int info;
 struct node* next;
};
int delete_last ( struct node **);
struct node * start;
start = NULL;
 item = delete_last(& start);
 int delete_last (struct node **q)
{
 int item;
 struct node * temp, *prev;
 temp = *q;
if (temp == NULL)
 {printf ("linked list is empty");
 return0;
 }
 while (temp -> next != NULL)
 {
  prev = temp;
  temp = temp -> next;
 }
 item = temp -> info;
 prev -> next = NULL;
 free (temp);
 return (item);
}
```

* Algorithm for count the No. of Node in the link list :-

step ① :- [Assign the value of a start into a variable]
p = start [then initialise the count variable to 0]
and set count = 0

step ② :- [check the link list is empty or not]
If, p == NULL
then print "link list is empty".
and return.

Downloaded from TwoWaits

Scanned by CamScanner

step ③ :- [start loop for count no. of nodes in the link list starting from 1st Node to last Node]
[Repeat step ④ until p is not equal to NULL)
p != NULL

step ④ :- [find the Address of Next node by Increment the count by 1)
p = next (p)
and Count = count + 1

step ⑤ :- [print the value of count variable]
print "count".

Step ⑥ :- STOP.

* Func^n in C to count no. of nodes in link list :- Here, q = p

```
struct node
{
 int info;
 struct node * next;
 };
void count (struct node *);
 struct node * start;
  start = NULL;
 count (start );
 void count ( struct node * q)
{ if (q == NULL)
 { printf (" linked list is empty\n");
  return ;
 }
 else while (q != NULL)
 {
  count = count + 1;
   q = q -> next ;
 }
 printf ("%d", count);
}
```

* Algorithm for insertion of node After a Given node at a particular location :-

let 'start' is a pointer to the 1st Node in the list. 'Loc' is the location of the node after which new node will be inserted.

step ① [Initialization of variable]
Set temp = start and i = 1

Downloaded from TwoWaits

Scanned by CamScanner

step (2) :- [To find the address of node present at location 'loc']
Repeat steps (3) to (5) until $i < loc$.

step (3) - [check that node at locⁿ 'loc' exists or not]
if, temp == NULL
then, print "link list has less No. of nodes" and return.

step (4) :- [Get the address of next node]
temp = next (temp)

step (5) - [Increase the value of i by 1]
i = i+1

step (6) - [get a new node]
p = getnode ( )
if (p == NULL), then print "Memory allocation unsuccessful",
and return.

step (7) - [write down the informⁿ into New Node]
info (p) = item

step (8) - [Insert the node after the node which is present at location "loc"]
next (p) = next (temp)
next (temp) = p.

step (9) - STOP...

* funcⁿ for inserting Node After a Given Node :-

```
struct node
{
  int info;
  struct node *next;
};
struct node * getnode ();
void insert_after (struct node **);
struct node * start;
start = NULL;
void insert_after (struct node **q)
{
  struct node * temp, *p;
  int loc, i;
  temp = *q;
  printf ("Enter the locⁿ after which a
          node get inserted");
  scanf ("%d", &loc);
  for (i=1; i<loc; i++)
  {
    if (temp == NULL)
    {
      printf ("lesser no. of nodes");
      return;
    }
    temp = temp -> next;
  }
  p = getnode ();
  if (p == NULL)
  {printf ("Memory allocⁿ unsuccessful");
    return;
```

Downloaded from TwoWaits

Scanned by CamScanner

```
}
printf ("enter inform");
scanf ("%d", &p -> info);
p -> next = temp -> next;
temp -> next = p;
}
```

* Write an Algo^m to delete a Node from linked list from a particular location.

let 'start' is a pointer to the first node in the list as 'loc' is the loc^n of the node which is to be deleted.

step ① - [Initialization of variable]
set temp = start and i=1.

step ② - [check that link list is empty or not]
if (temp == NULL), then print "link list is empty"
and return.

step ③ - [Find the address of the node, which is to be deleted and the address of the previous node]
Repeat steps ④, ⑤ and ⑥ until i < loc

step ④ - [Find the address of the previous and Next node]
set, prev = temp
temp = next (temp)

step ⑤ - [Increment the i by 1]
i = i+1.

step ⑥ - [check that node at the given loc^n exists or not]
if temp == NULL, then print "link list has lesser no. of nodes"
and return.

step ⑦ - [store the infor^n of the deleted node]
set item = info (temp)

step ⑧ - [Delete the node by adjusting the 'prev' and 'temp' pointer.]
next (prev) = next (temp)

step ⑨ - [Deallocate the Memory from the deleted node]
free (temp)

step ⑩ - [Return the deleted item]
return (item)

step ⑪ - STOP...

Downloaded from TwoWaits

Scanned by CamScanner

* func^n for deleting a Node at particular position :-

```
struct node
{
  int info;
  struct node * next;
};
  int delete_loc(struct node **);
  struct node * start;
  start = NULL;
  delete_loc(&start);
  int delete_loc( struct node ** q)
{
  struct node * temp, * prev;
  int i, loc, item;
  temp = *q;
  If (temp == NULL)
{ printf ("linked list is empty");
  return;
}
printf (" Enter loc^n which you want
          to delete");
scanf (" %d", &loc);
for (i=1; i<loc; i++)
{
  prev = temp;
  temp = temp -> next;
if (temp == NULL)
{ printf (" lesser no. of nodes ");
  return;
}
item = temp -> info;
prev -> next = temp -> next;
free (temp);
return (item);
}
```

* Write an Algo^m to implement push and pop oper^n using linked list

OR

Implementation of stack using linked list...

* linked list can be implemented as stack when both the insert and delete oper^n can be perform at the front of the linked list.

* Algorithm for "push" oper^n in stack from linked list :- (same Algo^m as inserting at beginning of the linked list)

Let "start" is the pointer to the last inserted node in the linked list.

step (1) :- [Get a new node]
p = getnode ()
if (p == NULL), then print" stack overflow" and return.

step (2) - [Write inform^n Data in the New node]
info (p) = Item

Downloaded from TwoWaits

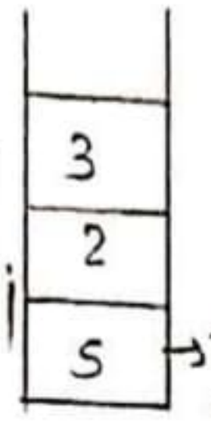
Scanned by CamScanner

step ③ :- [Insert this new node in the front of the linked list by adjusting start & next pointer field of p]

next(p) = start
start = p.

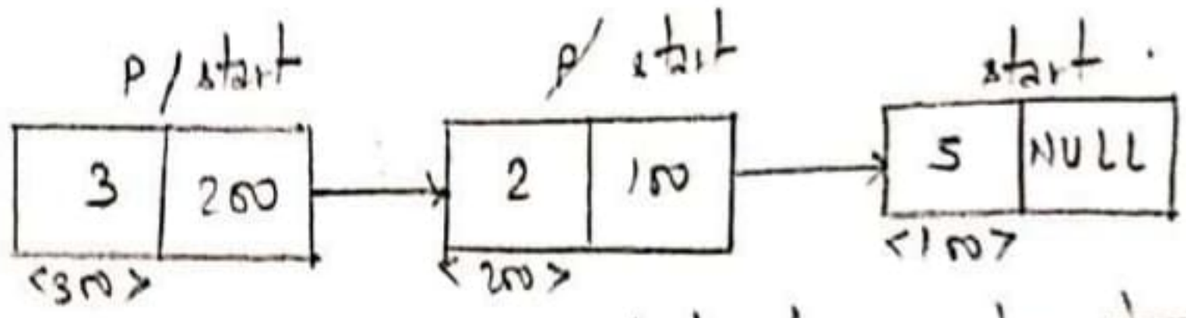
step ④ :- **STOP**...

Explanation :-



ये दोनों elements lost वाले के आगे लगते रहेंगे।

→ ये element linked list से lost होगा।



→ start change होता रहेगा।

→ Funcⁿ for push operⁿ in stack using linked list. (same as Inserⁿ at Begining)

```
struct node
{
   int info;
   struct node* next;
};
struct node* start;
   start = NULL;
void stack_push (struct node **);
stack_push (& start);
void stack_push (struct node ** q);
{
struct node *p;
 p = getnode ();
 if (p != NULL)
 {
  printf("Enter information");
  scanf ("%d", & p -> info);
  p -> next = * q;
  * q = p;
 }
}
```

getnode () :- This funcⁿ will Generate a Node and Return the Address of that Node.

→ Algorithm for "POP" operⁿ using linked list. (same as Deletion at the Begining)

let "start" be the pointer to the 1st inserted node in the linked list.

step ① :- [store the start address into another variable]
set p = start

step ② :- [check that linked list is Empty or not]
if, p = = NULL, then print "stack is underflow" and return.

step ③ :- [store the informⁿ Data of Deleted Node]
set item = info(p)

step ④ :- [Delete the front node by adjusting the start and pointer variable].

Downloaded from TwoWaits

Scanned by CamScanner

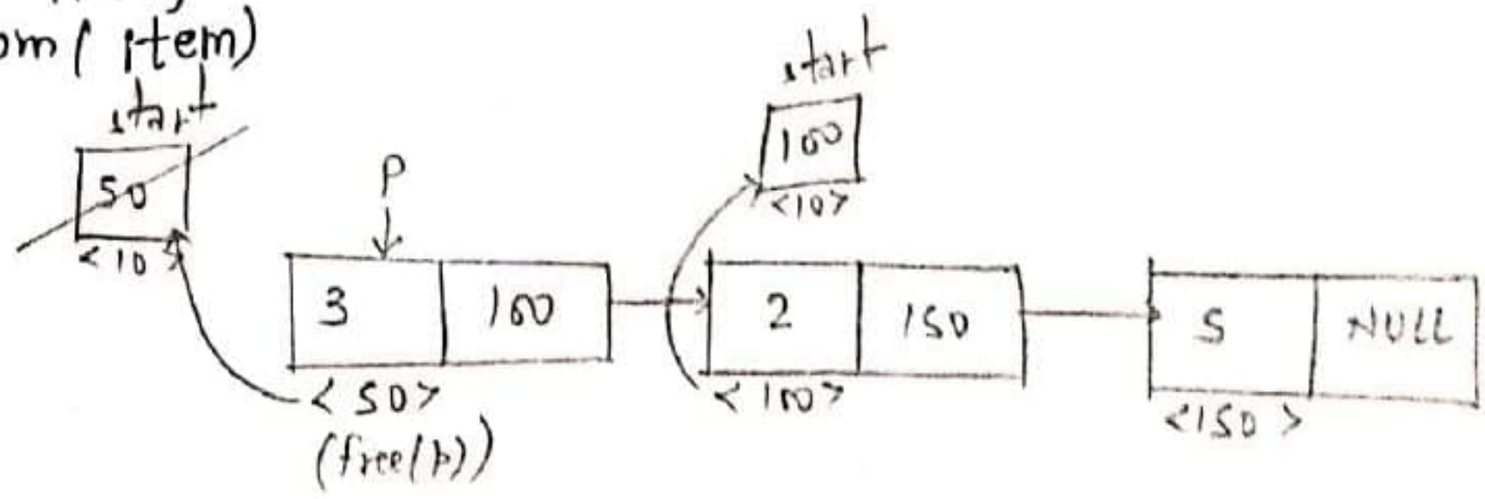
set start = next(p)

step ⑤ :- [Deallocate the memory from the deleted node]
free(p)

step ⑥ :- [Return the item]
return(item)

step ⑦ :- **stop.**

Explanation :-

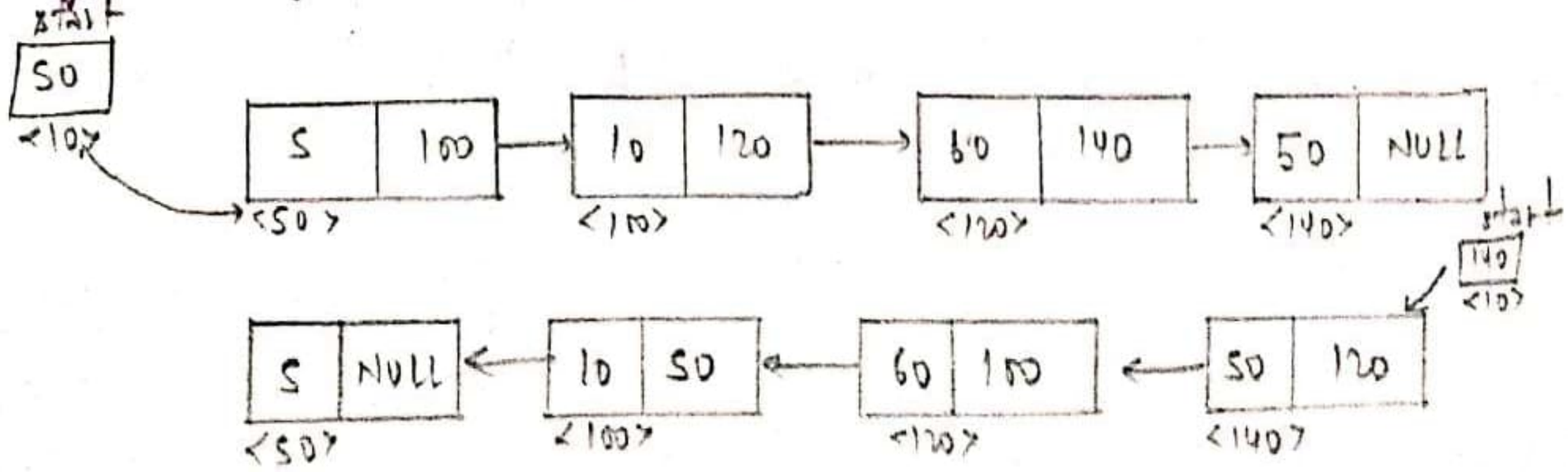


चूंकि पहला element है, → तो वो ही पहले pop होगा। तो link list में Deletion in Begining	3
	2
	5

* func^n for pop oper^n in stack using linked list :- (same as Delet^n at the Begining)

```
struct node
{
  int info;
  struct node * next;
};
start node * start;
start = NULL;
int stack_pop(struct node **);
  stack_pop(&start);
int stack_pop(struct node ** q);
{
  struct node *p;
  int k;
  p = *q;
  if (p == NULL)
  {
    printf("stack is empty");
    return;
  }
  else
    k = p->info;
    printf("\n");
    *q = p->next;
    free(p);
    return(k);
}
```

* Algorithm for Reverse of a linked list :-



Downloaded from TwoWaits

Scanned by CamScanner

step ① :- [Initialise the variable]
set temp = start and prev = NULL
[Declare H]

step ② :- [Find the address of each successive node and H's previous node]
[Repeat step ③ until temp is not equal to NULL]
temp != NULL'

step ③ :- [find the address of next & previous node]
H = prev
prev = temp
temp = next(temp)
next(prev) = H

step ④ :- [set pointer start at the begining of Reversed linked list]
set start = prev

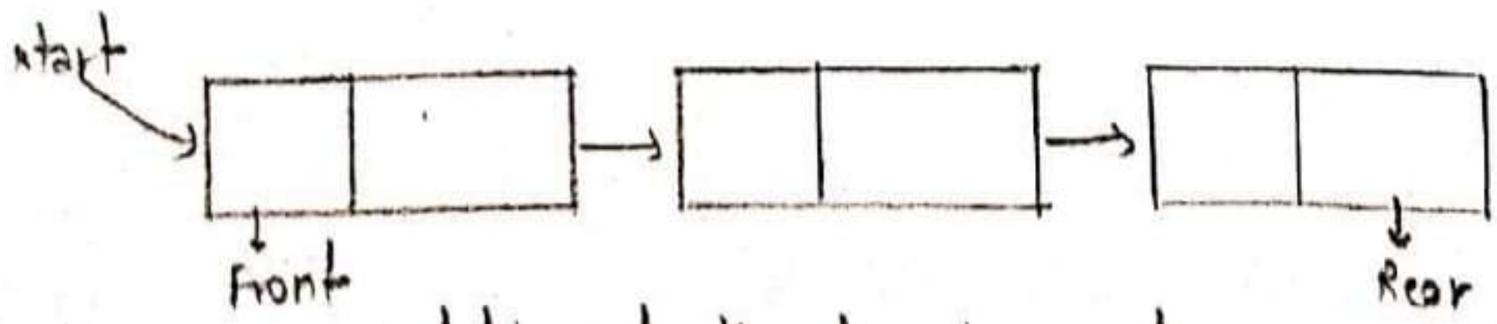
step ⑤ :- STOP. (इसमें '3' pointer variables लेते हैं)

* funcⁿ for Reversing the linked list :-

```
struct node
{
  int info;
  struct node * next;
};
void reverse (struct node **);
struct node * start;
  start = NULL;
reverse (& start);
void reverse (struct node **q)
{ struct node * temp, * prev, * H;
  temp = *q;
  prev = NULL;
  while (temp != NULL)
  {
    H = prev;
    prev = temp;
    temp = temp -> next;
    prev -> next = H
  }
  *q = prev;
}
```

* Implementation of Queue using linear link list :-

* ये मान कर चल रहे हैं, कि पहली Node front है, और Rear last Node है।



* Same as Deletion at the begining and Insertⁿ at the end in linked list.

Downloaded from TwoWaits

Scanned by CamScanner

* Algorithm for insertion in Queue using linked list... (same as insertn at the end)

let 'start' is the pointer variable which points the first node in the list.

step ① :- [Initialize the variable]
set temp = start

step ② :- [Find the address of the last node in the list]
[Repeat step ③ until temp is not equal to NULL]
temp != NULL

step ③ :- [Find address of next node]
set temp = next (temp)

step ④ :- [Get a node]
set, p = getnode()
if, p == NULL then print "Memory allocn is unsuccessful" and return.

step ⑤ :- [Write down inform into New Node]
info(p) = item

step ⑥ - [Insert the node at the end of the link list]
next (temp) = p and next (p) = NULL

step ⑦ :- STOP.

* Funcn for insertn in Queue using linked list :- (same as insertn at the end)

```
struct node
{
    int info;
    struct node * next;
};
void insert_queue (struct node**);
struct node * start;
start = NULL;
insert_queue (&start);
void insert_queue (structnode **q)
{
    struct node * temp, *p;
    temp = *q;
    while (temp -> next != NULL)
        temp = temp -> next;
        p = getnode ();
        if (p != NULL)
        {
        printf (" Enter informn for new node");
        scanf (" %d", &p -> info);
        p -> next = temp -> next;
        temp -> next = p;
        p -> next = NULL;
        }
}
```

* Algorithm for deletion in Queue using linked list :-
* Funcn for Deletion in Queue using linked list :-

} same as Deletion at the Begining.

Downloaded from TwoWaits

Scanned by CamScanner

* Algo^m for insertion in sorted linked list...

```
if (b==NULL)
{
 printf(" overflow \n");
 return;
}
printf(" Input new node");
scanf("%d", &b->info);
temp=start;
 while (temp->info < b->info)
 {
   r=temp;
  temp=temp->next;
 }
 r->next=b;
  b->next=temp;
}
```

for checking inform^n, { temp != NULL, item > temp(info)

for establishment Relation { (prev) = p, next(b) = temp

* func^n for insertion in sorted linked list :-

```
struct node
{
 int info;
 struct node * next;
};
void insert_sorted(struct node **);
struct node * start;
 start = NULL;
 insert_sorted(&start);
void insert_sorted(struct node ** q)
{
 struct node * temp, * prev, *u;
 temp= *q;
```

Downloaded from TwoWaits

Scanned by CamScanner

ADVANTAGES OF LINKED LIST :-

1. **Dynamic Data structure** :- link list can grow or shrink during the execution of a program. It means the size of a link list is not fixed at a time of compilation.
2. Insert^n and Deletion of nodes are Easier and efficient.
3. Accessibility of nodes in forward direct^n is easier.
4. **Non-contigeous Memory Blocks** :- there is no contigeous Memory alloc^n like Array. Data can store in non-contigeous Memory Block.

DISADVANTAGES OF LINKED LIST :-

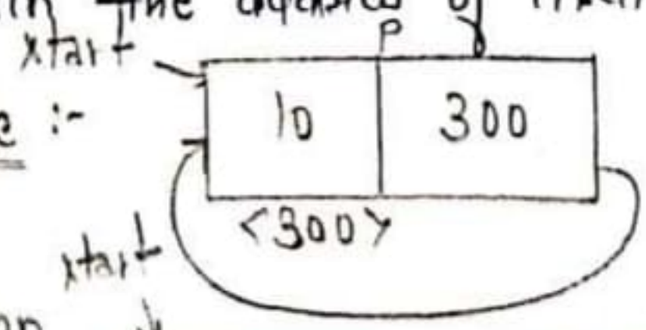
1. **Time consuming** :- Accessing a node which is present in last is time consuming compare to the node which is present in the start of the link list.
2. **More Memory** :- In the link list there is a special field known as **linked field or Next field** which holds the Address of Next node. So, link list requires extra space.

→ In Dynamic Memory alloc^n, we use malloc, calloc or Dealloc. Also, we use free() to dealloate the memory.

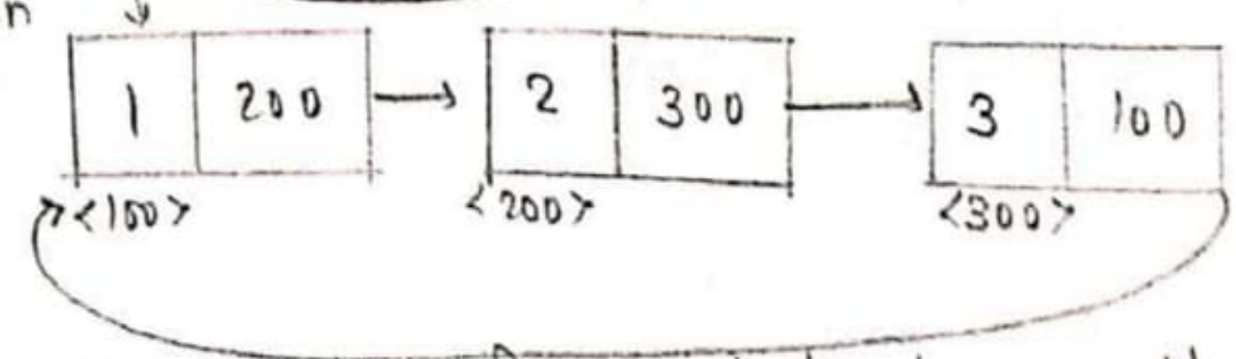
CIRCULAR LINKED LIST...

In the circular linked list last Node will contain the address of 1st Node insted of 'NULL' address.
In circular linked list we don't have Natural 1st Node and last node. One of the node address present in the circular link list must be store into external pointer variable (start), which can be used to access the entire circular link list, when only one node present in the circular link list then Next field of node will contain the address of itself.

→ for one Node :-



→ for more than one Node :-



→ अगर Circular link list में एक ही NODE है तो, वो अपना Address खुद ही Contain करेगा।

Downloaded from TwoWaits

Scanned by CamScanner

* Write algo^m and func^n for insert oper^n in circular link list :-

In the circular link list last node will contain the address of 1st Node, if external pointer variable 'start' will point to the last node of the circular link list, then implementation becomes easier.

step ① :- [Initialize the variable]
temp = start

step ② :- [Get a new node]
p = getnode ().
If, (p == NULL), then print "Memory alloc^n unsuccesful" and return.

step ③ :- [Write inform^n Data in the new node]
info (p) = item

step ④ :- [Insert this node at the front of linked list by adjusting 'p' and 'temp']. If, (temp == NULL)
then, next(p) = p
and start = p
otherwise, next (p) = next (temp)
next (temp) = p

* this algo^m is only for insert^n in Beg^n

FUNCTION...

```
struct node
{
   int info;
   struct node * next;
};
void insert_front_cc ( struct node **);
struct node * start;
start = NULL;
insert_front_cc ( & start);
void insert_front_cc (struct node **q);
{
   struct node * temp, *p;
   if p = getnode ();
      if (p == NULL)
      {
printf("Memory alloc$^n$ unsuccesful");
      return;
      }
```

* getnode() की Defin^n लिखना है।

```
printf("Enter inform$^n$ = ");
scanf("%d", &p->info);
if (temp == NULL)
{
   p->next = p;
   *q = p;
}
else
{
   p->next = temp->next;
   temp->next = p;
}
}
```

Downloaded from TwoWaits

Scanned by CamScanner