Unit 2 : LINEAR DATA STRUCTURE.

1. **Array**

   1. What is Array
   2. Types of array.
   3. Insert and delete element from array.
   4. Calculate address of 1-D array element
   5. Calculate address of 2-D array element.
   6. Representation of 2-D array(Row Major and Column Major order)
   7. Application of array.(Symbol manipulation ,Sparse matrix)

2. **Stack**

   1. What is Stack and define Operation on stack.
   2. Write algorithm of stack operation

      PUSH,POP, PEEP, CHANGE , DISPLAY
   3. WAP of stack operation.
   4. Explain Dynamic memory allocation.
   5. Difference between malloc() and calloc()
   6. Application of stack.
   7. Various types of notation and expression.

      Infix,prefix,postfix
   8. convert infix to postfix without stack
   9. convert infix to postfix in tabular form(using stack)
   10. Algorithm for conversion from infix to postfix.

11. WAP for for conversion from infix to postfix.
12. Evaluation of postfix notation in tabular form(using stack)
13. Algorithm for Evaluation of postfix notation in tabular form(using stack)
14. convert infix to prefix without stack.
15. convert infix to prefix in tabular form(using stack)
16. Algorithm for conversion from infix to prefix.
17. Evaluation of prefix notation in tabular form(using stack)
18. Algorithm for Evaluation of prefix notation in tabular form(using stack)
19. Recursion
20. Tower Of Hanoi

3. Queue

1. What is Queue and define Operation on Queue.
2. Define Types of queue.
3. Operation on Queue.
4. Write algorithm of Queue operation

    INSERT,DELETE,DISPLAY
5. WAP of Queue operation.
6. Limitation Of simple queue.
7. What is circular queue.
8. Write algorithm of circular Queue operation

    INSERT,DELETE,DISPLAY
9. WAP of circular Queue operation.
10. Difference between simple queue and circular queue.

11. Explain DEQUEUE(Double ended Queue)

12. Explain Priority Queue.

13. Write Application of queue.

14. Difference between stack and queue.

4. Linked List

1. Why we used linked list?

2. What is linked list.

3. Explain types of linked list.

4. Explain operation on link list.

5. Discuss advantages and disadvantages of linked list over array.

6. Explain operation on singly linked list.

1. Create singly linked list.
2. Traverse singly linked list.
3. Insertion into singly linked list.
   - At the beginning
   - At the end
   - After particular node.
   - In sorted.
4. Deletion from singly linked list.
   - At the beginning
   - At the end
   - Specific node.
   - Specific location.
5. Searching element into singly linked list
6. Count nodes in singly linked list.

7. Explain operation on Circular linked list.

1. Create circular linked list.
2. Traverse circular linked list.
3. Insertion into circular linked list.
   - At the beginning

- At the end
- After particular node

4. Deletion from circular linked list.

- .At the beginning
- At the end
- Specific node

8. Explain operation on Doubly linked list.

1. Create doubly linked list.
2. Traverse doubly linked list.
   I. Forward direction
   II. Backward direction
3. Insertion into doubly linked list.
   I.At the beginning
   II.At the end
   III. After specific node
   IV.In sorted list
4. Deletion from doubly linked list.
   I.From the beginning
   II.From the end
   III.From specific node

9. Application of linked list.

10. Implement stack using linked list.

11. Implement Queue using linked list

1. Array

1. What is Array

Ans:

- Array is a set of finite number of elements that have same data type .
- It means it contain one type of data only.
- Array stores its element in adjacent memory location.
- Ex .int a[5] it contains 5 integer type number.

A:	a[0]	a[1]	a[2]	a[3]	a[4]
Value:	10	20	33	40	30
Address:	100	102	104	106	108

❖ Characteristic of array

1. Array is non primitive and linear data structure.
2. Array is group of elements that have same data type.
3. Every elements has assign unique number which is called address of particular element.
4. Memory occupied by array is depending on data type and number of elements of array.
5. Array elements are stored sequentially or in linear fashion.
6. When array is declared and not initialized, it contain garbage values .If array is declared as static,all elements are initialized to zero.
7. In array insertion and deletion operation is slower and time consuming but searching and sorting operation is faster.

2. Types of array.

Ans:

- One Dimensional Array

- One dimensional array is an array having a single or one index.
- We give a name to the array and its elements are accessed by its index or subscript.

int a [2]
↑

- index range from 0 to size-1.It means if size=5 then array elements starts from a[0]- a[4]

- Declaration of 1-D array

- <u>Syntax:</u> Datatype arrayname[size];
- <u>Ex:</u> int A[5];

  double B[5*2],C[10]; // we declared array B with size 10 and in same line we declare

- ➢ Two dimensional array is an array having a two index .
- ➢ We give a name to the array and its elements are accessed by its index or subscript.

  int a [2] [3]

  $\uparrow$ $\uparrow$

- ➢ It is used in tabular form. So it arranging elements in the form of rows and columns.

- <u>Declaration of 2-D array</u>

- <u>Syntax:</u> Datatype arrayname[row][column];
- <u>Ex:</u> int A[5][3];
- ➢ Total size of an array = row * column
- ➢ Here array consider A as table and in this, table there is 5 rows and 3 columns.

- <u>Initialization of 2-D array :</u>

- ➢ a[2][3] = {1,2,3,4,5,6}
- ➢ a[2][3] = { {1,2,3} , {4,5,6} } //two row and each row contains 3 columns.

3. Insert and delete element from array.

Ans:

- <u>Insertion Operation</u>

- ➢ Insert a new element in array at particular position.

➢

Index	[0]	[1]	[2]	[3]
Value	8	5	6	3

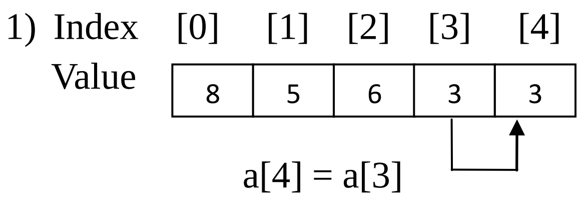
- ➢ Now if we want to insert a value 10 in 2nd position (index-1) then, after completion of this operation our array is,

- a[5] = { 8,10,5,6,3}
-

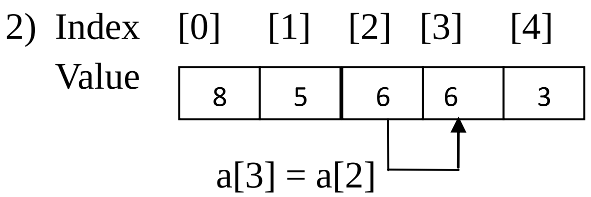
Index	[0]	[1]	[2]	[3]	[4]
Value	8	10	5	6	3

- In this operation shifting is performed. We have to shift a[1], a[2] and a[3] so that a[1] is free and we add 10 in a[1].

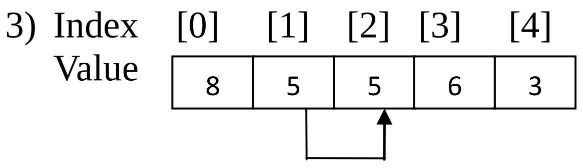
- Here, shifting start from last value of array so that overwrite is not done. So we have to shift 3 in next index(a[4]) then 6 in next index(a[3]) then 5 in next index(a[2]). After completion of shifting we add new value in a[1]. This operation graphically represented as below,



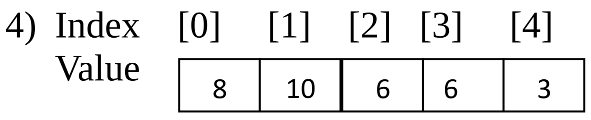
a[4] = a[3]



a[3] = a[2]



a[2] = a[1]



a[1] = 10 (Final array)

- <u>C program for Insertion operation.</u>

```
void main()
{
int a[15],i,n,pos,num;
clrscr();
printf("\nEnter how many numbers to be entered:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("Enter value:");
    scanf("%d",&a[i]);
}
printf("\nEnter new value:");
scanf("%d",&num);
printf("\nEnter position");
scanf("%d",&pos);
pos--;
for(i=n-1;i>=pos;i--)
{
    a[i +1]=a[i];
}
a[pos]=num;
printf("\nNew array\n");
for(i=0;i<=n;i++)
{
    printf("%d",a[i]);
}

getch();
}
```

- Output:

```
Enter how many numbers to be entered : 4
Enter value:8
Enter value:5
Enter value:6
Enter value:3
Enter new value:10
Enter position:2
8
10
5
6
3
```

- **<u>Deletion Operation of Array.</u>**

➢ Delete/Remove a element from array at particular position.

➢ a[5]={8,5,6,3,4}

Index	[0]	[1]	[2]	[3]	[4]
Value	8	5	6	3	4

➢ Now if we want to delete a from 2 position then, after completion of this operation our array is,

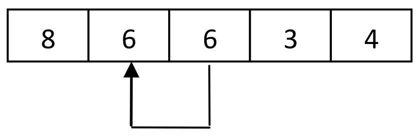
➢ a[5] = { 8,6,3,4}

Index	[0]	[1]	[2]	[3]
Value	8	6	3	4

➢ In this operation shifting is performed. We have to shift a[2],a[3] and a[4].

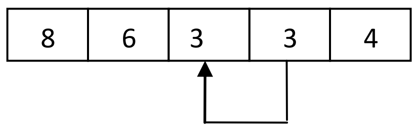
➢ Here, shifting start with value after the deleted value. So we have to shift 6 in previous index(a[1]) then 3 in previous index(a[2]) then 4 in previous index(a[3]).

1) Index [0] [1] [2] [3] [4]
   Value



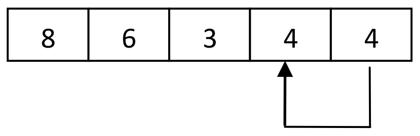
a[1] = a[2]

2) Index [0] [1] [2] [3] [4]
   Value



a[2] = a[3]

3) Index [0] [1] [2] [3] [4]
   Value



a[1] = a[2]

4) Index [0] [1] [2] [3]
   Value

Index	[0]	[1]	[2]	[3]
Value	8	6	3	4

- <u>C program for Deletion Operation.</u>

```
void main()
{
int a[15],i,n,pos;
clrscr();
printf("\nEnter how many numbers to be entered:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("Enter value:");
    scanf("%d",&a[i]);
}
printf("\nEnter position");
scanf("%d",&pos);
pos--;
for(i=pos;i<n-1;i++)
{
a[i]=a[i +1];
}
printf("\nNew array\n");
for(i=0;i<n-1;i++)
{
printf("%d",a[i]);
}
getch();
}
```

- Output:

```
Enter how many numbers to be entered : 4
Enter value:8
Enter value:5
Enter value:6
Enter value:3
Enter position:2
8
6
3
```

4. Calculate address of 1-D array element.

Ans: Find address of A[4]

- Ex: .int a[10] Base Adress (L0)=: 200.

A:	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Value:	10	20	33	40	30	10	20	33	40	30
Address:	200	202	204	206	208	210	212	214	216	218

Loc(A[i]) = L0 +size * i
=200 + 2 *4
=208

5. Representation of 2-D array(Row Major and Column Major order)array element.

Ans:

- All elements of array get stored in linear fashion .
- Two ways in which elements can be represent in computer's memory.
  1.Row major order.
  2.Column major order

<u>1.Row major order</u>

- In row major implementation memory location is done row wise.
- It means elements of array are stored in memory row by row.
- i.e.complete first row is stored,then complete second row is stored and so on.
- Ex: a[3][3] is stored in memory.

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]	a[2][0]	a[2][1]	a[2][2]

← Row1 → ← Row2 → ← Row3 ←

- Ex: int a[3][3]={{1,2,3},{4,5,6},{7,8,9}}

$$\begin{array}{c|ccc} & C0 & C1 & C2 \\ \hline R0 & 1 & 2 & 3 \\ R1 & 4 & 5 & 6 \\ R2 & 7 & 8 & 9 \end{array}$$

- Array stored in memory:-

1	2	3	4	5	6	7	8	9

2.Column major order

- In Column major implementation memory location is done Column wise.
- It means elements of array are stored in memory Column by Column.
- i.e.complete first Column is stored, then complete second Column is stored and so on.
- Ex: a[3][3] is stored in memory.

a[0][0]	a[1][0]	a[2][0]	a[0][1]	a[1][1]	a[2][1]	a[0][2]	a[1][2]	a[2][2]

$\longleftarrow$ Column1 $\longrightarrow$ $\longleftarrow$ Column2 $\longrightarrow$ $\longleftarrow$ Column3 $\longleftarrow$

- Ex: int a[3][3]={{1,2,3},{4,5,6},{7,8,9}}

	C0	C1	C2
R0	1	2	3
R1	4	5	6
R2	7	8	9

- Array stored in memory:-

1	4	7	2	5	8	3	6	9

6. Calculate address of 2-D array element in row major order.

Ans: Find address of A[3][3]

- Ex: .int a[4][4] Base Adress (L0)=: 200.

Loc(A[i][j]) = L0 +(row *Total no of columns +column)*size
=200 +(3 *4+3) *2
=200+(15*2)
=230

	0	1	2	3
0	200	202	204	206
1	208	210	212	214
2	216	218	220	222
3	224	226	228	230

7. Calculate address of 2-D array element in column major order.

Ans: Find address of A[0][2]

- Ex: .int a[4][4] Base Adress (L0)=: 200.

Loc(A[i][j]) = L0 +(column *Total no of rows +row)*size
Loc(A[0][2])=200 +(2 *4+0) *2
=200+(8*2)

	0	1	2	3
0	200	208	216	224
1	202	210	218	226
2	204	212	220	228
3	206	214	222	230

=216

8. Application of Array.

Ans: 1.Symbol Manipulation(matrix representation of polynomial equation)
2.Sparse matrix

- Symbol Manipulation

➢ We can use array for different kind of operations in polynomial equation such as addition, subtraction, division etc..

➢ We are interested in finding suitable representation for polynomial so that different operations like addition,subtraction etc.. can be performed in efficient manner.

➢ matrix representation of polynomial equation

	Y	Y^2	Y^3	Y^4
X	XY	XY^2	XY^3	XY^4
X^2	X^2Y	X^2Y^2	X^2Y^3	X^2Y^4
X^3	X^3Y	X^3Y^2	X^3Y^3	X^3Y^4
X^4	X^4Y	X^4Y^2	X^4Y^3	X^4Y^4

- ex : $2x^2+5xy+y^2$

		Y	Y^2	Y^3	Y^4
	0	0	1	0	0
X	0	5	0	0	0
X^2	2	0	0	0	0
X^3	0	0	0	0	0
X^4	0	0	0	0	0

➢ Once we have algorithm for converting the polynomial equation to array representation and another algorithm for converting array to polynomial equation ,then different operations in array will be corresponding operations of polynomial equation.

- Sparse matrix.

- The matrix in which many elements are zero is known as sparse matrix.
- The matrix that is not a sparse matrix is called dense matrix.
- We can device a simple representation scheme whose space requirement equals the size of the non zero elements.
- If we use 2D array for sparse matrix then it waste lots of space .
- Ex: 100 ×100 =10000 elements
- In that only 10 non zero elements are presents.
- To access 10 non zero element scanning for 10000 times.It is tedious task.
- So instead of storing zero with non zero elements we only store non zero elements.
- Two methods for this representation

    - 1.Triplet representation
    - 2. Linked representation.

- <u>1.Triplet representation</u>
- We use three field Row ,Column,Value
- EX:5×6 array Non zero elements:=6

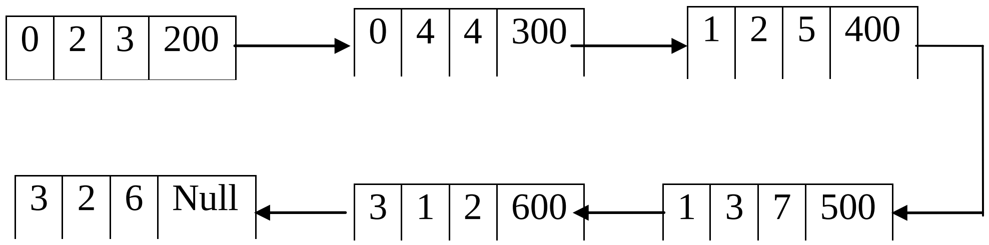
$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 1 & 0 & 8 & 0 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 5 \\ 4 & 0 & 0 & 2 & 0 & 0 & 0 \end{array} \Rightarrow$$

Rows	**Columns**	**Values**
5	**6**	**6**
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

- <u>2.Linked List representation</u>
- We use node to represent non zero elements.
- Each node has four fields.
- Row,Column,Value,Next node address.
- Row: Index of row where non zero element is located.
- Column:Ondex of column where non zero element is located.
- Value:Value of non zero
- Next node address:address of next node.

	0	1	2	3	4	5
0	0	0	0	0	9	0
1	0	8	0	0	0	0
2	4	0	0	2	0	0
3	0	0	0	0	0	5
4	0	0	2	0	0	0

Row	Column	Value	Address of next node

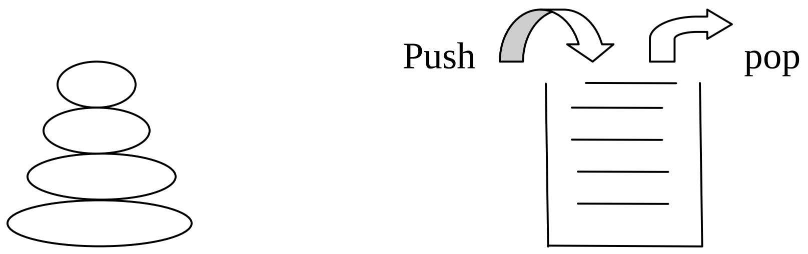


2. STACK

1.What is Stack and define operation on stack.

Ans:

- A stack is a non primitive data structure ,it is ordered list in which insertion and deletion of data is done from only one end known as TOP of stack.
- Last element inserted will be on top of stack.
- Since deletion is done from same end, last element inserted will be first to be removed out from stack and so on.
- So it is also called LIFO (Last In First Out).
- Ex: stack of plates on counter, during time of dinner customers take plates from top of the stack and waiter puts washed plates on top of stack.



Stack representation

7. Operation on stack.

- There are two main operation on stack.
  1.Push
  2.Pop

- 1.Push
  - Insertion in stack is known as PUSH .
  - Push means to add (insert) element at top of stack.
  - Suppose all the elements are added in stack it means if stack is full and when we try to insert one more element it is not possible to insert any new element.
  - This situation is called stack overflow condition.
  - In push top pointer is incremented by 1.

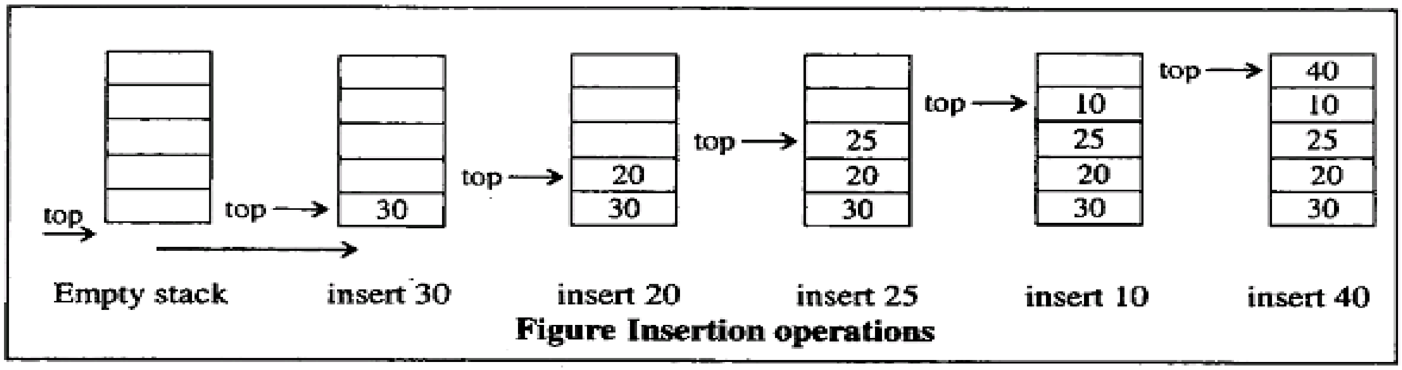
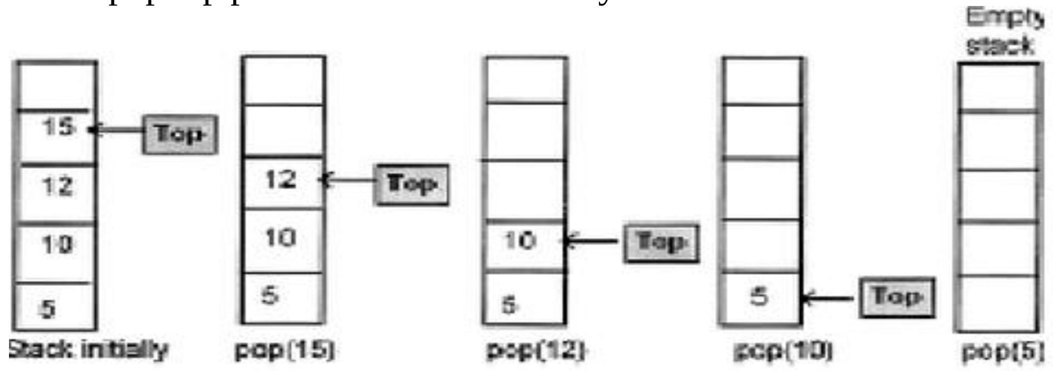


Fig. Insertion Operations

- 2.Pop
  - ➢ Deletion from stack is known as POP.
  - ➢ Pop means to delete (remove) element from top of stack.
  - ➢ When all the elements are deleted, top points to bottom of stack .
  - ➢ When stack is empty it is not possible to deleted any element and this situation is called stack underflow.
  - ➢ In pop top pointer is decremented by 1



2. **Write Algorithm of Stack operation(PUSH,POP,PEEP,CHANGE)**

Ans:

- **PUSH:**

```
    Initially top= -1
    PUSH(s,top,X)
Step 1:     [Check for overflow]
            If top>=N-1 then
            Write "stack overflow"
            Exit
Step 2:     [increment top pointer ]
```

```
            top←top+1
Step 3:     [perform insertion]
            S[top]=X
Step 4:     [finished]
            Exit
```

- **POP**

➢ Initially top=-1

```
POP(s,top)
Step 1:     [Check for underflow]
            If top<0 then
            Write "stack underflow"
            Exit
Step 2:     [return top element]
            X=S[top]
            Write X
Step 3:   [decrement top pointer ]
            top←top-1
Step 4:     [finished]
            exit
```

- **PEEP**

➢ PEEP it means to extract element or to display particular element.

➢ This operation returns value of ith element from top of the stack.

➢ Initially top=-1

```
PEEP(s,top,X)
Step 1:     [Check for underflow]
            If top-i+1<=0 then
            Write "stack underflow"
            Exit
Step 2:     [return ith element element]
            X=S[top-i]
            Write X
Step 3:     [finished]
            exit
```

- **CHANGE**

- CHANGE it means user can change content of i^{th} element from top of the stack.
- Initially top=-1

```
CHANGE(s,top,X)
Step 1:    [Check for underflow]
           If top-i+1<0 then
           Write "stack underflow"
           Exit
Step 2:    [Change element value from top of stack]
           S[top-i]←X

Step 3:    [finished]
           exit
```

3.Write Implementation of Stack

Ans:

```
void main()
{
        int s[3],no,n,ch,top=-1,i,x,y;
       printf("\n\nEnter the size of stack:-> ");
       scanf("%d",&n);
       printf("1:PUSH \n");
       printf("2:POP \n");
       printf("3:PEEP \n");
       printf("4:UPDATE \n");
       printf("5:DISPLAY \n");
       do
       {
                printf("\n\nEnter the choice:-> ");
                scanf("%d",&ch);
               switch(ch)
               {
                       case 1 :
                               if(top>=n-1)
                               {
                                       printf("\n\n------------STACK OVERFLOW-------------");
                                       break;
```

```
          }
          printf("\n\nEnter the element:-> ");
          scanf("%d",&no);
           top=top+1;
            s[top]=no;
break;
case 2:
           if(top==-1)
          {
                    printf("\n\n------------------STACK UNDERFLOW-------------");
                    break;
          }
           y=s[top];
          top=top-1;
           printf("\n\nDeleted Element is :-> %d",y);
break;
case 3:
          printf("\n\nEnter Position For PEEP :-> ");
           scanf("%d",&i);
          if(top-i+1<0)
          {
                    printf("\n\n--------STACK UNDERFLOW ON PEEP----------");
                    break;
          }
          y=s[top-i];
          printf("\n\nElement is :-> %d",y);
          break;
case 4:
          printf("\n\n Enter Position For Updation :->");
          scanf("%d",&i);
          if(top-i+1<0)
           {
                    printf("\n\n-----STACK UNDERFLOW ON UPDATION-----");
                    break;
          }
          printf("\n\nEnter Element U Want to Update :-> ");
          scanf("%d",&x);
           s[top-i] = x;
 break;
case 5:
```

```
                                        printf("\n\nElements of stack are :->\n");
                                        printf("\n\n---------------------------\n");
                                        for(i=0;i<=top;i++)
                                        {
                                                printf("%d \n",s[i]);
                                        }
                                         printf("---------------------------");
                                break;
                        }
                }while(ch>=1 && ch<=5);
        }
```

4.Explain Dynamic memory allocation.

Ans:

- ➢ Memory allocation is the process of giving space for variables. There are two basic types of memory allocation:
- ➢ static and dynamic memory allocation.
- ➢ Static memory allocation is the allocation of memory at compile time before the associated program is executed,
- ➢ The process of allocating memory at runtime is known as dynamic memory allocation
- ➢ In C, the exact size of array is unknown until compile time, i.e., the time when a compiler compiles your code into a computer understandable language. So, sometimes the size of the array can be insufficient or more than required.
- ➢ Dynamic memory allocation allows your program to obtain more memory space while running, or to release it if it's not required.
- ➢ So, Dynamic memory allocation allows you to manually handle memory space for your program.
- ➢ "memory management functions" are used for allocating and freeing memory during execution of a program. These functions are defined in stdlib.h.

Function	Use
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	allocates multiple blocks of memory each of same size returns a pointer to memory
free()	deallocate the previously allocated space

realloc()	Change the size of previously allocated space

- **malloc()**

  - The name malloc stands for "memory allocation".
  - The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.

Syntax of malloc()

pointer variable = (cast-type*) malloc(byte-size)

  - Here, ptr is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.
  - ptr = (int*) malloc(100 * sizeof(int));
  - This statement will allocate 200 according to size of int 2 bytes and the pointer points to the address of first byte of memory.

- **calloc()**

  - The name calloc stands for "contiguous allocation".
  - The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax of calloc()

pointer variable = (cast-type*)calloc(n, element-size);

  - This statement will allocate contiguous space in memory for an array of *n* elements. For example:
  - ptr = (float*) calloc(25, sizeof(float));
  - This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e, 4 bytes.

- **free()**

  - Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on its own. You must explicitly use free() to release the space.

syntax of free()

free(ptr);

- This statement frees the space allocated in the memory pointed by ptr.

- **realloc()**

- If the previously allocated memory is insufficient or more than required, you can change the previously allocated memory size using realloc().

Syntax of realloc()

ptr = realloc(ptr, newsize);

- Here, *ptr* is reallocated with size of newsize.

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using malloc() function.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int num, i, *ptr, sum = 0;

    printf("Enter number of elements: ");
    scanf("%d", &num);

    ptr = (int*) malloc(num * sizeof(int));  //memory allocated using malloc

    printf("Enter elements of array: ");
    for(i = 0; i < num; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }
    printf("Sum = %d", sum);
    free(ptr);
    }
```

5.Difference between malloc() and calloc()

Ans:

Malloc	**calloc**
The name malloc stands for memory allocation.	The name calloc stands for contiguous allocation.
malloc() allocates single block of memory	calloc() allocates multiple blocks of memory each of same size
malloc() takes one argument that is, number of bytes.	calloc() take two arguments those are: number of blocks and size of each block

syntax of malloc():
void *malloc(size_t n);
 	
Allocates n bytes of memory. If the allocation succeeds, a void pointer to the allocated memory is returned. Otherwise NULL is returned.
	syntax of calloc():
	void *calloc(size_t n, size_t size);
	
	Allocates a contiguous block of memory large enough to hold n elements of size bytes each.

malloc is faster than calloc.	calloc takes little longer than malloc because of the extra step of initializing the allocated memory by zero.
malloc() does not initialize the memory allocated, it means it contains garbage value	while calloc() initializes the allocated memory to ZERO.
ptr = (cast-type*) malloc(byte-size)	ptr = (float*) calloc(25, sizeof(float));

6.Application of stack.

Ans:

- Arithmetic expression like conversion from infix to postfix or prefix.
- Stack machine ,certain computers perform stack operation at the hardware or machine level.
- Representation of polish notations.
- To reverse string.
- It is used by compiler for parsing syntax of expression.
- In recursive functions.

7.Various types of notation and expression.(Infix,prefix,postfix)

Ans:

- Process of writing operators of expression either before their operands or after operands are called polish notation.
- 1.Infix 2.Prefix or polish notation 3.Postfix or reverse polish notation

- 1.Infix: operator placed between operands Ex: A + B
- 2.Prefix: Operator placed before operands Ex: +AB
- 3.Postfix:Operator placed after operands Ex: AB+

8. convert infix to postfix without stack.

Ans:

- Highest priority are converted first.
- Priority of operators:
  1. [ ] ,( )
  2. ^ or $ or ↑(right to left)
  3. * , / (left to right)
  4. + , - (left to right)

1 .A+(B*C)
=A+ (BC*)
Ans =ABC*+

2. A^B*C-D+E/F/(G+H)
=(AB^) * C –D + E / F / (GH+)
=(AB^C*) – D + (EF/) /(GH+)
=(AB^C*D-) + (EF/GH+/)
Ans =AB^C*D-EF/GH+/+

3. (A+B) *C Ans:AB+C*
4. (A+B) * (C-D) Ans:AB+CD-*
5 .(A+B) *C/D Ans:AB+C*D/
6. A+B/C-D Ans:ABC/+D-
7. A+(B*C-(D/E^F)*G)*H Ans: ABC+DEF^/G*-H*+

9. Algorithm for conversion from infix to postfix(RPN).

Ans:

- **POSTFIX(Q,P):**

Q:- It is arithmetic expr in infix notation
P:- It is equivalent postfix expression.

```
Step 1:   Push '(' into the stack and
          add ')' to the end of expression Q

Step 2:   Scan Q (expression) from left to right and repeat step 3 to 6 for each element of Q
          until stack is empty.

Step 3:   if operand is found add it to P

Step 4:   if left parenthesis '(' is found Push to stack.

Step 5:   If operator is found then
          (a) If the precedence of OP is less than or equal to the precedence of the symbol on
              the Top of the stack then
                  -  Pop from the stack ,add to P
                  -  Push OP to stack

          (b) If the precedence of OP is greater than the precedence of the symbol on the Top
              of the stack then
                  -  Push OP to stack

Step 6:   If right parenthesis ')' is found then repetedly POP from the stack and add to P
          until left parenthesis '(' is found.
```

10.convert infix to postfix in tabular form(using stack)

Ans:

1. A+B*C Ans: ABC*+

Input symbol	**stack**	**postfix**
A	(	A
+	(+	A
B	(+	AB
*	(+*	AB
C	(+ *	ABC
)	-	ABC *+

2. A+B-C+D-E Ans: AB+C-D+E-

Input symbol	**stack**	**postfix**
A	(	A
+	(+	A
B	(+	AB
-	(-	AB+
C	(-	AB+C
+	(+	AB+C-
D	(+	AB+C-D
-	(-	AB+C-D+
E	(-	AB+C-D+E
)		AB+C-D+E-

3. A+(B-C*D) -E Ans: ABCD*-+E-

Input symbol	**stack**	**postfix**
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
-	(+(-	AB
C	(+(-	ABC
*	(+(-*	ABC
D	(+(-*	ABCD

)	(+	ABCD*-
-	(-	ABCD*-+
E	(-	ABCD*-+E
)	(	ABCD*-+E-

4.X*Y-Z/P+(Q+R)*S Ans: XY*ZP/-QR+S*+

Input symbol	**stack**	**postfix**
X	(	X
*	(*	X
Y	(*	XY
-	(-	XY*
Z	(-	XY*Z
/	(-/	XY*Z
P	(-/	XY*ZP
+	(+	XY*ZP/-
(	(+(	XY*ZP/-
Q	(+(	XY*ZP/-Q
+	(+(+	XY*ZP/-Q
R	(+(+	XY*ZP/-QR
)	(+	XY*ZP/-QR+
*	(+*	XY*ZP/-QR+
S	(+*	XY*ZP/-QR+S
)	-	XY*ZP/-QR+S*+

5.A+(B*C-(D/E*F)*G)*H Ans: ABC*DE/F*G*-H*+

6.A+(B-(C+(D-E))) Ans: ABCDE-+-+

11. WAP for conversion from infix to postfix.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#define N 100
char S[N];
int top=-1;
```

```
void Push(char);
char Pop();
int Prec(char);
void main()
{
        char infix[100],postfix[100],x,ch;
        int i,j=0,len;
        clrscr();
        printf("\n \t Enter infix expression ");
        scanf("%s",infix);
        Push('(');
        strcat(infix,")");
        len=strlen(infix);
        for(i=0;i<=len-1;i++)
        {
                x=infix[i];
                if(isalnum(x))
                {
                        postfix[j]=x;
                        j++;
                }

                else if(x=='(')
                {
                        Push(x);
                }

                else if(x==')')
                {
                        do
                        {
                                ch=Pop();
                                if(ch!='(')
                                {
                                        postfix[j]=ch;
                                        j++;
                                }
                                else
                                {
                                        break;
                                }
                        }while(1);
                }

                else
                {
                        while(Prec(x)<=Prec(S[top]))
                        {
                                ch=Pop();
                                postfix[j]=ch;
                                j++;
                        }
                Push(x);
        }
    }
    postfix[j]='\0';
    printf("\n \t Postfix expression is %s \n",postfix);
        getch();
}

void Push(char a)
{
        if(top>=N-1)
        {
        printf("\n \t Stack overflow..........\n \t");
                return;
        }
        else
        {
                top=top+1;
                //printf("\n top = %d \n",top);
                S[top]=a;
        }
}

char Pop()
{
        char a;
        if(top==-1)
        {
```

```
                printf("\n \t Stack is underflow..........\n");

        }
        else
        {
                a=S[top];
                top=top-1;
                return a;
        }
        return a;
}

int Prec(char b)
{
        int k;
        switch(b)
        {
                case '+':
                case '-':
                k=1;
                break;

                case '*':
                case '/':
                k=2;
                break;

                case '^':
                k=3;
                break;

                case '(':
                k=0;
                break;
        }
        return k;
}
```

12.Algorithm for Evaluation of postfix notation in tabular form(using stack)

Ans:

- **EVALUATE POSTFIX(Q):**

Q:- It is arithmetic expr in infix notation

```
Step 1:     INITIALIZE Top =-1

Step 2:     Scan Q (expression) from left to right and repeat step 3  for each element of Q.

Step 3:     if Digit is found then push Digit to stack

            else If operator is found then
                op2= pop()
                op1= pop()
            select case(operator)
            {
                case '+' :
                ans<- op1+op2;
                case '-' :
                ans<- op1-op2;
                case '*' :
                ans<- op1*op2;
                case '/' :
                ans<- op1/op2;
            }
            push(ans)

    Step 4:Finished
```

13.Evaluation of postfix notation in tabular form(using stack)

Ans:

1) 934 * 8 + 4/- Ans: 4

		4		8		4		
	3	3	12	12	20	20	5	
9	9	9	9	9	9	9	9	4
9	3	4	*	8	+	4	/	-

2) 562 +*124/-+ Ans: 40.5

							4			
		2				2	2	0.5		
	6	6	8		1	1	1	1	0.5	
5	5	5	5	40	40	40	40	40	40	40.5
5	6	2	+	*	1	2	4	/	-	+

14.WAP for postfix evaluation .

Ans:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define N 100
char S[N];
int top=-1;
void push(int);
int pop();

void main()
{
        char P[100];
        int ans,x,a,b,i,j,len;
        printf("\n \t Enter infix expression ");
        scanf("%s",P);
        len=strlen(P);
        for(i=0;i<=len-1;i++)
        {
                x=P[i];
                if(isdigit(x))
                {       x=x-'0';
                        push(x);
                }

                else
                {
                        b=pop();
                        a=pop();
                switch(x)
                {
                case '+':
                        ans=a+b;
                        push(ans);
                        break;

                case '-':
                        ans=a-b;
                        push(ans);
                        break;
                case '*':
                        ans=a*b;
                        push(ans);
                        break;
                case '/':
                        ans=a/b;
                        push(ans);
                        break;
                }
        }

    }
    printf("%d",S[top]);
    }

    void push(int d)
```

```
{
        if(top>=N-1)
        {
                printf("\n \t Stack overflow..........\n \t");
                return;
        }
        else
        {
                top=top+1;
                //printf("\n top = %d \n",top);
                S[top]=d;
        }
}

int pop()
{
        int x;
        if(top==-1)
        {
                printf("\n \t Stack is underflow..........\n");

        }
        else
        {
                x=S[top];
                top=top-1;

        }
        return x;
}
```

15.convert infix to prefix without stack.

Ans:

1.A+B*C
=A+ (*BC)
Ans: =+A *BC

2.(A-B/C)*(D*E-F)
=(A- (/BC)) * ((*DE) – F)

=(A – P) * (Q - F) P=/BC,Q=*DE
=(-AP) * (-QF))
=*-AP-QF
Ans:= *-A/BC-*DEF

3.(A*B + (C/D)) –F Ans: - + * AB/CDF
4.A/B^C-D Ans: -/A^BCD

16.Algorithm for conversion from infix to prefix.

Ans:

- **PREFIX(Q,P):**

Q:- It is arithmetic expr in infix notation
P:- It is equivalent prefix expression.

Step 1: Reverse Input expression
Replace '(' with ')' and ')' with '('

Step 2: convert expression into prefix using same method(same algo.) as postfix
Step 3: reverse answer .

17.convert infix to prefix in tabular form(using stack)

Ans:
1. E -) D * C – B (+ A Ans: +A - - B * CDE
reverse string: E – (D * C – B) +A

Input symbol	**stack**	**postfix**
E	(-	E
-	(- (	E
(	(- (	E
D	(-(	ED
*	(- (*	ED
C	(-(*	EDC
-	(-(-	EDC*
B	(-(-	EDC*B
)	(-	EDC*B-
+	(+	EDC*B--
A	(+	EDC*B - - A

)		EDC*B - -A+

18.Algorithm for Evaluation of prefix.

- Ans: **<u>EVALUATE PREFIX(Q):</u>**

Q:- It is arithmetic expr in infix notation

```
Step 1:    INITIALIZE Top =-1

Step 2:    Scan Q (expression) from right to left and repeat step 3  for each element of Q.

Step 3:    if Digit is found then push Digit into stack

           else If operator is found then
                op1= pop()
                op2= pop()
           select case(operator)
           {
                case '+' :
                ans<- op1+op2;
                case '-' :
                ans<- op1-op2;
                case '*' :
                ans<- op1*op2;
                case '/' :
                ans<- op1/op2;
           }
           push(ans)

    Step 4:Finished
```

19.Evaluation of prefix notation in tabular form(using stack)

Ans:

1) -9/+*3484 Ans: 4

			3					
		4	4	12				
	8	8	8	8	20		9	
4	4	4	4	4	4	5	5	4
4	8	4	3	*	+	/	9	-

20. Recursion

Ans:

- Recursion is technique for defining function in terms of call itself.
- It is defined as "function call itself"
- Thus "chain of process occurs"
- There are two important condition that must be satisfied by any recursive procedure.
- 1.Each time procedure calls itself ,it must br "nearer" in some sense solution
- 2.there must be a decision criterion for stopping process.
- Ex: 1.To find factorial of given number

.

<u>1.Programme for factorial:</u>

```
int fact(int x);
void main()
{
        int i,n,a;
        clrscr();
        printf("enter value");
        scanf("%d",&n);
        a=fact(n);
        printf("%d",a);
        getch();
}
int fact(int x)
{
        if (x==1)
        return 1;
        else
        return x*fact(x-1);
}
```

- Output 1: enter value
  5
  Ans:120

  Output 2: enter value
  4
  Ans:24

Execution

fact(5);
5 * fact(4)
5 * 4 * fact(3)
5 * 4 * 3 * fact(2)
5 * 4 * 3 * 2 * fact(1)
5 * 4 * 3 * 2 * 1

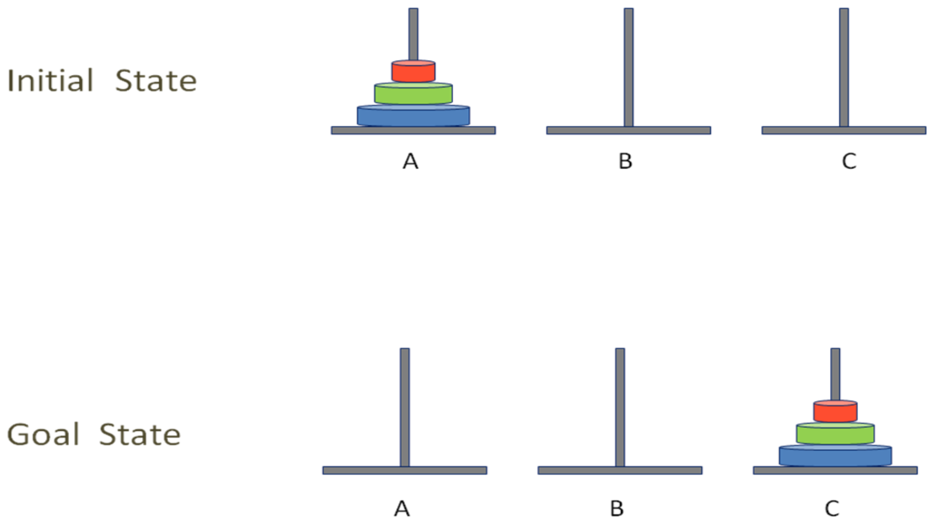
21. Difference between Recursion and Iteration

Ans:

Recursion	**Iteration**
If function call itself then it is known as recursion.	In iteration loops are used like for,while,do while
It is slower than iteration due to overhead of function call and return	It is faster.
It use more memory than iteratation	Ues less memoey
Ex: factorial,Tower of Hanoi	Ex: fiboanci

22.Tower Of Hanoi.

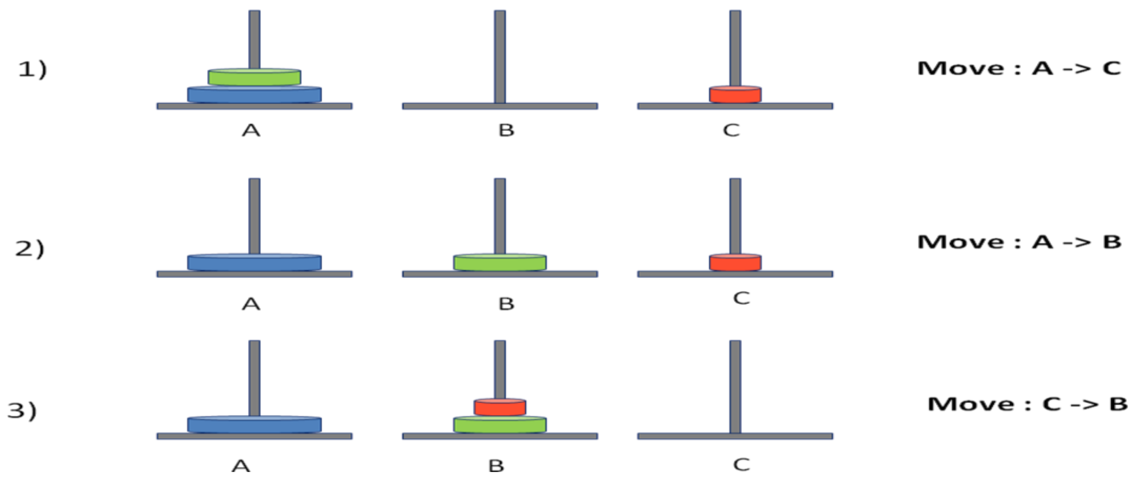
Ans:The tower of Hanoi of n disks is as follows

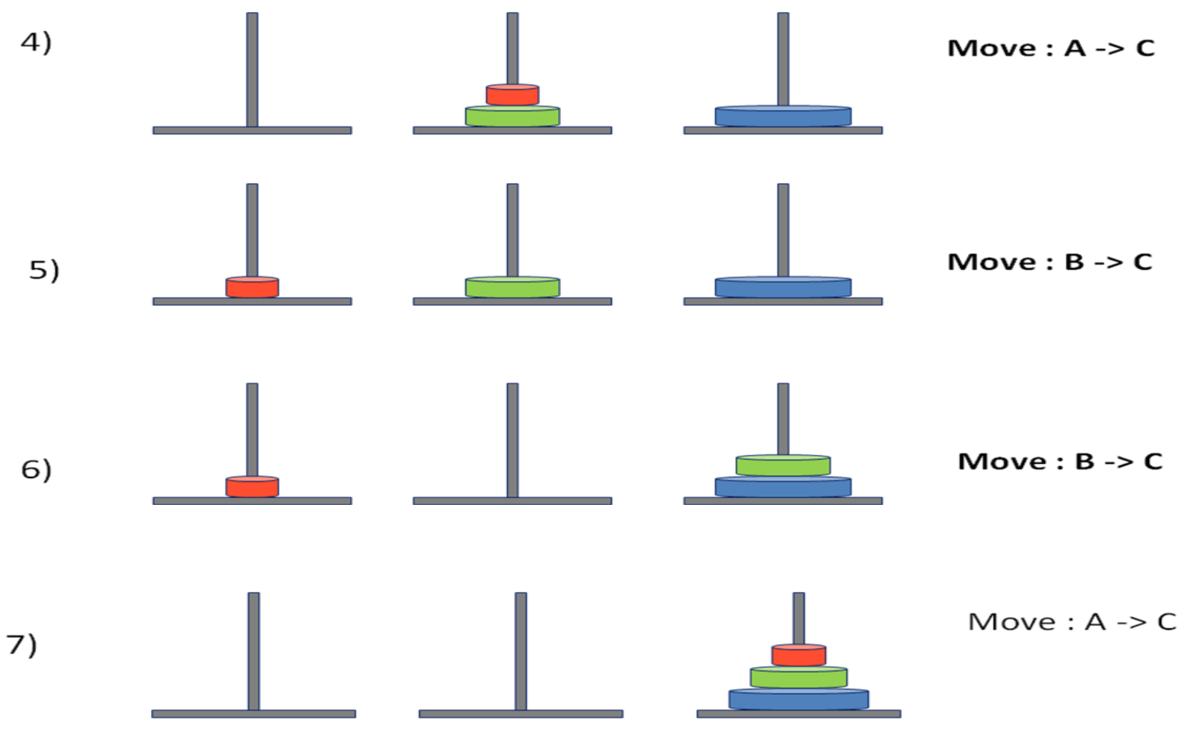


- We have n discs in Tower A. We need to move all n discs from Tower A to Tower C in such a way that smallest disc appears at the top of the Tower and Largest Disc appears at the bottom most position as shown in the figure.”
- We can use one extra Tower B.

- **Rules for TOH**

1) Only one disk can be moved at a time.
2) A larger Disc cannot be placed above a smaller Disc.
3) Only topmost Disc can be moved from any tower and it can be placed only on the top of the another Disc





No. Of Disks : n

Total number of movements : $2^n - 1$

Here, n = 3

So, Total number of movements : 7

Time Complexity = $O(2^n)$

<u>WAP for TOH</u>

```
#include<stdio.h>
#include<conio.h>
int count=0;
void TOH(int,char,char,char);
void main()
{
        int n;

        printf("Enter No. of Discs : ");
        scanf("%d",&n);
        TOH(n,'A','C','B');
        printf("\n\n TOTAL NO. OF DISC MOVEMENT : %d",count);
void TOH(int n, char fr, char to, char ax)
{
        if(n==1)
        {
                printf("\n Move disk 1 from %c to %c",fr,to);
                return;
        }
        TOH(n-1,fr,ax,to);
        printf("\n Move disk %d from %c to %c",fr,to);
        TOH(n-1,ax,to,fr);
}
```