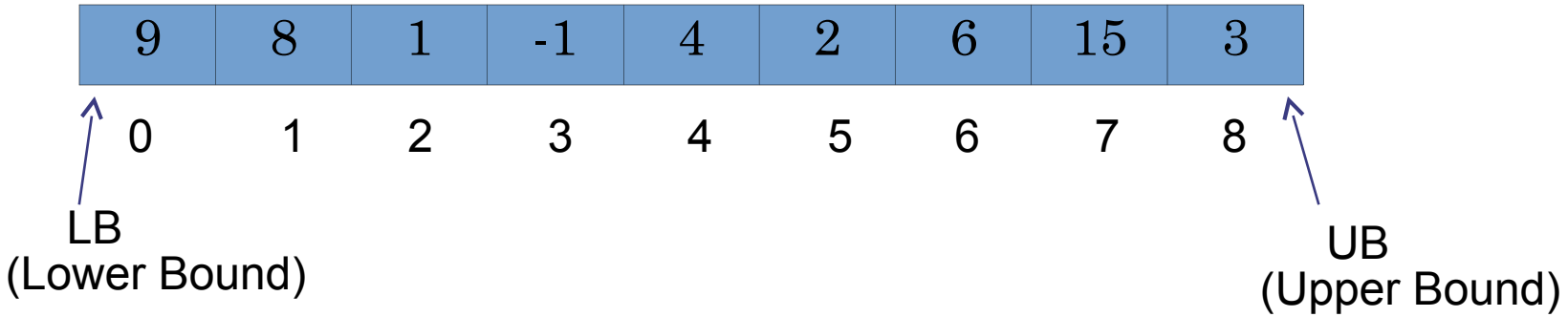
ARRAYS

INTRODUCTION

- An Array is a **Data Structure** with which we can perform operations on a collection of similar data type such as simple list or tables of information.

- All elements in the array are of same type i.e. int, char, float etc.

- The individual elements within the array can be accessed by the integer called ***Index.***
- eg-: int array[9]



- Length of an Array=UB-LB+1

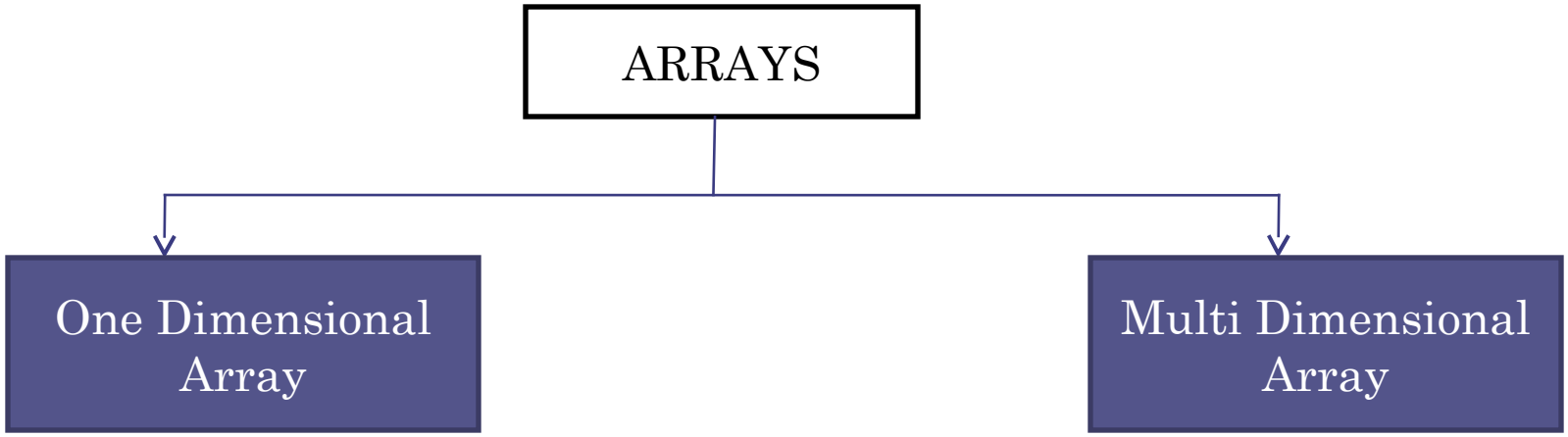
- The zeroth element (9) in the list can be accessed by array[0].

- An ***Index*** is also called a ***Subscript.***

- Therefore, individual elements of an Array are called ***subscripted*** variables eg. array[0], array[1] etc.

- Thus, an Array can be defined as a finite ordered collection of items of same type.

Types of Array



(An Array whose elements are specified by a single subscript
Eg. A[0])

(An Array whose elements are specified by two or more than two subscripts
Eg. A[10][10] is a two dimensional array)

One Dimensional Array-:

- 1-D Arrays are suitable for processing lists of items of identical types.

- They are very useful for problems that require the same operation to be performed on a group of data items.

Operations defined on 1-D Arrays-:

1. **TRAVERSAL-:**

   An Array can be traversed by visiting its element starting from the zeroth element to the last element in the list.

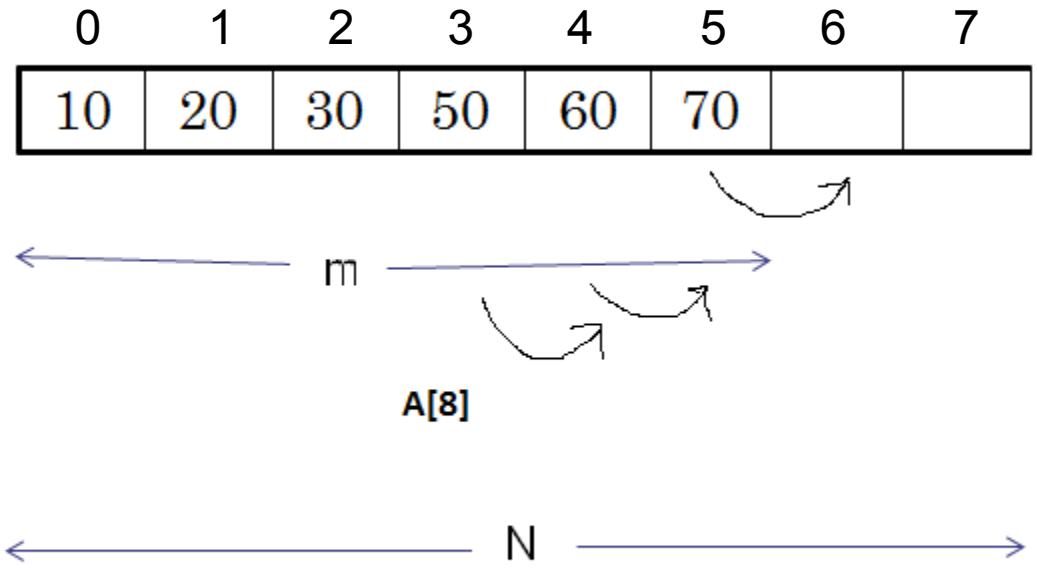
   Processing of the visited element can be done as per the requirement of the problem.

```
Algorithm Traversal (A [], N)
{
  //Description:- A is an array of size N.
    1. for (i=0 to N-1)
            1.1 Print A[i]
}

//Time Complexity-: O(N)
```

2. **INSERTION of N elements in an array**

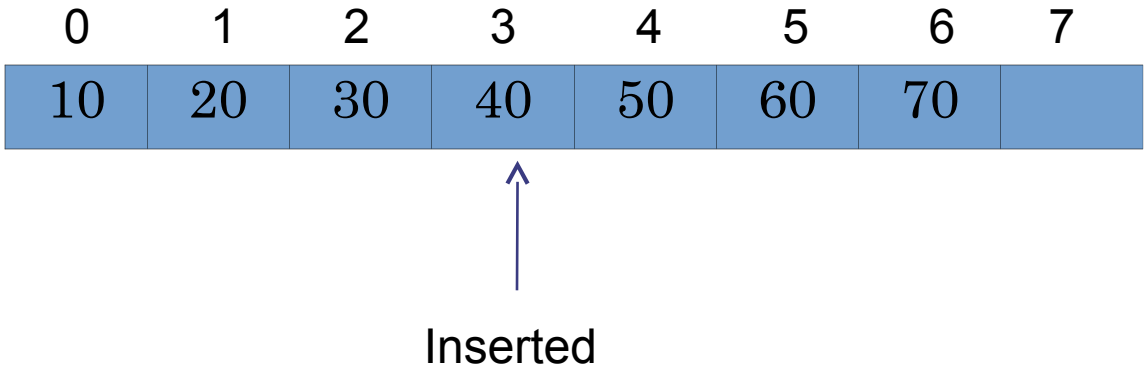
```
Algorithm Insertion (A [], N)
{
//Description:- N is the number of elements to
be inserted in array A.
    1. for( i=0 to N-1)
            1.1 Enter A[i]
}

//Time Complexity-: O(N)
```

3. INSERTION of an element at a particular position-:



Insert 40 at location 4th

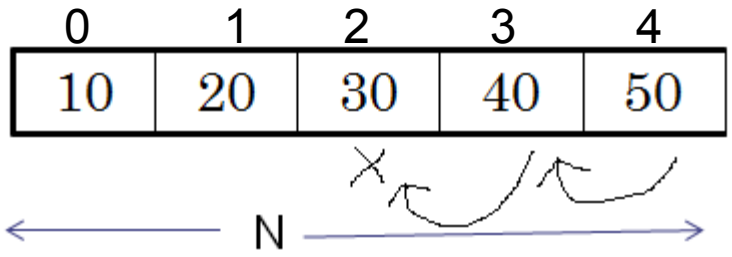
After Insertion-:



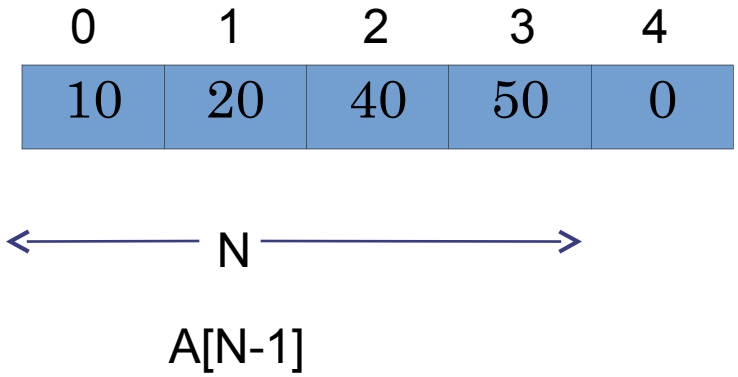
```
Algorithm InsertionAtPos(A[], N, val, loc)
{
  Description-: A is an array having N elements.
val is the value to be inserted at location loc.
  1. if (loc>N)
        4.1 Print Insertion Not Possible
        4.2 Exit
  2. for (i=N-1 to loc-1)
       2.1 A[i+1] = A[i]
  3. A[loc-1] = val
  4. N=N+1
}
```

//Time Complexity-: O(N)

4. Deletion of an element from an array-:



Delete an element at location 3rd

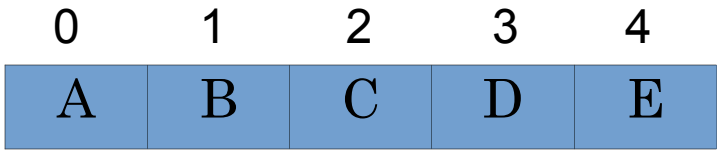


```
Algorithm Deletion(A[],N, loc)
{
  Description-: A is an Array of size N
                loc is the location of deletion.
1. if (loc>N)
        1.1 Print Deletion Not Possible
        1.2 Exit
2.for (i=loc to N-1)
       2.1 A[i-1]=A[i]
3. N=N-1
}
```

Time Complexity-: O(N)

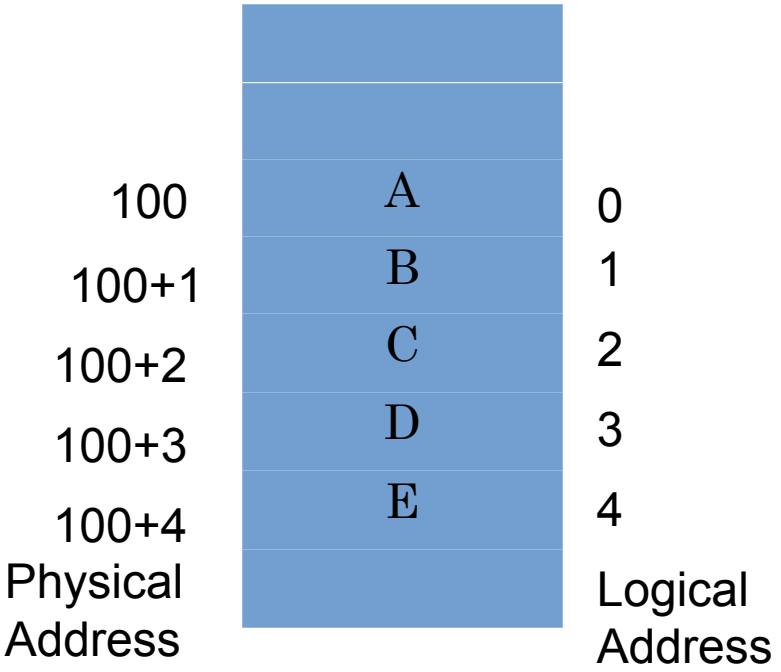
ADDRESS CALCULATION-:

- One Dimensional Array-:



List[5]

Logical View-:



◦The array elements are stored in contiguous memory locations by sequential allocation techniques.
◦The Address of ***ith*** element of the array can be obtained if we know-:

1. The starting address i.e. the address of the first element called ***Base Address*** denoted by **B**
2. The size of the element in the array denoted by **W**

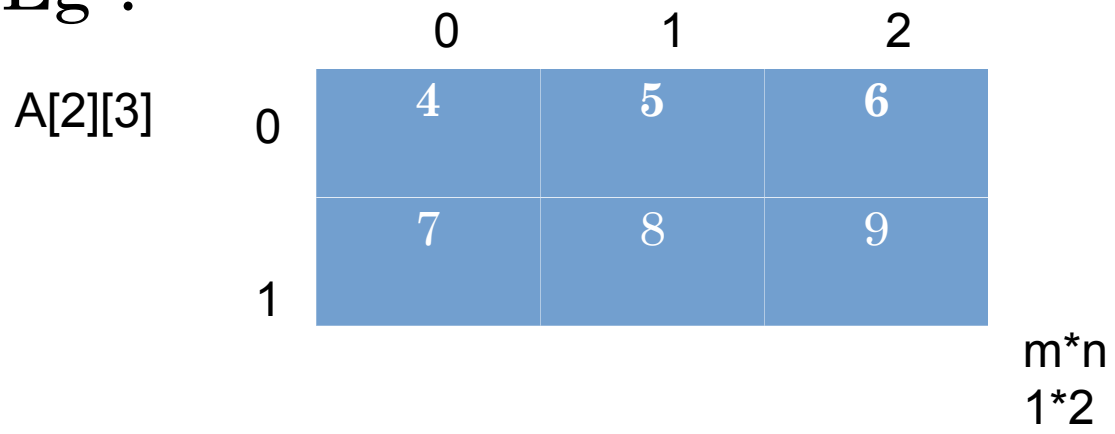
Address of List [i] = B + (i – LB) * W
where LB is Lower Bound of the array

List[5] = List[0---4] ; LB=0

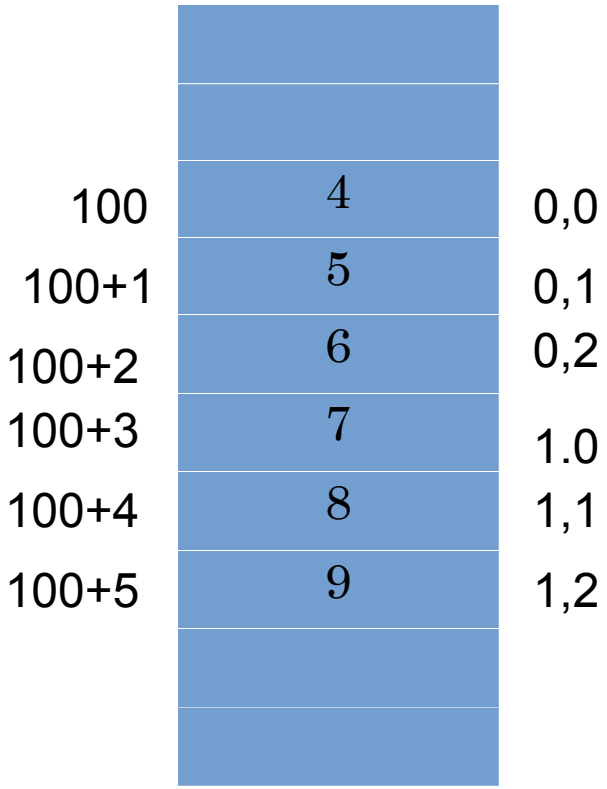
○Two Dimensional Array-:

A [m] [n] represents an array A where m is no.of rows and n is no.of columns.

Eg-:



Row Major Order-:



In row-major storage, a multidimensional array in linear memory is accessed such that rows are stored one after the other.

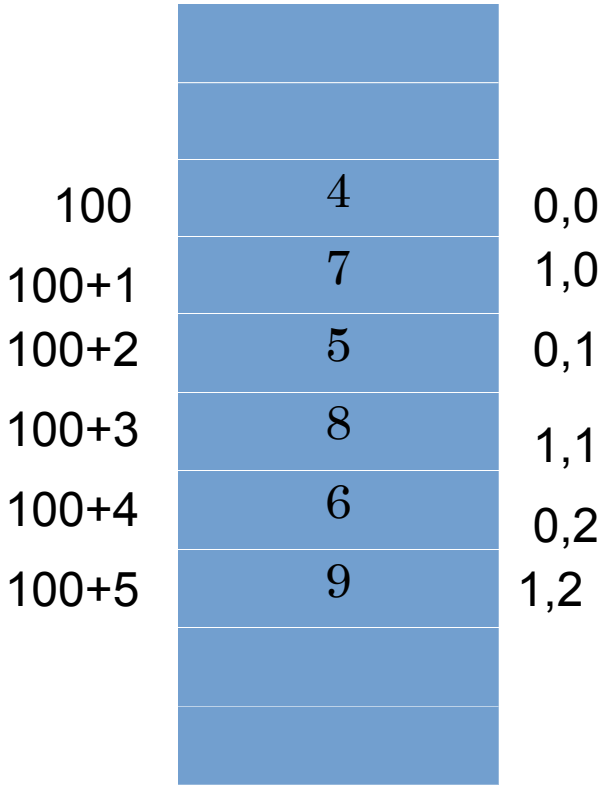
Address of A [i] [j]= B + [(i-LB_R)*n + (j-LB_C)]*W

where :

- LB_R is the Lower Bound of Rows
- LB_C is the Lower Bound of Columns
- W is the size of element
- n is the no.of columns
- Eg-: A[1][2]; m=2;n=3;LB_R=0;LB_C=0

A [1] [2] = 100 + ((1-0) * 3 + (2-0)) *1
= 105

Column Major Order-:



In column-major storage, a multidimensional array in linear memory is accessed such that columns are stored one after the other.

Address of A [i] [j] = B+ [(i-LB_R) + (j-LB_C)*m]*W

where

- LB_R is the Lower Bound of Rows
- LB_C is the Lower Bound of Columns
- W is the size of element
- m is the no.of rows