Data Structure

A mathematical and logical model of data is known as *Data Structure*.

Primitive data structure: The data structure, which is available in the compiler, is known as a *primitive data structure*. Example: Array
Non-primitive data structure: The data structure, which is not available in the compiler, is known as *non-primitive data structure*. Examples: Stack, Queue, Linked List

Linear Data Structure: The data structure in which each element can access maximum one predecessor element and one successor element is known as *linear data structure*. Example: Stack, Queue etc.
Non-linear Data Structure: The data structure in which each element can access any number of predecessor elements and any number of successor elements is known as *Non-linear data structure*. Example: Tree, Graphs, etc.

Static Data Structure: The data structure in which the number of elements is fixed, is known as *Static Data Structure*. Example: Arrays
Dynamic Data Structure: The data structure in which the number of elements is not fixed, is known as *Dynamic Data Structure*. Example: Linked List.

Array

It is a *static primitive data structure*. It is a homogeneous collection of data. The elements in the array are stored on consecutive memory locations. Array is also known as a subscripted variable, e.g., **A[i]** is **i**th element of the array **A** (i.e., **i** is the subscript with variable **A**). As we know that the elements of the array are stored on consecutive memory locations, it becomes convenient to find out the address of memory location of **i**th element, for given base address (address of first element) and **W** (i.e. the number of memory locations required by one element).

One dimensional array
Number of elements `N= UB-LB+1` Where `LB`-Lower Bound `UB`-Upper Bound
Memory Location of `A[i]`; `Loc(A[i])=Base(A)+W*(i-LB)`
`Loc(A[i])=Base(A)+W*I`
Where N is given as in C++, `LB` is assumed as 0

Two dimensional array
Number of elements = `ROWSxCOLS=(UBI-LBI+1)x(UBJ-LBJ+1)`

Row Major: `Loc(A[I][J])=Base(A) + W*(COLS*(I-LBI)+(J-LBJ))`
where `Base(A)` is address of first element's memory location
`COLS` is number of columns = `UBJ-LBJ+1`
`W` is number of memory locations required by one element
`LBI` is Lower Bound of row
`UBJ` is upper bound of column
`LBJ` is Lower bound of column
`Loc(A[I][J])=Base(A) + W*(COLS*I+J)`
where `COLS` is the number of columns, `LBI=0` and `LBJ=0`

Column Major:`Loc(A[I][J])=Base(A) + W*(ROWS*(J-LBJ)+(I-LBI))`
where `Base(A)` is address of first element's memory location
ROWS is the number of `rows = UBI-LBI+1`
`W` is number of memory locations required by one element
`LBJ` is Lower Bound of Column
`UBI` is upper bound of Row
`LBI` is Lower bound of Row
`Loc(A[I][J])=Base(A) + W*(ROWS*J+I)`

where R is number of Rows, `LBI=0` and `LBJ=0`

Exercise 1 A one-dimensional array P[100] is stored in memory with a base address as 5000. Find out addresses of P[15] and P[40], if each element of this array requires 4 bytes.

```
Given,      Base(P)      =    5000
            W            =    4
            Loc(P[I])    =    Base(P)    +    W*I
            Loc(P[15])   =    5000       +    4*15
                         =    5000       +    60
                         =    5060
            Loc(P[40])   =    5000       +    4*40
                         =    5000       +    160
                         =    5160
```

Exercise 2 A one-dimensional array A[-5..25] is stored in memory with each element requiring 2 bytes. If the base address is 8000, find out the following:

a) Address of A[5] and A[-3]
b) Total no. of elements present in the array

```
Given,      Base(A)      =    8000
            W            =    2
            LB           =    -5
            Loc(A[I])    =    Base(A)    +    W*(I-LB)
            Loc(A[5])    =    8000       +    2*(5-(-5))
                         =    8000       +    20
                         =    8020
            Loc(A[-3])   =    8000       +    2*(-3-(-5))
                         =    8000       +    4
                         =    8004
            Total
            No. of Elements=  UB-LB+1    =25-(-5)+1 =31
```

Exercise 3 A two-dimensional array Q[5][15] is stored in memory along the row with each element requiring 2 bytes. If the base address is 6500, find out the following:

a) Addresses of Q[5][10] and Q[3][5]
b) Total no. of elements present in the array

```
Given,      Base(Q)      =    6500
            W            =    2
            COLS         =    5
Row Major,  Loc(Q[I][J]) =    Base(Q)    +    W*(COLS*I+J)
            Loc(Q[5][10])=    6500       +    2*(15*5+10)
                         =    6500       +    170
                         =    6670
            Loc(Q[3][5]) =    6500       +    2*(15*3+5)
                         =    6500       +    100
                         =    6600
            Total
            No. of Elements=  ROWS*COLS  =5*15       =75
```

Exercise 4 R[-4..4,7..17] is a two-dimensional array, stored in the memory along the column with each element requiring 4 bytes. If the base address is 5000, find out the following:

a) Addresses of R[2][10] and R[3][15]
b) Total no. of elements present in the array

```
Given,      Base(R)      =    5000
            W            =    4
            ROWS         =    UBI-LBI+1 =4-(-4)+1=9
            COLS         =    UBJ-LBJ+1 =17-7+1=11

Column Major,Loc(R[I][J]) =   Base(R)    +    W*(ROWS*(J-LBJ)+(I-LBI))
            Loc(R[2][10])=    5000       +    4*(9*(10-7)+(2-(-4))
                         =    5000       +    4*(27+6)
                         =    5132
            Loc(R[3][15])=    5000       +    4*(9*(15-7)+(3-(-4))
                         =    5000       +    4*(72+7)
                         =    5316
            Total No. of Elements=    ROWS*COLS    =    9*11   =    99
```