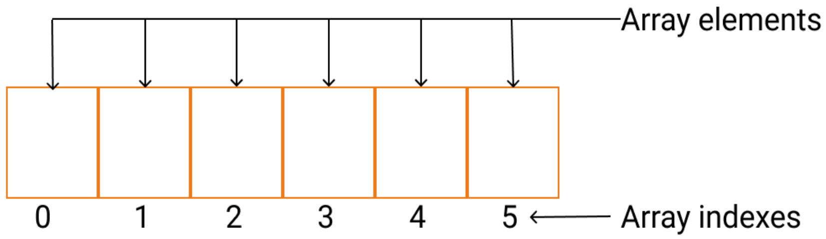
Array : Static, Sequential, Linear, Homogeneous Data Structure

Prof. Rupa G. Mehta

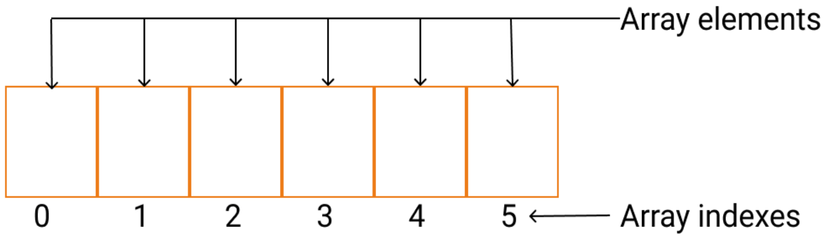
DOCSE, SVNIT, Surat

Array

- Collection of data items
  - Having similar data types
  - Stored in contiguous memory locations
- By knowing the address of the first item we can easily access all items/elements of an array.



Array



- Array Description:
  - **Array Index:**
    - The location of an element in an array has an index, which identifies the element.
    - Normally array index starts from 0.
  - **Array Element:**
    - Items stored in an array is called an element.
    - The elements can be accessed via its index.
  - **Array Length:**
    - The length of an array is defined based on the number of elements an array can store. In the above example, array length is 6 which means that it can store 6 elements.

- When an array of size and type is declared, the compiler allocates enough memory to hold all elements of data.
- E.g. an array face [10] will have 10 elements with index starting from 0 to 9 and the memory allocated contiguously will be 20 bytes (for integer of 2 bytes).
- The compiler knows the address of the first byte of the array only. Also, the address of the first byte is considered as the memory address for the whole array.

Types of Arrays

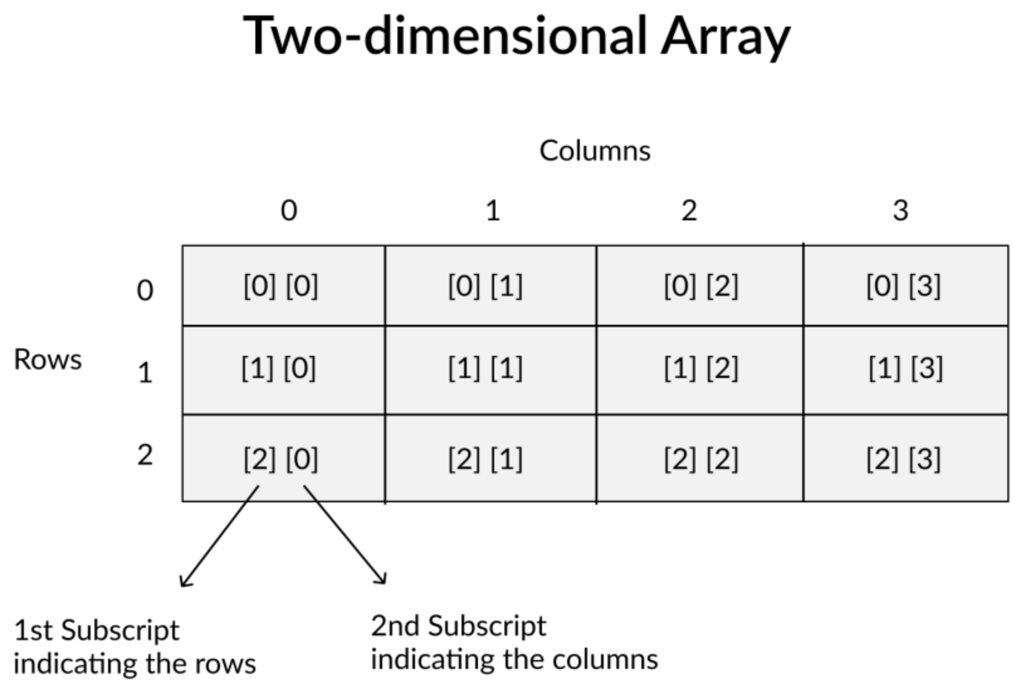
- One dimensional array
- Multi-dimensional array

One-Dimensional Array

- A one-dimensional array is also called a single dimensional array
- where the elements will be accessed in sequential order.
- This type of array will be accessed by the subscript of either a column or row index.

Multi-Dimensional Array

- When the number of dimensions specified is more than one, then it is called as a multi-dimensional array.
- Multidimensional arrays include 2D arrays and 3D arrays.



Example

- **A two-dimensional array**
  - Accessed with subscript of row and column index
  - For traversal the value of the rows and columns will be considered
  - E.g. **face [3] [4],** the first index specifies the number of rows and the second index specifies the number of columns and the array can hold 12 elements (3 * 4)

A three-dimensional array

- The array **face [5] [10] [15]** can hold 750 elements (5 * 10 * 15).

Address Calculation:

Actual Address of the 1st element of the array is known as **Base Address (B)**
Here it is 1100

Memory space acquired by every element in the Array is called **Width (W)**
Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

Lower Limit/Bound of Subscript **(LB)**

- Address of an element A[I] is calculated using the following formula:
- For i=1: 1100 + (1 - 0) * 4 = 1104
- For i=4: 1100 + (4 - 0) * 4 = 1116
  - **Address of A [I] = B + W * (I – LB)**
  - Where,
    B = Base address
    W = Storage Size of one element stored in the array (in byte)
    I = Subscript of element whose address is to be found
    LB = Lower limit / Lower Bound of subscript, if not specified assume 0 (zero)

Example 2:

Base address of an array B[1300, 1400, ...1800, 1900] as 1020 and size of each element is 2 bytes in the memory. Find the address of B[1700]

B = 1020, LB = 1300, W = 2, I = 1700

$$\begin{aligned}
\text{Address of } A[I] &= B + W * (I - LB) \\
&= 1020 + 2 * (1700 - 1300) \\
&= 1020 + 2 * 400 \\
&= 1020 + 800 \\
&= 1820
\end{aligned}$$

Why does Array Indexing start with 0?

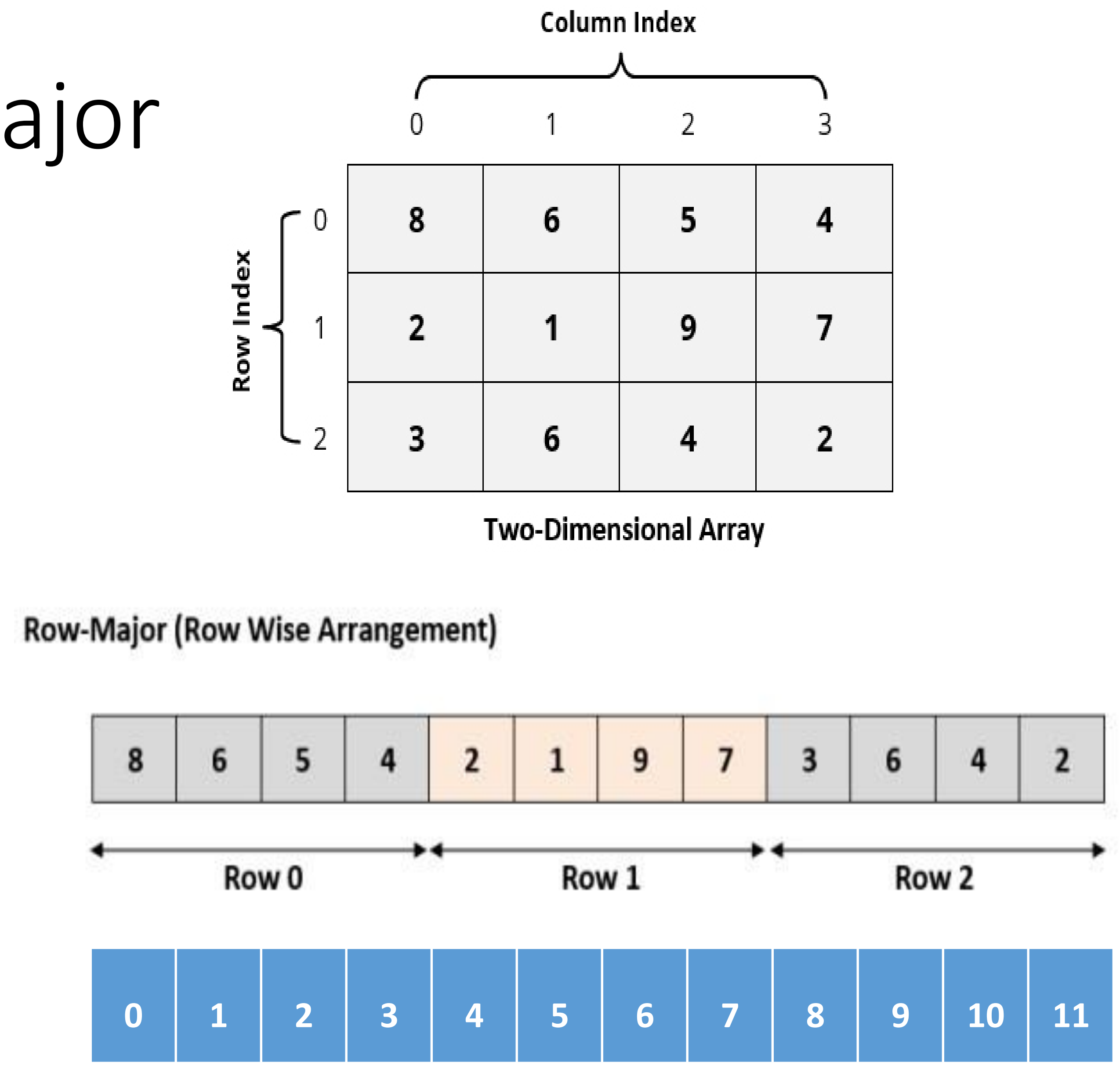
- **In a[n], a** is a pointer which contains the memory location of the first element of the array
- First element can be accessed with **a[0]**
  - Internally decoded by the compiler as ***(a + 0)**
- Second element can be accessed by **a[1]** or ***(a + 1)**
- **a** contains the address of the first element = base address
- Index describes the offset from the first element, i.e. The distance from the first element.

- If array indexing starts at 1 instead of 0
- The first element can be accessed by **a[1]**
  - Which is internally decoded as ***(a + 1 – 1)**.
- **One extra operation i.e. Subtraction by 1**
  - **Increase the complexity**
- Thus to avoid this extra operation and improve the performance, array indexing starts at 0 and not at 1.

Organization of Two Dimensional Array

- While storing the elements of a 2-D array in memory, these are allocated contiguous memory locations.
- A 2-D array must be linearized so as to enable their storage.
- There are two alternatives to achieve linearization:
  - Row-Major
  - Column-Major

Row Major

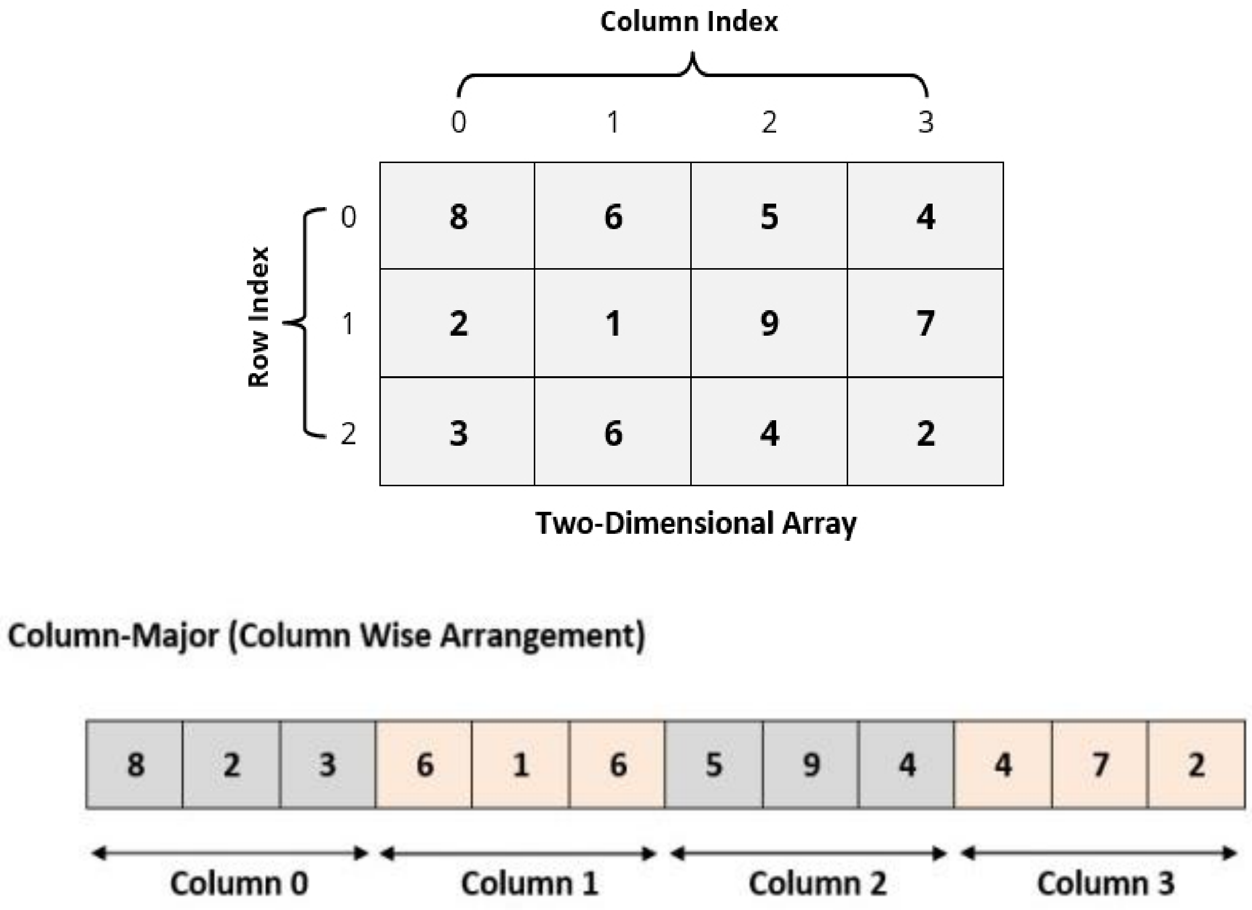

Column Index
0 1 2 3
Row Index
0 1 2
8 6 5 4
2 1 9 7
3 6 4 2
Two-Dimensional Array
Row-Major (Row Wise Arrangement)
8 6 5 4 2 1 9 7 3 6 4 2
Row 0
Row 1
Row 2
0 1 2 3 4 5 6 7 8 9 10 11

Address Calculation in Row Major 3D array

- Address of an element A[I][J] = = B + W * [N * (I – Lr) + (J – Lc)] for the array declared as A[M][N]
- Where
- **B** = Base address
  I = Row subscript of element whose address is to be found
  J = Column subscript of element whose address is to be found
  W = Storage Size of one element stored in the array (in byte)
  Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)
  Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)
  M = Number of row of the given matrix
  N = Number of column of the given matrix

- Example : for the given A[4][4] with A[0..3][0..3]
- Address of A[1][1] = B + W * [4 * (1-0) + (1-0)]
  = B + W * [5]

Column Oriented storage :



Address calculation for Column Major 2D Array

- **Address of A[I][J] = B + W * [M*(J – Lc) + (I – Lr)]**

- Where
- **B** = Base address
  I = Row subscript of element whose address is to be found
  J = Column subscript of element whose address is to be found
  W = Storage Size of one element stored in the array (in byte)
  Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)
  Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)
  M = Number of row of the given matrix
  N = Number of column of the given matrix

Example

An array X [-15……….10, 15……………40] requires **one byte of storage**. If beginning location is 1500 determine the location of X [15][20].

- **Solution:**
- Number or rows say **M = (Ur – Lr) + 1** = [10 – (- 15)] +1 = 26
  Number or columns say **N = (Uc – Lc) + 1** = [40 – 15)] +1 = 26

- **Row Major Wise Calculation of above equation**
- The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, N = 26
- Address of A [I][J] = A [15][20] = B + W * [N * (I – Lr) + (J – Lc)]

= 1500 + 1* [26 * (15 – (-15))) + (20 – 15)]

= 1500 + 1 * [26 * 30 + 5]

= 1500 + 1 * [780 + 5]

= 1500 + 785

= 2285

Example (cont.)

An array X [-15……….10, 15……………40] requires **one byte of storage**. If beginning location is 1500 determine the location of X [15][20].

- **Solution:**
- Number or rows say **M = (Ur – Lr) + 1** = [10 – (- 15)] +1 = 26

  Number or columns say **N = (Uc – Lc) + 1** = [40 – 15)] +1 = 26

Column Major Wise Calculation of above equation

- The given values are: B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, M = 26
- Address of A [I][J] = B + W * [(I – Lr) + M * (J – Lc)]

  = 1500 + 1 * [(15 – (-15)) + 26 * (20 – 15)]

  = 1500 + 1 * [30 + 26 * 5]

  = 1500 + 1 * [160]

  = 1660 **[Ans]**