Data Structure and Algorithm
CS-102

Dr. Sambit Bakshi
Dept. of CSE, NIT Rourkela

Operation on Linear Structure

- **Traversal** : Processing each element in the list
- **Search** : Finding the location of the element with a given value or the record with a given key
- **Insertion**: Adding a new element to the list
- **Deletion**: Removing an element from the list
- **Sorting** : Arranging the elements in some type of order
- **Merging** : Combining two list into a single list

Array

Linear Arrays

- A linear array is a list of a finite number of **n** homogeneous data elements (that is data elements of the same type) such that
  - The elements of the arrays are referenced respectively by an **index set** consisting of **n** consecutive numbers
  - The elements of the arrays are stored respectively in **successive memory locations**

Linear Arrays

- The number **n** of elements is called the length or size of the array.
- The index set consists of the integer **1, 2, … n**
- **Length** or the number of data elements of the array can be obtained from the index set by

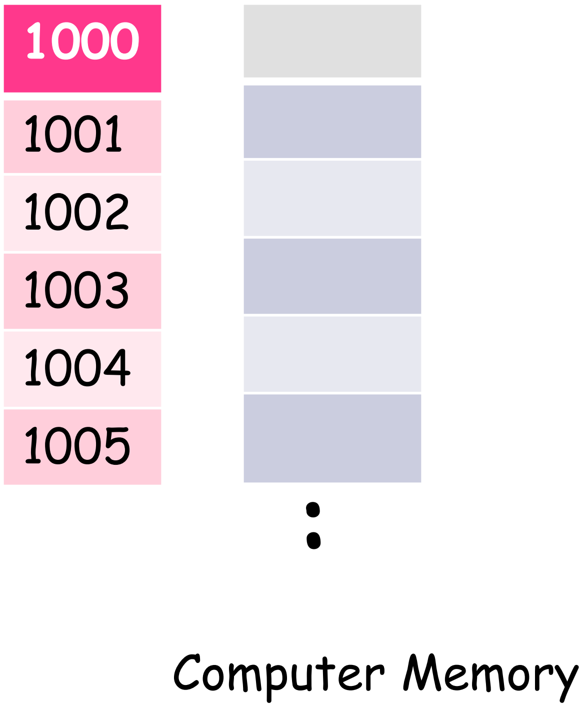
  Length = UB – LB + 1 where **UB** is the largest index called the **upper bound** and **LB** is the smallest index called the **lower bound** of the arrays

Linear Arrays

- Element of an array **A** may be denoted by
  - Subscript notation **A_1, A_2, , …. , A_n**
  - Parenthesis notation **A(1), A(2), …. , A(n)**
  - Bracket notation **A[1], A[2], ….. , A[n]**

- The number **K** in A[K] is called subscript or an index and A[K] is called a **subscripted variable**

Representation of Linear Array in Memory



Representation of Linear Array in Memory

- Let **LA** be a linear array in the memory of the computer
- **LOC(LA[K]) = address of the element LA[K] of the array LA**
- The element of **LA** are stored in the successive memory cells
- Computer does not need to keep track of the address of every element of **LA**, but need to track only the address of the first element of the array denoted by **Base(LA)** called the **base address** of LA

Representation of Linear Array in Memory

- **LOC(LA[K]) = Base(LA) + w(K – LB)** where **w** is the number of words per memory cell of the array LA [**w** is the size of the **data type**]

Example 1

Find the address for LA[6]
Each element of the array occupy 1 byte

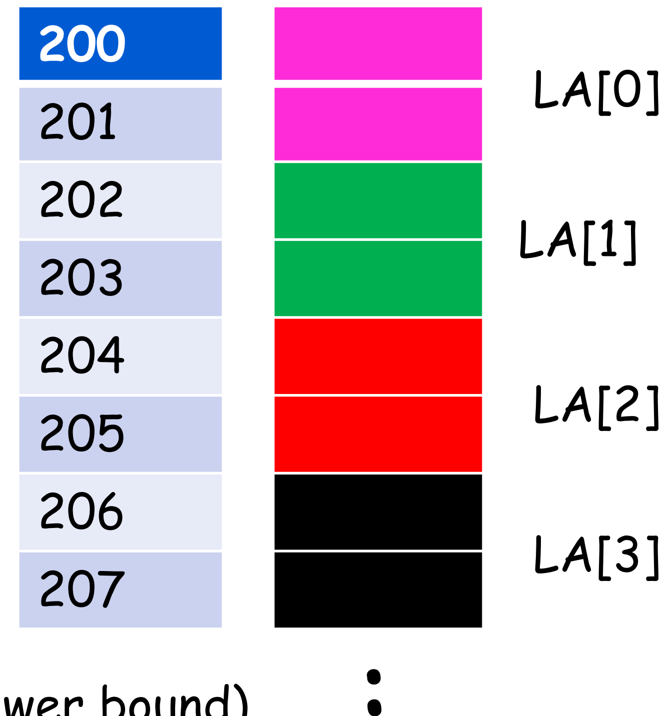
200		LA[0]
201		LA[1]
202		LA[2]
203		LA[3]
204		LA[4]
205		LA[5]
206		LA[6]
207		LA[7]
	:	

LOC(LA[K]) = Base(LA) + w(K – lower bound)

LOC(LA[6]) = 200 + 1(6 – 0) = 206

Example 2

Find the address for LA[15]
Each element of the array
occupy 2 bytes



LOC(LA[K]) = Base(LA) + w(K – lower bound)

LOC(LA[15]) = 200 + 2(15 – 0) = 230

Representation of Linear Array in Memory

- Given any value of **K**, time to calculate **LOC(LA[K])** is same
- Given any subscript **K** one can access and locate the content of **LA[K]** without scanning any other element of **LA**
- A collection **A** of data element is said to be **indexed** if any element of **A** called A_k can be located and processed in time that is independent of **K**

Traversing Linear Arrays

- Traversing is accessing and processing each element of the data structure exactly once.

Traversing Linear Arrays

- Traversing is accessing and processing each element of the data structure exactly once.

Traversing Linear Arrays

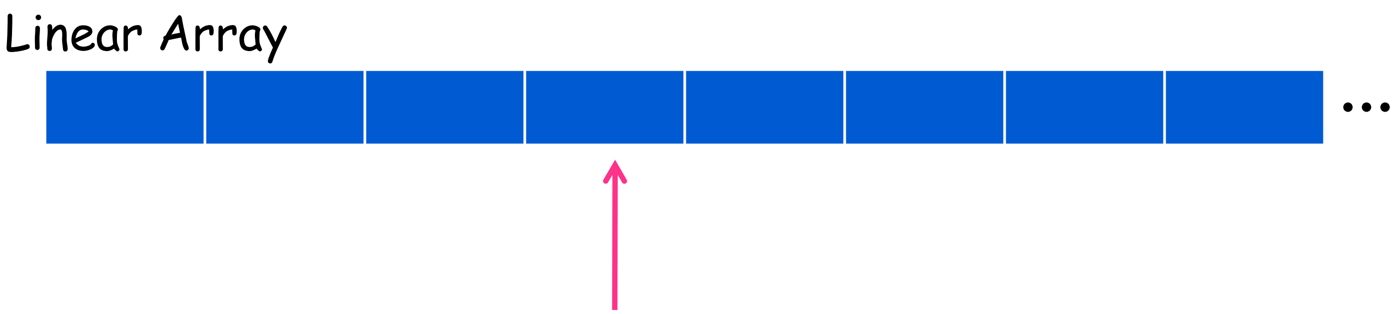
- Traversing is accessing and processing each element of the data structure exactly once.

Traversing Linear Arrays

- Traversing is accessing and processing each element of the data structure exactly once.

Traversing Linear Arrays

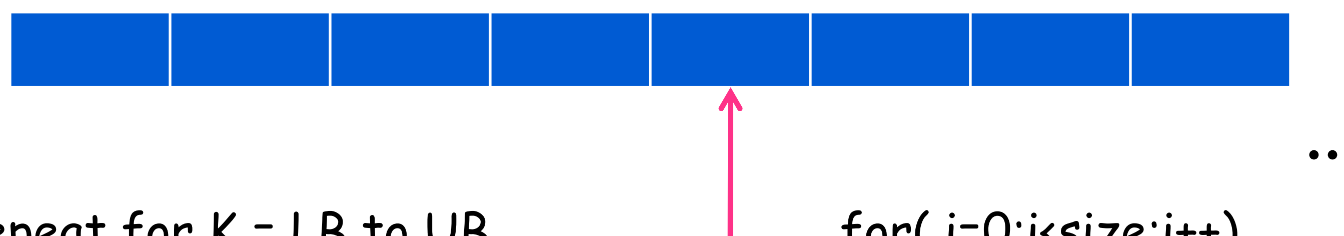
- Traversing is accessing and processing each element of the data structure exactly once.



Traversing Linear Arrays

- Traversing is accessing and processing each element of the data structure exactly once.

Linear Array



1. Repeat for K = LB to UB
   Apply PROCESS to LA[K]
   [End of Loop]
2. Exit

```
for( i=0;i<size;i++)
  printf("%d",LA[i]);
```

Inserting and Deleting

- **Insertion**: Adding an element
  - Beginning
  - Middle
  - End

- **Deletion**: Removing an element
  - Beginning
  - Middle
  - End

Insertion

1	**Brown**
2	Davis
3	Johnson
4	Smith
5	Wagner
6	
7	
8	

1	**Brown**
2	Davis
3	Johnson
4	Smith
5	Wagner
6	Ford
7	
8	

Insert Ford at the End of Array

Insertion

1	**Brown**	1	**Brown**	1	**Brown**	1	**Brown**
2	Davis	2	Davis	2	Davis	2	Davis
3	Johnson	3	Johnson	3	Johnson	3	Ford
4	Smith	4	Smith	4		4	Johnson
5	Wagner	5		5	Smith	5	Smith
6		6	Wagner	6	Wagner	6	Wagner
7		7		7		7	
8		8		8		8	

Insert Ford as the 3rd Element of Array

Insertion is not Possible without loss of data if the array is FULL

Insertion Algorithm

- **INSERT (LA, N , K , ITEM)** [LA is a linear array with N elements and K is a positive integers such that K ≤ N. This algorithm inserts an element ITEM into the K^{th} position in LA]
  1. [Initialize Counter] Set J := N
  2. Repeat Steps 3 and 4 while J ≥ K
  3. [Move the J^{th} element downward] Set LA[J + 1] := LA[J]
  4. [Decrease Counter] Set J := J -1
  5. [Insert Element] Set LA[K] := ITEM
  6. [Reset N] Set N := N +1;
  7. Exit

Deletion

1	**Brown**
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	Wagner
8	

1	**Brown**
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	
8	

Deletion of Wagner at the End of Array

Deletion

1	**Brown**	1	**Brown**	1	**Brown**	1	**Brown**
2	Davis	2		2	Ford	2	Ford
3	Ford	3	Ford	3		3	Johnson
4	Johnson	4	Johnson	4	Johnson	4	
5	Smith	5	Smith	5	Smith	5	Smith
6	Taylor	6	Taylor	6	Taylor	6	Taylor
7	Wagner	7	Wagner	7	Wagner	7	Wagner
8		8		8		8	

Deletion of Davis from the Array

Deletion

1	**Brown**
2	Ford
3	Johnson
4	Smith
5	Taylor
6	Wagner
7	
8	

No data item can be deleted from an empty array

Deletion Algorithm

- **DELETE (LA, N , K , ITEM)** [LA is a linear array with N elements and K is a positive integers such that $K \leq N$. This algorithm deletes K^{th} element from LA]

  1. Set ITEM := LA[K]
  2. Repeat for J = K+1 to N:

     [Move the J^{th} element upward] Set LA[J-1] := LA[J]
  3. [Reset the number N of elements] Set N := N - 1;
  4. Exit

Multidimensional Array

- One-Dimensional Array
- Two-Dimensional Array
- Three-Dimensional Array

Two-Dimensional Array

- A Two-Dimensional **$m \times n$** array **A** is a collection of **$m \cdot n$** data elements such that each element is specified by a pair of integers (such as J, K) called subscripts with property that

  $1 \leq J \leq m$ and **$1 \leq K \leq n$**

The element of **A** with first subscript **J** and second subscript **K** will be denoted by **$A_{J,K}$** or **A[J][K]**

2D Arrays

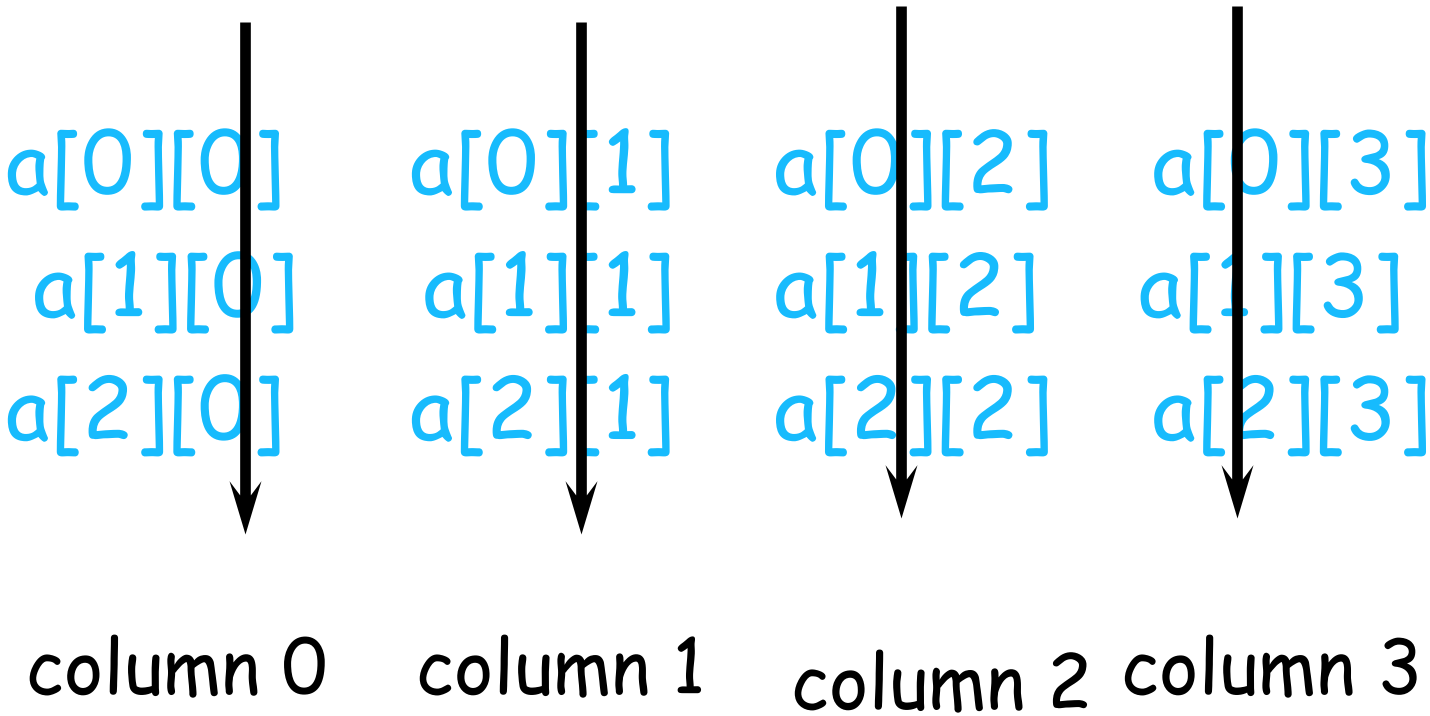
The elements of a 2-dimensional array a is shown as below

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Rows Of A 2D Array

a[0][0]	a[0][1]	a[0][2]	a[0][3]	row 0
a[1][0]	a[1][1]	a[1][2]	a[1][3]	row 1
a[2][0]	a[2][1]	a[2][2]	a[2][3]	row 2

Columns Of A 2D Array



2D Array

- Let ***A*** be a two-dimensional array **m x n**
- The array ***A*** will be represented in the memory by a block of **m x n** sequential memory location
- Programming language will store array ***A*** either
  - **Column by Column** (Called Column-Major Order) Ex: Fortran, MATLAB
  - **Row by Row** (Called Row-Major Order) Ex: C, C++ , Java

2D Array in Memory

A	Subscript	
	(1,1)	
	(2,1)	Column 1
	(3,1)	
	(1,2)	
	(2,2)	Column 2
	(3,2)	
	(1,3)	
	(2,3)	Column 3
	(3,3)	
	(1,4)	
	(2,4)	Column 4
	(3,4)	

Column-Major Order

A	Subscript	
	(1,1)	
	(1,2)	Row 1
	(1,3)	
	(1,4)	
	(2,1)	
	(2,2)	
	(2,3)	Row 2
	(2,4)	
	(3,1)	
	(3,2)	Row 3
	(3,3)	
	(3,4)	

Row-Major Order

2D Array

- $LOC(LA[K]) = Base(LA) + w(K - 1)$

- **LOC(A[J,K]) of A[J,K]**

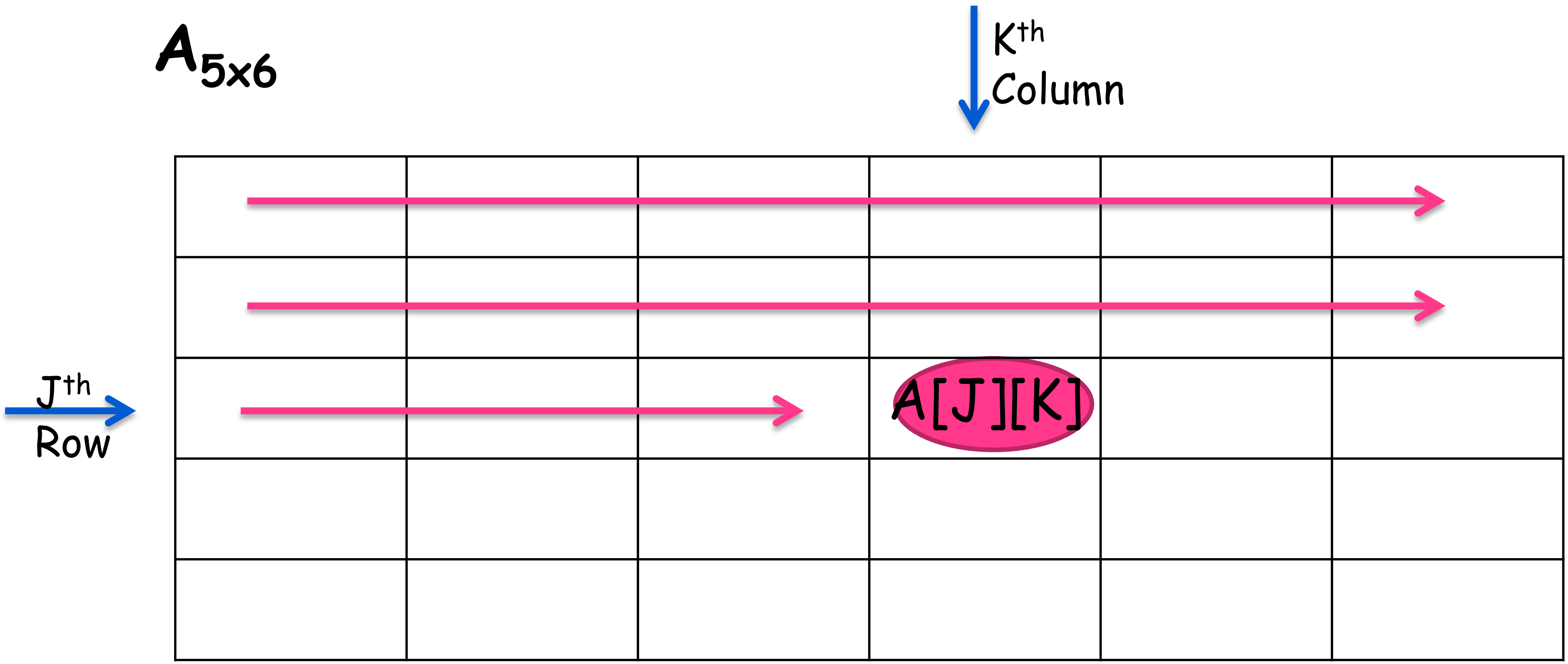
  Column-Major Order

  $LOC(A[J,K]) = Base(A) + w[m(K-LB) + (J-LB)]$

  Row-Major Order

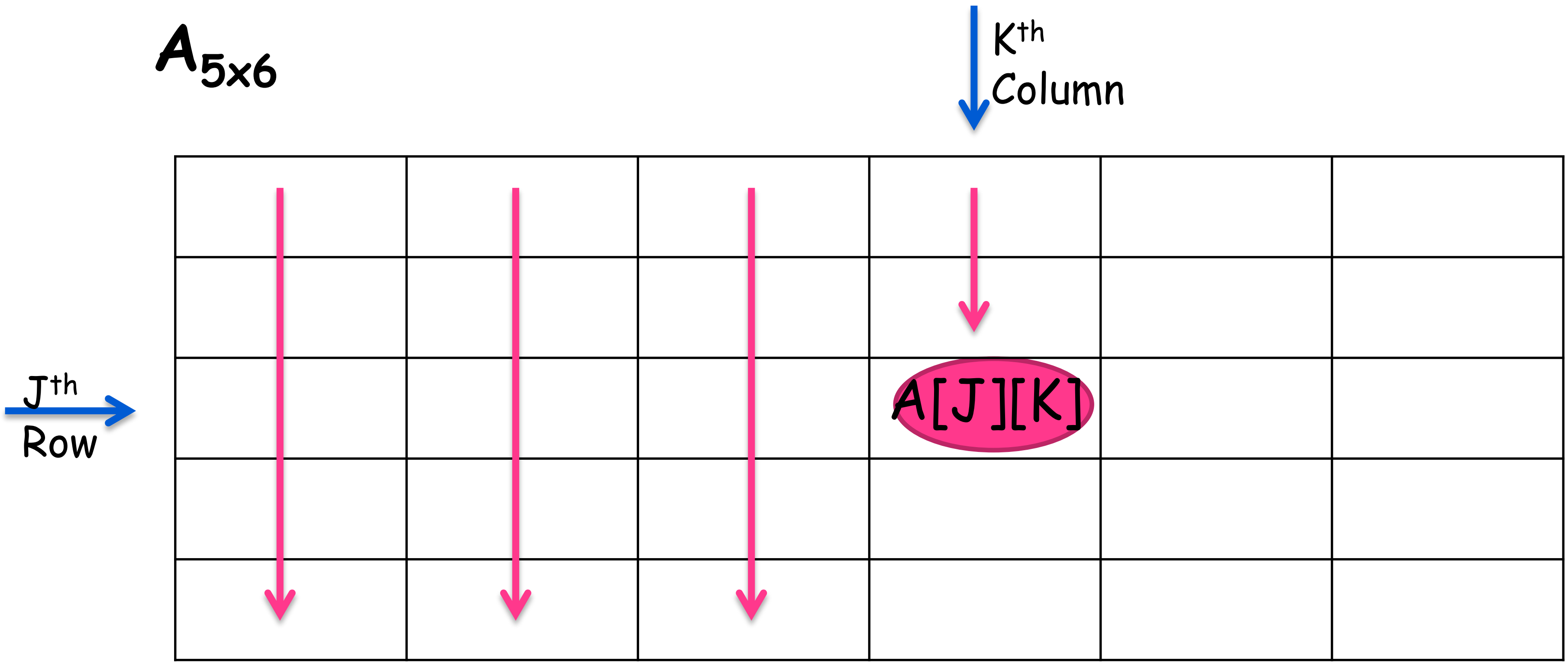
  $LOC(A[J,K]) = Base(A) + w[n(J-LB) + (K-LB)]$

2D Array (Row Major)



$$LOC(A[J,K]) = Base(A) + w[n(J-LB) + (K-LB)]$$

2D Array (Column Major)



$$LOC(A[J,K]) = Base(A) + w[m(K-LB) + (J-LB)]$$

2D Array Example

- Consider a 25 x 4 array A. Suppose the Base(A) = 200 and w =2. Suppose the programming stores 2D array using row-major. Compute LOC(A[12,3])

- LOC(A[J,K]) = Base(A) + w[n(J-LB) + (K-LB)]

- LOC(A[12,3]) = 200 + 2[4(12-1) + (3 -1)]
  = 292

Multidimensional Array

- An **n**-dimensional $m_1 \times m_2 \times .... \times m_n$ array **B** is a collection of $m_1.m_2...m_n$ data elements in which each element is specified by a list of **n** integers indices – such as $K_1, K_2, ..., K_n$ – called subscript with the property that

  $1 \le K_1 \le m_1, \quad 1 \le K_2 \le m_2, \quad .... \quad 1 \le K_n \le m_n$

The Element **B** with subscript $K_1, K_2, ..., K_n$ will be denoted by

$$B_{K1,K2,\ ...,Kn} \qquad \text{or} \qquad B[K_1,K_2,....,K_n]$$

Multidimensional Array

- Let C be a n-dimensional array
- Length L_i of dimension i of C is the number of elements in the index set

  $L_i = UB_i - LB_i + 1$
- For a given subscript K_i, the effective index E_i of K_i is the number of indices preceding K_i in the index set

  $E_i = K_i - LB_i$

Multidimensional Array

- Address $LOC(C[K_1, K_2, \ldots, K_n])$ of an arbitrary element of C can be obtained as

Column-Major Order

$$Base(C) + w[(((\ldots (E_N L_{N-1} + E_{N-1})L_{N-2} + E_{N-2}) + \ldots\ldots + E_3)L_2 + E_2)L_1 + E_1]$$

Row-Major Order

$$Base(C) + w[(\ldots ((E_1 L_2 + E_2)L_3 + E_3)L_4 + \ldots\ldots + E_{N-1})L_N + E_N]$$

Multidimensional Array

Column-Major Order

$$\text{Base}(C) + w \sum_{i=N}^{1} \left(E_i \left(\prod_{\substack{j=i-1 \\ j \geq 1}}^{1} L_j \right) \right)$$

Row-Major Order

$$\text{Base}(C) + w \sum_{i=1}^{N} \left(E_i \left(\prod_{\substack{j=i+1 \\ j \leq N}}^{N} L_j \right) \right)$$

Example

- MAZE(2:8, -4:1, 6:10)
- Calculate the address of **MAZE[5,-1,8]**
- Given: Base(MAZE) = 200, w = 4, MAZE is stored in Row-Major order
- L1 = 8-2+1 = 7, L2 = 6, L3 = 5
- E1 = 5 -2 = 3, E2 = 3, E3 = 2

Example Contd ..

- Base(C) + w[(… ((E_1L_2 + E_2)L_3 + E_3)L_4 + ….. +E_{N-1})L_N +E_N]
- E_1L_2 = 3 . 6 = 18
- E_1L_2 + E_2 = 18 + 3 = 21
- (E_1L_2 + E_2)L_3 = 21 . 5 = 105
- (E_1L_2+E_2)L_3 + E_3 = 105 + 2 = 107
- MAZE[5,-1,8] = 200 + 4(107) = 200 + 428 = 628