

# ARRAYS

# ARRAYS

- An array is a collection of similar data elements.
- The elements of the array are stored in consecutive memory locations and are referenced by an **index** (also known as the subscript).
- Declaring an array means specifying three things:
  - Data Type
  - Array Name
  - Array Size
- Arrays are declared using the following syntax.

# ARRAYS

## CALCULATING THE ADDRESS OF ARRAY ELEMENTS

**Address of data element,  $A[k] = BA(A) + w(k - \text{lower\_bound})$**

Here, **A** is the array

**k** is the index of the element of which we have to calculate the address

**BA** is the base address of the array **A**.

**w** is the word size of one element in memory, for example, size of int is 2.

## CALCULATING THE LENGTH OF THE ARRAY

**Length = upper\_bound – lower\_bound + 1**

Where, upper\_bound is the index of the last element

and lower\_bound is the index of the first element in the array

# ARRAYS

## Operations:

- **Traversing of Array**
- **Insertion in Arrays**
  - **Insert at end**
  - **Insert at front**
  - **Insert at a given position**
  - **Insert after a given value**
- **Deletion in Arrays**
  - **Delete from end**
  - **Delete from front**
  - **Delete at a given position**
  - **Delete a given value**
- **Linear Search**
- **Binary Search**





## **DELETE\_LOC (Arr, SIZE, N, LOC)**

**Step 1. If  $N == 0$  then**

**PRINT “No data...No deletion”**

**End If**

**Step 2. If  $N > 0$  AND  $LOC < N$  then**

**a. DELETE Arr [LOC]**

**b. Set  $I = LOC + 1$**

**c. Repeat While  $I \leq N-1$  do**

**i. Set Arr [I-1] = Arr [I]**

**ii. Set  $I = I + 1$**

**Done**

**d. DELETE Arr [N-1]**

**e. Set  $N = N - 1$**

**Else**

**PRINT “LOC  $\geq$  N, so no data to delete”**

**End If**

**Step 3. Exit**

# Introduction

- A binary search is an advanced type of search algorithm that finds and fetches data from a sorted list of items.
- Its core working principle involves dividing the data in the list to half until the required value is located and displayed to the user in the search result.
- Binary search is commonly known as a **half-interval search** or a **logarithmic search**.



# Binary Search

BINARY\_SEARCH (A, LB, UB, VAL)

Step 1. Set  $BEG = LB$ ,  $END = UB$ ,  $POS = -1$ .

Step 2. Repeat While  $BEG \leq END$  do

a.  $MID = \text{INT} (BEG + END) / 2$

b. If  $A[MID] = VAL$  then

i. PRINT “Element found at MID position”

ii.  $POS = MID$

iii. Go to Step 4.

Else If  $A[MID] > VAL$  then

$END = MID - 1$

Else

$BEG = MID + 1$

End If


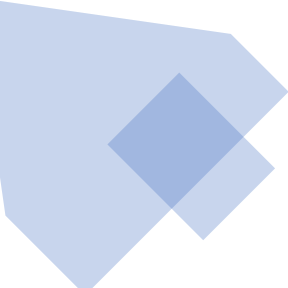
Done

Step 3. If  $POS == -1$  then

PRINT “VAL not found”

End If

Step 4. Exit



0	1	2	3	4	5	6	7	8
11	13	25	28	35	45	55	60	65

# 2-Dimensional Arrays

Rows/Columns	Col 0	Col 1	Col2	Col 3	Col 4
Row 0	Marks[0][0]	Marks[0][1]	Marks[0][2]	Marks[0][3]	Marks[0][4]
Row 1	Marks[1][0]	Marks[1][1]	Marks[1][2]	Marks[1][3]	Marks[1][4]
Row 2	Marks[2][0]	Marks[2][1]	Marks[2][2]	Marks[2][3]	Marks[2][4]

## *Column Major:*

$$\text{Address}(A[I][J]) = \text{Base\_Address} + w \{ M ( J - \text{LB} ) + ( I - \text{LB} ) \}$$

## *Row major:*

$$\text{Address}(A[I][J]) = \text{Base\_Address} + w \{ N ( I - \text{LB} ) + ( J - \text{LB} ) \}$$

where,  $w$  is the number of memory bytes stored per element

$M$ , is the number of rows

$N$ , is the number of columns

$I$  and  $J$  are the indices of the array element

# 2-Dimensional Arrays

## Operations:

- Traversing
- Transpose
- Creation
- Addition of two matrices
- Subtraction of two matrices
- Multiplication of two matrices

# 2-Dimensional Arrays

TRAVERSE (A, M, N)

Step 1. Repeat For  $I = 0$  to  $M-1$  do

Repeat For  $J = 0$  to  $N-1$  do

PRINT (A[I][J])

Done

Done

Step 2. Exit

# 2-Dimensional Arrays

TRANSPOSE (A, M, N, B)

Step 1. Repeat For I = 0 to M-1 do

    Repeat For J = 0 to N-1 do

        Set  $B[J][I] = A[I][J]$

    Done

Done

Step 2. Repeat For I = 0 to M-1 do

    Repeat For J = 0 to N-1 do

        PRINT  $B[I][J]$

    Done

Done

Step 3. Exit

# 2-Dimensional Arrays

Matrix-Multiplication (X, Y, Z) //  $X_{m \times n}$  multiplied by  $Y_{n \times p}$  to form  $Z_{m \times p}$

Step 1. Repeat For  $i = 0$  to  $m-1$  do

Repeat For  $j = 0$  to  $p-1$  do

a. Set  $Z[i][j] = 0$

b. Repeat For  $k = 0$  to  $n-1$  do

Set  $Z[i][j] = Z[i][j] + X[i][k] \times Y[k][j]$

Done

Done

Done

Step 2. Repeat For  $I = 0$  to  $m-1$  do

Repeat For  $J = 0$  to  $p-1$  do

PRINT  $Z[I][J]$

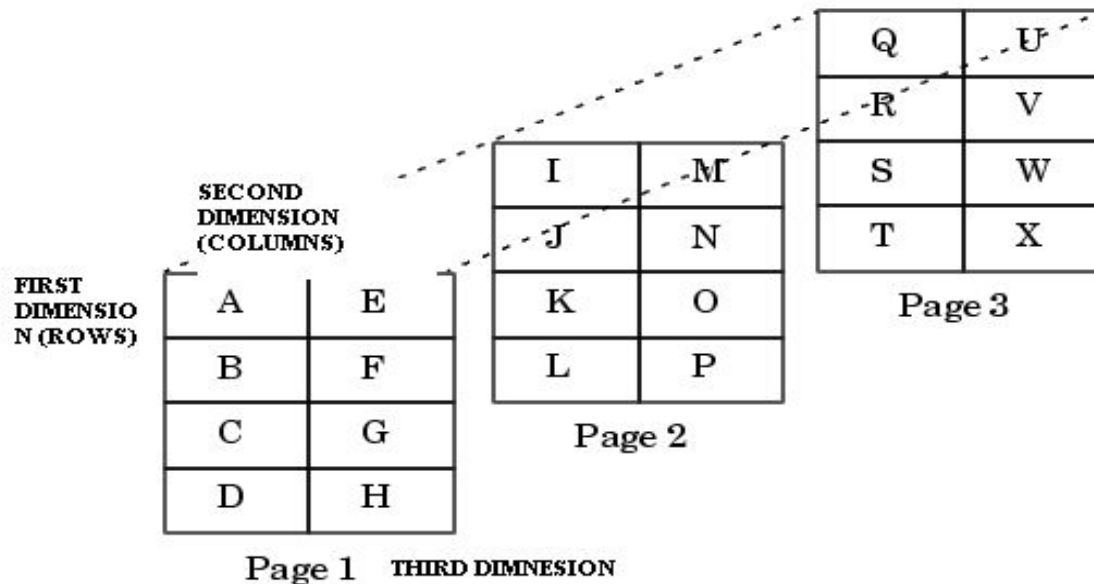
Done

Done

Step 3. Exit

# MULTI DIMENSIONAL ARRAYS

- A multi dimensional array is an array of arrays.
- Like we have one index in a single dimensional array, two indices in a two dimensional array, in the same way we have n indices in a n-dimensional array or multi dimensional array.





# **SPARSE MATRIX**

- Sparse matrix is a matrix that has many elements with a value zero.
- In order to efficiently utilize the memory, specialized algorithms and data structures that take advantage of the sparse structure of the matrix should be used. Otherwise, execution will slow down and the matrix will consume large amounts of memory.
- Types-
  - Lower Triangular matrix
  - Upper Triangular matrix
  - Diagonal matrix

# SPARSE MATRIX

- In the second variant of a sparse matrix, elements with a non-zero value can appear only on the diagonal or immediately above or below the diagonal. This type of matrix is also called a **tridiagonal matrix**.
- In a tridiagonal matrix,  $A[i][j] = 0$  where  $|i - j| > 1$ . Therefore, if elements are present on:
  1. The main diagonal the, it contains non-zero elements for  $i=j$ . In all there will be  $n$  elements
  2. Diagonal below the main diagonal, it contains non zero elements for  $i=j+1$ . In all there will be  $n-1$  elements
  3. Diagonal above the main diagonal, it contains non zero elements for  $i=j-1$ . In all there will be  $n-1$  elements

# Question 1

A program P reads in 500 integers in the range [0 to 100] representing the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies?  
(GATE CS 2005)

- (A) An array of 50 numbers
- (B) An array of 100 numbers
- (C) An array of 500 numbers
- (D) A dynamically allocated array of 550 numbers

## Question 2

Which of the following operations is not  $O(1)$  for an array of sorted data. You may assume that array elements are distinct.

- (A) Find the  $i$ th largest element
- (B) Delete an element
- (C) Find the  $i$ th smallest element
- (D) All of the above

## Question 3

- a. Write an algorithm to reverse an array of unique elements in place without using any other array.
- b. Write an algorithm to print the common elements in the given 2 sorted arrays. Assume unique element are present in the arrays and are sorted in ascending order.