C++ Programming

Post Graduate Diploma in Advanced Computing (PG-DAC)

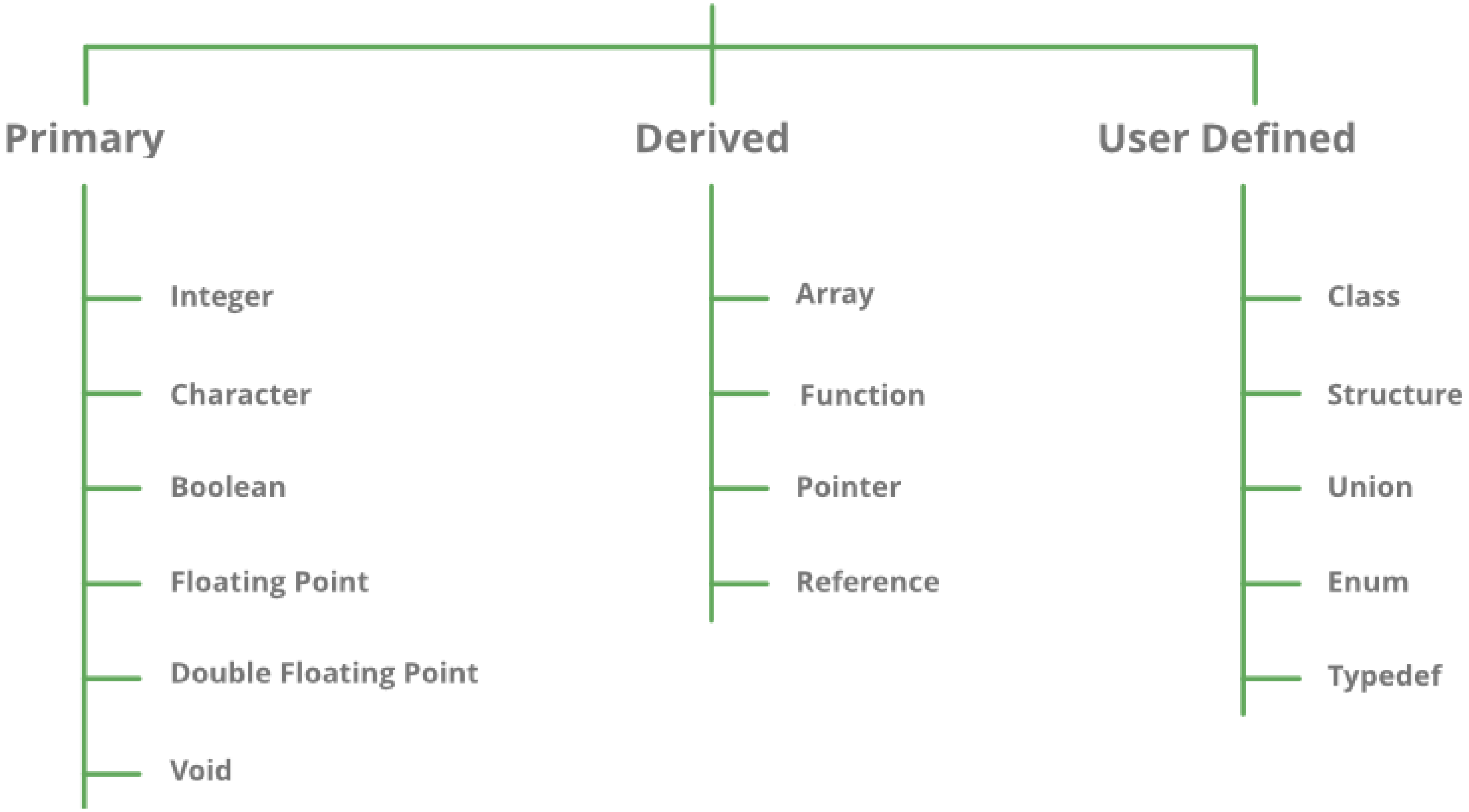
ACTS, C-DAC Bangalore

Topics Covered:

- Data Types in C++
- Arrays
  - 1D Array
  - 2D Array
- Search & Sort Operation in Array
- Sparse Matrix

Data Types in C++

DataTypes in C / C++



Primary

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Void

Derived

- Array
- Function
- Pointer
- Reference

User Defined

- Class
- Structure
- Union
- Enum
- Typedef

Array

1-D Array

Arrays

- a set of homogeneous data items
- stored as a common name
- contiguous block of memory gets allocated for the array
- array is a derived data type
- each of the element referenced by an index value
- the array indexing starts from 0 ; last element index is 1 less than the size of the array as the index value starts from 0

Use Cases For Array

- List of Employees in Organization
- Test Scores of a class of students
- List of temperatures recorded every hour in a day, or a month, or a year.
- List of products and their cost sold by a store

Types of Array

- Single Dimension Array
  - A list of item can be given one variable name using only one subscript and such a variable is called a single subscripted variable or One- dimensional array
  - Example: List
- Two Dimensional Array
  - Data stored in the form of table is categorized as 2D Array.
  - Example: Matrix
- Multi Dimensional Array
  - More than 2 dimensions.
  - Example: JSON objects

Single Dimensional Array

- List form of data structure
- consists of single dimension
- Consider the example of marks scored by student in 6 subjects. This can be represented in array as : float marks[6] where 6 is the size of array marks is name of array and will store float type values.

Array Declaration : 1D Array

- Syntax:

  datatype array-name[size].

- Example:

  ```
  int arr[5];
  ```

- Declaration can be optionally done along with the initialization as:

  ```
  int arr[5] = {1,2,3,4,5}
  ```

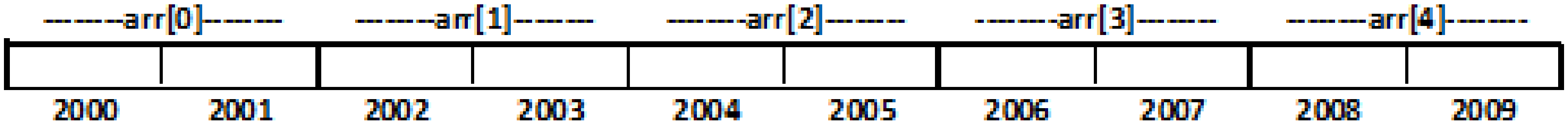
- Declaration statement provides the type of value it stores, the number of values that can be stored and the size of memory to be allocated for array.

Storage & Access of Array : 1D Array

- Consider array:

  int arr[5];

- Since int type, 2B for each element; total 5 elements then total memory allocated is 5*2 = 10B
- Each elements is referenced with its index value as arr[0], arr[1], arr[2], arr[3], arr[4]
- Elements are stored in contiguous block assuming from address 2000 to 2009
- Each element of array has a default garbage value stored



-------arr[0]------- -------arr[1]------- -------arr[2]------- -------arr[3]------- -------arr[4]-------

2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

Initialization (Storing Values) of Array: 1D Array

- At Compile Time
  1) Complete array initialization.
     - **all the locations** of an array is assigned with some value during declaration.
     - Example: int arr[5]={10,15,1,3,20};

10	15	1	3	20
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]

  2) Partial array initialization.
     - **few locations** from starting location of an array is assigned with some value during declaration.
     - Example: int a[5]={10,15};

10	15	0	0	0
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]

  3) Initialization without size.
     - size is not specified during the declaration but the values are specified.
     - Based on value array size is allocated
     - Example: int arr [ ]={10, 15, 1, 3, 20};

10	15	1	3	20
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]

  4) String initialization.
     - string is initialized to the array and the data type will be character only.
     - Example: char str[6]="GRAPH";

'G'	'R'	'A'	'P'	'H'	\0
str[0]	str[1]	str[2]	str[3]	str[4]	str[5]

- At Run Time
  - Using Loops, of assignment statement for each index, values at run time be initialized as user input.

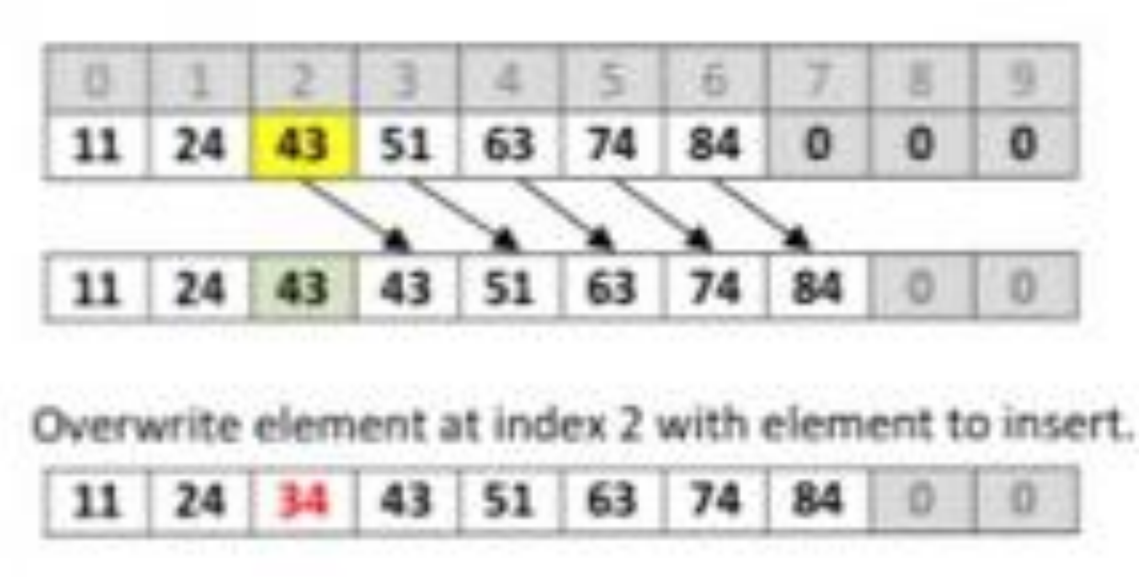
Operations With Array

- Read & Display the array
- Inserting element to array
- Deleting element from array
- Searching element in array
- Sorting array

Read & Display Array : 1D Array

```
int arr[MAX], n, i;
cout<<"Enter the size of the array : ";
cin>>n;

cout<<"\nReading the elements to an array :";
for (i = 0; i < n; i++)
{
    cout<<"\narr[%d] = ", i;
    cin>>arr[i];
}

cout<<"Displaying the %d elements of the array", n;
for (i = 0; i < n; i++)
{
        cout<<arr[i];
}
```

Inserting Element in Array



0	1	2	3	4	5	6	7	8	9
11	24	43	51	63	74	84	0	0	0

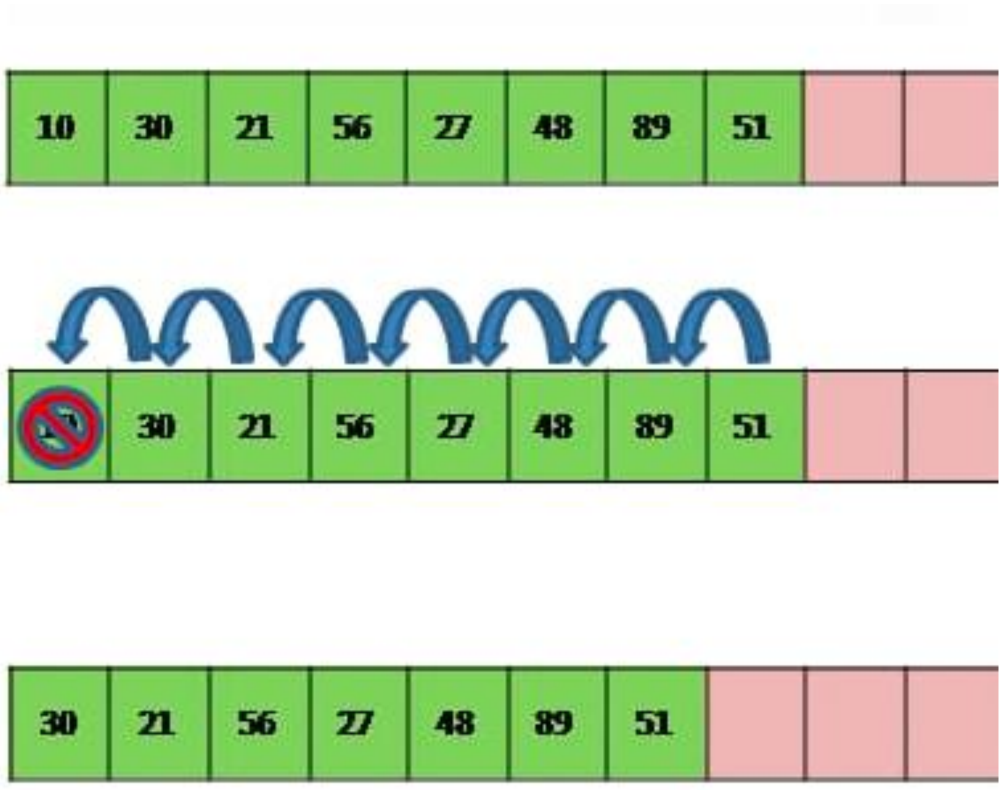
11	24	43	43	51	63	74	84	0	0

Overwrite element at index 2 with element to insert.

11	24	34	43	51	63	74	84	0	0

```
int arr[MAX], n, i;
int item, pos;

if (pos >= n)
{
    cout<<"\nPosition entered is not
    valid! Inserting at end of array!";
    pos = n-1;
}
for (int i = n - 1; i >= pos; i--)
        arr[i + 1] = arr[i];
n++;
arr[pos] = item;
```

Deleting Element From Array : 1D Array



10	30	21	56	27	48	89	51		
🚫	30	21	56	27	48	89	51		
30	21	56	27	48	89	51			

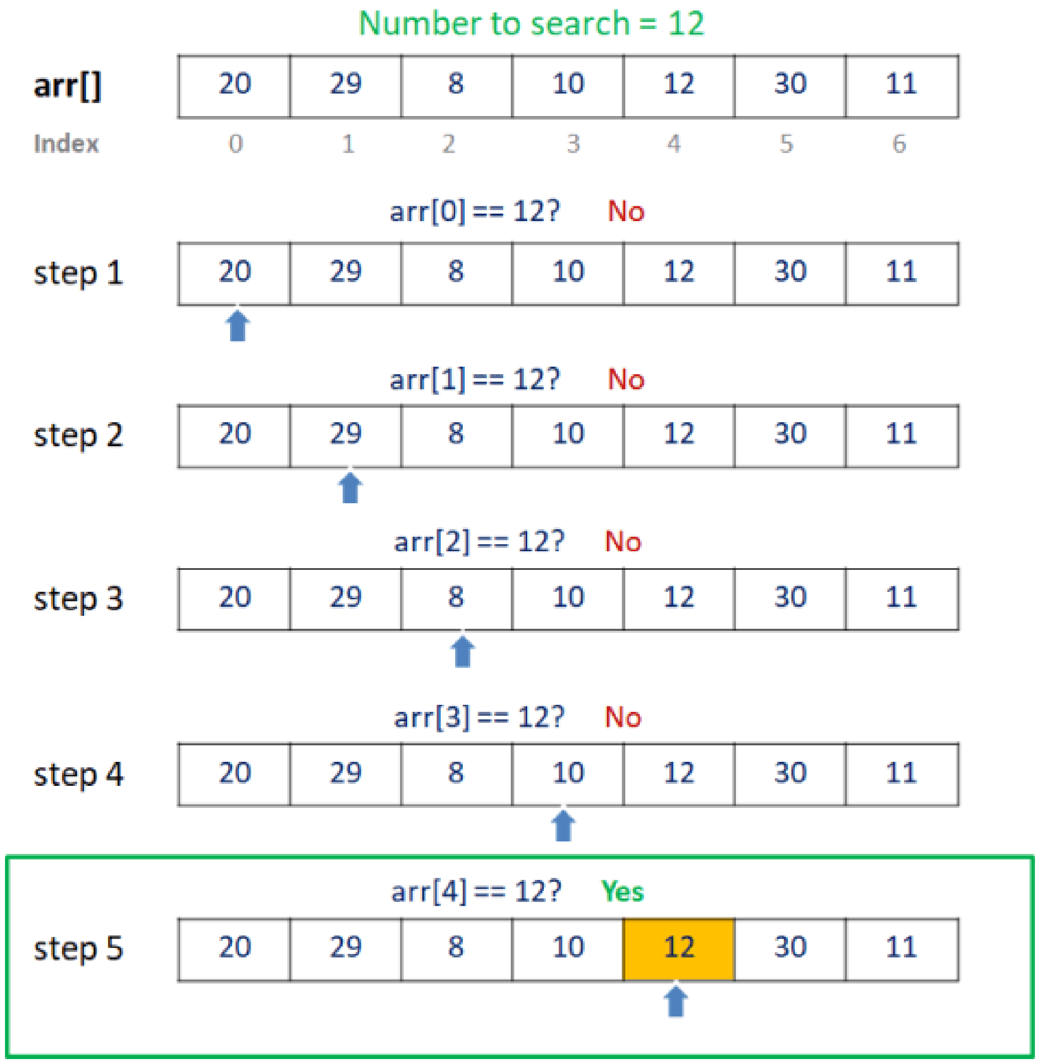
```
int arr[MAX], n, i;
int item, pos;

if (pos >= n)
{
    cout<<"\nPosition entered is not valid!
    Deleting from end of array!";
    pos = n - 1;
}

item = arr[pos];

for (int i = pos; i < n; i++)
        arr[i] = arr[i+1];
n--;
```

Searching Element in 1D Array

- Linear Search
  - Start from beginning, search by comparing elements one by one

- Binary Search
  - Find the middle value and keep comparing till either element is found or list exhausted
  - List should be sorted for the binary search to be performed

Linear Search



Number to search = 12

arr[]	20	29	8	10	12	30	11
Index	0	1	2	3	4	5	6

arr[0] == 12? No

step 1	20	29	8	10	12	30	11

arr[1] == 12? No

step 2	20	29	8	10	12	30	11

arr[2] == 12? No

step 3	20	29	8	10	12	30	11

arr[3] == 12? No

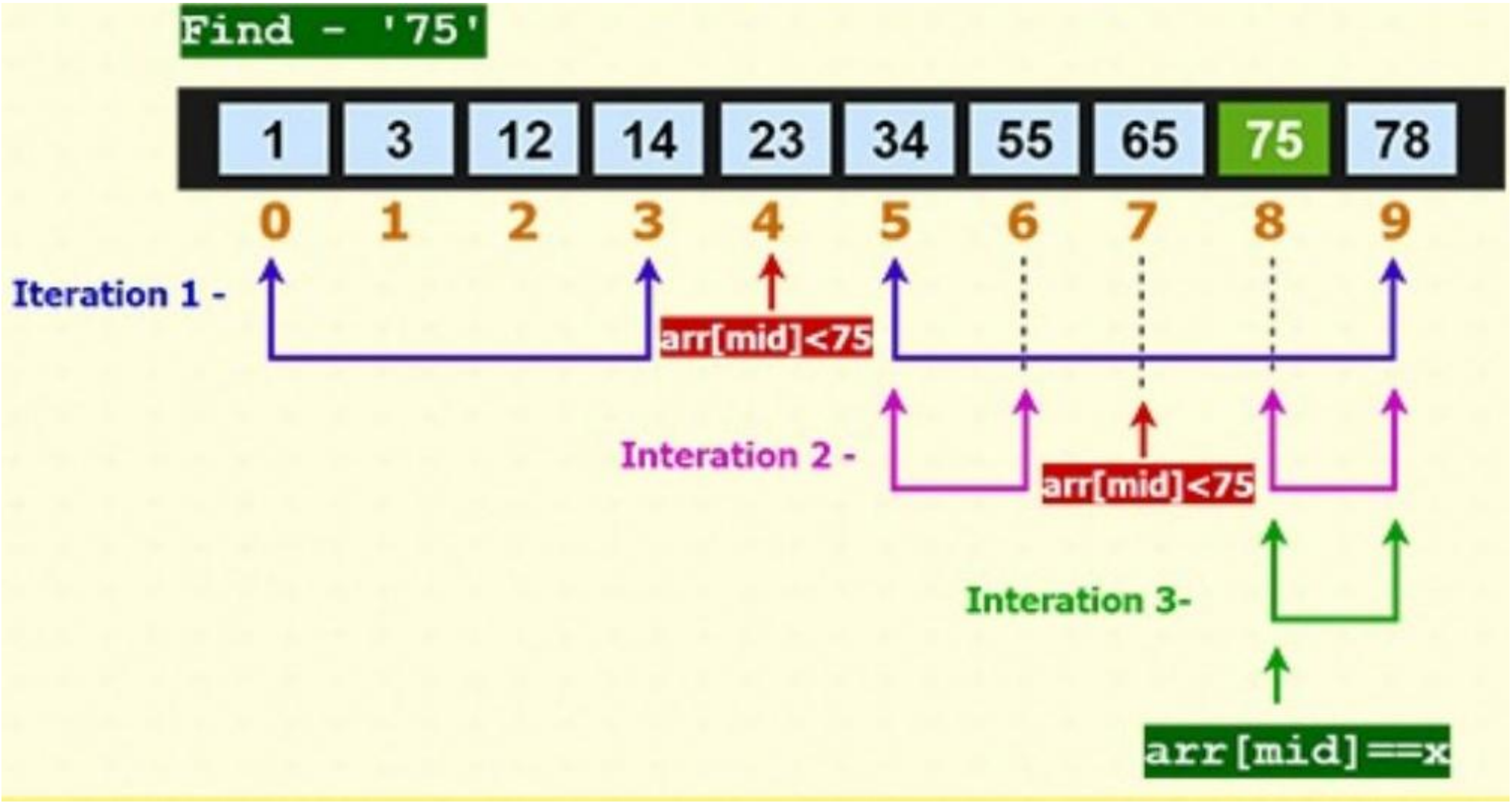
step 4	20	29	8	10	12	30	11

arr[4] == 12? Yes

step 5	20	29	8	10	12	30	11

```
int arr[MAX], n, i, item, pos, isfound = 0;

for (i = 0; i < n; i++)
{
    if (arr[i] == item)
    {
        isfound = 1;
        pos = i;
        break;
    }
}
if (isfound == 1)
{
    cout<<"\nElement found";
}
else
{
    cout<<"\nElement not found";
}
```

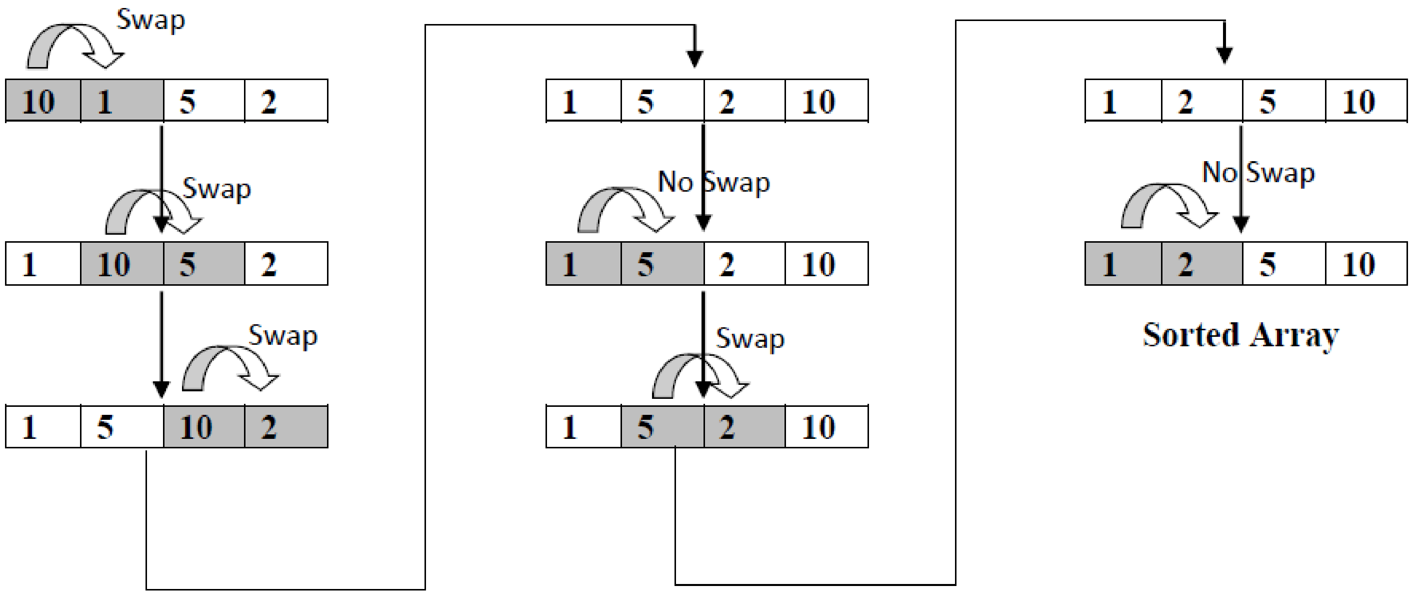
Binary Search



1	3	12	14	23	34	55	65	75	78
0	1	2	3	4	5	6	7	8	9

```
int arr[MAX], n, i, item;
int pos, isfound = 0;
int low, high, mid;
low = 0;
high = n - 1;
do{
    mid = (low + high) / 2;
    if (item == arr[mid])
    {
        isfound = 1;
        pos = mid;
        break;
    }
    else if (item < arr[mid])
        high = mid - 1;
    else
        low = mid + 1;
} while (low <= high);
if (isfound == 1)
    cout<<"\nElement found";
else
    cout<<"\nElement not found";
```

Bubble Sort : 1D Array



```
int arr[MAX], n, i, j, temp;

cout<<"\nPerforming Bubble Sort";

for (i = 0; i < n - 1; i++)
{
    for (j = 0; j < n - i - 1; j++)
    {
        if (arr[j] > arr[j + 1])
        {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}
```

Array

2-D Array

Two Dimensional Array

- tabular representation of data in terms of two dimensions namely rows and columns
- Each element of the array is accessed by two index values namely rows and columns
- Memory allotted for 2D array is a collection of contiguous block of memory of size governed by type and rows and columns in 2D Array
- Some examples are working with matrix based operations.
- Eg: Marks in 5 subjects for a set of 50 students

2D Array : Declaration

- Syntax:

  <datatype> <array-name>[<row-count>][<column-count>];

- Size of Memory Block:

  size = sizeof(datatype) * row-count * column-count;

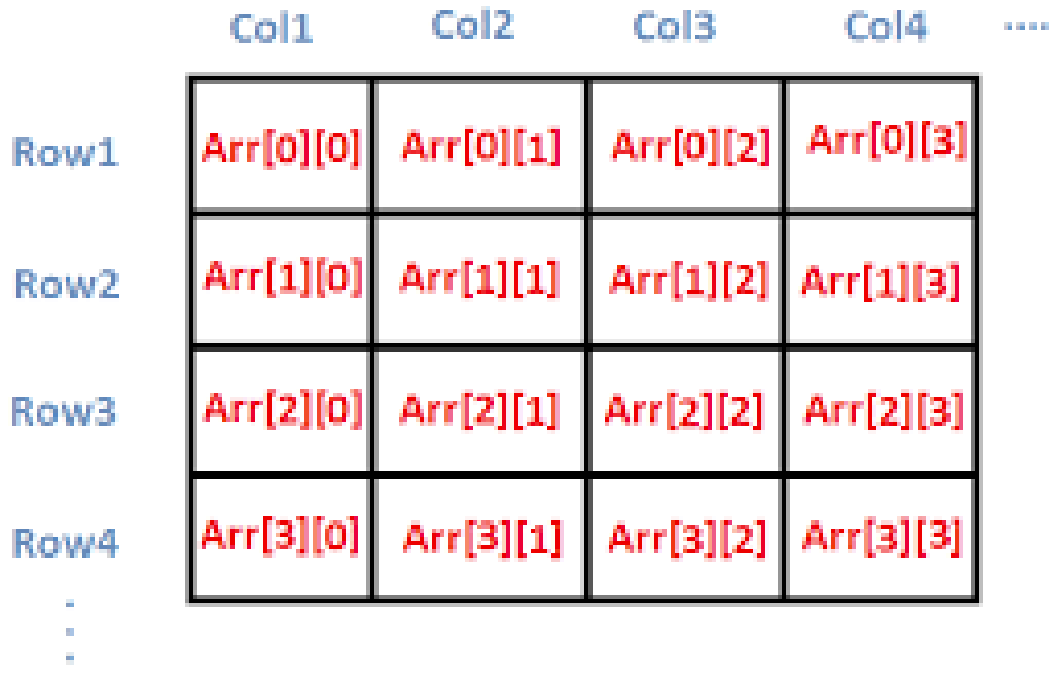
- Example:

  int matrix[2][3];

  This consists of 2 rows and 3 columns for elements referred by matrix & stores element of type int. Size of memory allotted : 2 * 3 * 2 = 12Bytes

2D Array Representation

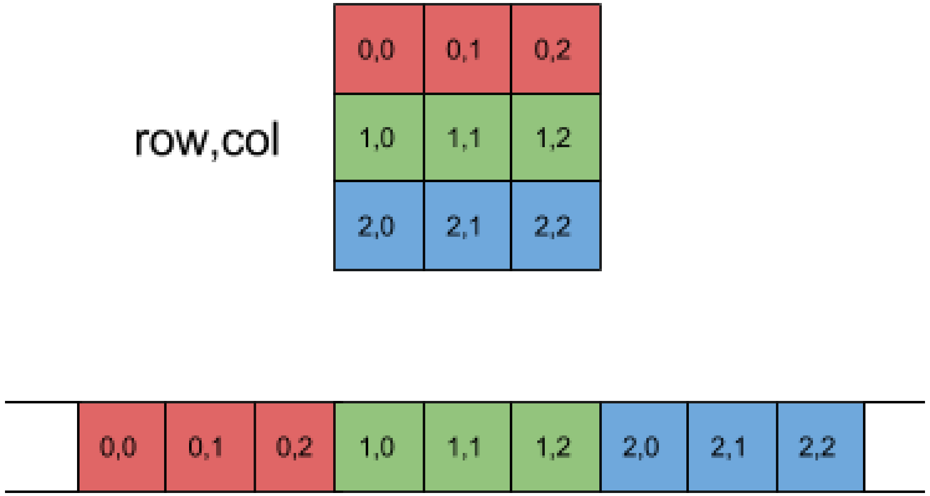
- Each of the array element is represented by the array name followed by the index value for row and index value of column.
- Woking with 2D array involves scrolling across every column for each of the row.
- Storage of array in memory can be in Row major or column major form



	Col1	Col2	Col3	Col4	....
Row1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]	
Row2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]	
Row3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]	
Row4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]	
⋮					

2D Array Storage

Row Major Representation



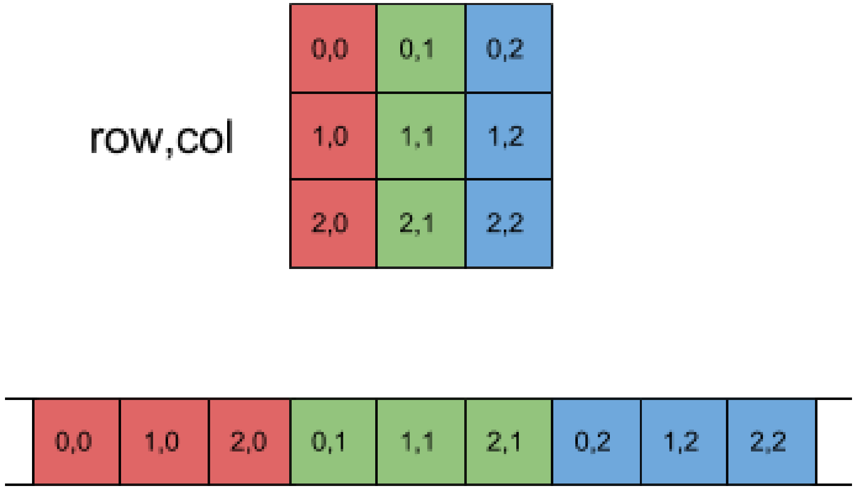
row,col

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2

$$\mathrm{loc}(a[i][j]) = \mathrm{base}(a) + w(n *i +j)$$

Column Major Representation



row,col

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	1,0	2,0	0,1	1,1	2,1	0,2	1,2	2,2

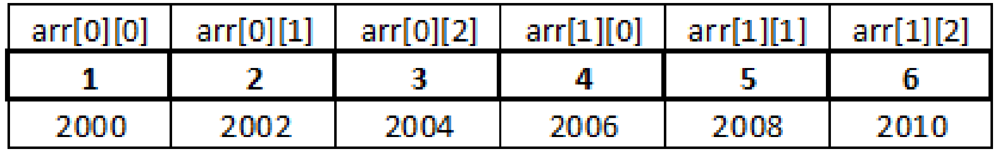
$$\mathrm{loc}(a[i][j]) = \mathrm{base}(a) + w(m *j +i)$$

base(a): base address in pointer a ; **w:** size of each element ; **m:** total rows in array a ; **n:** total columns in array a

2D Array: Initialization

- At Compile Time
  - Complete Array Initialization

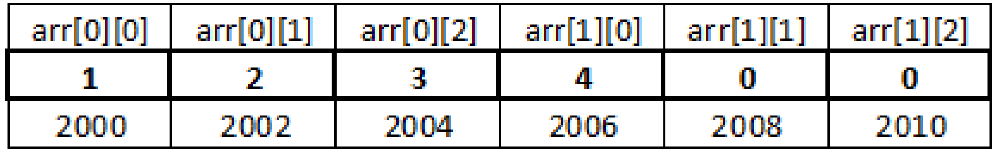
```
int arr[2][3] = {{1,2,3},{4,5,6}};
int arr[2][3] = {1,2,3,4,5,6};
int arr[][3] = {1,2,3,4,5,6};
```



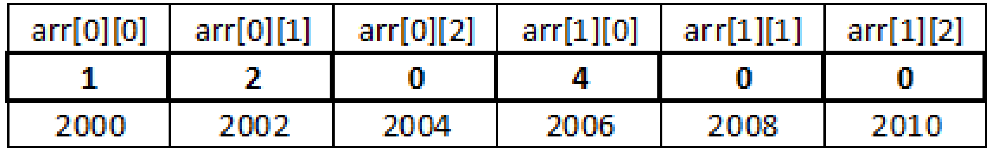
arr[0][0]	arr[0][1]	arr[0][2]	arr[1][0]	arr[1][1]	arr[1][2]
1	**2**	**3**	**4**	**5**	**6**
2000	2002	2004	2006	2008	2010

  - Partial Array Initialization

```
int arr[2][3] = {1,2,3,4};
int arr[2][3] = {{1,2},{4}};
```



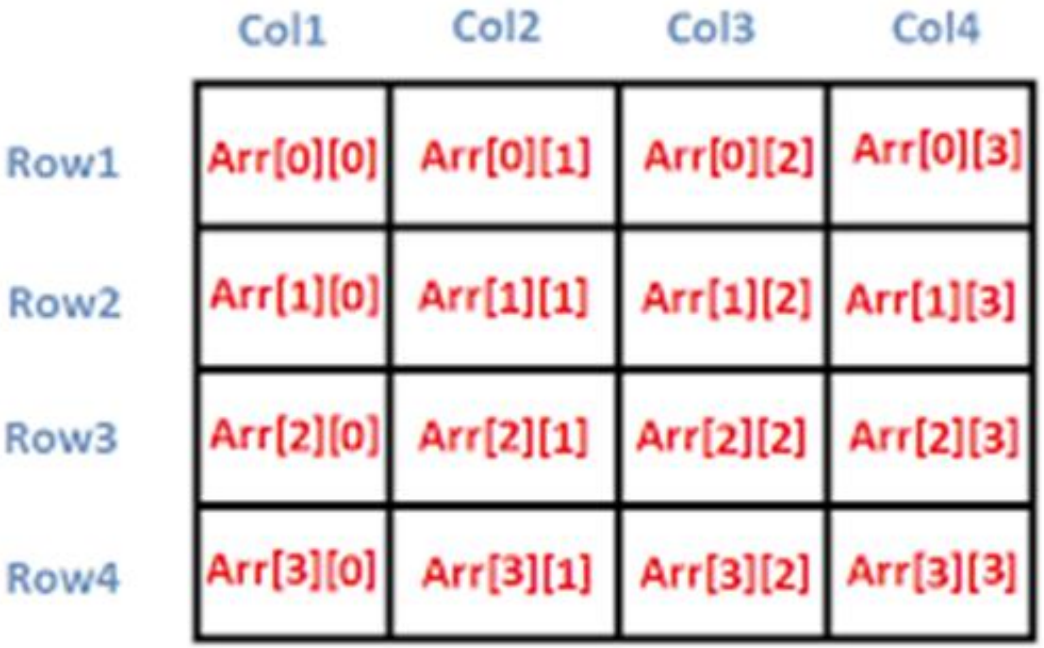
arr[0][0]	arr[0][1]	arr[0][2]	arr[1][0]	arr[1][1]	arr[1][2]
1	**2**	**3**	**4**	**0**	**0**
2000	2002	2004	2006	2008	2010



arr[0][0]	arr[0][1]	arr[0][2]	arr[1][0]	arr[1][1]	arr[1][2]
1	**2**	**0**	**4**	**0**	**0**
2000	2002	2004	2006	2008	2010

- At Run Time
  - Using input statement inside 2 loops (one for row, other for column)

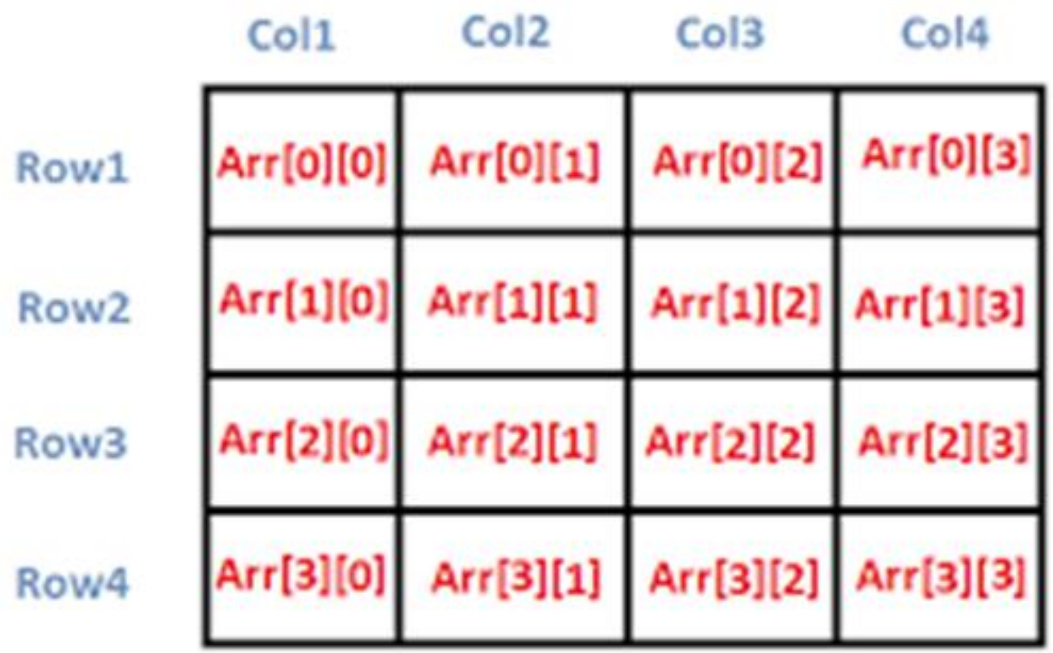
2D Array: Reading and Displaying Elements



	Col1	Col2	Col3	Col4
Row1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]
Row2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]
Row3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]
Row4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]

```
int Arr[ROW_MAX][COL_MAX], row, col, i, j;

cout<<"Enter the size of matrix :";
cout<<"\n Rows : ";
cin>>row;//4
cout<<"\n Columns : ";
cin>>col; //4

cout<<"Enter the “<< row*col <<“array
elements\n";

for (i = 0; i < row; i++)
{
    for (j = 0; j < col; j++)
    {
        cin>>Arr[i][j];
    }
}
```

2D Array: Reading and Displaying Elements



	Col1	Col2	Col3	Col4
Row1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]
Row2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]
Row3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]
Row4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]

```
int Arr[ROW_MAX][COL_MAX], row, col, i, j;

for (i = 0; i < row; i++)
{
    for (j = 0; j < col; j++)
    {
        cout<<"%d ", Arr[i][j];
    }
    cout<<"\n";
}
```

Example Programs With 2D Array

- Matrix Addition
- Matrix Multiplication
- Matrix Transpose

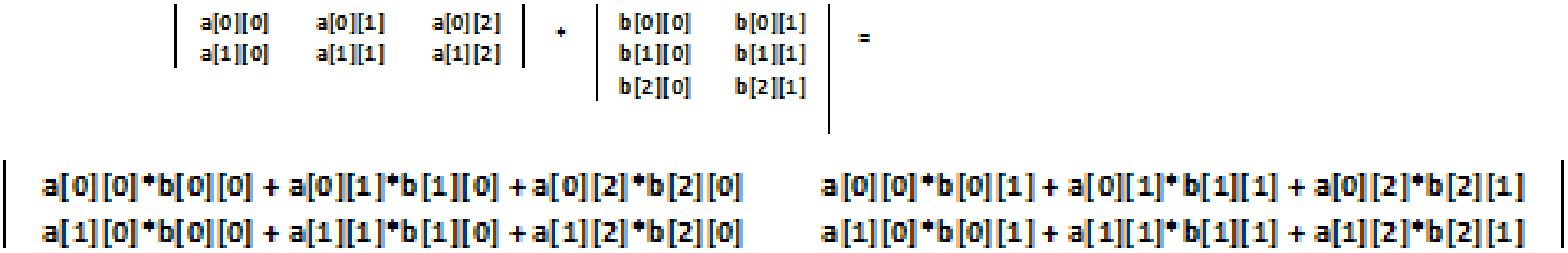
Matrix Addition

$$\overset{A}{\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}} \pm \overset{B}{\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}}$$

$$= \overset{A \pm B}{\begin{pmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & a_{13} \pm b_{13} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & a_{23} \pm b_{23} \\ a_{31} \pm b_{31} & a_{32} \pm b_{32} & a_{33} \pm b_{33} \end{pmatrix}}$$

```
if (row1 != row2 || col1 != col2)
{
    cout<<"Matrix Addition Not Possible!";
}
else
{
    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < col1; j++)
        {
            matadd[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}
```

Matrix Multiplication

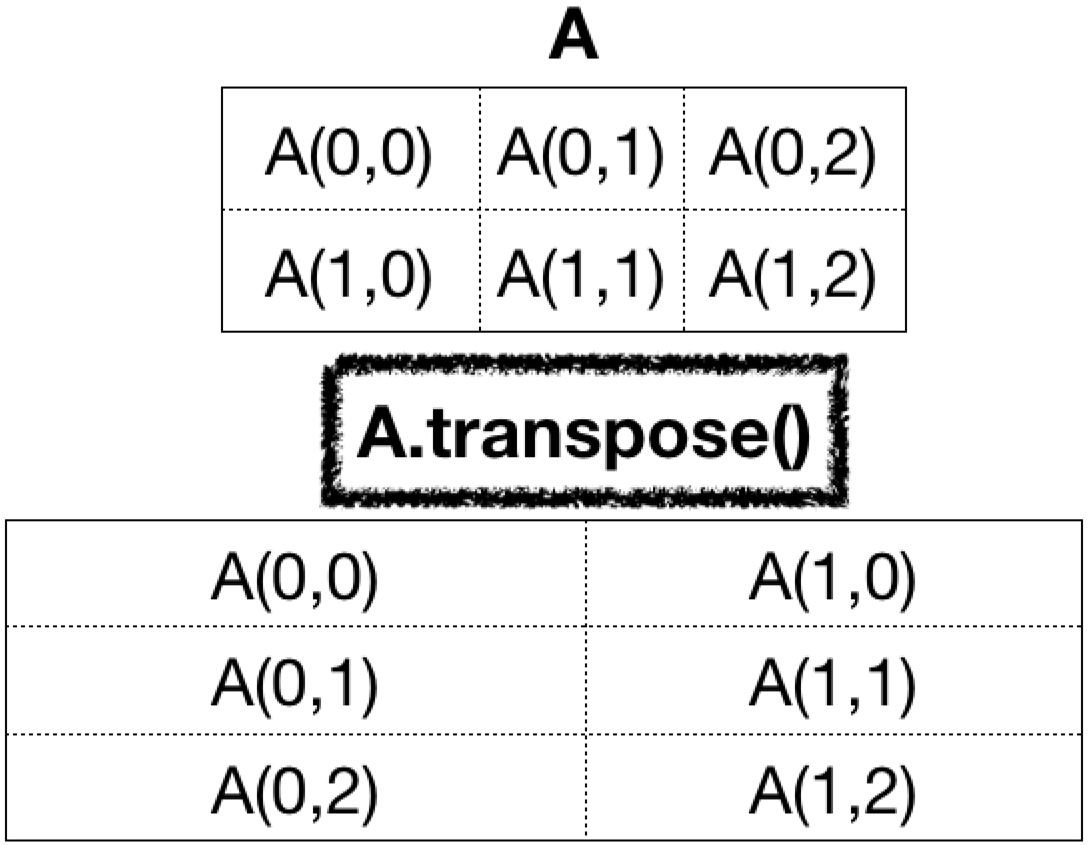


$$\begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \end{vmatrix} * \begin{vmatrix} b[0][0] & b[0][1] \\ b[1][0] & b[1][1] \\ b[2][0] & b[2][1] \end{vmatrix} =$$

$$\begin{vmatrix} a[0][0]*b[0][0] + a[0][1]*b[1][0] + a[0][2]*b[2][0] & a[0][0]*b[0][1] + a[0][1]*b[1][1] + a[0][2]*b[2][1] \\ a[1][0]*b[0][0] + a[1][1]*b[1][0] + a[1][2]*b[2][0] & a[1][0]*b[0][1] + a[1][1]*b[1][1] + a[1][2]*b[2][1] \end{vmatrix}$$

```
for (i = 0; i < row1; i++)
{
    for (j = 0; j < col2; j++)
    {
        matmul[i][j] = 0;
        for (k = 0; k < col1; k++)
        {
            matmul[i][j] += mat1[i][k] * mat2[k][j];
        }
    }
}
```

Matrix Transpose



A

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)

A.transpose()

A(0,0)	A(1,0)
A(0,1)	A(1,1)
A(0,2)	A(1,2)

```
for (i = 0; i < col; i++)
{
    for (j = 0; j < row; j++)
    {
        mattrans[i][j] = mat[j][i];
    }
}
```

Sparse Matrix

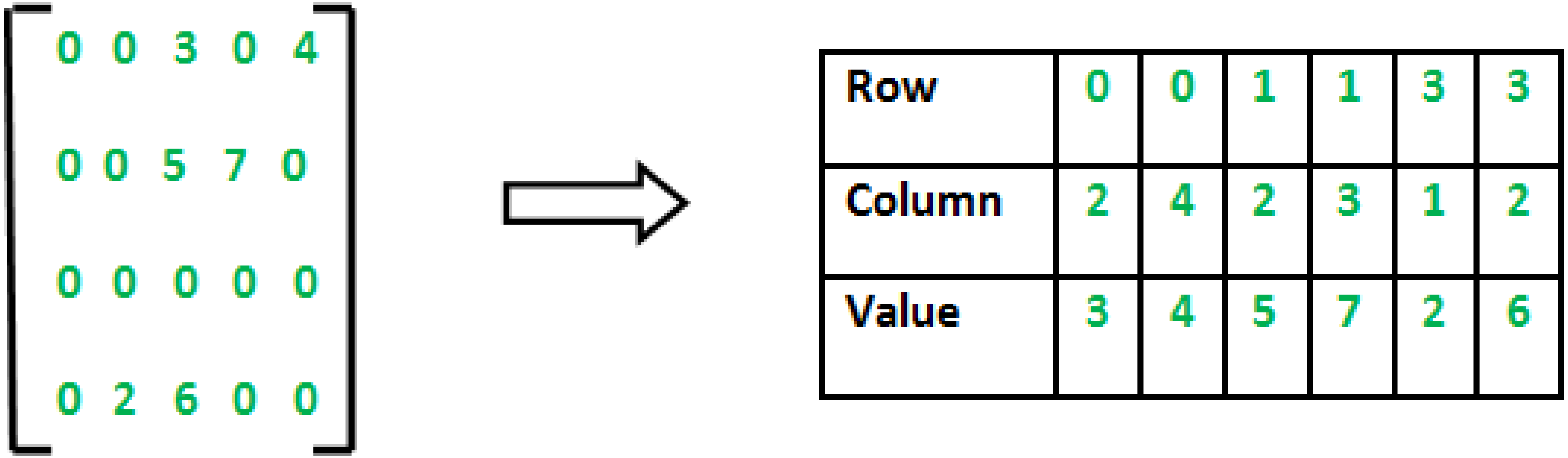
- matrix where maximum elements are zero (0)
- representing such matrix as a normal array representation in form of rows and columns involves wastage of memory as maximum memory is used storing non essential information.
- This kind of matrix can be representation in a different manner using array

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

<u>Using Normal Matrix Representation</u>

Total Memory Assigned: 4*5* sizeof(int)
= 4*5*2 = **40B**
Useful Data Space: 6*2 = **12B**
Space Wasted: 40B – 12B = **38B**

Sparse Matrix Representation



$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Representation of Sparse Matrix as Array consisting of 3 rows and multiple columns equal to number of non-zero elements