CS1002 – Programming Fundamentals

Lecture # 19
Wednesday, October 26, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

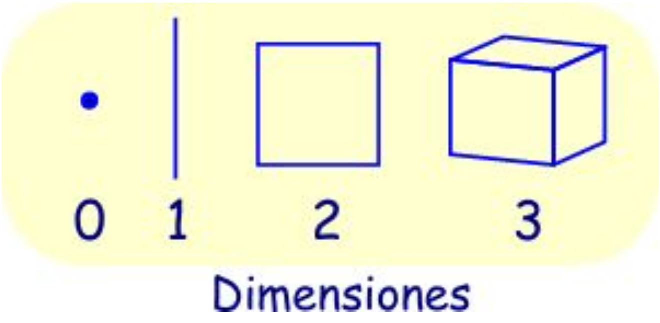
Muhammad Yousaf

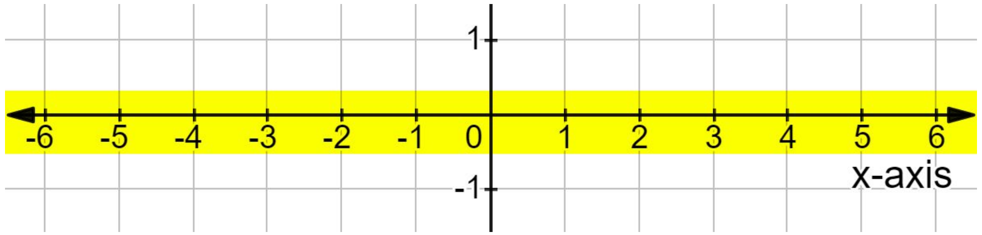
Outline

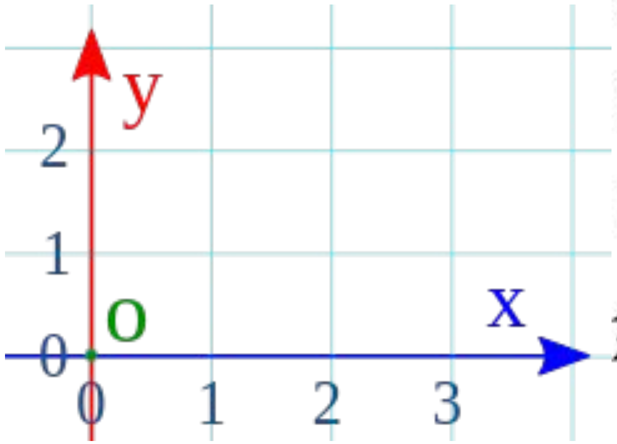
- One Dimensional Arrays (Previously)
- Two Dimensional Arrays
- Multi-Dimensional Arrays
- Operations on Multi-Dimensional Arrays

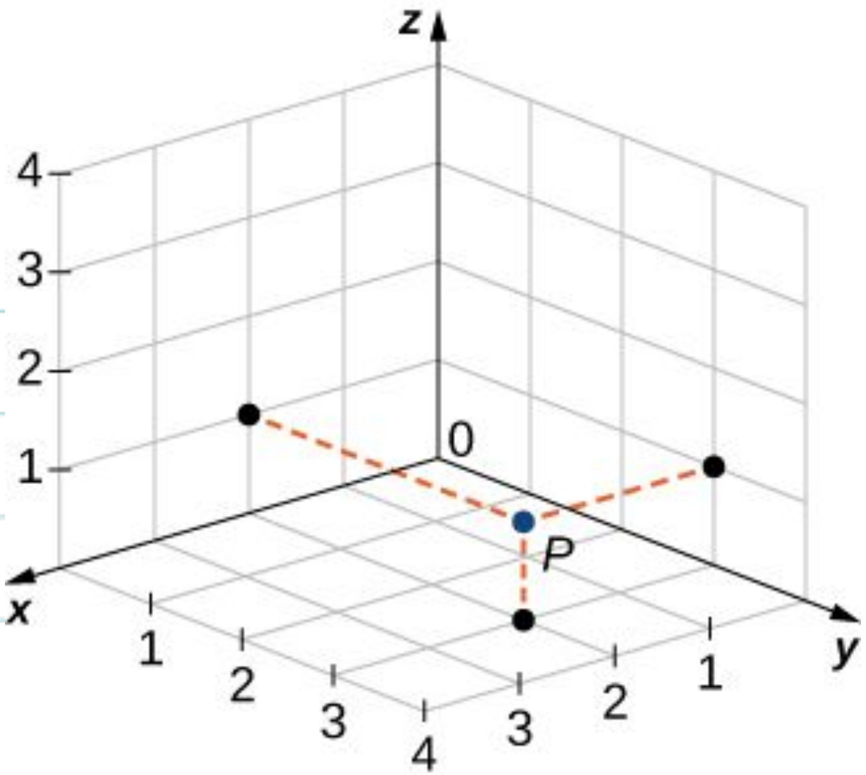
Maths: Cartesian Plane (Dimensions)

- Points on x, y and z-axis on cartesian plane.









1D point: 0,1,2,3

2D point: (0,0)

3D point: (0,0,0)

Multidimensional Arrays

- **Arrays** with more than one dimension are called multi-dimensional arrays.

```
dataType arrayName[size1][size2]....[sizeN];
```

  - data_type: Type of data to be stored in the array.
  - array_name: Name of the array
  - size1, size2,... ,sizeN: Sizes of the dimension
- Examples:
  - Two dimensional array: int two_d [10] [20];
  - Three dimensional array: int three_d [5] [10] [20];

Size of Multidimensional Arrays

- The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.
- The array int two_d [10] [20]; can store total (10*20) = 200 elements.
- Array int three_d [10] [20] [30]; can store total (5*10*20) = 1000 elements.
- Number of subscripts determines the dimensionality of the array.
  - X [i] refers to an element of one-dimensional array
  - X [i] [j] refers to an element of two-dimensional array
  - X [i] [j] [k] refers to an element of three-dimensional array

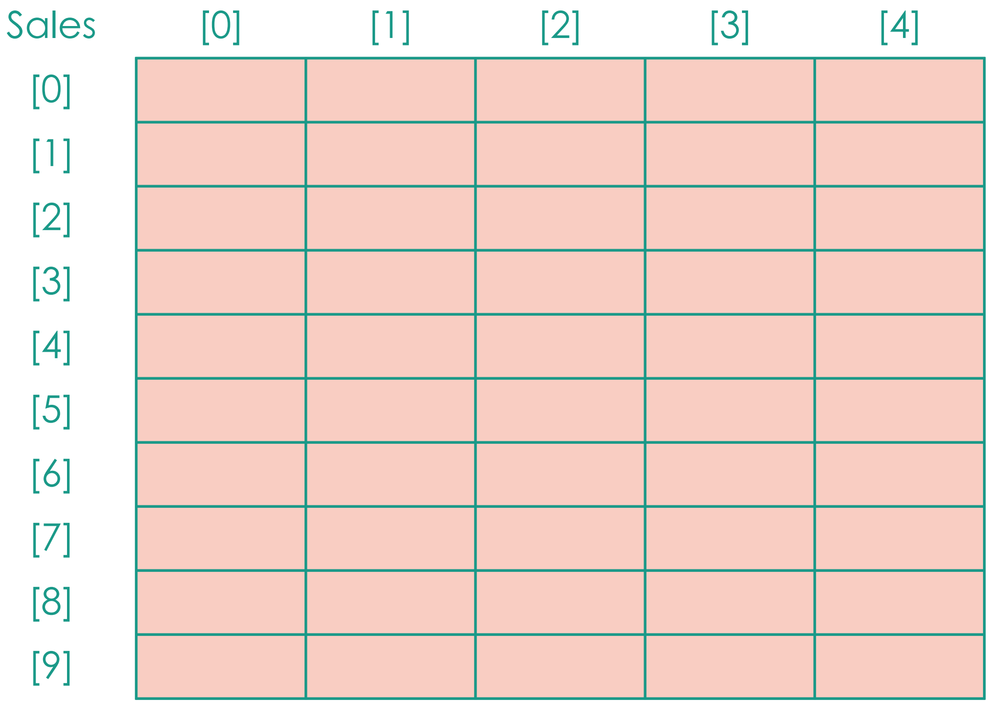
Two Dimensional Arrays

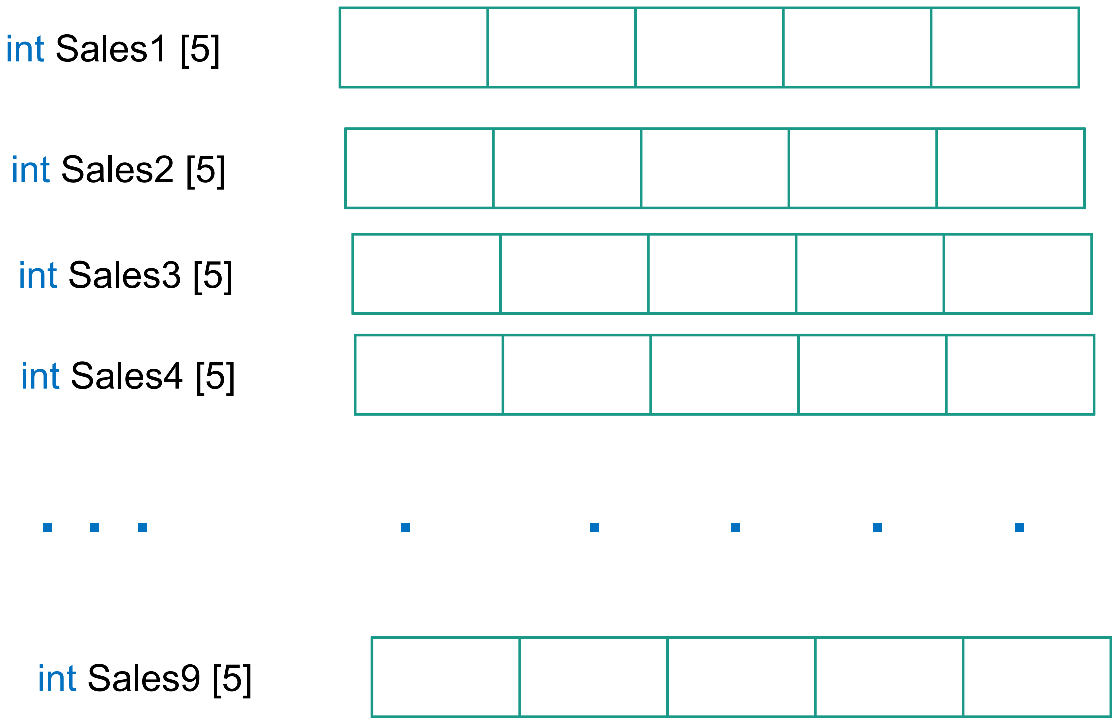
- **Two-dimensional array:** collection of a fixed number of components (of the same type) arranged in two dimensions
- Sometimes called matrices or tables
- Declaration syntax:

```
dataType arrayName[size1][size2];
```

- where **size1** and **size2** are expressions yielding positive integer values, and specify the **number of rows** and the **number of columns**, respectively, in the array

Comparison: 2D vs 1D arrays





int Sales1 [5]

int Sales2 [5]

int Sales3 [5]

int Sales4 [5]

. . . .

int Sales9 [5]

-Create **Ten** separate 1D arrays of size=5 with 10 different names?
OR
-Create 2D array of 10x5 with a single name?

Comparison: 2D vs 1D arrays

```
int inStock [ ] [ ];
```

inStock	[RED]	[BROWN]	[BLACK]	[WHITE]	[GRAY]
[GM]	10	7	12	10	4
[FORD]	18	11	15	17	10
[TOYOTA]	12	10	9	5	12
[BMW]	16	6	13	8	3
[SUZUKI]	10	7	12	6	4
[VOLVO]	9	4	7	12	11

-How many alternative 1D arrays?

-Size of each 1D array?

Accessing Array Components

- Syntax:

```
arrayName[indexDim1][indexDim2]
```

- Where **indexDim1** and **indexDim2** are expressions yielding nonnegative integer values, and specify the row and column position

Accessing Array Components (cont'd.)

Suppose that:

```
int i = 6;

int j = 2;
```

Then, the following statement:

```
    sales[6][2] = 69.85;
```

Is equivalent to:

```
    sales[i][j] = 69.85;
```

So the indices can also be variables.

Sales	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]					
[3]					
[4]					
[5]					
[6]			69.85		
[7]					
[8]					
[9]					

Two-Dimensional Array Initialization During Declaration

- Two-dimensional arrays can be initialized when they are declared:

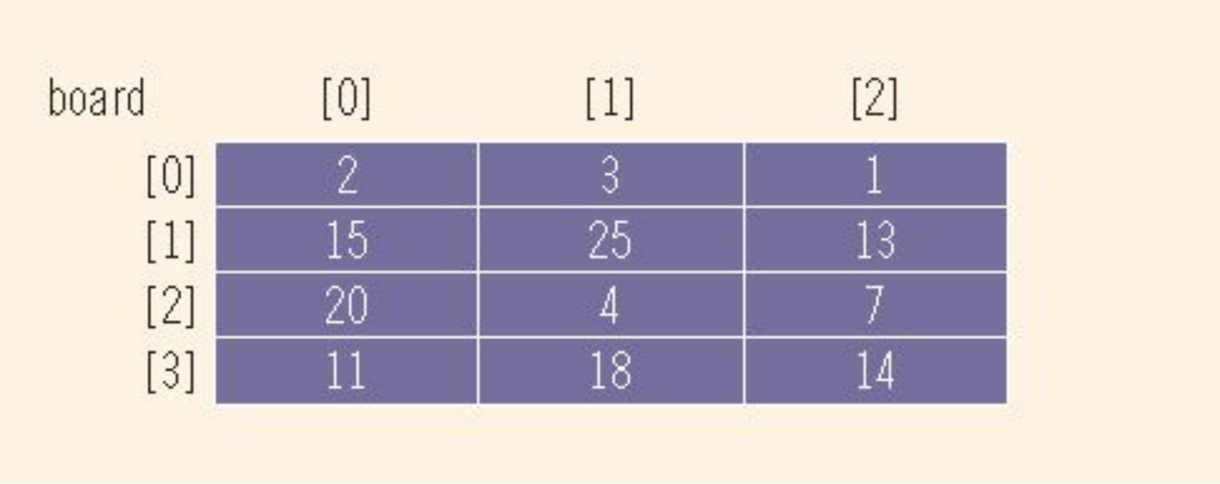
```
int board[4][3] = {{2, 3, 1},
          {15, 25, 13},
          {20, 4, 7},
          {11, 18, 14} };
```

- Elements of each row are enclosed within braces and separated by commas
- All rows are enclosed within braces
- For number arrays, if all components of a row aren’t specified, unspecified ones are set to 0

Example

```
int x[4][3] = {1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11, 12};

int board[4][3] = {   {2, 3, 1},
           {15, 25, 13},
           {20, 4, 7},
           {11, 18, 14}};
```



board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7
[3]	11	18	14

Processing Two-Dimensional Arrays

- Ways to process a two-dimensional array:
  - Process the entire array
  - Process a particular row of the array, called row processing
  - Process a particular column of the array, called column processing
- Each row and each column of a two-dimensional array is a one-dimensional array
  - To process, use algorithms similar to processing one-dimensional arrays

Processing Two-Dimensional Arrays (cont'd.)

```
const int NUMBER_OF_ROWS = 7;//This can be set to any number
const int NUMBER_OF_COLUMNS = 6;//This can be set to any number

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

Figure 9-15 shows Tw

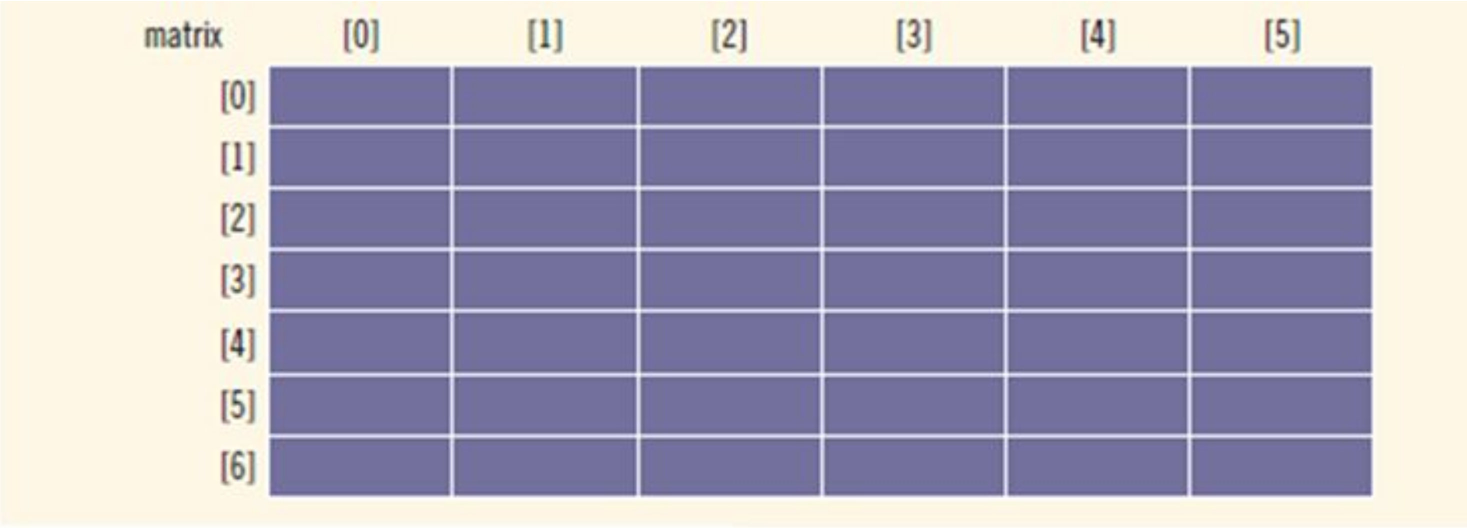


FIGURE 9-15 Two-dimensional array `matrix`

Initialization

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To initialize row number 5 (i.e., sixth row) to 0:

```
row = 5;
for(int col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
    matrix[row][col] = 0;
```

- To initialize the entire matrix to 0:

```
for(row = 0 ; row < NUMBER_OF_ROWS ; row++)
for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
    matric[row][col] = 0 ;
```

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

Print

- To print data from each component of matrix:

```
for(row = 0 ; row < NUMBER_OF_ROWS ; row++)
{
    for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
        cout << setw(5) << matrix[row][col] << “ ” ;
    cout << endl;
}
```

Input

```cpp
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To input data into each component of matrix:

```cpp
for(row = 0 ; row < NUMBER_OF_ROWS ; row++)

    for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)

        cin >> matrix[row][col] ;
```

Sum by Row

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To find the sum of row number 3 of matrix:

```
sum = 0 ;
row = 3 ;
for(col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
sum += matrix[row][col] ;
```

- To find the sum of each individual row:

```
//Sum of each individual row

for (row = 0; row < NUMBER_OF_ROWS; row++)
{
     sum = 0;
     for (col = 0; col < NUMBER_OF_COLUMNSS; col++)
          sum += matrix[row][col];
     cout << “Sum of Row ” << row + 1
          << “ = ” << sum << endl;
}
```

Sum by Column

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To find the sum of each individual column:

```
//Sum of each individual column
for (col = 0 ; col < NUMBER_OF_COLUMNS ; col++)
{
    sum = 0 ;
    for(row = 0 ;  row < NUMBER_OF_ROWS ; row++)
        sum += matrix[row][col] ;

    cout << “Sum of Column ” << col + 1
        << “ = ” << sum << endl ;
    }
```

Sum of Matrix

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

- To find the sum of complete matrix:

```
//Sum of complete matrix

sum = 0 ;

for (row = 0 ; row < NUMBER_OF_ROWS ; row++)

{

    for(col = 0 ;  col < NUMBER_OF_COLUMNS ; col++)

        sum += matrix[row][col] ;

}

cout << “Sum of Matrix = ” << sum << endl ;
```

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

Largest Element in Each Row

```
//Largest number in each row
for (row = 0 ; row < NUMBER_OF_ROWS ; row++)
{
    largest = matrix[row][0] ;
    for(col = 1 ;  col < NUMBER_OF_COLUMNS ; col++)
        if(matrix[row][col] > largest)
            largest = matrix[row][col];
    cout << “The largest element in row ” << row + 1
        << “ = ” << largest << endl;
}
```

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

Largest Element in Each Column

```
//Largest number in each col

for (col = 0 ; col < NUMBER_OF_COLS ; col++)

{

    largest = matrix[0][col] ;

    for(row = 1 ; row < NUMBER_OF_ROWS ; row++)

        if(matrix[row][col] > largest)

            largest = matrix[row][col];

    cout << “The largest element in col ” << col + 1

        << “ = ” << largest << endl;

}
```

Largest Element in Matrix

```
const int NUMBER_OF_ROWS = 7;
const int NUMBER_OF_COLUMNS = 6;

int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
int row;
int col;
int sum;
int largest;
int temp;
```

```
//Largest number in the matrix

largest = matrix[0][0] ;

for (row = 0 ; row < NUMBER_OF_ROWS ; row++)      {

    for(col = 0 ;  col < NUMBER_OF_COLUMNS ; col++)

        if(matrix[row][col] > largest)

            largest = matrix[row][col] ;

}

cout << “The largest element in matrix = ”

    << largest << endl;
```

Array Arithmetic

- If base address of the array is known the address of any index in the two dimensional array can be calculated.
- Address of arr[row][col] provided base address of the array is ‘b’ and size of data type is ‘s’ and columns per row are COLS
- Address of arr[row][col] = b + (row * COLS + col)* s
- Or arr[row][col] = b + row * COLS * s + col * s

e.g. for int Arr[5][6]; provided base address is 100

Address of Arr[1][2] = 100 + (1*6 + 2) * 4 = 100 + 32 = 132

Address of Arr[0][0] = 100 + (0*6 + 0) * 4 = 100 + 0 = 100

Address of Arr[4][0] = 100 + (4*6 + 0) * 4 = 100 + 96 = 196

Array Arithmetic

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

In an array

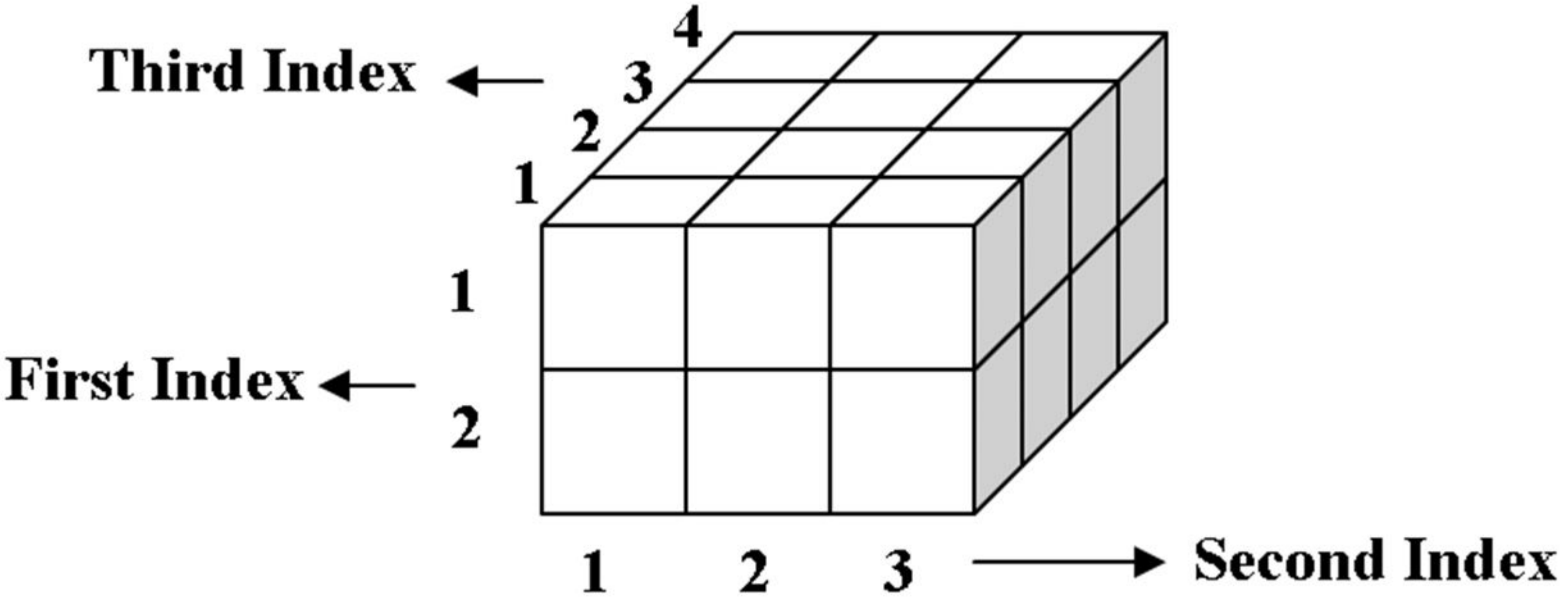
```
int arr[5][4];
```

To reach the cell number arr[2][3] it will have to travel

- Row number 0 and row number 1 completely
- Up to Column 3 in row number 2
- Lets assume the base address of the arrays is 100

arr[2][3] = 100 + (2 * 4) *4 + 3 * 4
= 100 + (2 * 4 + 3) * 4
= 100 + 44
= 144

Example



Three-dimensional array with twenty four elements

Multidimensional Arrays

- Multidimensional array: collection of a fixed number of elements (called components) arranged in n dimensions (**n >= 1**)
- Also called an n-dimensional array
- Declaration syntax:

```
dataType arrayName[intDim1][intDim2] … [intDimn];
```

- To access a component:

```
arrayName[indexDim1][indexDim2] … [indexDimn]
```

Example

For example
double carDealers[10][5][5];

- The base address of the array carDealers is the address of the first array component—that is, the address of carDealers[0][0][0]
- The total number of components in the array carDealers is
  - 10 * 5 * 5 = 250

```
carDealer[5][3][2] = 15009.65; // sets the value of
                               //carDealer[5][3][2] to 15009.65
```

- **Initialize all of the elements in array to 0.0**

```
for (int i = 0; i<10; i++)
    for (int j = 0; j<5; j++)
        for (int k = 0; k<5; k++)
            carDealer[i][j][k] = 0.0;
```

Questions