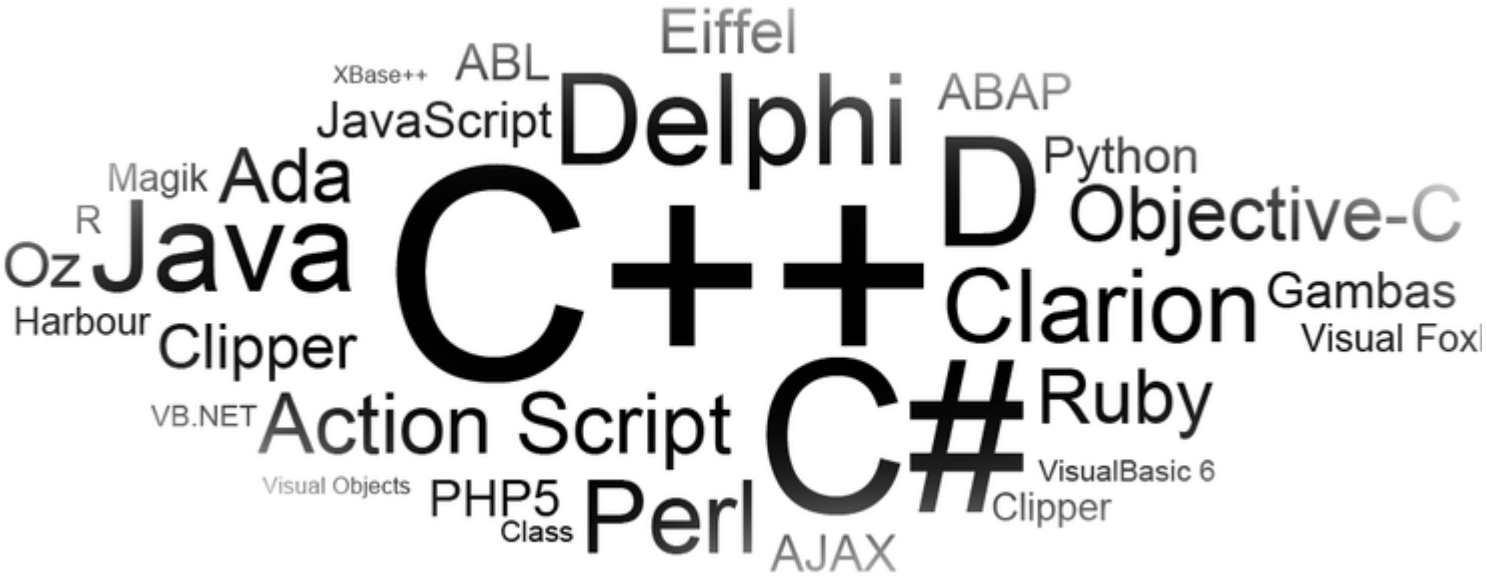
Programming Fundamentals II

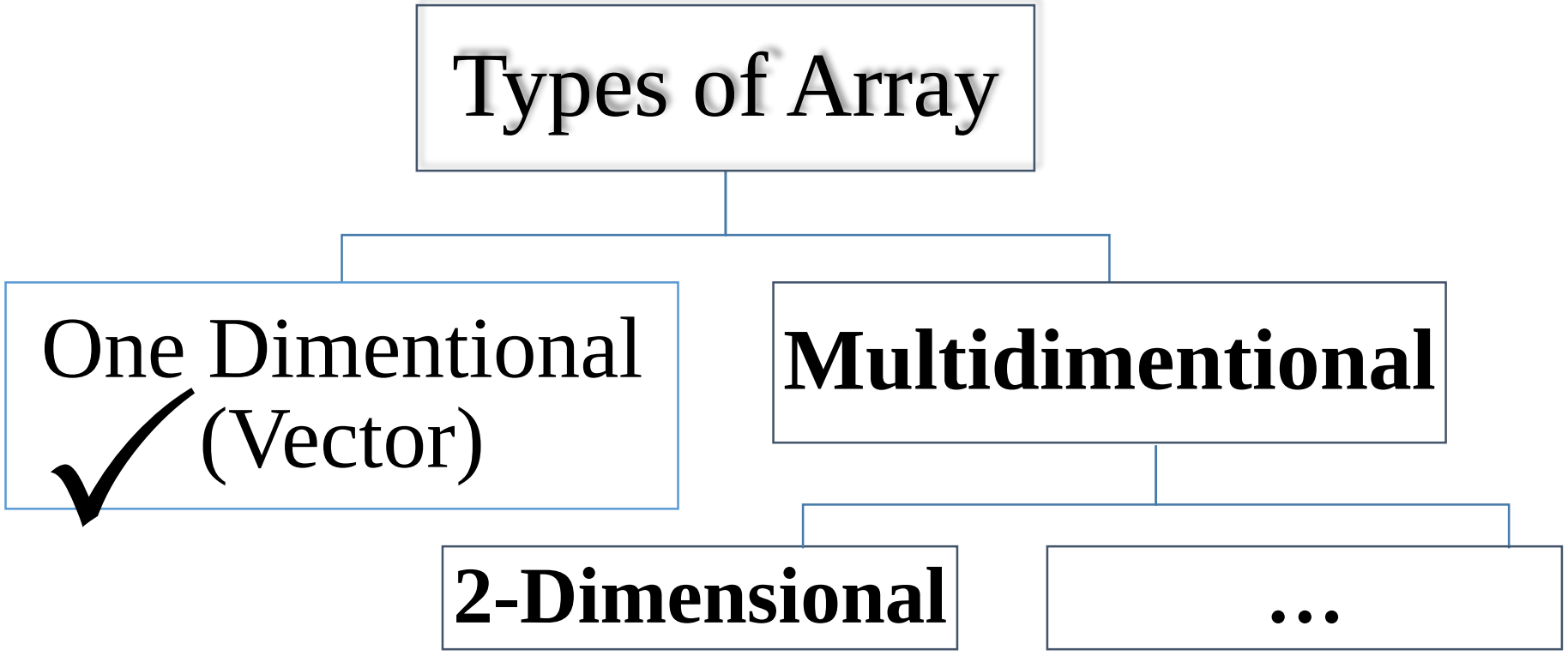
Lecture 5: Arrays



Outline

- Types of Arrays
  - 2-Dimensional Array
- 2-D Array Declarations
- 2-D Array Initialization
- Reading and Writing 2-D Arrays
- Examples

Arrays


Types of Array
One Dimentional (Vector)
Multidimentional
2-Dimensional
…

Two Dimensional Arrays

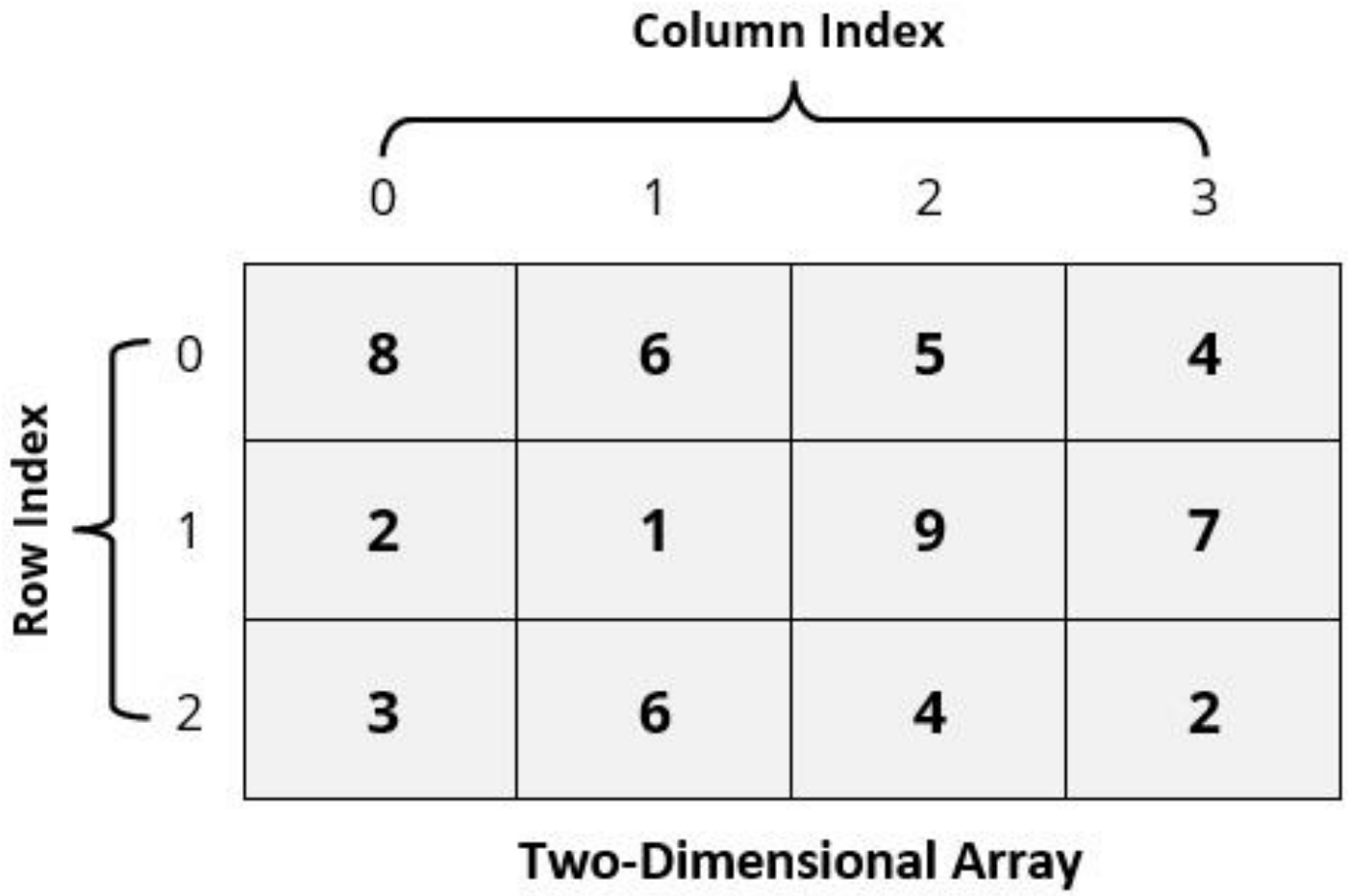
- The arrays we have used previously have all been *one-dimensional*, which means that they are *linear* or *sequential*.
- An array of arrays is called *multidimensional* array.
- A one-dimensional array of one-dimensional array is called two-dimensional array
  - Example:- double a[5][6]
- A one-dimensional array of two-dimensional arrays is called three-dimensional array, …etc.
  - Example:- double a[5][6][3]

Two Dimensional Arrays

(Matrix)

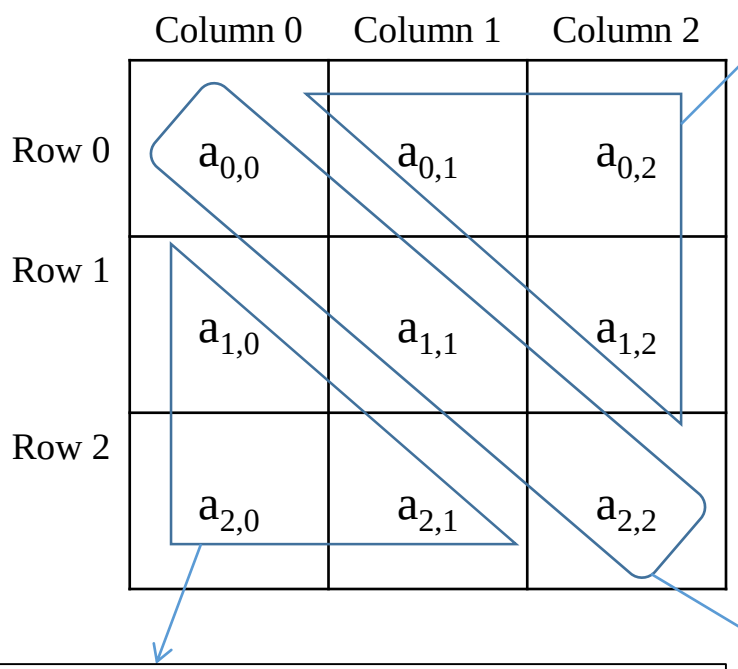
$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Two Dimensional Arrays



Two Dimensional Arrays

Square Matrix (A_{3x3})



Above Main diagonal i < j

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

Below Main diagonal i > j

Main diagonal i = j

Two Dimensional Arrays
2-D ARRAY STORAGE

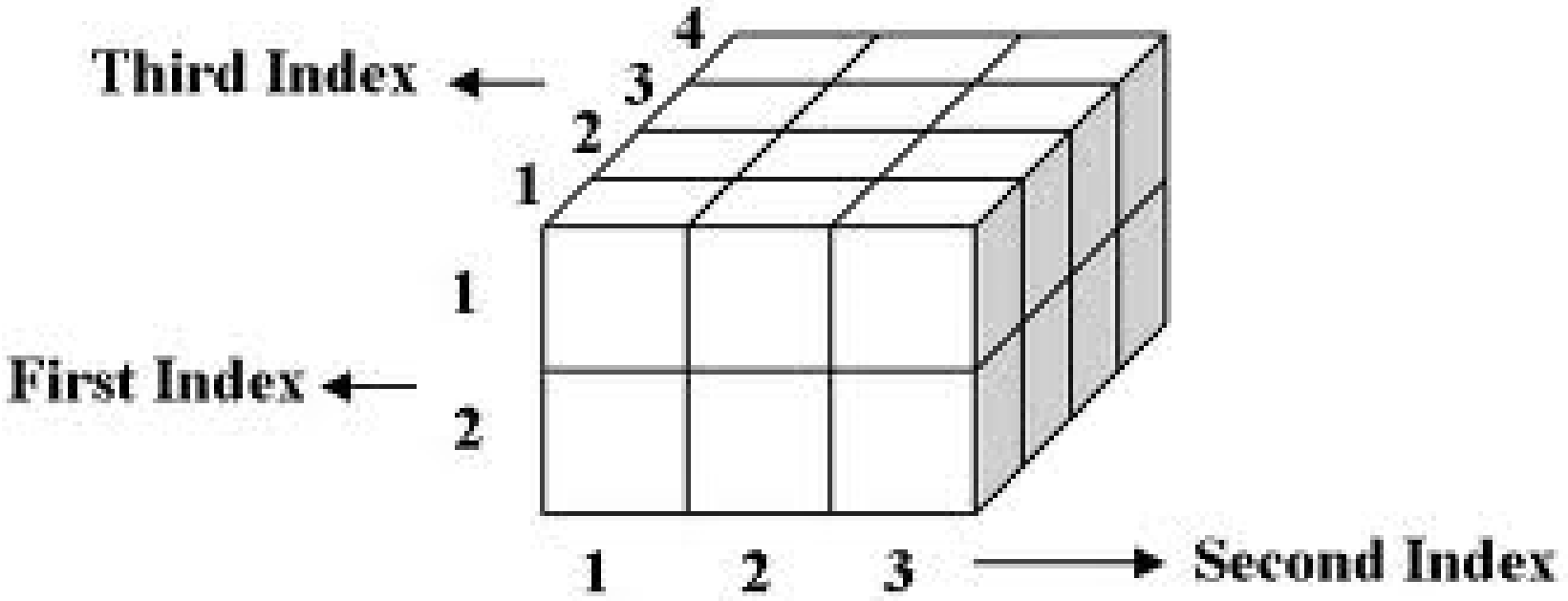
Conception of 2-D Array

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$

Actual Storage

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$

Three Dimensional Arrays



Three-dimensional array with twenty four elements

2-D Array Declaration

- An array must be declared before use.
- Syntax
  - *DataType identifier[integerConstant][integerConstant];*
- *DataType* is the type of each element
  - Can be simple or complex type, doesn't matter
- *identifier* is the name used to access the array
- *integerConstant* is a literal or named constant
  - Defines the number of elements in the array.

2-D Array Initialization

- Using two loops to initialize 2-D arrays, like following:

```
const  int   ROWS = 3;                  // Number of rows
const  int   COLUMNS = 2;               //  Number of columns
int   array[ROWS][COLUMNS];
for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLUMNS; j++){
        // set initial value to 0
        array[i][j] = 0;
    }
 }
```

- Initializing arrays at declaration with same size and number of elements of the array:

```
int array[ROWS][COLUMNS] = {{3,5},{7,4},{1,6}};
```

2-D Array Initialization

- Unassigned elements are initialized to 0:

```
int array[ROWS][COLUMNS] = {{3,5},{7,4}};
// array[0][0] = 3 , array[0][1] = 5
// array[1][0] = 7 , array[1][1] = 4
// array[2][0] = 0 , array[2][1] = 0
```

- Providing too many initial values is *compile-time error*:

```
// ERROR!
int array[ROWS][COLUMNS] = {{3,5},{7,4},{1,6},{8,9}};
```

Reading & Writing 2-D Arrays

- **Example 1**:- Write a C++ program that reads and prints out matrix (A_{nxm}).

```
//Declaration of the Matrix and its size
const  int  ROW_SIZE = 3;
const  int  COL_SIZE   = 3;
int  array[ROW_SIZE][COL_SIZE];
cout<<"Enter the Matrix elements\n";
//Reading the Matrix
for(int i = 0; i<ROW_SIZE; i++){
       for(int j = 0; j<COL_SIZE; j++){
          cin>>array[i][j];
       }
}
…
```

```
//Printing the Matrix
for(int i = 0; i<ROW_SIZE; i++){
       for(int j = 0; j<COL_SIZE; j++){
          cout<<array[i][j];
       }
       cout<<endl;
}
```

Reading & Writing 2-D Arrays

- **Example 1**:- Write a C++ program that reads and prints out matrix (A_{nxm}).

Output

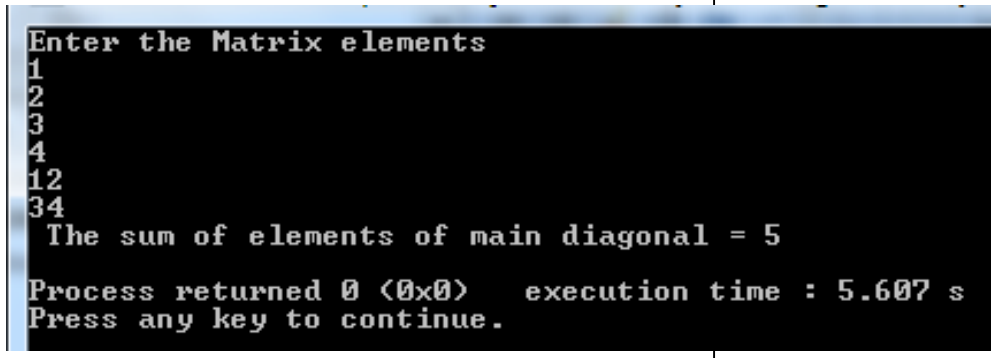
```
Enter the Matrix elements
1
2
3
4
5
6
7
8
9
123
456
789

Process returned 0 (0x0)   execution time : 11.358 s
Press any key to continue.
```

Example 2

- Write a C++ program that reads matrix (A_{nxm}), then find and print the sum of the elements of *main diagonal*.

```
…
…
//Find sum of elements of main diagonal
int  sum = 0;
for(int i = 0; i<ROW_SIZE; i++){
        for(int j = 0; j<COL_SIZE; j++){
            if(i == j){
                sum = sum + array[i][j];
            }
        }
}
cout << " The sum of elements of main diagonal = " << sum << endl;
…
```



Example 3

- Write a C++ program that reads matrix (A_{nxm}), then find and print the sum of the elements above *main diagonal*.

```
…
…
//Find sum of elements of main diagonal
int  sum = 0;
for(int i = 0; i<ROW_SIZE; i++){
        for(int j = 0; j<COL_SIZE; j++){
            if(i < j){
                sum = sum + array[i][j];
            }
        }
}
cout << " The sum of elements of main diagonal = " << sum << endl;
…
```

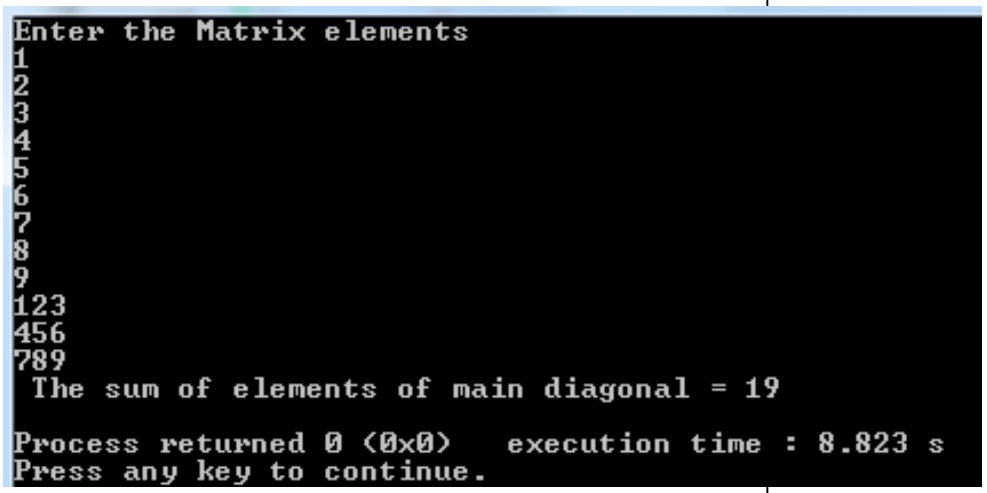
```
Enter the Matrix elements
1
2
3
4
5
6
7
8
9
123
456
789
 The sum of elements of main diagonal = 11

Process returned 0 (0x0)   execution time : 11.163 s
Press any key to continue.
```

Example 4

- Write a C++ program that reads matrix (A_{nxm}), then find and print the sum of the elements below *main diagonal*.

```
…
…
//Find sum of elements of main diagonal
int  sum = 0;
for(int i = 0; i<ROW_SIZE; i++){
        for(int j = 0; j<COL_SIZE; j++){
            if(i > j){
                sum = sum + array[i][j];
            }
        }
}
cout << " The sum of elements of main diagonal = " << sum << endl;
…
```

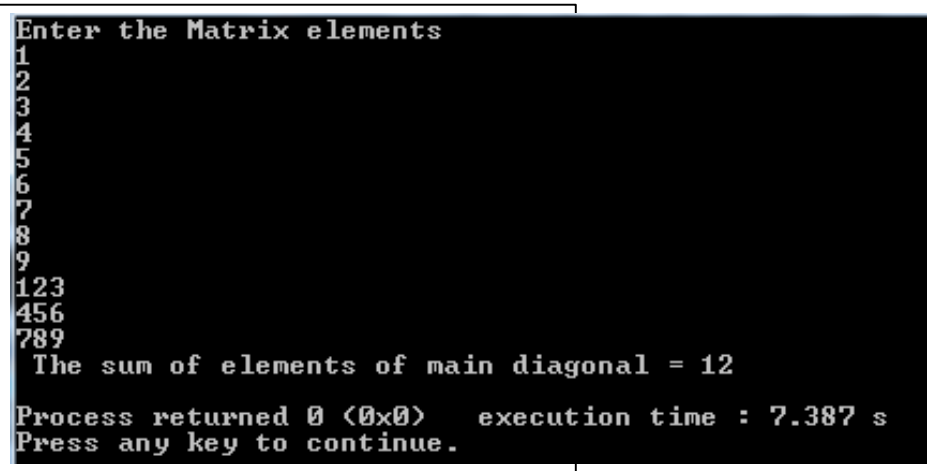


Example 5

- Write a C++ program that reads matrix (A_{nxm}), then find and print the sum of the elements of the first column.

```
…
…
//Find sum of elements of main diagonal
int  sum = 0;
for(int i = 0; i<ROW_SIZE; i++){
        sum = sum + array[i][0];
}

cout << " The sum of elements of main diagonal = " << sum << endl;
…
```

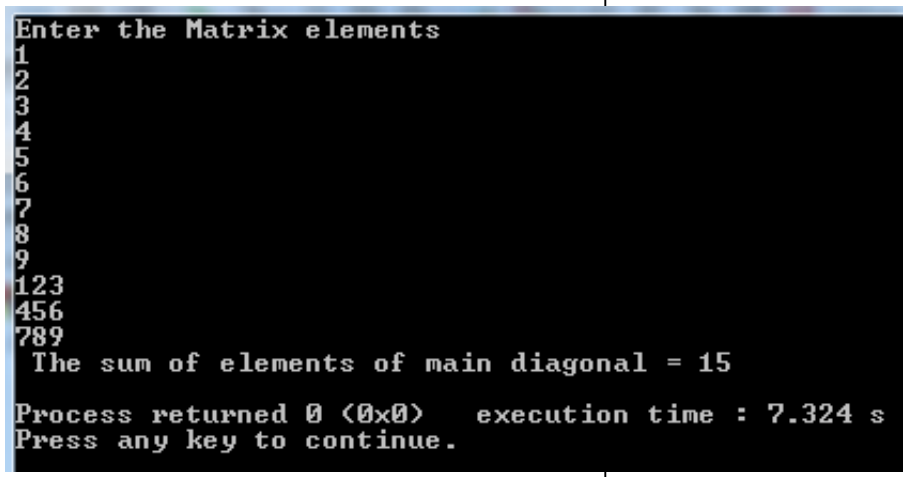


Example 6

- Write a C++ program that reads matrix (A_{nxm}), then find and print the sum of the elements of the second row.

```
…
…
//Find sum of elements of main diagonal
int  sum = 0;
for(int j = 0; j<COL_SIZE; j++){
        sum = sum + array[1][j];
}

cout << " The sum of elements of main diagonal = " << sum << endl;
…
```



Questions?