Multidimensional Arrays

- Arrays with more than one dimension are called multidimensional arrays.
- An array of two dimensions can be declared as follows:
  - **data_type array_name[size1][size2];**
  - Here, data_type is the name of some type of data, such as int. Also, size1 and size2 are the sizes of the array’s first and second dimensions, respectively.
- A three-dimensional array, such as a cube, can be declared as follows:
  - **data_type array_name[size1][size2][size3]**

Two dimensional Array

Computer memory is essentially one dimensional with memory location running straight from 0 to highest. A multidimensional array cannot be stored in memory as grid.

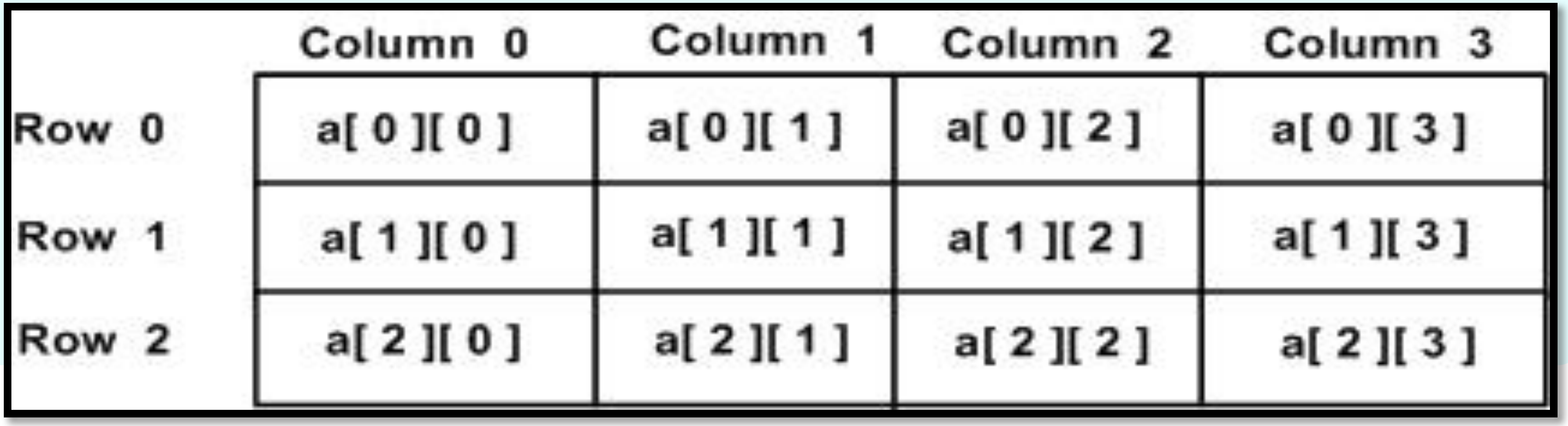
Row 0	1	2	3
Row 1	4	5	6
Row 2	7	8	9

In computer, this array will be stored as below

1	2	3	4	5	6	7	8	9
Row 0			Row 1			Row 2		

Two dimensional array

- A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.
- Thus, every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initializing two dimensional array

- Multidimensional arrays may be initialized by specifying bracketed value for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {{0,1,2,3},    /* Initializers for row indexed by 0 */
               {4,5,6,7},    /* Initializers for row indexed by 1 */
               {8,9,10,11} /* Initializers for row indexed by 2 */
               };
```

- The nested braces, which indicate the intended row, are optional. The following example is equivalent to the previous example-

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing two dimensional array elements

- An element in a two dimensional array is accessed by using the subscript i.e, row index and column index of the array. For example -

```
int val = a[2][3];
```

Accessing Two-Dimensional Array Elements

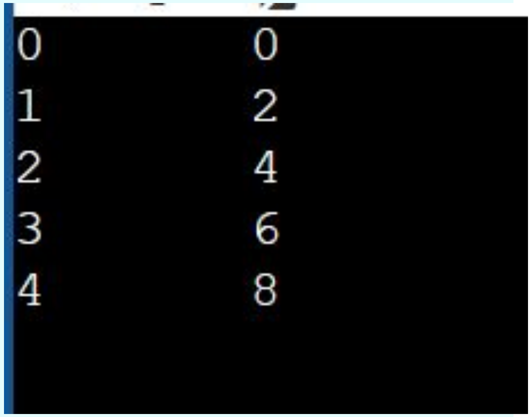
```
#include <stdio.h>
int main ()
{
    /* an array with 5 rows and 2 columns*/
     int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
    int i, j;
    /* output each array element's value */
    for ( i = 0; i < 5; i++ ) // for rows
    {
        for ( j = 0; j < 2; j++ ) //for columns
        {
            printf("a[%d][%d] = %d\n", i,j, a[i][j] );
        }
    }
    return 0;
}
```

0	0
1	2
2	4
3	6
4	8

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

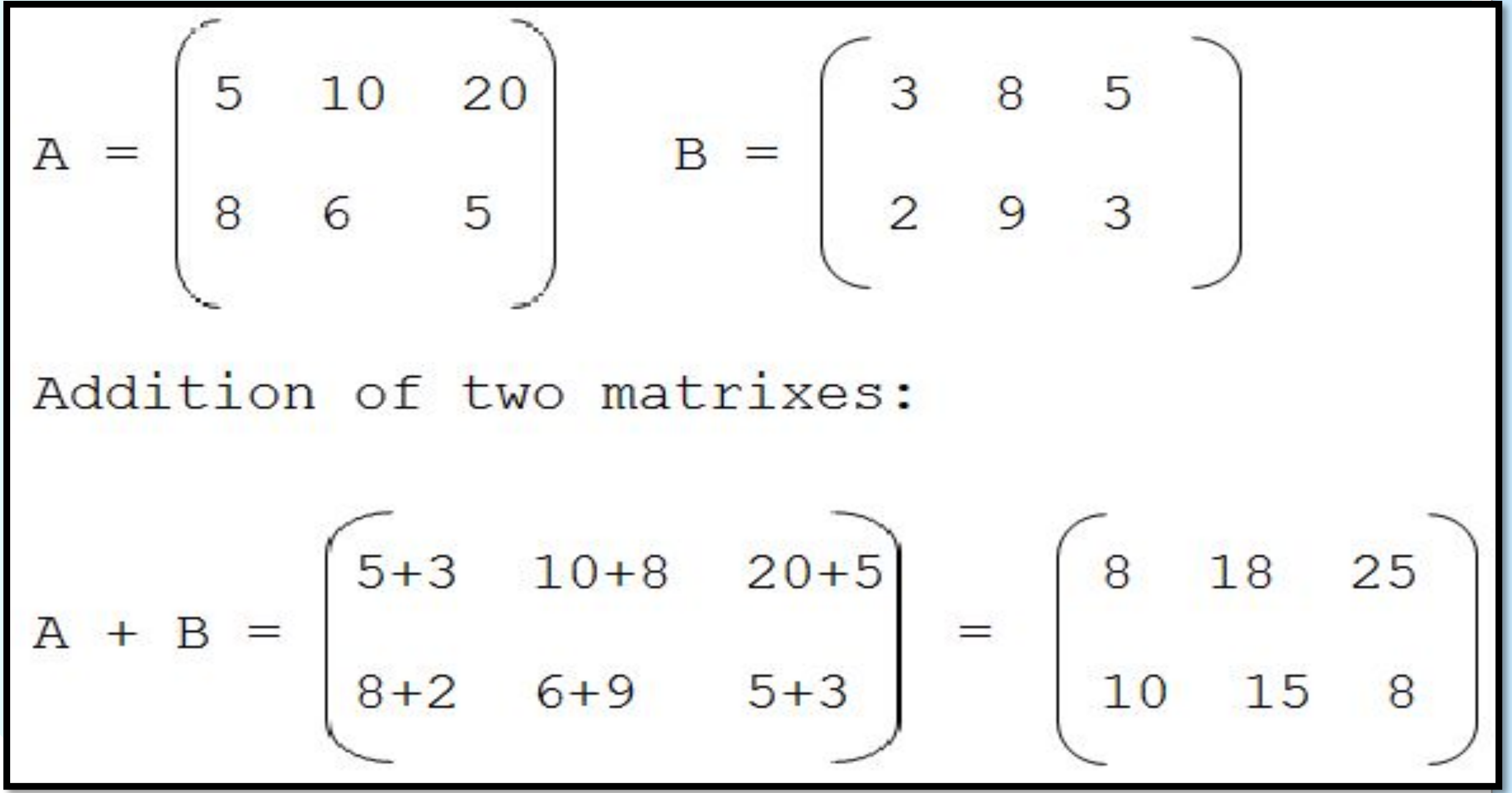
Matrix form printing

```
#include <stdio.h>
int main ()
{
    /* an array with 5 rows and 2 columns*/
     int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
    int i, j;
    /* output each array element's value */
    for ( i = 0; i < 5; i++ ) // for rows
    {
        for ( j = 0; j < 2; j++ ) //for columns
        {
            printf("%d",  a[i][j] );
            printf("\t");
        }
        printf("\n");
    }
    return 0;
}
```



```
0       0
1       2
2       4
3       6
4       8
```

Matrix - Addition



$$A = \begin{pmatrix} 5 & 10 & 20 \\ 8 & 6 & 5 \end{pmatrix} \qquad B = \begin{pmatrix} 3 & 8 & 5 \\ 2 & 9 & 3 \end{pmatrix}$$

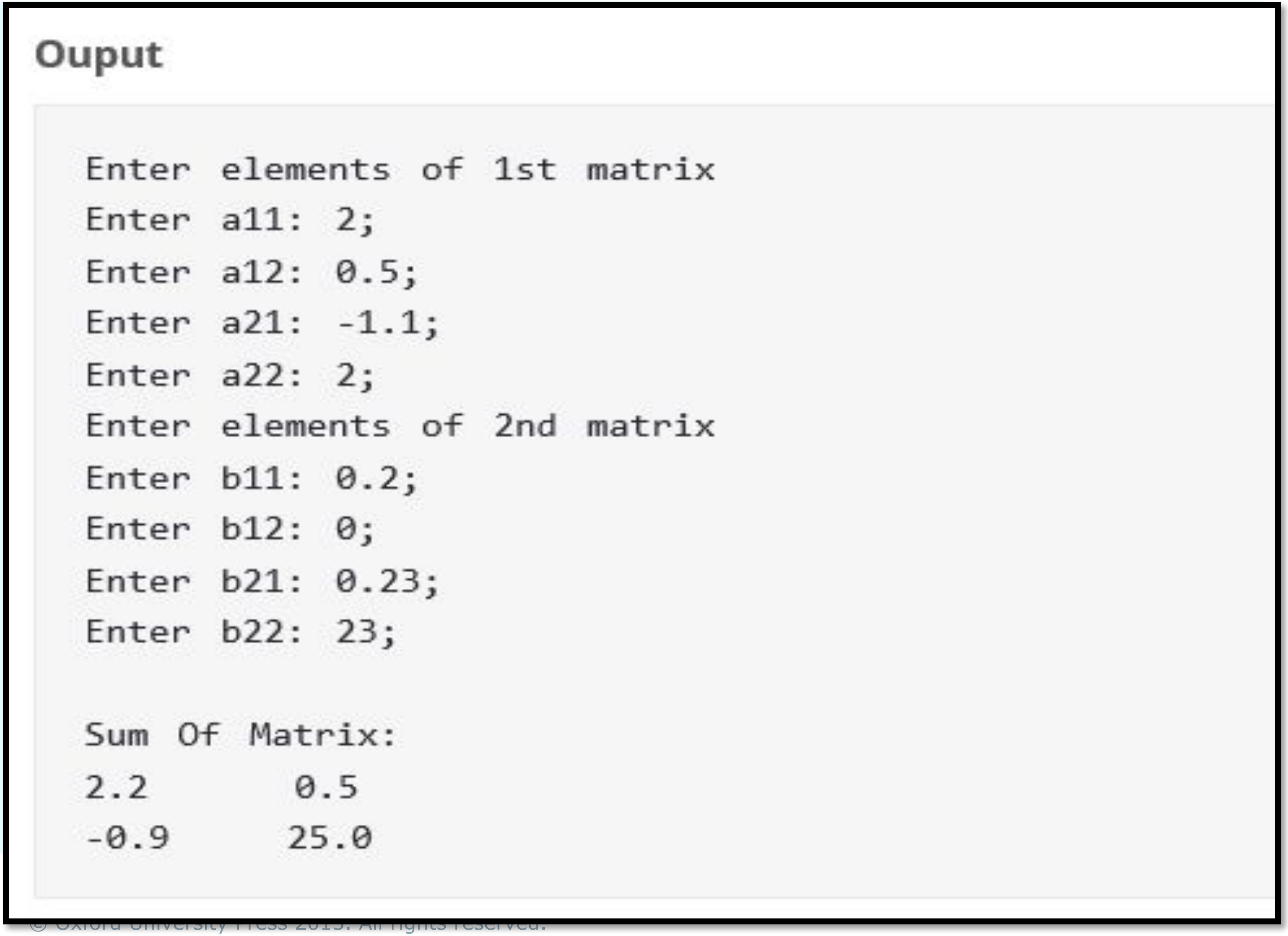
Addition of two matrixes:

$$A + B = \begin{pmatrix} 5+3 & 10+8 & 20+5 \\ 8+2 & 6+9 & 5+3 \end{pmatrix} = \begin{pmatrix} 8 & 18 & 25 \\ 10 & 15 & 8 \end{pmatrix}$$

sum of two matrices of order 2*2

```
#include <stdio.h>
int main()
 {
float a[2][2], b[2][2], c[2][2];
int i, j;
 // Taking input using nested for loop
printf("Enter elements of 1st matrix\n");
for(i=0; i<2; ++i)
       for(j=0; j<2; ++j)
      {
      printf("Enter a%d%d: ", i+1, j+1);
scanf("%f", &a[i][j]);
      }

 // Taking input using nested for loop
printf("Enter elements of 2nd matrix\n");
for(i=0; i<2; ++i)
       for(j=0; j<2; ++j)
       {
      printf("Enter b%d%d: ", i+1, j+1);
scanf("%f", &b[i][j]);
      }
```

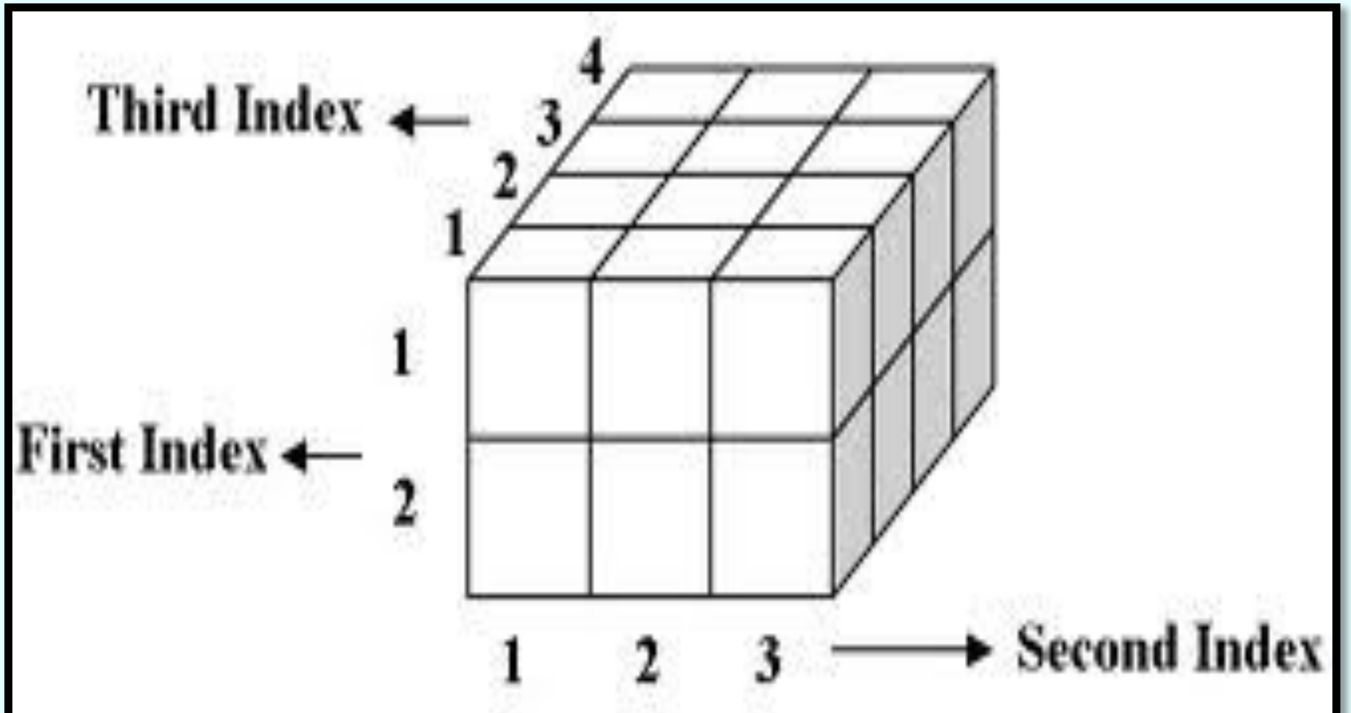
```
// adding corresponding elements
of two arrays
for(i=0; i<2; ++i)
      for(j=0; j<2; ++j)
      {
      c[i][j] = a[i][j] + b[i][j];
       }

// Displaying the sum
printf("\nSum Of Matrix:"); for(i=0;
i<2; ++i)
      for(j=0; j<2; ++j)
      {
      printf("%.1f\t", c[i][j]);
if(j==1)
            printf("\n"); } return 0;
}
```



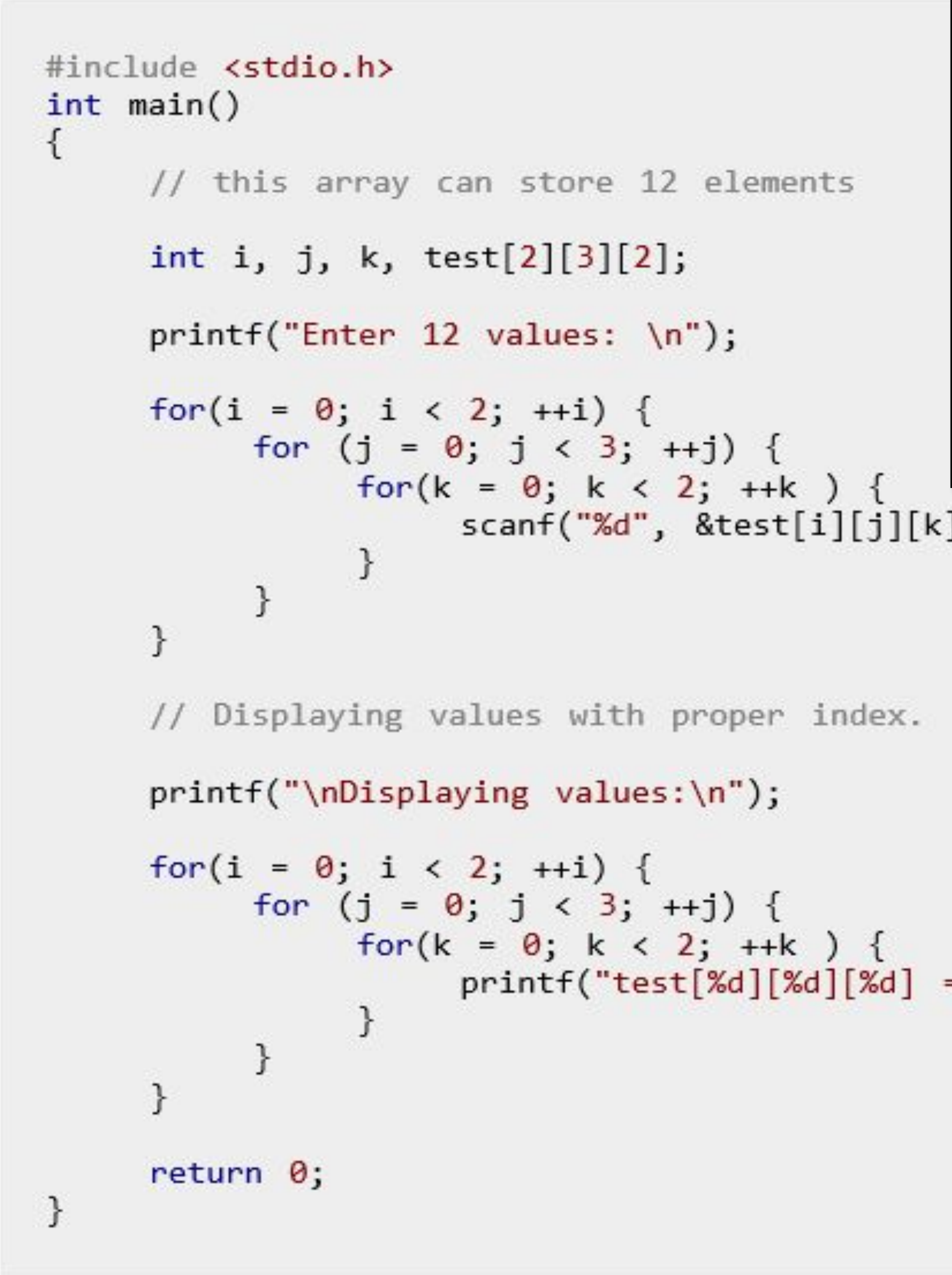
Ouput

```
Enter elements of 1st matrix
Enter a11: 2;
Enter a12: 0.5;
Enter a21: -1.1;
Enter a22: 2;
Enter elements of 2nd matrix
Enter b11: 0.2;
Enter b12: 0;
Enter b21: 0.23;
Enter b22: 23;

Sum Of Matrix:
2.2       0.5
-0.9     25.0
```

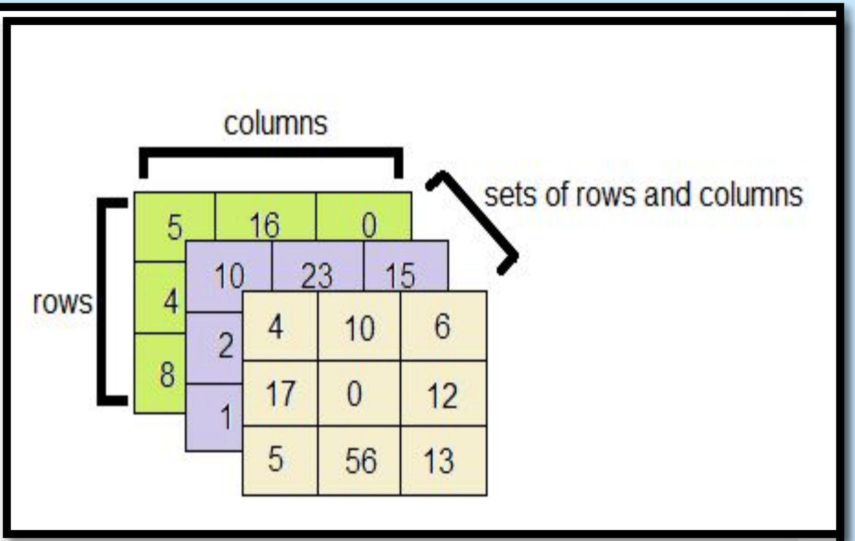
3 D Arrays



Three-dimensional array with twenty four elements



```c
#include <stdio.h>
int main()
{
      // this array can store 12 elements

      int i, j, k, test[2][3][2];

      printf("Enter 12 values: \n");

      for(i = 0; i < 2; ++i) {
            for (j = 0; j < 3; ++j) {
                  for(k = 0; k < 2; ++k ) {
                        scanf("%d", &test[i][j][k]);
                  }
            }
      }

      // Displaying values with proper index.

      printf("\nDisplaying values:\n");

      for(i = 0; i < 2; ++i) {
            for (j = 0; j < 3; ++j) {
                  for(k = 0; k < 2; ++k ) {
                        printf("test[%d][%d][%d] = %d\n", i, j, k, test[i][j][k]);
                  }
            }
      }

      return 0;
}
```



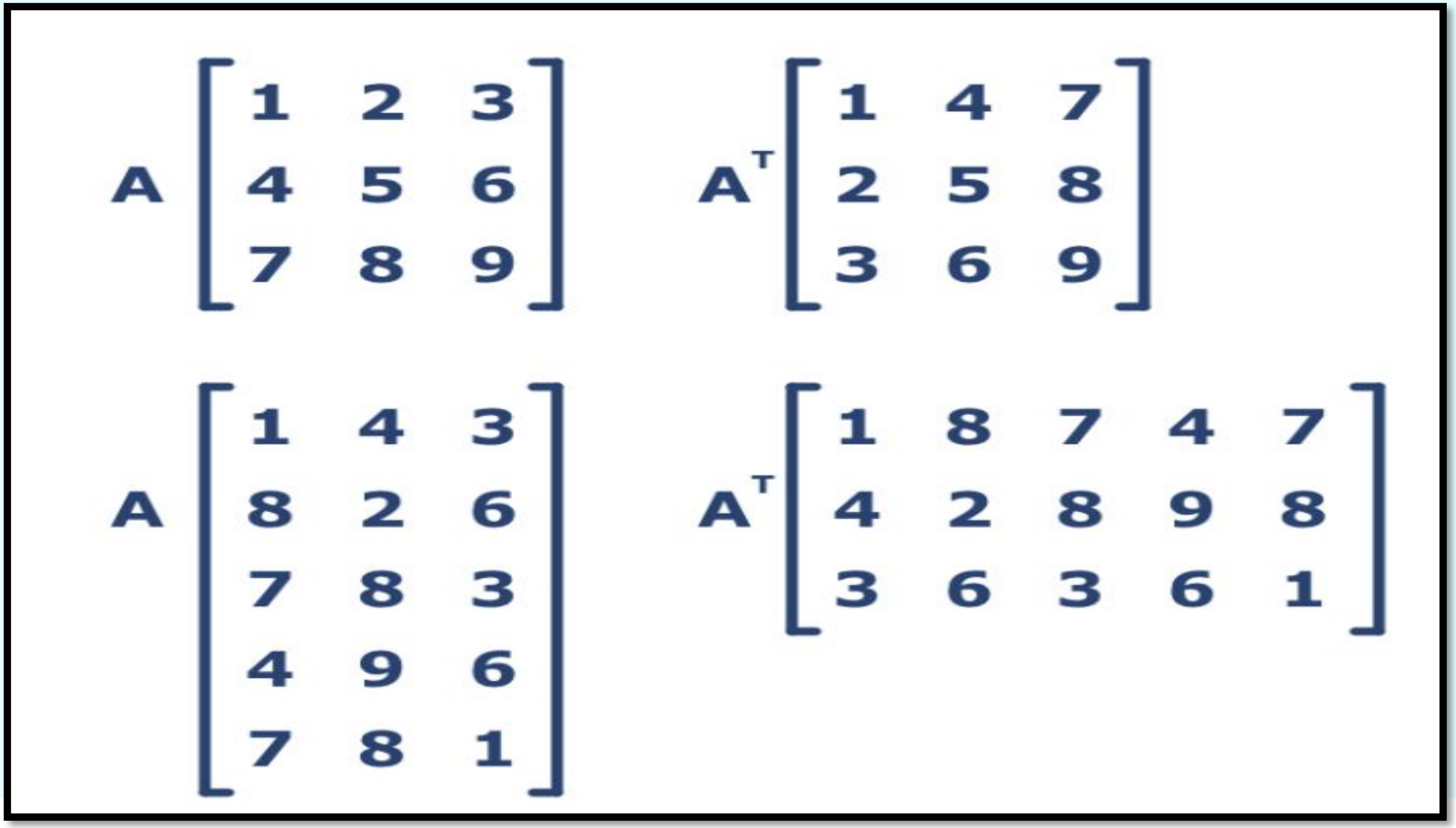
Multidimensional Arrays

- **Unsized Array Initializations**
  - C compiler automatically creates an array big enough to hold all the initializers. This is called an unsized array.
  - The following are examples of declarations with initialization.
    - **char e1[] =“read error\n”;**
    - **char e2[] =“write error\n”;**

Multidimensional Arrays

- Multi-dimensional arrays are kept in computer memory as a linear sequence of variables.
- The elements of a multi-dimensional array are stored contiguously in a block of computer memory.
- The number of subscripts determines the *dimensionality* of an array.
- The separation of initial values into rows in the declaration statement is not necessary.
- If unsized arrays are declared, the C compiler automatically creates an array big enough to hold all the initializers.

Transpose of a matrix



$$A \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad A^{T} \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

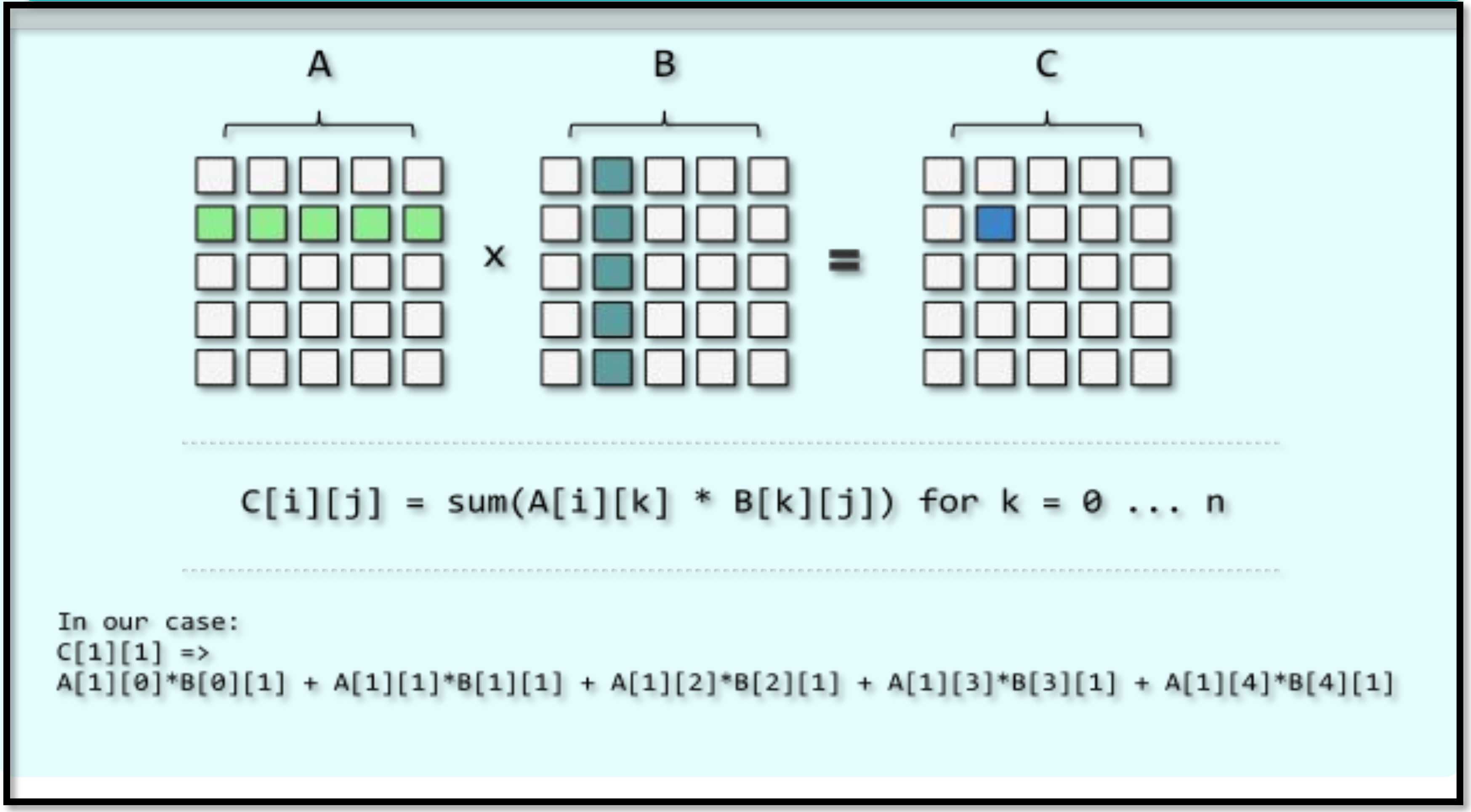
$$A \begin{bmatrix} 1 & 4 & 3 \\ 8 & 2 & 6 \\ 7 & 8 & 3 \\ 4 & 9 & 6 \\ 7 & 8 & 1 \end{bmatrix} \qquad A^{T} \begin{bmatrix} 1 & 8 & 7 & 4 & 7 \\ 4 & 2 & 8 & 9 & 8 \\ 3 & 6 & 3 & 6 & 1 \end{bmatrix}$$

```
#include <stdio.h>
 void main()
{
static int array[10][10];
int i, j, m, n;
printf("Enter the order of the matrix \n");
scanf("%d %d", &m, &n);
printf("Enter the coefficients of the
matrix\n");
for (i = 0; i < m; ++i)
{
        for (j = 0; j < n; ++j)
        {
                scanf("%d", &array[i][j]);
        }
}
printf("The given matrix is \n");
for (i = 0; i < m; ++i)
        {
        for (j = 0; j < n; ++j)
        {
                printf(" %d", array[i][j]);
        }
printf("\n");}
```

```
printf("Transpose of matrix is \n");
for (j = 0; j < n; ++j)
{
        for (i = 0; i < m; ++i)
        {
        printf(" %d", array[i][j]);
        }
printf("\n");
}
}
```

```
Enter the order of the matrix 3 3
Enter the coefficients of the matrix
3 7 9
2 7 5
6 3 4
The given matrix is
3 7 9
2 7 5
6 3 4
Transpose of matrix is
3 2 6
7 7 3
 9 5 4
```

Multiplication matrix



```
C[i][j] = sum(A[i][k] * B[k][j]) for k = 0 ... n
```

```
In our case:
C[1][1] =>
A[1][0]*B[0][1] + A[1][1]*B[1][1] + A[1][2]*B[2][1] + A[1][3]*B[3][1] + A[1][4]*B[4][1]
```

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

```
for(i=0;i<m;i++)
{
          for(j=0;j<q;j++)
          {=
                    c[i][j]=0;
                    for(k=0;k<n;k++)
                    {
                    c[i][j]+=a[i][k]*b[k][j];
                    }
          }
}
```

$$\begin{bmatrix} aj + bm + cp & ak + bn + cq & al + bo + cr \\ dj + em + fp & dk + en + fq & dl + eo + fr \\ gj + hm + ip & gk + hn + iq & gl + ho + ir \end{bmatrix}$$

Multiplication of two matrix

PROGRAM CODE

```
#include<stdio.h>
int main()
{
int a[10][10],b[10][10],c[10][10];
int i,j,k,m,n,p,q;
clrscr();
printf("\nThe row & column of Matrix A :");
scanf("%d%d",&m,&n);
fflush(stdin);
printf("\nThe row & column of Matrix B :");
scanf("%d%d",&p,&q);
```

read

```
if (n==p)
{
printf("\nFor Matrix A:-\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
    printf("\nEnter values for A[%d][%d]=> ",i,j);
    scanf("%d",&a[i][j]);
    }
}
```

Read First matrix

```
printf("\nFor Matrix B:-\n");
for(i=0;i<p;i++)
{
    for(j=0;j<q;j++)
    {
    printf("\nEnter values for B[%d][%d]=> ",i,j);
    scanf("%d",&b[i][j]);
    }
}
//Matrix Multiplication Logic
for(i=0;i<m;i++)
{
    for(j=0;j<q;j++)
    {
        c[i][j]=0;
        for(k=0;k<n;k++)
        {
            c[i][j]+=a[i][k]*b[k][j];
        }
    }
}
```

Read Second matrix

Multiplication logic

Print 1st Matrix

```
printf("\nMatrix A is\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        printf("%d\t",a[i][j]);
        printf("\n");
}
printf("\n");
```

Print 2nd matrix

```
printf("\nMatrix B is\n");
    for(i=0;i<p;i++)
    {
        for(j=0;j<q;j++)
        printf("%d\t",b[i][j]);
        printf("\n");
    }
printf("\n");
```

Print result matrix

```
printf("\nMultiplication Matrix C is\n");
for(i=0;i<m;i++)
{
    for(j=0;j<q;j++)
    printf("%d\t",c[i][j]);
    printf("\n”);
}}
```

```
else
printf("\nMultiplication is not
possible.\n");
return 0;
}
```

Arrays of Strings: Two-dimensional Character Array

- A two-dimensional array of strings can be declared as follows:
  - <data_type> <string_array_name>[<row_size>][<columns_size>];

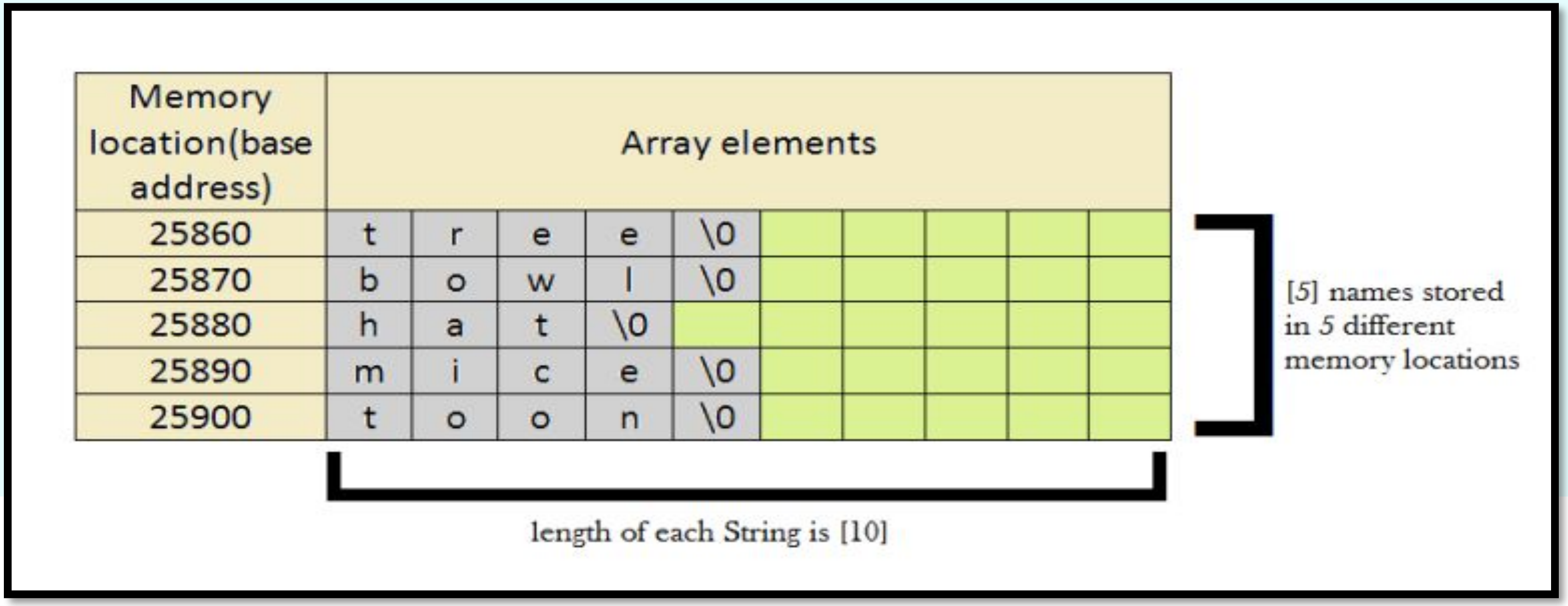
- Consider the following example on declaration of a two-dimensional array of strings.

```
char s[5][30];
```

2 D char array

```
char name[5][10]={
                 "tree",
                 "bowl",
                 "hat",
                 "mice",
                 "toon"
               };
```

The areas marked in green shows the memory locations that are reserved for the array but are not used by the string. Each character occupies **1 byte** of storage from the memory.



Memory location(base address)	Array elements									
25860	t	r	e	e	\0	🟩	🟩	🟩	🟩	🟩
25870	b	o	w	l	\0	🟩	🟩	🟩	🟩	🟩
25880	h	a	t	\0	🟩	🟩	🟩	🟩	🟩	🟩
25890	m	i	c	e	\0	🟩	🟩	🟩	🟩	🟩
25900	t	o	o	n	\0	🟩	🟩	🟩	🟩	🟩

[5] names stored in 5 different memory locations

length of each String is [10]

Initialization

- Two-dimensional string arrays can be initialized as shown
  - ✔ char s[5][10] ={“Cow”,”Goat”,”Ram”,”Dog”,”Cat”};

- which is equivalent to
  - ✔ s[0] C o w \0
  - ✔ S[1] G o a t \0
  - ✔ S[2] R a m \0
  - ✔ S[3] D o g \0
  - ✔ S[4] C a t \0

- Here every row is a string. That is, s[i] is a string. Note that the following declarations are invalid.
  - ✔ char s[5][] ={“Cow”,“Goat”,“Ram”,”Dog”,“Cat”};
  - ✔ char s[][] ={“Cow”,“Goat”,“Ram”,“Dog”,“Cat”};

Asignment

1. WAP to check whether a given matrix of order 3X3 is orthogonal or not.
2. WAP to find the number of odd and number of even elements in the matrix of order 3X3.
3.