2D ARRAYS

2 dimensional Array

- It is an ordered table of homogeneous elements.
- It can be imagined as a two dimensional table made of elements, all of them of a same uniform data type.
- It is generally referred to as **matrix**, of some rows and some columns.
- It is also called as a **two-subscripted variable**.

2 dimensional Arrays

For example

```
int marks[5][3];
float matrix[3][3];
char page[25][80];
```

- ✓The first example tells that marks is a 2-D array of 5 rows and 3 columns.
- ✓The second example tells that matrix is a 2-D array of 3 rows and 3 columns.
- ✓Similarly, the third example tells that page is a 2-D array of 25 rows and 80 columns.

2 dimensional Arrays

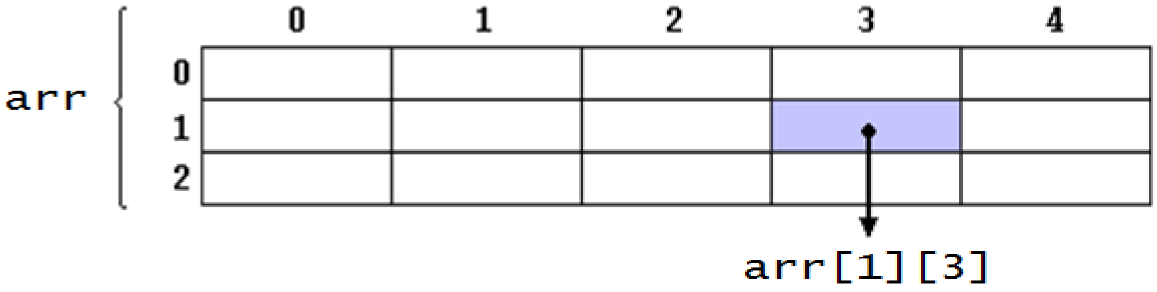
Declaration

type array_name[row_size][column_size];

For example,

int arr [3][5];

✓arr represents a two dimensional array or table having 3 rows and 5 columns and it can store 15 integer values.



An array int mark[5][4] is represented as follows

Student [subscript]	Tests1 [0]	Test 2 [1]	Test 3 [2]	Test 4 [3]
1 [0]	**20 (mark[0][0])**	**20 (mark[0][1])**	**21 (mark[0][2])**	**22 (mark[0][3])**
2 [1]	**18 (mark[1][0])**	**23 (mark[1][1])**	**22 (mark[1][2])**	**20 (mark[2][3])**
3 [2]	**11 (mark[2][0])**	**22 (mark[2][1])**	**15 (mark[2][2])**	**16 (mark[2][3])**
4 [3]	**22 (mark[3][0])**	**21 (mark[3][1])**	**23 (mark[3][2])**	**24 (mark[3][3])**
5 [4]	**17 (mark[4][0])**	**15 (mark[4][1])**	**16 (mark[4][2])**	**18 (mark[4][3])**

2 Dimensional Arrays

Initialization of two dimensional arrays

type array-name [row size] [col size] ={list of values};

```
int table [2][3]={0,0,0,1,1,1};
```

→ initializes the elements of the first row to zero and the second row to 1.

Initialization is always done row by row.

The above statement can be equivalently written as

```
int table [2][3]={{0,0,0},{1,1,1}};
```

OR in matrix form it can be written as

```
int table [2][3]=      {      {0,0,0},
                              {1,1,1}     };
```

2 Dimensional Arrays

When array is completely initialized with all values, need not necessarily specify the first dimension.

```
int table [] [3]=    {          {0,0,0},
                                {1,1,1 }
                     };
```

If the values are missing in an initializer, they are set to zero

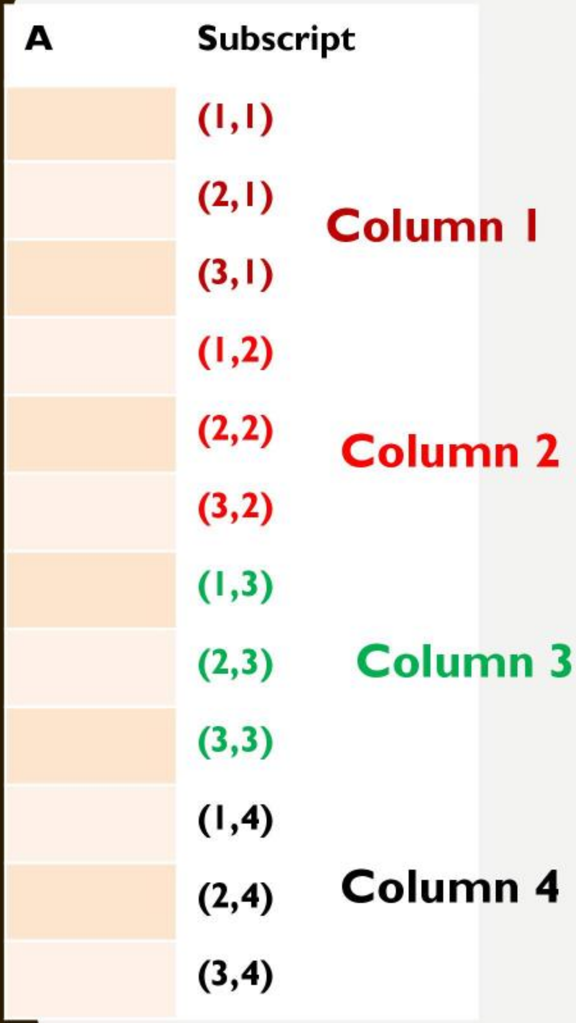
```
int table [2] [3]=      {         {1,1},
                                  {2}
                        };
```

will initialize the first two elements of the first row to 1, the first element of the second row to two, and all other elements to zero.

To set all elements to zero

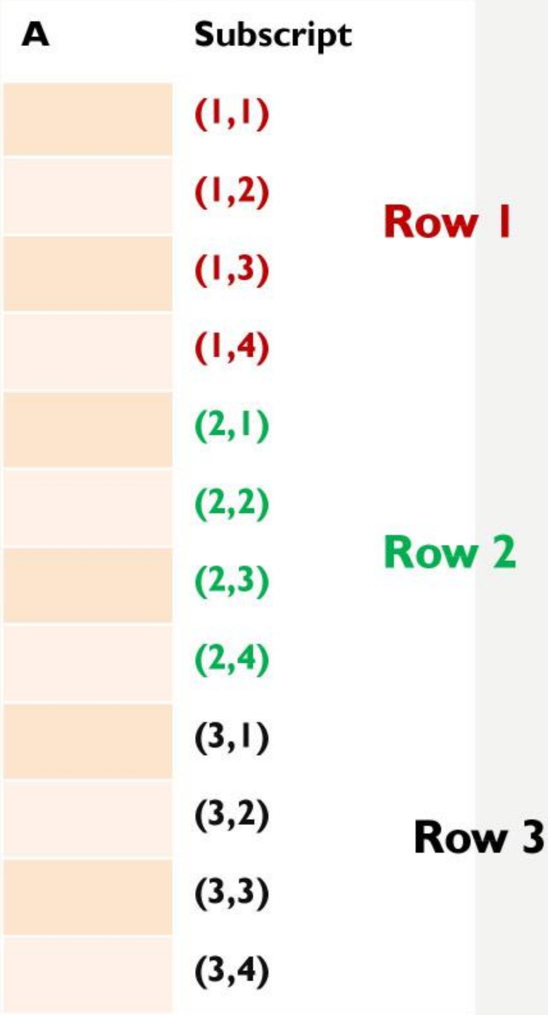
```
int table [3] [3]={{0},{0},{0}};
```

2D ARRAY IN MEMORY



A	Subscript	
	(1,1)	Column 1
	(2,1)	
	(3,1)	
	(1,2)	Column 2
	(2,2)	
	(3,2)	
	(1,3)	Column 3
	(2,3)	
	(3,3)	
	(1,4)	Column 4
	(2,4)	
	(3,4)	

Column-Major Order



A	Subscript	
	(1,1)	Row 1
	(1,2)	
	(1,3)	
	(1,4)	
	(2,1)	Row 2
	(2,2)	
	(2,3)	
	(2,4)	
	(3,1)	Row 3
	(3,2)	
	(3,3)	
	(3,4)	

Row-Major Order

2D ARRAY

$LOC(LA[K]) = Base(LA) + w(K - 1)$

- **$LOC(A[J,K])$ of $A[m,n]$**

Column-Major Order

$$LOC(A[J,K]) = Base(A) + w[m(K-1) + (J-1)]$$

Row-Major Order

$$LOC(A[J,K]) = Base(A) + w[n(J-1) + (K-1)]$$

2D ARRAY EXAMPLE

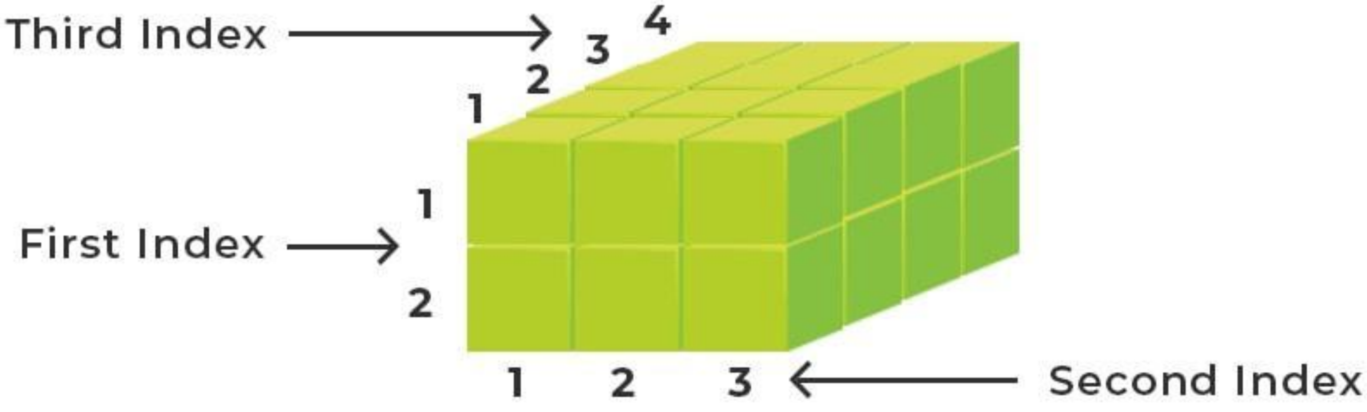
- Consider a 25 x 4 array A. Suppose the Base(A) = 200 and w =4. Suppose the programming store 2D array using row-major. Compute LOC(A[12,3])

- LOC(A[J,K]) = Base(A) + w[n(J-1) + (K-1)]

- LOC(A[12,3]) = 200 + 4[4(12-1) + (3 -1)]

  = 384

3D Array



Three-Dimensional Array
with 24 Elements

MULTIDIMENSIONAL ARRAY

- An **n**-dimensional m_1 x m_2 x …. X m_n array **B** is a collection of $m_1.m_2…m_n$ data elements in which each element is specified by a list of **n** integers – such as $K_1, K_2, ….., K_n$ – called subscript with the property that

  $1 \le K_1 \le m_1, \ 1 \le K_2 \le m_2, \ …. \ 1 \le K_n \le m_n$

The Element **B** with subscript $K_1, K_2, …,K_n$ will be denoted by

$B_{K1,K2, …,Kn}$ or $B[K_1,K_2,….,K_n]$

MULTIDIMENSIONAL ARRAY

- Let **C** be a **n**-dimensional array
- Length L_i of dimension **i** of **C** is the number of elements in the index set

  $L_i = UB - LB + 1$

- For a given subscript K_i, the effective index E_i of L_i is the number of indices preceding K_i in the index set

  $E_i = K_i - LB$

MULTIDIMENSIONAL ARRAY

Address LOC(C[K_1,K_2, …., K_n]) of an arbitrary element of C can be obtained as

Column-Major Order

$$\text{Base(C)} + w[(((\ldots (E_N L_{N-1} + E_{N-1})L_{N-2}) + \ldots.. +E_3)L_2+E_2)L_1+E_1]$$

Row-Major Order

$$\text{Base(C)} + w[(\ldots ((E_1 L_2 + E_2)L_3 + E_3)L_4 + \ldots.. +E_{N-1})L_N +E_N]$$

EXAMPLE

- MAZE(2:8, -4:1, 6:10)
- Calculate the address of **MAZE[5,-1,8]**
- Given: Base(MAZE) = 200, w = 4, MAZE is stored in Row-Major order
- L1 = 8-2+1 = 7, L2 = 6, L3 = 5
- E1 = 5 -2 = 3, E2 = 3, E3 = 2

EXAMPLE CONTD ..

- Base(C) + w[(… ((E_1L_2 + E_2)L_3 + E_3)L_4 + ….. +E_{N-1})L_N +E_N]
- E_1L_2 = 3 . 6 = 18
- E_1L_2 + E_2 = 18 + 3 = 21
- (E_1L_2 + E_2)L_3 = 21 . 5 = 105
- (E_1L_2+E_2)L_3 + E_3 = 105 + 2 = 107
- MAZE[5,-1,8] = 200 + 4(107) = 200 + 248 = 628

Read a matrix and display it

```
int main()
{
int i, j, m, n, a[100][100];

printf("enter dimension for a:“);
scanf(“%d %d”,&m,&n);

cout<<"\n enter elements\n";
for(i=0;i<m;i++)
{
     for(j=0;j<n;j++)
     {
          scanf(“%d”, &a[i][j]);
     }
}
```

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

return 0;
}
```

Addition of two Matrices

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i, j, m, n, p, q, a[10][10],
b[10][10], c[10][10];
printf("enter dimension for a \n“);
scanf(“%d  %d”,&m,&n);
printf("enter dimension for b\n“);
scanf(“%d  %d”,&p,&q);
```

```
 if (m!=p||n!=q)
 {
 printf("cannot add \n“);
  exit(0);            }
//Reading the elements
printf("enter elements for a \n“);
for (i=0;i<m;i++)
            for(j=0;j<n;j++)
                scanf(“%d”,&a[i][j]);
```

Matrix Addition

```
printf("\n enter elements for b\n)";
          for(i=0;i<p;i++)
                    for(j=0;j<q;j++)
                              scanf(“%d”,
&b[i][j]);
          //Addition
          for(i=0;i<m;i++)
                    for(j=0;j<n;j++)
                    c[i][j]=a[i][j]+b[i][j];
```

```
//Display
 printf("\n final matrix is \n“);
          for(i=0;i<m;i++)
          {
                    for(j=0;j<n;j++)

          printf(“%d\t”,c[i][j]);
                    printf("\n“);
          }
```

Row Sum & Column Sum of a matrix

```
int a[10][10];
int rowsum[10], colsum[10];
printf("enter dimension for a \n“);
scanf(“%d  %d”,&m, &n);
 //Reading
printf("enter elements for a \n“);
for (i=0;i<m;i++){
  for(j=0;j<n;j++)
     scanf(“%d”, &a[i][j]);
}
```

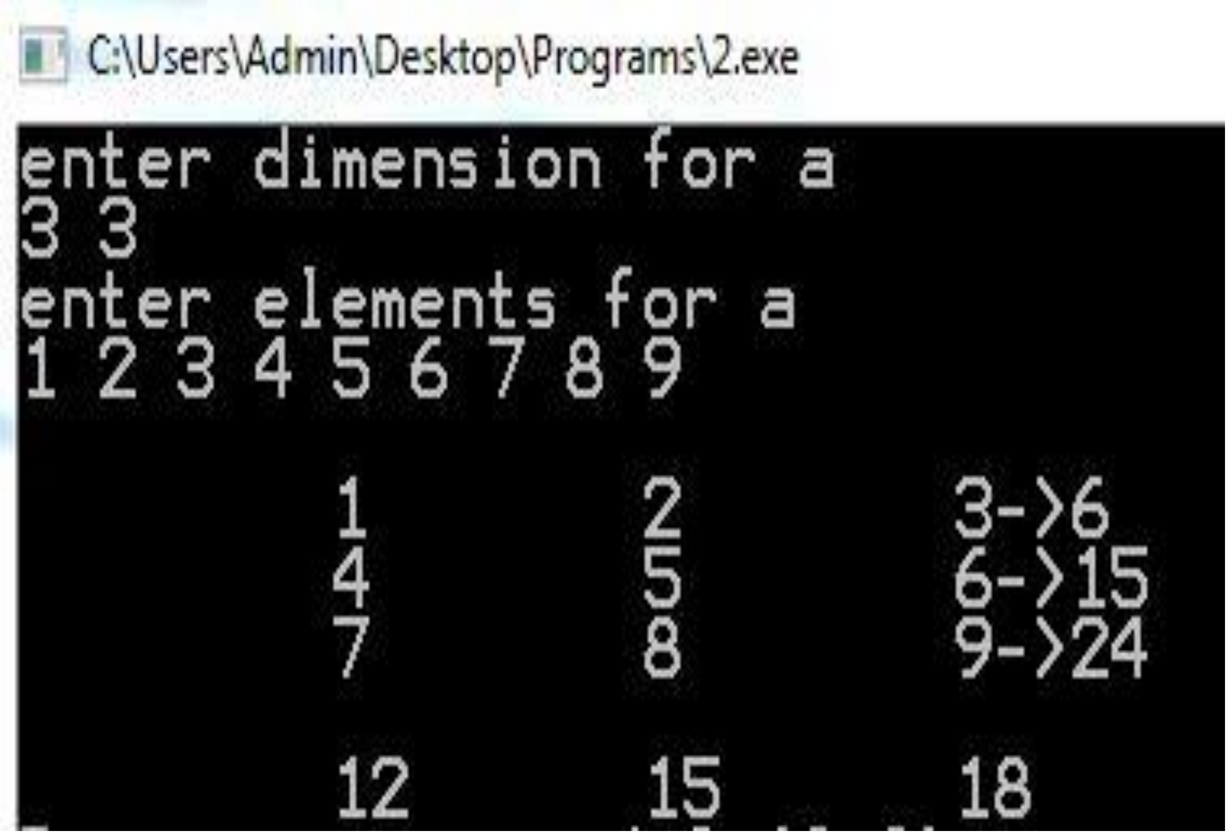
```
//Row sum
for(i=0;i<m;i++)
{
  rowsum[i]=0;
  for(j=0;j<n;j++)
  rowsum[i]=rowsum[i]+a[i][j];
}
printf("\n“);
```

Row Sum & Column Sum of a matrix

```
//Column sum
for(j=0;j<n;j++)
{
   colsum[j]=0;
   for(i=0;i<m;i++)
      colsum[j]=colsum[j]+a[i][j];
}
```

```
//Display
for(i=0;i<m;i++)
{
   for(j=0;j<n;j++)
      printf("\t %d“,a[i][j]);
   printf(“->“)
printf(“%d\n”,rowsum[i]);
}
printf("\n“);

for(i=0;i<n;i++)
printf("\t %d”,colsum[i]);
```

Row Sum & Column Sum of a matrix



```
enter dimension for a
3 3
enter elements for a
1 2 3 4 5 6 7 8 9

        1       2       3->6
        4       5       6->15
        7       8       9->24

        12      15      18
```

Trace and Norm of a Matrix

Trace is sum of principal diagonal elements of a square matrix.

Norm is Square Root of sum of squares of elements of a matrix.

```
int trace=0, sum=0,i,j,norm;
int m=3,n=3;
printf("enter elements for a \n");
for (i=0;i<m;i++){
 for(j=0;j<n;j++)

scanf("%d",&a[i][j]);
}


for(i=0;i<m;i++)
   trace=trace + a[i][i];
```

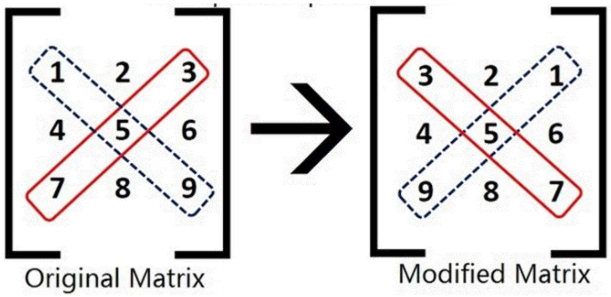
```
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
  sum=sum+a[ i ][ j]*a[ i ][ j ];
}
norm=sqrt(sum);

printf(" trace is %d", trace );
printf(" norm is %d", norm );
```

Check whether a given Matrix is Symmetric or not

```
printf("enter dimension \n“);
scanf(“%d %d”,&m,&n);
if(m!=n)
{
printf("it is not a square \n“);
else
{ printf("enter elements \n“);
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    scanf(“%d’,&a[i][j]);
```

```
for(i=0;i<m;i++){
  for(j=0;j<n;j++){
    if (a[ i ][ j ]!=a[ j ][ i ]){
      printf("\n matrix is not
        symmetric \n“);
      exit(0);    }
  }  }
printf("\n matrix is symmetric“);
}
```

Exchange the elements of principal diagonal with secondary diagonal in an N dimensional Square matrix



```
int main(){
int i, j, temp, arr[4][4],n;

printf("\nEnter dimension: “);
scanf(“%d”,&n);

printf(“\nEnter elements:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf(“%d”,&arr[i][j]);
```

```
for(i=0;i<n;i++)
for(j=0;j<n;j++)
   if(i==j){
     temp=arr[i][j];
     arr[i][j]=arr[i][n-i-1];
     arr[i][n-i-1]=temp;
  }
printf("\nModified Matrix:\n“);
for(i=0;i<n;i++){
  for(j=0;j<n;j++)
    printf("  “);
Printf(“%d”,arr[i][j]);
  printf("\n“);
  }
return 0;
}
```

Exchange the Rows and Columns of a ‘*mxn*’ matrix

```
/*read ‘mxn’ matrix */
printf("\nEnter the rows to exchange: “);
scanf(“%d %d”,&r1,&r2);
/*Row exchange r1 ⇔ r2 */
for(j=0;j<n;j++) {
   temp=arr[r1-1][j];
   arr[r1-1][j]=arr[r2-1][j];
   arr[r2-1][j]=temp;  }
```

```
printf(“\nEnter the cols to exchange: “);
scanf(“%d %d”,&c1,&c2);
/*Column exchange : c1 ⇔ c2 */
for(i=0;i<m;i++) {
   temp=arr[i][c1-1];
   arr[i][c1-1]=arr[i][c2-1];
   arr[i][c2-1]=temp; }
```

Multiplication of two Matrices

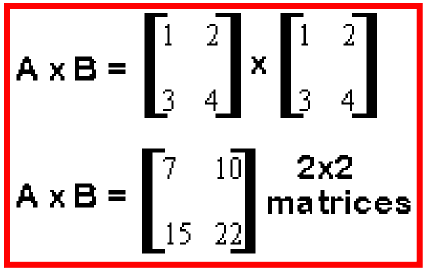
```
#include <stdlib.h>

int main(){ int i, j, m, n, p, q;

int a[10][10], b[10][10],c[10][10];

printf("enter dimension for a \n“);

scanf(“%d %d”,&m,&n);

printf("\n enter dimension for b\n“);

scanf(“%d %d”, &p,&q);

if(n!=p){

  printf("not multiplicable \n“);

  exit(0);  }

printf("enter elements for a \n“);

for (i=0;i<m;i++)

{

   for(j=0;j<n;j++)

      scanf(“%d”,&a[i][j]);

}

printf("\n enter elements for b\n“

for(i=0;i<p;i++)

{  for(j=0;j<q;j++)

scanf(“%d”,&b[i][j]);

}
```

$$A \times B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$ 2x2 matrices

Multiplication of two Matrices

```
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]=c[i][j]+a[i][k]*b[k][j];
   }
}
```



$$A \times B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

2x2 matrices

```
printf("\n The product matrix is \n“);
for(i=0;i<m;i++){
  for(j=0;j<q;j++)
    printf(“%d\t“,c[i][j]);
  printf("\n“);
}
```