



2 D A R R A Y S



Objectives

To learn and appreciate the following concepts

- **2D Array declaration, initialization**
- **Programs using 2D arrays**



Session outcome

At the end of session student will be able to

- Declare, initialize and access 2D array
- Write programs using 2D array

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

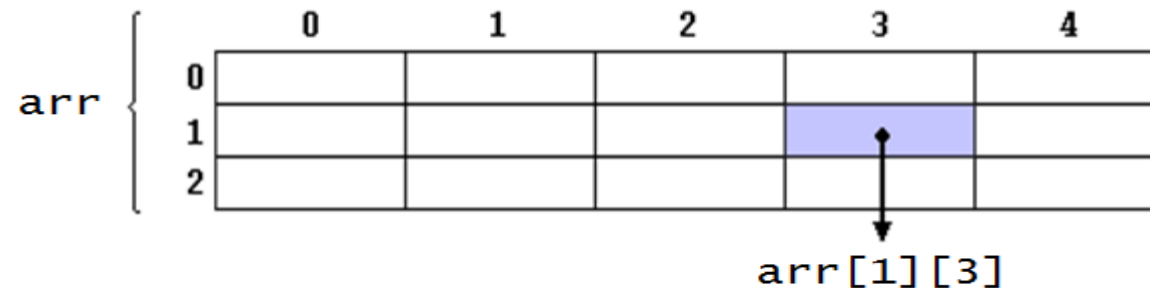
Declaration

`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.



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};
```


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);

    printf("\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",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 ("\n")  
    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
```

Multiplication of two Matrices

```
#include <stdio.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]);
```

```
}
```

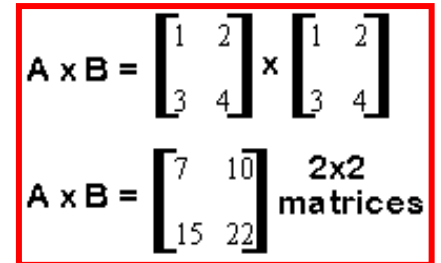
```
printf("\n enter elements for b\n");
```

```
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];
    }
}
```

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

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

$$\begin{matrix} \vec{b}_1 \\ \downarrow \\ \vec{b}_2 \end{matrix}$$

printf("\n The product matrix is \n");

$$\begin{matrix} \vec{a}_1 \rightarrow \\ \vec{a}_2 \rightarrow \end{matrix} \begin{matrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \\ A \end{matrix} \cdot \begin{matrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \\ B \end{matrix} = \begin{matrix} \begin{bmatrix} \vec{a}_1 \cdot \vec{b}_1 & \vec{a}_1 \cdot \vec{b}_2 \\ \vec{a}_2 \cdot \vec{b}_1 & \vec{a}_2 \cdot \vec{b}_2 \end{bmatrix} \\ C \end{matrix}$$

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


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

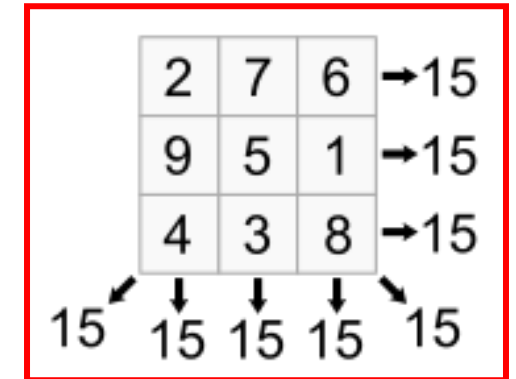
```
printf("enter dimension \n");  
  
scanf("%d %d",&m,&n);  
  
if(m!=n){  
    printf("Enter a square matrix \n");  
    exit(0);  
}  
  
printf("enter elements \n");  
for(i=0; i<m; i++)  
    for(j=0; j<n; j++)  
        scanf("%d", &a[i][j]);
```

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

Problem...

Write program to check the given matrix is a magic square or not

A **magic square** of order n is an arrangement of n^2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant.



2	7	6	→15
9	5	1	→15
4	3	8	→15
15	15	15	15

A **normal** magic square contains the integers from 1 to n^2 .

Magic Square

```
//Matrix is Magic square or not
void main()
{
int  mag[10][10], i, j, row, col, rowsum[10], colsum[10];
int pd=0, sd=0, k, x=0, b[100];
clrscr();
printf("enter dimension \n");
scanf("%d %d",&row,&col);
if(row!=col) // checking for square matrix
{
printf("matrix is not square");
exit(0);
}
```

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

Magic Square

//inputting elements to the array

```
printf("\n enter elements for a \n");  
for(i=0; i<row; i++)  
{  
    for(j=0; j<col; j++)  
        scanf("%d",&mag[i][j]);  
}
```

//copying elements to 1D

```
for(i=0; i<row; i++)  
    for(j=0; j<col; j++)  
        b[x++]=mag[i][j];
```

//checking for uniqueness

```
for(k=0; k<x-1; k++)  
    for(j=k+1; j<x; j++)  
        if(b[k]==b[j])  
        {  
            printf("elements are no distinct\n");  
            printf("matrix is not magic");  
            exit(0);  
        }
```

Magic Square

//Finding sum of elements on principal Diagonal

```
for(i=0; i<row; i++)  
    pd=pd + mag[i][i];
```

//Row sum

```
for(i=0; i<row; i++)  
{  
    rowsum[i]=0;  
    for(j=0; j< col; j++)  
        rowsum[i]=rowsum[i]+mag[i][j];
```

//comparing rowsum and principal diagonal sum

```
    if(rowsum[i]!=pd)  
    {  
        printf("matrix is not magic");  
        exit(0);  
    }  
}
```

Magic Square

//Finding column sum

```
for(i=0;i<col;i++)  
{  
    colsum[i]=0;  
    for(j=0;j<row;j++)  
        colsum[i]=colsum[i]+mag[j][i];
```

//comparing Columnsum and principal diagonal sum

```
    if(colsum[i]!=pd){ printf("matrix is not magic");  
exit(0);  
}  
}
```

Magic Square

//finding secondary diagonal sum

i=row-1;

k=i;

for(j=col-1;j>=0;j--,i--)

sd=sd+mag[i][k-j];

if(sd!=pd) {

printf("matrix is not magic");

exit(0);

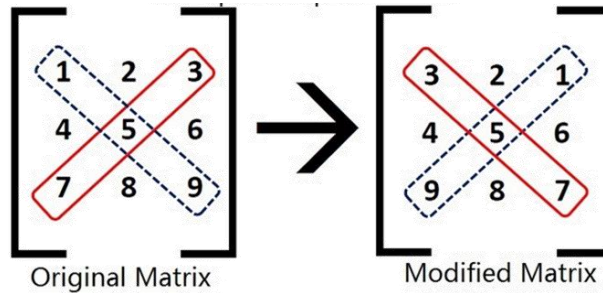
}

printf("Matrix is magic\n");

return 0;

}

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

```
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;
}
```

```
/*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;
}
```

Syntax

Declaration:

data-type array_name[row_size][column_size];

Initialization of two dimensional arrays:

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

Reading a Matrix:

```
int a[100][100];
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        scanf("%d",&a[i][j]);
}
```

Display a Matrix:

```
int a[100][100];
for(i=0;i<m;i++){
    for(j=0;j<n;j++)
        printf("%d",a[i][j]);
    printf(" ");
    printf("\n");
}
```



Tutorials

- Write a program to check whether the given matrix is sparse matrix or not.
- Write a program to find the sum of the elements above and below diagonal elements in a matrix.
- Write a program to find the determinant of a matrix.



Summary

- 2 Dimensional Arrays