TWO-DIMENSIONAL ARRAYS

Till now, we have only discussed one-dimensional arrays. One-dimensional arrays are organized linearly in only one direction. But at times, we need to store data in the form of grids or tables. Here, the concept of single-dimension arrays is extended to incorporate two-dimensional data structures.

A two-dimensional array is specified using two subscripts where the first subscript denotes the row and the second denotes the column. The C compiler treats a two-dimensional array as an array of one-dimensional arrays.

Figure 3.1 shows a two-dimensional array which can be viewed as an array of arrays.

3.1 Declaring Two-dimensional Arrays

Any array must be declared before being used. The declaration statement tells the compiler **the name of the array**, **the data type of each element** in the array, and **the size of each dimension**. A two-dimensional array is declared as:

data_type array_name[row_size][column_size];

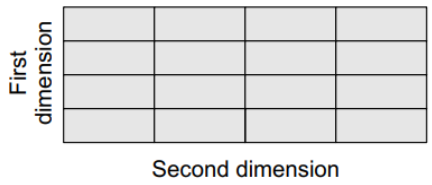

Figure 3.1 Two-dimensional array

Therefore, a two-dimensional **m × n array** is an array that contains **m × n data elements** and each element is accessed using two subscripts, **i** and **j**, **where i<=m and j<=n**.

For example, if we want to store the marks obtained by **three students** in **five different subjects**, we can declare a two-dimensional array as: **int marks[3][5];**

In the above statement, a **two-dimensional array** called **marks** has been declared that has **m(3) rows** and **n(5) columns**.

The pictorial form of a two-dimensional array is shown in **Fig. 3.2**.

Rows / Columns	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Figure 3.2 Two-dimensional array

Hence, we see that a 2D array is treated as a collection of 1D arrays. Each row of a **2D array** corresponds to a **1D array** consisting of **n elements**, where **n** is the number of columns. To understand this, we can also see the representation of a two-dimensional array as shown in **Fig. 3.3**.

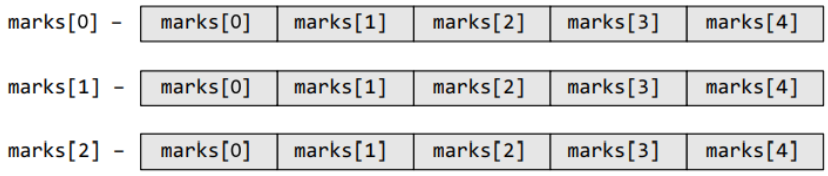

Figure 3.3 Representation of two-dimensional array marks[3][5]

Although we have shown a rectangular picture of a two-dimensional array, **in the memory, these elements actually will be stored sequentially**.

<u>There are two ways of storing a two-dimensional array in the memory:</u>

- **The first way** is the row major order.
- **The second way** is the column major order.

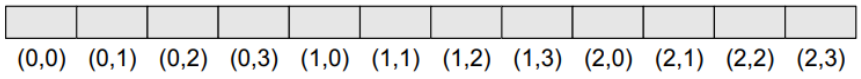

Figure 3.4 Elements of a 3 × 4 2D array in row major order

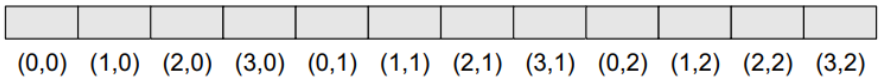

Figure 3.5 Elements of a 4 × 3 2D array in column major order

In one-dimensional arrays, we have seen that the computer does not keep track of the address of every element in the array. It stores only the address of the first element and calculates the address of other elements from the base address (address of the first element).

Same is the case with **a two-dimensional array**. Here also, the computer stores **the base address**, and **the address of the other elements** is calculated using **the following formula**:

⇨ If the array elements are stored in column major order,

$$\text{Address}(A[I][J]) = \text{Base\_Address} + w\{M(J-1) + (I-1)\}$$

⇨ And if the array elements are stored in row major order,

$$\text{Address}(A[I][J]) = \text{Base\_Address} + w\{N(I-1) + (J-1)\}$$

where

w is the number of bytes required to store one element,
N is the number of columns,
M is the number of rows,
and **I** and **J** are the subscripts of the array element.

Example 3.1 : Consider a 20 × 5 two-dimensional array marks which has its base address = 1000 and the size of an element = 2. Now compute the address of the element, marks[18][4] assuming that the elements are stored in row major order.

Solution

$$\begin{aligned}
\text{Address}(A[I][J]) &= \text{Base_Address} + w\{N(I-1) + (J-1)\} \\
\text{Address}(marks[18][4]) &= 1000 + 2\{5(18-1) + (4-1)\} \\
&= 1000 + 2\{5(17) + 3\} \\
&= 1000 + 2(88) \\
&= 1000 + 176 \\
&= 1176
\end{aligned}$$

3.2 Initializing Two-dimensional Arrays

Like in the case of other variables, declaring a two-dimensional array only reserves space for the array in the memory. No values are stored in it. A two-dimensional array is initialized in the same way as a one-dimensional array is initialized. **For example**,

Int marks[2][3]={90, 87, 78, 68, 62, 71};

Note that the initialization of a two-dimensional array is done row by row. The above statement can also be written as:

Int marks[2][3]={{90,87,78},{68, 62, 71}};

The above two-dimensional array has two rows and three columns. First, the elements in the first row are initialized and then the elements of the second row are initialized.

Therefore, marks[0][0] = 90 marks[0][1] = 87 marks[0][2] = 78
marks[1][0] = 68 marks[1][1] = 62 marks[1][2] = 71

In the above example, each row is defined as a one-dimensional array of three elements that are enclosed in braces. Note that the commas are used to separate the elements in the row as well as to separate the elements of two rows.

In case of one-dimensional arrays, we have discussed that if the array is completely initialized, we may omit the size of the array. The same concept can be applied to a two-dimensional array, except that only the size of the first dimension can be omitted. Therefore, the declaration statement given below is valid.

Int marks[][3]={{90,87,78},{68, 62, 71}};

In order to initialize the entire two-dimensional array to zeros, simply specify the first value as zero. That is,

Int marks[2][3] = {0};

3.3 Accessing the Elements of Two-dimensional Arrays

The elements of a 2D array are stored in contiguous memory locations. In case of one-dimensional arrays, we used a single for loop to vary the index i in every pass, so that all the elements could be scanned.

Since the two-dimensional array contains two subscripts, we will use two for loops to scan the elements. The first for loop will scan each row in the 2D array and the second for loop will scan individual columns

for every row in the array. Look at the programs which use two for loops to access the elements of a 2D array.

PROGRAMMING EXAMPLES

16. Write a program to print the elements of a 2D array.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int arr[2][2] = {12, 34, 56,32};
        int i, j;
        for(i=0;i<2;i++)
        {
              printf("\n");
              for(j=0;j<2;j++)
                      printf("%d\t", arr[i][j]);
        }
        return 0;
}
```

Output

```
12      34
56      32
```

3.4 OPERATIONS ON TWO-DIMENSIONAL ARRAYS

Two-dimensional arrays can be used to implement the mathematical concept of matrices. In mathematics, a matrix is a grid of numbers, arranged in rows and columns. Thus, using twodimensional arrays, we can perform **the following operations on an m×n matrix:**

- **Transpose**: Transpose of an **m×n** matrix **A** is given as a **n×m** matrix **B**, where $B_{i,j} = A_{j,i}$.
- **Sum:** Two matrices that are compatible with each other can be added together, storing the result in the **third matrix**. **Two matrices are said to be compatible when they have the same number of rows and columns**. The elements of two matrices can be added by writing:

$$C_{i,j} = A_{i,j} + B_{i,j}$$

- **Difference:** Two matrices that are compatible with each other can be subtracted, storing the result in the third matrix. Two matrices are said to be compatible when they have the same number of rows and columns. The elements of two matrices can be subtracted by writing:

$$C_{i,j} = A_{i,j} - B_{i,j}$$

- **Product:** Two matrices can be multiplied with each other if the number of columns in the first matrix is equal to the number of rows in the second matrix. Therefore, **m × n** matrix **A** can be multiplied with a **p × q** matrix **B** if **n=p**. The dimension of the product matrix is **m × q**. The elements of two matrices can be multiplied by writing:

$$C_{i,j} = \sum A_{i,k} B_{k,j} \quad \text{for } \mathbf{k=1} \text{ to } \mathbf{n}$$

PROGRAMMING EXAMPLES

20. Write a program to read and display a 3×3 matrix.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, j, mat[3][3];
        clrscr();
        printf("\n Enter the elements of the matrix ");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                    scanf("%d",&mat[i][j]);
                }
        }
        printf("\n The elements of the matrix are ");
        for(i=0;i<3;i++)
        {
                printf("\n");
                for(j=0;j<3;j++)

                        printf("\t %d",mat[i][j]);
        }
        return 0;
}
```

Output

```
Enter the elements of the matrix
1 2 3 4 5 6 7 8 9
The elements of the matrix are
1 2 3
4 5 6
7 8 9
```

21. Write a program to transpose a 3×3 matrix.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, j, mat[3][3], transposed_mat[3][3];
        clrscr();
        printf("\n Enter the elements of the matrix ");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        scanf("%d", &mat[i][j]);
                }
        }
        printf("\n The elements of the matrix are ");
        for(i=0;i<3;i++)
        {
                printf("\n");
                for(j=0;j<3;j++)
                        printf("\t %d", mat[i][j]);
        }
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                        transposed_mat[i][j] = mat[j][i];
        }
        printf("\n The elements of the transposed matrix are ");
        for(i=0;i<3;i++)
        {
                printf("\n");
                for(j=0;j<3;j++)
                        printf("\t %d",transposed_ mat[i][j]);
        }
        return 0;
}
```

Output

```
Enter the elements of the matrix
1 2 3 4 5 6 7 8 9
The elements of the matrix are
1 2 3
4 5 6
7 8 9
The elements of the transposed matrix are
1 4 7
2 5 8
3 6 9
```

22. Write a program to input two m × n matrices and then calculate the sum of their corresponding elements and store it in a third m × n matrix.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, j;
        int rows1, cols1, rows2, cols2, rows_sum, cols_sum;
        int mat1[5][5], mat2[5][5], sum[5][5];
        clrscr();
        printf("\n Enter the number of rows in the first matrix : ");
        scanf("%d",&rows1);
        printf("\n Enter the number of columns in the first matrix : ");
        scanf("%d",&cols1);
        printf("\n Enter the number of rows in the second matrix : ");
        scanf("%d",&rows2);
        printf("\n Enter the number of columns in the second matrix : ");
        scanf("%d",&cols2);
        if(rows1 != rows2 || cols1 != cols2)
        {
             printf("\n Number of rows and columns of both matrices must be equal");
             getch();
             exit();
        }
        rows_sum = rows1;
        cols_sum = cols1;
        printf("\n Enter the elements of the first matrix ");
        for(i=0;i<rows1;i++)
        {
                for(j=0;j<cols1;j++)
                {
                        scanf("%d",&mat1[i][j]);
                }
        }
        printf("\n Enter the elements of the second matrix ");
        for(i=0;i<rows2;i++)
        {
                for(j=0;j<cols2;j++)
                {
                        scanf("%d",&mat2[i][j]);
                }
        }
        for(i=0;i<rows_sum;i++)
        {
                for(j=0;j<cols_sum;j++)
                        sum[i][j] = mat1[i][j] + mat2[i][j];
        }
        printf("\n The elements of the resultant matrix are ");
        for(i=0;i<rows_sum;i++)
        {
                printf("\n");
                for(j=0;j<cols_sum;j++)
                        printf("\t %d", sum[i][j]);
        }
        return 0;
}
```

Output

```
Enter the number of rows in the first matrix: 2
Enter the number of columns in the first matrix: 2
Enter the number of rows in the second matrix: 2
Enter the number of columns in the second matrix: 2
Enter the elements of the first matrix
1 2 3 4
Enter the elements of the second matrix
5 6 7 8
The elements of the resultant matrix are
6 8
10 12
```

23. Write a program to multiply two `m × n` matrices.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, j, k;
        int rows1, cols1, rows2, cols2, res_rows, res_cols;
        int mat1[5][5], mat2[5][5], res[5][5];
        clrscr();
        printf("\n Enter the number of rows in the first matrix : ");
        scanf("%d",&rows1);
        printf("\n Enter the number of columns in the first matrix : ");
        scanf("%d",&cols1);
        printf("\n Enter the number of rows in the second matrix : ");
        scanf("%d",&rows2);
        printf("\n Enter the number of columns in the second matrix : ");
        scanf("%d",&cols2);
        if(cols1 != rows2)
        {
                printf("\n The number of columns in the first matrix must be equal
                to the number of rows in the second matrix");
                getch();
                exit();
        }
        res_rows = rows1;
        res_cols = cols2;
        printf("\n Enter the elements of the first matrix ");
        for(i=0;i<rows1;i++)
        {
                for(j=0;j<cols1;j++)
                {
                        scanf("%d",&mat1[i][j]);
                }
        }
        printf("\n Enter the elements of the second matrix ");
        for(i=0;i<rows2;i++)
        {
                for(j=0;j<cols2;j++)
                {
                        scanf("%d",&mat2[i][j]);
                }
        }
        for(i=0;i<res_rows;i++)
        {
                for(j=0;j<res_cols;j++)
```

```
                {
                        res[i][j]=0;
                        for(k=0; k<res_cols;k++)
                        res[i][j] += mat1[i][k] * mat2[k][j];
                }
        }
        printf("\n The elements of the product matrix are ");
        for(i=0;i<res_rows;i++)
        {
                printf("\n");
                for(j=0;j<res_cols;j++)
                        printf("\t %d",res[i][j]);
        }
        return 0;
}
```

Output

```
Enter the number of rows in the first matrix: 2
Enter the number of columns in the first matrix: 2
Enter the number of rows in the second matrix: 2
Enter the number of columns in the second matrix: 2
Enter the elements of the first matrix
1 2 3 4
Enter the elements of the second matrix
5 6 7 8
The elements of the product matrix are
19 22
43 50
```