# ARRAYS IN C

## 4.1 INTRODUCTION

- An array is a set of elements of the same data type that are referred to by the same name.
- An array is a fixed-size collection of elements having same data type.
- An array in C programming language can be defined as "number of memory locations, each of which can store the same data type and which can be references through the same variable name".

**OR**

- Array can be defined as, "a collection of variables belongings to the same data type".
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element, (See Fig. 4.1).

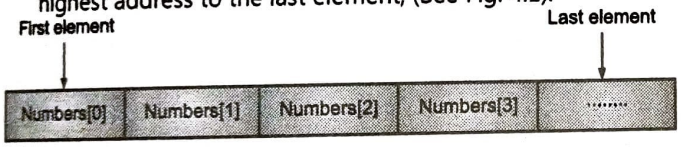



**Fig. 4.1**

- Various properties of arrays are:
  - The size of an array is the number of elements in each dimension.
  - The number of dimensions.
  - The type of an array is the data type of its elements.
  - The name of an array.
- Fig. 4.2 shows types of arrays in C.

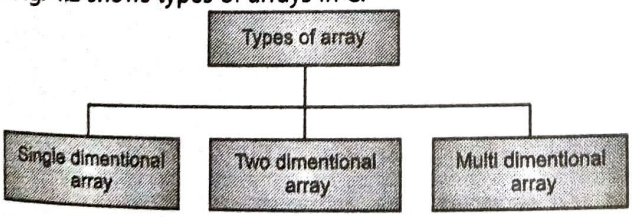



**Fig. 4.2: Types of arrays**

- Fig. 4.2 shows following types of arrays:
  - **Single Dimensional Array:** Single dimensional array is used to represent and store data in a linear form. Array having only one subscript variable is called one dimensional array. It is also called as one dimensional array or linear array.
  - **Two Dimensional Array:** Two dimensional array requires two subscript variables. It stores the values in the form of matrix. One subscript variable denotes the "Row" of a matrix. Another subscript variable denotes the "Column" of a matrix.
  - **Multi-Dimensional Array:** Array having more than one subscript variable is called multi-dimensional array. Multi-dimensional array is also called as matrix.

**Advantages of Array:**

- **Easy Access of Elements:** We can access any element of array using array name and index. We can access all elements serially by iterating from index 0 to size-1 using a loop.
- **Random Access:** We can access any elements of array in O(1) time complexity.
- **Less Amount of Code:** Using array we can aggregate N variables of same data type in a single data structure. Otherwise, we have to declare N individual variables.
- **Easy to Implement Algorithms:** Certain algorithms can be easily implemented using array like searching and sorting, finding maximum and minimum elements.

## 4.2 ARRAY DECLARATION

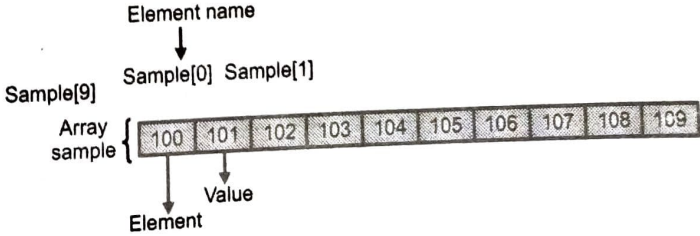
- Array is a collection of homogenous data stored under unique name.
- The values in an array are called as 'elements of an array.' These elements are accessed by numbers called as 'subscripts or index numbers.'
- Arrays may be of any variable type.
- Array is also called as 'subscripted variable.'
- An array is declared as,

```
data_type name_of_array[size];
```

**Example:**

```
int sample[10];
```

Here, int = data type of all individual data items
sample = name of the array
10 = 10 different integer values can be stored in this array or site of array.



- The data elements are referred to as sample[0], sample[1], sample[2], sequentially upto sample[9].

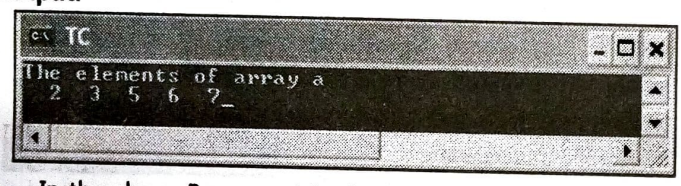
  i.e., sample[0] = 100;
  sample[1] = 101;
  :
  :
  sample[9] = 109;

- These addresses of the individual array elements are also known as indexes or subscripts.

**Program 4.1:** Program for simple array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i, a[5] = {2, 3, 5, 6, 7};
   clrscr();
   printf("The elements of array a \n");
   for(i = 0; i <= 4; i++)
   printf("%3d", a[i]);
   getch();
}
```

**Output:**




- In the above Program 4.1 a is declared to be an array of int type and of size five and also initialized while it is declared. i is a variable of int type and it is used to traverse the elements of a.
- To display the elements of a, we have used a for loop to repeatedly call printf(). The loop variable i is made to range from 0 to 4. When i takes 0, the first element is displayed. When i takes 1, the second element is displayed and so on.

**Accessing Array Elements and Displaying Array Elements:**

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

  **For example:**

  double salary = balance[9];

- The above statement will take 10th element from the array and assign the value to salary variable.

## 4.3 INTIALIZING CHARACTER ARRAY

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows –

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++ –

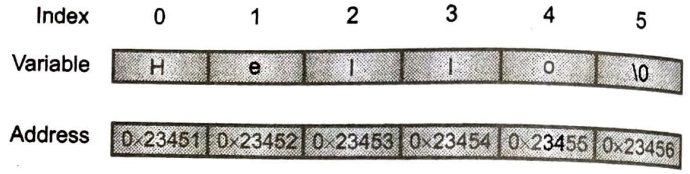



**Fig. 4.3**

Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string –

```
#include <stdio.h>
int main () {
  char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  printf("Greeting message: %s\n", greeting );
  return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Greeting message: Hello
```

C supports a wide range of functions that manipulate null-terminated strings –

| S. N. | Function and Purpose |
|---|---|
| 1 | **strcpy(s1, s2);**<br>Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**<br>Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**<br>Returns the length of string s1. |
| 4 | **strcmp(s1, s2);**<br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**<br>Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**<br>Returns a pointer to the first occurrence of string s2 in string s1. |

The following example uses some of the above-mentioned functions –

```
#include <stdio.h>
#include <string.h>
int main () {

 char str1[12] = "Hello";
 char str2[12] = "World";
 char str3[12];
 int  len ;

 /* copy str1 into str3 */
 strcpy(str3, str1);
 printf("strcpy( str3, str1) :  %s\n", str3 );

 /* concatenates str1 and str2 */
 strcat( str1, str2);
 printf("strcat( str1, str2):   %s\n", str1 );

 /* total lenghth of str1 after concatenation */
 len = strlen(str1);
 printf("strlen(str1) :  %d\n", len );

 return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

## 4.4 SINGLE DIMENSIONAL ARRAY

- A list of items can be given one variable name using only one subscript such a variable is called as a single subscripted variables or one dimensional array.
- **Definition:** An array which is having either a single row or single column is termed as a one dimensional array. One dimensional array have the subscripts in a linear manner.

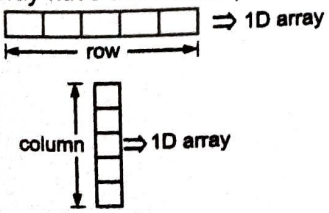



**Fig. 4.4: Single dimensional array**

### 4.4.1 Declaration of One Dimensional Array

- Like other general variables, the array is also declared in the program before its use.
- Syntax to declare a one dimensional array is,

```
data-type variable-name[size];
```

The 'data-type' specifies type of the array. That is, int, char, float etc., and the 'variable-name' is the name given to the array, by which we will refer it. Finally, size is the maximum number of elements that an array can hold.

**Example:**

```
int var[10];
```

- This declares the array 'var' containing 10 different integer elements. We can store 10 integer constants in the variable 'var'.

### 4.4.2 Initialization of One Dimensional Array

- After an array is declared, it must be initialized. Otherwise they will contain the garbage values.
- There are two different ways in which we can initialize the array i.e., Compile time and Run time.

**(i) Compile Time Initialization:**

- This initialization is done while writing the program itself. Following is the syntax of initialization of the array.

```
data-type array-name[size] = { list of values };
```

**For example:**

```
int var[5] = {5, 7, 9, 3, 4};
```

- Here, the array variables are initialized with the values given in front of it. That is, the 0th element of array will contain 5, 1st will contain 7 and so on. Remember, the array indexing or subscripting is always done from 0th position. The values of the array will be get stored as shown in Fig. 4.5.

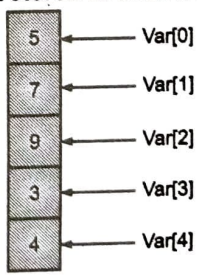



**Fig. 4.5**

- Each element in the array will be referred individually by its index such as, var[0], var[1], var[2], . . . They can also be initialized individually such as:

```
var[0] = 5;
var[1] = 7;
var[2] = 9;
```

- In the array declaration, the size can also be omitted. In such cases the compiler will automatically identify the total number of elements and size will be given to it.

**For example:**

```
int max[ ] = {8, 6, 7, 4, 5, 3, 0, 1};
```

- This array is not declared with size, but initialized with values. Here, size of the array will be automatically set to 8. Now it can be referred as the general array onwards.
- Compile time initialization can also be done partially. Such as,

```
float x[5] = {1.2, 6.9, 4.1};
```

Here, array 'x' is having size 5, but only 0th, 1st, and 2nd elements are declared with values. The remaining 3rd and 4th element of the array will be automatically set to 0. This property is applicable to all numerical type of the array in C. remember, however, if we have more values than the declared size, the compiler will give an error.

**For example:**

```
int arr[5] = {56, 10, 30, 74, 56, 32};
```

This is the illegal statement in C.

**(ii) Run Time Initialization:**

- An array can initialized at run time by the program or by taking the input from the keyboard.
- Generally, the large arrays are declared at run time in the program it self. Such as:

```
int sum[20];
for (i = 0;i<20;i++)
sum[i] = 1;
```

Here, all the elements of the array 'sum' are initialized with the value 1. Remember the loop control structures are applicable in these cases. We can also use the scanf() function to input the values from the user. In such cases, the loop control structure is applicable.

**For example:**

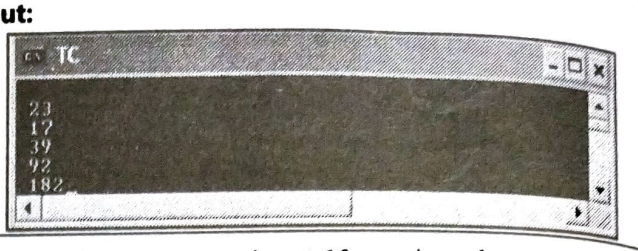
```
int sum[10], x;
printf("Enter 10 numbers: ");
for(x=0;x<10;x++)
scanf("%d",&sum[x]);
```

Here, the array 'sum' is initialized by taking the values as input from the keyboard.

**Program 4.2:** Program to print contents of array which is initialized.

```
#include<stdio.h>
void main()
{
    int array[5] = { 23, 17, 39, 92, 182};
    int i;
    clrscr();
    for(i=0;i<5;i++)
    {
    printf("\n %d", array[i]);
    }
    getch();
}
```

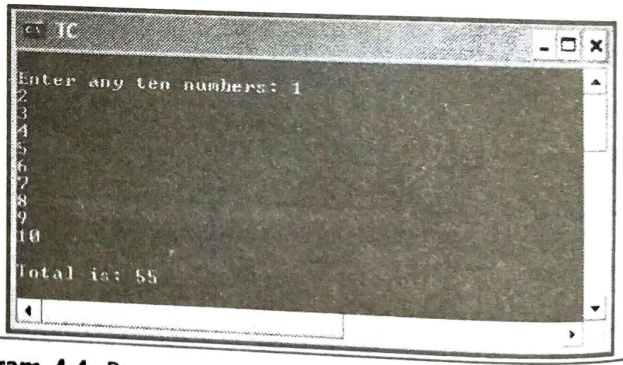
**Output:**




**Program 4.3:** Program to input 10 numbers from user and find the total of all of them.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int val[10], i, total=0;
    clrscr();
    printf("\n Enter any ten numbers: ");
    for(i=0;i<10;i++)
    {
        scanf("%d",&val[i]);  // input numbers
    }
    for(i=0;i<10;i++)
    {
        total = total + val[i];   // find total
    }
    printf("\n Total is: %d", total);
    getch();
}
```

**Output:**




**Program 4.4:** Program to find the smallest and largest numbe from an array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10],n,i,large,small;
    printf("enter the number of elements:\n");
    scanf("%d",&n);
    printf("enter the elements:\n");
    for(i=0;i<n;i++)
```

```
    scanf("%d",&a[i]);
    large=a[0];
    small=a[0];
    for(i=1;i<n;i++)
    {
    if(a[i]>large)
    large=a[i];
    if(a[i]<small);
    small=a[i];
    }
    printf("Largest element in the array is: %d\n",large);
    printf("Smallest element in the array is:
    %d\n",small);
    getch();
}
```

Output:

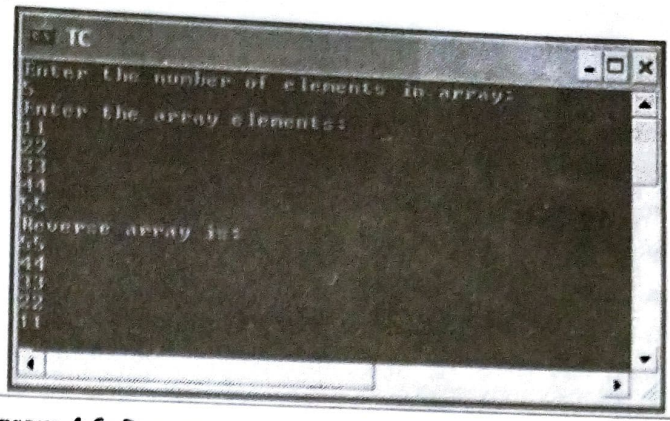
```
enter the number of elements: 5
enter the elements:
78
23
7
19
35
Largest element in the array is: 78
Smallest element in the array is: 7
```

**Program 4.5:** Program to reverse an array.

```
#include<stdio.h>
#include<conio.h>
void main()

{
int n, i, j, a[100], b[100];
clrscr();
printf("Enter the number of elements in array:\n");
scanf("%d", &n);
printf("Enter the array elements:\n");
for (i = 0; i < n ; i++)
scanf("%d", &a[i]);
for (i = n - 1, j = 0; i >= 0; i--, j++)
b[j] = a[i];
for (i = 0; i < n; i++)
a[i] = b[i];
printf("Reverse array is:\n");
for (i = 0; i < n; i++)
printf("%d\n", a[i]);
getch();
}
```

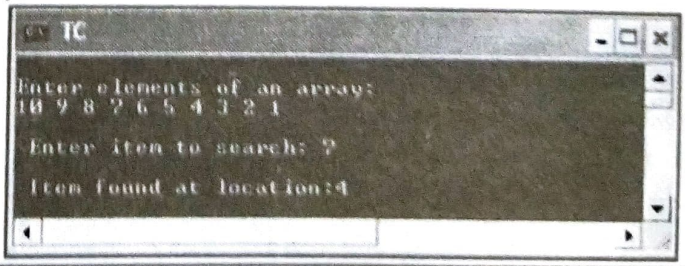
Output:




**Program 4.6:** Program to search an element in the array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10], i, item;
    clrscr();
    printf("\n Enter elements of an array:\n");
    for (i=0; i<=9; i++)
    scanf("%d", &a[i]);
    printf("\n Enter item to search: ");
    scanf("%d", &item);
    for (i=0; i<=9; i++)
    {
        if (item == a[i])
        {
        printf("\n Item found at location:%d", i+1);
            break;
        }
    }
    if (i > 9)
    printf("\n Item doesnot exist.");
    getch();
}
```

Output:




## 4.5 TWO DIMENSIONAL ARRAYS

- The array which is used to represent and store data in a tabular form is called as 'two dimensional array.' Such type of array specially used to represent data in a matrix form.
- It is also called as array of arrays.

- A two dimensional array is referred as a matrix or a table.
- A matrix has two subscripts, one denotes the row and the other denotes the column. The first dimension (i.e. the first index) might refer to the row number, and the second dimension (i.e. the second index) to the column number.
- Two-dimensional array can be defined as, "those type of array, which has finite number of rows and finite number of columns".
- The other option is to have a single array with 2-dimension (rows and columns), which will store all the information as shown in the Fig. 4.6.

| | column 0 | column 1 ..... | | | column n |
|---|---|---|---|---|---|
| | Subject and Roll no | Maths | Computer | | |
| row 0 | 4001 | ..... | ..... | ..... | ..... |
| row 1 | 4002 | ..... | ..... | ..... | ..... |
| : | : | ..... | ..... | ..... | ..... |
| : | : | ..... | ..... | ..... | ..... |
| row n | : | ..... | ..... | ..... | ..... |

**Fig. 4.6: Two-dimensional array**

### 4.5.1 Declaration and Initialization of Two-Dimensional Array

- Declaration syntax of two-dimensional array is as follows:

```
data_type array_name[rows][columns];
```

Where,

| | | |
|---|---|---|
| data_type | : | The type of the data stored in the array |
| array_name | : | Name of the array |
| rows | : | Maximum number of rows in the array |
| columns | : | Maximum number of columns in the array |
| Example | : | int value[3][3]; // implies 3 rows and 3 columns |
| Initialization | : | Two-dimensional arrays may be initialized by their declaration with a list of initial values enclosed in braces, like |

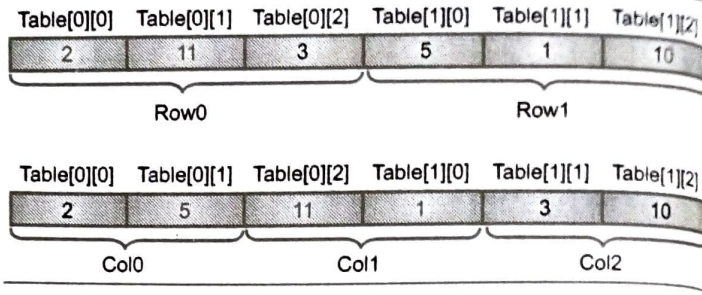
```
int table[2][3] = {2, 11, 3, 5, 1, 10};
```

- Above initialized two-dimensional array can be stored in the memory in any of the three foms mentioned below:

**1. Two-Dimensional Form as Follows:**

| | col 0 | col 1 | col 2 |
|---|---|---|---|
| row 0 | 2 | 11 | 3 |
| Row 1 | 5 | 1 | 10 |

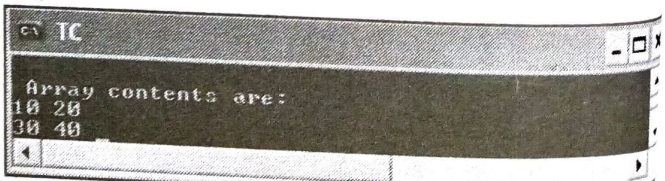
**2. Row-Wiselinear List:** Elements of row0 are stored first and then row1 is stored.




**Program 4.7:** Program to initialized two dimensional array a print the contents.

```
#include<stdio.h>
void main()
{
    int array[2][2] =
    {
        {10,20},
        {30,40}
    };
    int i=0,j=0;
    clrscr();
    printf("\n Array contents are:");
    for(i=0;i<2;i++)
    {
        printf("\n");
        for(j=0;j<2;j++)
        {
        printf("%d ", array[i][j]);
        }
    }
    getch();
}
```

**Output:**




## 4.6 MULTI-DIMENSIONAL ARRAY

- C language allows arrays of three or more dimensions called as multi-dimensional array.
- The syntax is:

```
type array_name[s1], [s2], ... , [sn];
```

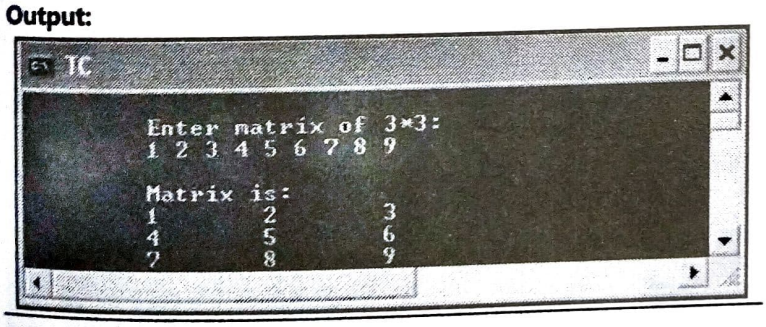
**Example:**

```
int a[4][5][6];
float t[10][20][30][40];
```

**Program 4.8:** Program to accept elements for the matrix and display the contents.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3], i, j;
    clrscr();
    printf("\n\t Enter matrix of 3*3: ");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
        scanf("%d",&a[i][j]); //read 3*3 array
        }
    }
    printf("\n\t Matrix is: \n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
        printf("\t %d",a[i][j]); //print 3*3 array
        }
        printf("\n");
    }
    getch();
}
```

**Output:**




**Program 4.9:** C program to add two matrix.

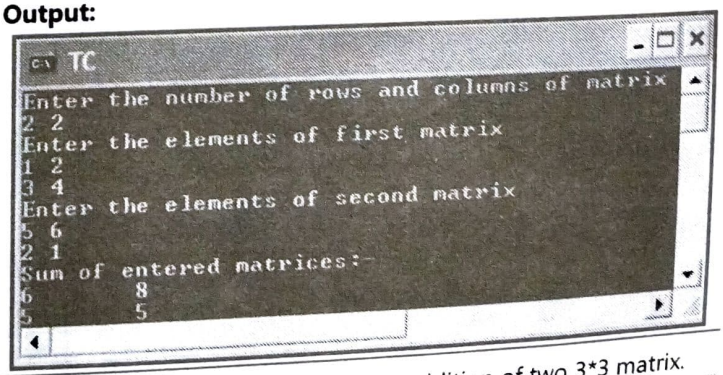
This program adds two matrix i.e., compute the sum of two matrices and then print it, firstly user will be asked to enter the order of matrix (number of rows and columns) and then two matrices. For example, if the user entered order as 2, 2 i.e. two rows and two columns and matrices as,

```
First Matrix:
    1 2
    3 4
Second matrix:
    4 5
    -1 5
```

then output of the program (sum of First and Second matrix) will be,

```
    5 7
    2 9
#include<stdio.h>
int main()
{
    int m, n, c, d, first[10][10], second[10][10], sum[10][10];
    clrscr();
    printf("Enter the number of rows and columns of
    matrix\n");
    scanf("%d%d", &m, &n);
    printf("Enter the elements of first matrix\n");
    for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
    scanf("%d", &first[c][d]);
    printf("Enter the elements of second matrix\n");
    for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
    scanf("%d", &second[c][d]);
    for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
    sum[c][d] = first[c][d] + second[c][d];
    printf("Sum of entered matrices:-\n");
    for ( c = 0 ; c < m ; c++ )
    {
        for ( d = 0 ; d < n ; d++ )
        printf("%d\t", sum[c][d]);
        printf("\n");
    }
    return 0;
}
```

**Output:**

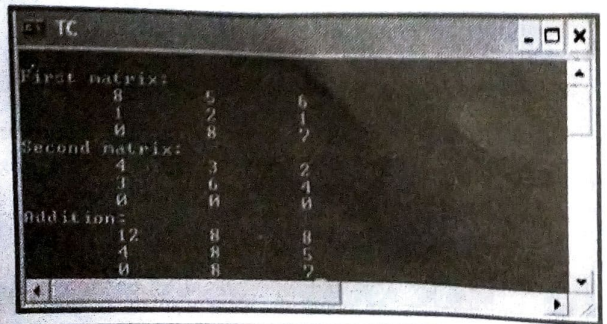



**Program 4.10:** Program to perform addition of two 3*3 matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
    int x[][3] = { {8,5,6},
                   {1,2,1},
                   {0,8,7}
                 };
    int y[][3] = { {4,3,2},
                   {3,6,4},
                   {0,0,0}
                 };
    int i,j;
    clrscr();
    printf("\n First matrix: ");
    for(i=0;i<3;i++)
    {
        printf("\n");
        for(j=0;j<3;j++)
        printf("\t%d", x[i][j]);
    }
    printf("\n Second matrix: ");
    for(i=0;i<3;i++)
    {
        printf("\n");
        for(j=0;j<3;j++)
        printf("\t%d",y[i][j]);
    }
    printf("\n Addition: ");
    for(i=0;i<3;i++)
    {
        printf("\n");
        for(j=0;j<3;j++)
        printf("\t%d", x[i][j]+y[i][j]);
    }
    getch();
}
```

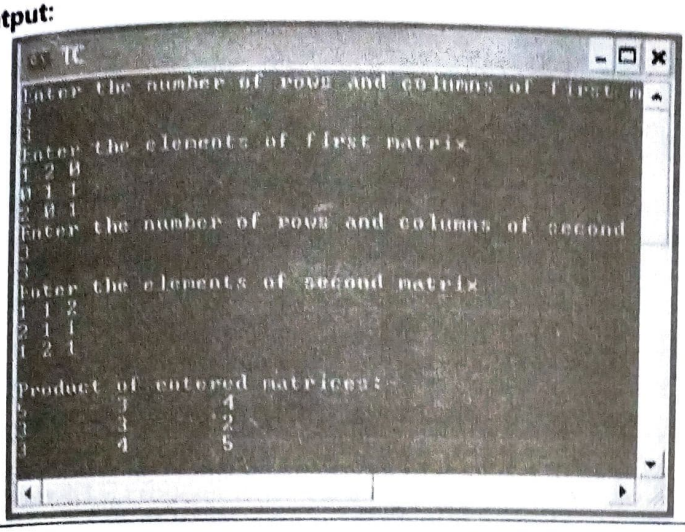
**Output:**




**Program 4.11:** Program to perform multiplication of two matrix.

```
#include<stdio.h>
int main()
{
    int m, n, p, q, c, d, k, sum = 0;
    int first[10][10], second[10][10], multiply[10][10];
    clrscr();
    printf("Enter the number of rows and columns of first
    matrix\n");
    scanf("%d%d", &m, &n);
    printf("Enter the elements of first matrix\n");
    for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
    scanf("%d", &first[c][d]);
    printf("Enter the number of rows and columns of second
    matrix\n");
    scanf("%d%d", &p, &q);
    if ( n != p )
    printf("Matrices with entered orders can't be multiplied
    with each other.\n");
    else
    {
        printf("Enter the elements of second matrix\n");
        for ( c = 0 ; c < p ; c++ )
        for ( d = 0 ; d < q ; d++ )
        scanf("%d", &second[c][d]);
        for ( c = 0 ; c < m ; c++ )
        {
        for ( d = 0 ; d < q ; d++ )
                {
                 for ( k = 0 ; k < p ; k++ )
                   {
        sum = sum + first[c][k]*second[k][d];
                   }
                 multiply[c][d] = sum;
                 sum = 0;
                }
        }
        printf("\n Product of entered matrices:-\n");
        for ( c = 0 ; c < m ; c++ )
        {
        for ( d = 0 ; d < q ; d++ )
        printf("%d\t", multiply[c][d]);
        printf("\n");
        }
    }
    return 0;
}
```

**Output:**




**Program 4.12:** Program to transpose a matrix.

In C transpose of a matrix is obtained by interchanging rows and columns of a matrix. For example, if a matrix is

```
1   2
3   4
5   6
```

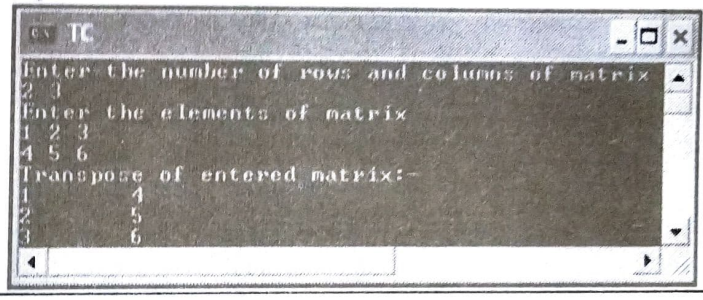
then transpose of above matrix will be,

```
1   3   5
2   4   6
```

When we transpose a matrix then the order of matrix changes, but for a square matrix order remains same.

```
#include<stdio.h>
int main()
{
    int m, n, c, d, matrix[10][10], transpose[10][10];
    clrscr();
    printf("Enter the number of rows and columns of
matrix");
    scanf("%d%d",&m,&n);
    printf("Enter the elements of matrix \n");
    for( c = 0 ; c < m ; c++ )
    {
        for( d = 0 ; d < n ; d++ )
        {
        scanf("%d",&matrix[c][d]);
        }
    }
    for( c = 0 ; c < m ; c++ )
    {
        for( d = 0 ; d < n ; d++ )
        {
        transpose[d][c] = matrix[c][d];
        }
    }
    printf("Transpose of entered matrix:-\n");
    for( c = 0 ; c < n ; c++ )
    {
        for( d = 0 ; d < m ; d++ )
        {
        printf("%d\t",transpose[c][d]);
        }
        printf("\n");
    }
    return 0;
}
```

**Output:**




**Program 4.13:** Program to merge two arrays.

```
#include<stdio.h>
void merge(int [], int, int [], int, int []);
int main()
{
    int a[100], b[100], m, n, c, sorted[200];
    clrscr();
    printf("Input number of elements in first array\n");
    scanf("%d", &m);
    printf("Input %d integers\n", m);
    for (c = 0; c < m; c++)
    {
    scanf("%d", &a[c]);
    }
    printf("Input number of elements in second array\n");
    scanf("%d", &n);
    printf("Input %d integers\n", n);
    for (c = 0; c < n; c++)
    {
    scanf("%d", &b[c]);
    }
    merge(a, m, b, n, sorted);
    printf("Sorted array:\n");
    for (c = 0; c < m + n; c++)
    {
    printf("%d\n", sorted[c]);
    }
```