Arrays

When solving problems, it is important to be able to visualize the data related to the problem. Sometimes the data consists of just a single number, such as the radius of a circle. At other times, the data may be a coordinate in a plane that can be represented as a pair of numbers, with one number representing the *x*-coordinate and the other number representing the *y*-coordinate. There are also times when we want to work with a set of similar data values, but we do not want to give each value a separate name. For example, suppose that we have to read set of 1000 students marks and store it, so that we could use it to perform several computations.

Obviously, we do not want to use 1000 different names for each student marks, so we need a method for working with a group of values using a single identifier. One solution to this problem uses a data structure called an **array**.

"*Array is a data structure that consists of a group of elements of same data type. An array is a derived data type since it contains a collection of variables belonging to the same data type. The elements of an array are stored in contiguous memory locations.*"

Array prototype:

Code:

Data_type array_name[array_size];

Here, employeeId is an array of integer type of size 10.

Array Memory Allocation:

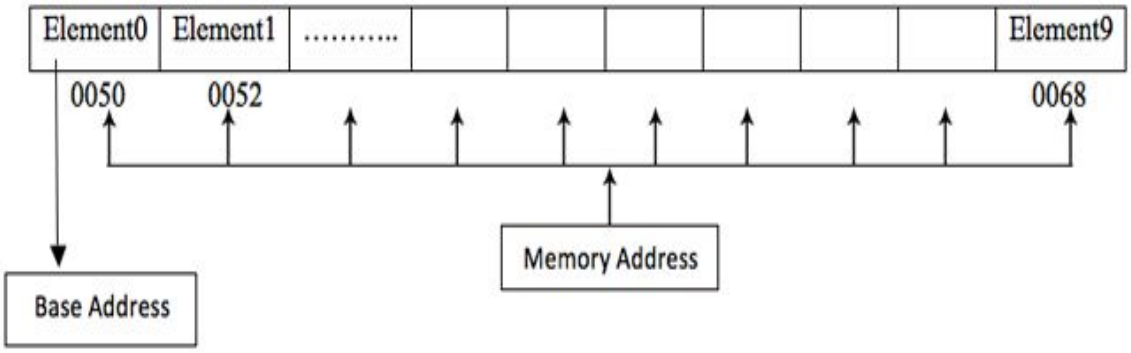
The whole array is stored in a single contiguous memory block. Let, you have an array of integer:

Code:

```
int employeeId[10];
```

If size of an integer is 4 byte, then 40(10*4) bytes will be required to store this array in memory and these

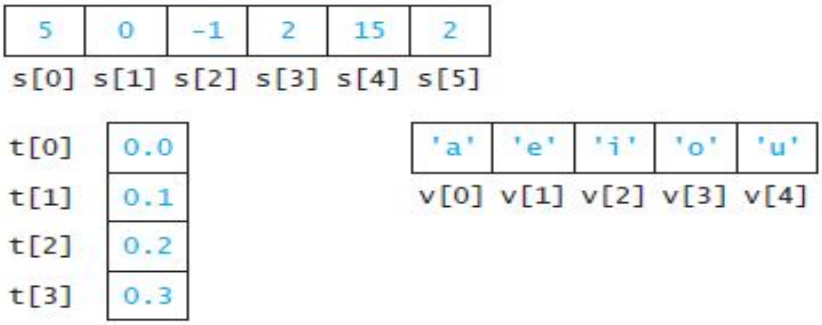
40 bytes must be contiguous.



Address of the first element is called the base address of the array. Address of ith element of the array can be achieved by following formula:

Address of i element = Address of base + size of element * i;

A one-dimensional array can be visualized as a list of values arranged in either a row or a column, as follows:



Internal View of an Array

Array variables are like pointers. When you create an array, the array variable can be used as a pointer to the start of the array in memory. When C sees a line of code in a function like this:

int A[25]={11,12,13,14……34,35};

It will associate the address of the first element of the array with the **A** variable. Every time the **A** variable is used in the code, the computer will substitute it with the address of the first character in the array.

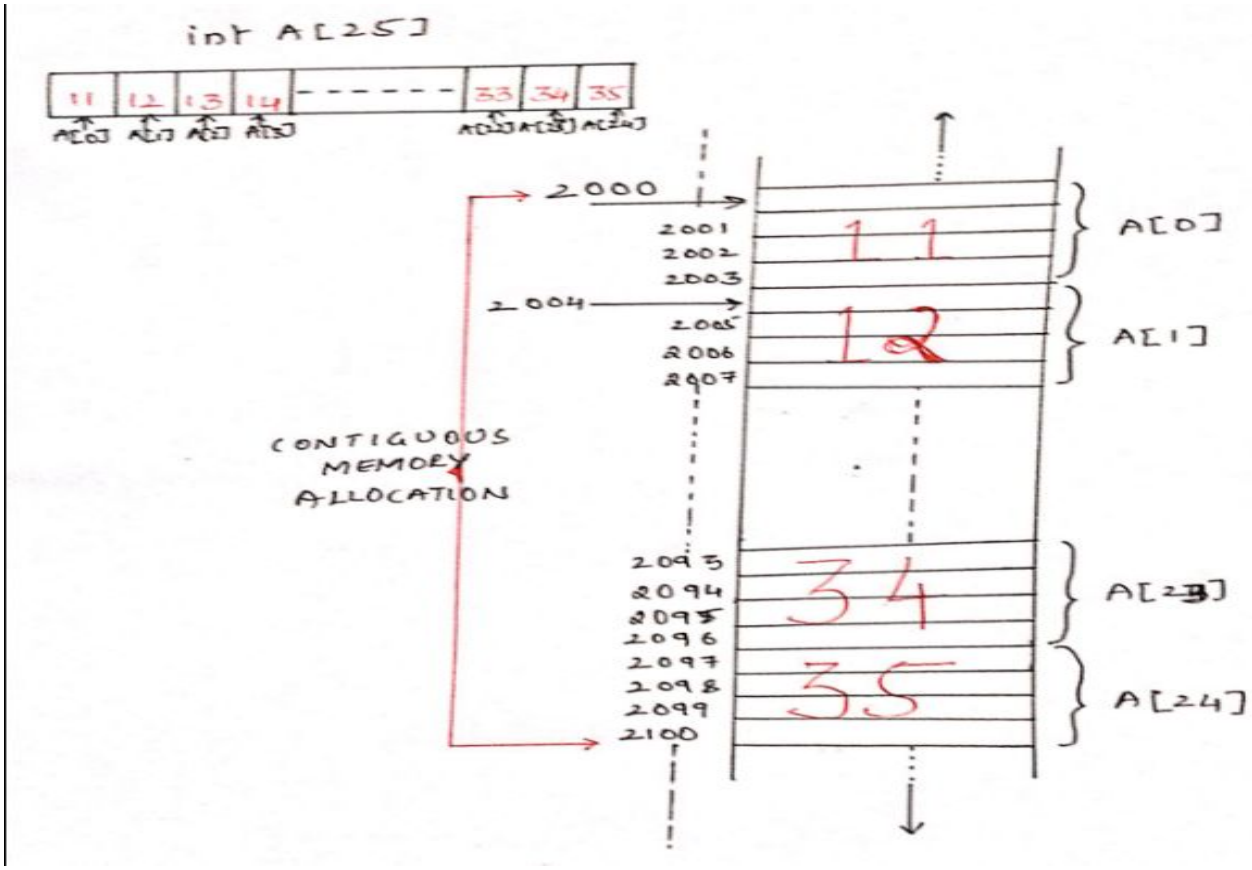
A= Base Address of the Array (the address of the first element) =**2000.**

How can we access any element from the array using Base Address?

- **A[1]=12** → → A + Index no of the item + No of bytes associated with the data Type →2000+1+4 → 2004 → Now points to 2nd element in the array A(12).

In the above example 2000 is the base address of the array provided by the name of the array (i.e. **A**), next the index number is added and finally add the size of the data type depending upon the internal architecture of the machine (i.e. in this case it 4 bytes for int data type).

Note: - The example considered above is with the assumption that the machine is Byte Addressable, which means 1byte is addressed at a time.



How do we distinguish between elements or values in the array?

- **We assign an identifier (name) to an array, and then we distinguish between elements or values in the array using *subscripts/index number*.**

In C, the subscripts always start with 0 and increment by 1. Thus, by using the above example arrays,

- the first value in the s array is referenced by s[0],
- the third value in the t array is referenced by t[2],
- And the last value in the array v is referenced by v[4].

Arrays are convenient for storing and handling large amounts of data.

Array Subscripts/index

We use a subscript to differentiate between the individual array elements and to specify which array element is to be manipulated. We can use any expression of ***type int*** as an array subscript. However, to create a valid reference, the value of this subscript must lie between 0 and one less than the declared size of the array.

EXAMPLE 1: - Understanding the distinction between an array subscript value and an array element value is essential. The original array x is as shown below. The subscripted variable x[i] references a particular element of this array. If i has the value 0, the subscript value is 0, and x[0] is referenced. The value of x[0] in this case is 16.0. If i has the value 2, the subscript value is 2, and the value of x[i] is 6.0. If i has the value 8, the subscript value is 8, and we cannot predict the value of x[i] because the subscript value is out of the allowable range.

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

EXAMPLE 2: - Table below lists some sample statements involving the array x above. The variable i is assumed to be of type int with value 5. Make sure you understand each statement.

TABLE 7.2 Code Fragment That Manipulates Array x

Statement	Explanation
`i = 5;`	
`printf("%d %.1f\n", 4, x[4]);`	Displays `4` and `2.5` (value of `x[4]`)
`printf("%d %.1f\n", i, x[i]);`	Displays `5` and `12.0` (value of `x[5]`)
`printf("%.1f\n", x[i] + 1);`	Displays `13.0` (value of `x[5]` plus 1)
`printf("%.1f\n", x[i] + i);`	Displays `17.0` (value of `x[5]` plus 5)
`printf("%.1f\n", x[i + 1]);`	Displays `14.0` (value of `x[6]`)
`printf("%.1f\n", x[i + i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f\n", x[2 * i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f\n", x[2 * i - 3]);`	Displays `-54.5` (value of `x[7]`)
`printf("%.1f\n", x[(int)x[4]]);`	Displays `6.0` (value of `x[2]`)
`printf("%.1f\n", x[i++]);`	Displays `12.0` (value of `x[5]`); then assigns `6` to `i`
`printf("%.1f \n", x[--i]);`	Assigns `5` (`6 - 1`) to `i` and then displays `12.0` (value of `x[5]`)
`x[i - 1] = x[i];`	Assigns `12.0` (value of `x[5]`) to `x[4]`
`x[i] = x[i + 1];`	Assigns `14.0` (value of `x[6]`) to `x[5]`
`x[i] - 1 = x[i];`	Illegal assignment statement

The two attempts to display element x[10], which is not in the array, may result in a run-time error, but they are more likely to print incorrect results. Consider the call to printf that uses (int)x[4] as a subscript expression. Since this expression evaluates to 2, the value of x[2] (not x[4]) is printed. If the value of (int)x[4] were outside the range 0 through 7, its use as a subscript expression would not reference a valid array element.

Definition and Initialization

An array is defined using declaration statements. The identifier is followed by an integer expression in brackets that specifies the number of elements in the array. ***Note that all elements in an array must be the same data type.***

The declaration statements for the three example arrays are as follows:

```
int s[6];
```

char v[5];

double t[4];

How can we initialize an array?

- ***An array can be initialized with declaration statements or with program statements.***

Case1:- Arrays can also be initialized with declaration statements

To initialize the array with a declaration statement, the values are specified in a sequence that is separated by commas and enclosed in braces. To define and initialize the sample arrays s, v, and t, use the following statements:

int s[6]={5,0,-1,2,15,2};

char v[5]={'a','e','i','o','u'};

double t[4]={0.0,0.1,0.2,0.3};

What if the initializing sequence is shorter than the array?

- ➔ If the initializing sequence is shorter than the array, then the rest of the values are initialized to *zero*.
- ➔ Example-If we want to define an integer array of 100 values, where each value is also initialized to zero, we would use the following statement:

  int s[100]={0};

What if an array is specified without a size, but with an initialization sequence?

- ➔ If an array is specified without a size, but with an initialization sequence, the size is defined to be equal to the number of values in the sequence, as follows:

  int s[]={5,0,-1,2,15,2};

  double t[]={0.0,0.1,0.2,0.3};

The size of an array must be specified in the declaration statement by using either a constant within brackets or by an initialization sequence within braces.

Exercise#1

Let x be the array shown in Fig. Notice that x[1] is the second array element and x[7], not x[8], is the last array element. A sequence of statements that manipulate this array is shown in Table. The contents of array x after execution of these statements are shown after Table. Only x[2] and x[3] are changed.

```
double x[8];
```

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

TABLE 7.1 Statements That Manipulate Array x

Statement	Explanation
`printf("%.1f", x[0]);`	Displays the value of `x[0]`, which is `16.0`.
`x[3] = 25.0;`	Stores the value `25.0` in `x[3]`.
`sum = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]`, which is `28.0` in the variable `sum`.
`sum += x[2];`	Adds `x[2]` to `sum`. The new `sum` is `34.0`.
`x[3] += 1.0;`	Adds `1.0` to `x[3]`. The new `x[3]` is `26.0`.
`x[2] = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]` in `x[2]`. The new `x[2]` is `28.0`.

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	28.0	26.0	2.5	12.0	14.0	-54.5

Case2:- Arrays can also be initialized with program statements.

For example, suppose that we want to fill a double array g with the values 0.0, 0.5, 1.0, 1.5…, 10.0. Because there are 21 values, listing the values on the declaration statement would be tedious. Thus, we use the following statements to define and initialize this array:

```
/*  Declare variables.  */
int k;
double g[21];
...
/*  Initialize the array g.  */
for (k=0; k<=20; k++)
   g[k] = k*0.5;
```

Note: - It is important to recognize that the condition in this for statement must specify a final subscript value of **20**, and **not 21**, since the array elements are g[0] through g[20]. It is a common mistake to

specify a subscript that is one value more than the largest valid subscript, and this error can be very difficult to find because it accesses values outside the array.

What if we try Accessing values outside of an array?

➔ Accessing values outside of an array can produce execution errors such as "***segmentation fault"***. More often, this error is not detected during the program execution, but will cause unpredictable program results, since your program has modified a memory location outside of your array. It is important to be careful about exceeding the array subscripts.

TWO DIMENSIONAL ARRAYS

A two - dimensional array is an array of one - dimensional arrays. Every element of all these one - dimensional arrays can be accessed using a single variable name.

Why two - dimensional arrays?

A one - dimensional array is linear in nature. It stores data in a single dimension. It is suitable for applications where we want to store data like marks of a particular subject of a set of students or the average temperature of a city for a week and so on. But if there arises a situation where we need to store a set of related data but the data has to be accessed not in a single dimension, but in two dimensions, then we make use of two dimensional arrays. For example, let us suppose, we want to store marks of all the students in a class in all subjects. Here, we want to store and access data in two dimensions. One dimension is that of a student. You want to know the marks of a particular student in all subjects. The other dimension is that of a subject. You want to know the marks of all students in a subject. Such data access becomes difficult using a single dimensional array and there lies the need for two dimensional arrays.

Declaration of a two dimensional array:

```
int a[10][10];
```

Definition of a two dimensional array:

```
int b[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

Internal Structure of a two dimensional array:

In a two - dimensional array also, the elements are stored in contiguous memory locations. And since elements are stored in contiguous memory locations, there are two possibilities in which it can be done.

One is Row Major Ordering where all elements of one row are stored followed by all elements of the next row and so on.

The other is Column Major Ordering where all elements of one column are stored followed by all elements of the next column and so on.

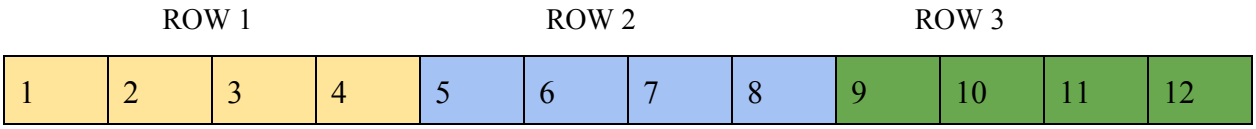
Example:

int arr[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};

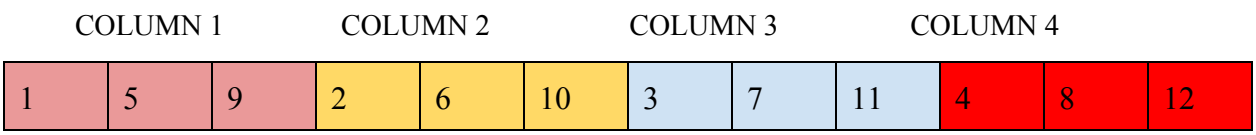
The first subscript indicates how many 1D arrays are there and the second subscript indicates how many elements are present in each of these 1D arrays.

1	2	3	4
5	6	7	8
9	10	11	12

Row Major Arrangement:



Column Major Arrangement:



Address Calculation in a two dimensional array

A[i][j] = Base_Address +((i * No. of columns in every row) + j)*size of every element;

PROGRAMS RELATED TO ARRAYS

//Program which demonstrates array declaration and initialization

```
#include<stdio.h>
void main()
{
        //Fixed Length Array Declaration
        //Syntax: data_type array_name[size];

        int arr1[20];

        //Initialization

        //Array size and number of elements are the same
        int arr2[4] = {4, 2, 21, 1};

        //Array size is greater than the number of elements
        int arr3[5] = {12, 7, 52};

        //Array size is lesser than the number of elements
        int arr4[5] = {21, 11, 8, 12, 15, 9};

        /*Size of the array can be skipped if initialization is done
        at the time of declaration*/
        int arr5[] = {1,2,3,4,5};

        int i;

        printf("\n\narr1:\n\n");
        //Displaying the values in an array
```

```
	//arr1 is uninitialized. Contains junk values.
	for(i=0; i<20; i++)
		printf("%d\n", arr1[i]);

	printf("\n\narr2:\n\n");

	/*arr2 is initialized. Array size and number of elements are same.*/
	for(i=0; i<4; i++)
		printf("%d\n", arr2[i]);



	printf("\n\narr3:\n\n");



	/*arr3 is initialized. Array size is greater than the number of elements..*/
	for(i=0; i<5; i++)
		printf("%d\n", arr3[i]);

	printf("\n\narr4:\n\n");

	/*arr4 is initialized. Array size is smaller than the number of elements.*/
	for(i=0; i<5; i++)
		printf("%d\n", arr4[i]);

	printf("\n\narr5:\n\n");

	/*arr5 is initialized.*/
	for(i=0; i<5; i++)
		printf("%d\n", arr5[i]);
}
```

OUTPUT:

arr1:

1966830442
-191260937
4199104
4199104
0
4201184
6422240
6422296
6422476
1966854640
-2118589981
-2
1966830442
1966830685
4201184
6422420
4201275
4201184
0
4124672

arr2:

4
2
21
1

arr3:

12
7
52
0
0

arr4:

21
11
8
12
15

arr5:

1
2
3
4
5

Program to demonstrate variable length array declaration and taking elements of an array as input from the user.

```
#include<stdio.h>

void main()
{
	int n;
	printf("Enter the number of elements you want to enter:\n");
	scanf("%d",&n);

	int arr[n];
	int i;

	//Taking elements of the array as input from the user
	printf("\nEnter the elements of the array:\n\n");
	for(i=0; i<n; i++)
		scanf("%d",&arr[i]);

	//Displaying the elements of the array
	printf("\nThe elements of the array are:\n");
	for(i=0; i<n; i++)
		printf("%d\n",arr[i]);
}
```

OUTPUT:

```
Enter the number of elements you want to enter:
5

Enter the elements of the array:
```

15
26
2
8
20

The elements of the array are:
15
26
2
8
20

Program which finds the sum of the elements of an array

```
#include<stdio.h>

void main()
{
        int n;
        printf("Enter the number of elements in the array:\n");
        scanf("%d",&n);

        int i, sum = 0;
        int arr[n];
        printf("\nEnter the elements of the array:\n");
        for(i=0; i<n; i++)
                scanf("%d",&arr[i]);

        for(i=0; i<n; i++)
```

```
            sum = sum + arr[i];

    printf("\n\nThe sum of the elements of the array is %d.\n",sum);

}
```

OUTPUT:

```
Enter the number of elements in the array:
5
Enter the elements of the array:
5
10
15
20
25
The sum of the elements of the array is 75!
```

Program which determines the number of elements in an array divisible by 11

```
#include<stdio.h>
void main()
{
    int n, i;
    int count = 0;
    printf("Enter the number of elements you want to store:\n");
    scanf("%d",&n);
    int a[n];
    printf("\nEnter the elements of the array:\n");
    for(i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
```

```
            if(a[i] % 11 == 0)
                count++;
        }
        printf("The number of elements divisible by 11 is %d!\n",count);
}
```

OUTPUT:

```
Enter the number of elements you want to store:
5
Enter the elements of the array:
1
11
121
12121
1212121
The number of elements divisible by 11 is 2!
```

Program which determines the sum of elements in even indices and sum of elements in odd indices

```
#include<stdio.h>
void main()
{
        int n, i;
        int odd = 0;
        int even = 0;
        printf("Enter the number of elements you want to store:\n");
        scanf("%d",&n);
        int a[n];
```

```
        printf("Enter the elements of the array:\n");
        for(i=0; i<n; i++)
        {
                scanf("%d",&a[i]);
                if(i % 2 == 0)
                        even = even + a[i];
                else
                        odd = odd + a[i];
        }
        printf("The sum of elements in even positions is %d!\n",even);
        printf("The sum of elements in odd positions is %d!\n",odd);
}
```

OUTPUT:

```
Enter the number of elements you want to store:
5
Enter the elements of the array:
2
4
6
8
10
The sum of elements in even positions is 18!
The sum of elements in odd positions is 10!
```

Program to demonstrate inserting an element at a specified index, moving the rest of the elements one position to their right

```
#include<stdio.h>
void main()
{
	int a[6] = {1, 2, 3, 4, 5, 6};
	int ele,index,i,temp1,temp2;
	printf("Enter the element to be inserted:\n");
	scanf("%d",&ele);
	printf("Enter the index at which to insert the element\n");
	scanf("%d",&index);

	//Moving the elements one position to their right
	temp1=a[index];
	for(i=index+1; i<6; i++)
	{
		temp2 = temp1;
		temp1 = a[i];
		a[i] = temp2;
	}

	//Assigning the user entered element at the user entered index
	a[index] = ele;

	//Displaying the elements of the array
	printf("\n\n");
	for(i=0; i<6; i++)
		printf("%d\n",a[i]);
}
```

OUTPUT:

```
Enter the number of elements:
5
Enter the elements:
3
5
7
11
13
Enter the element to be inserted:
17
Enter the element in the array after which insertion has to take place:
5

3
5
17
7
11
```

Program to move all elements after a user specified element one position to the left and insert 0 at the last position

```
#include<stdio.h>
void main()
{
        int n;
        int ele;
        int i, j;
        printf("Enter the number of elements:\n");
        scanf("%d",&n);
```

```
        int a[n];
        printf("\nEnter the elements:\n");
        for(i=0; i<n; i++)
                scanf("%d",&a[i]);
        printf("Enter the element to delete:\n");
        scanf("%d",&ele);
        i = 0;
        while(a[i] != ele)
        {
                i++;
        }
        for(j=i+1; j<n; j++)
        {
                a[j-1] = a[j];
        }
        a[j-1]=0;
        printf("\n");
        for(i=0; i<n; i++)
                printf("%d\n",a[i]);
}
```

OUTPUT:

```
Enter the number of elements:
5

Enter the elements:
12
24
48
36
42
```

```
Enter the element to delete:
48

12
24
48
36
0
```

Program to demonstrate matrix addition using two dimensional arrays

```
#include<stdio.h>

void main()
{
 int arr1[10][10];
 int arr2[10][10];
 int arr3[10][10];
 int i, j, r, c;

 printf("Enter the number of rows:\n");
 scanf("%d",&r);
 printf("Enter the number of columns:\n");
 scanf("%d",&c);

 printf("Enter elements of the first matrix:\n");
 for(i=0; i<r; i++)
 {
  for(j=0; j<c; j++)
```

```
 {
  scanf("%d",&arr1[i][j]);
 }
}

printf("Enter elements of the second matrix:\n");
for(i=0; i<r; i++)
{
 for(j=0; j<c; j++)
 {
  scanf("%d",&arr2[i][j]);
 }
}

for(i=0; i<r; i++)
{
 for(j=0; j<c; j++)
 {
  arr3[i][j] = arr1[i][j] + arr2[i][j];
 }
}

printf("The resultant matrix is:\n");
for(i=0; i<r; i++)
{
 for(j=0; j<c; j++)
 {
  printf("%d\t",arr3[i][j]);
 }
```

```
  printf("\n");
 }
}
```

OUTPUT:

Enter the number of rows:

3

Enter the number of columns:

2

Enter elements of the first matrix:

2

4

6

8

6

8

Enter elements of the second matrix:

1

2

3

1

1

3

The resultant matrix is:

3 6

9 9

7 11

Program to demonstrate matrix subtraction using two dimensional arrays

```
#include<stdio.h>

void main()
{
 int arr1[10][10];
 int arr2[10][10];
 int arr3[10][10];
 int i, j, r, c;

 printf("Enter the number of rows:\n");
 scanf("%d",&r);
 printf("Enter the number of columns:\n");
 scanf("%d",&c);

 printf("Enter elements of the first matrix:\n");
 for(i=0; i<r; i++)
 {
  for(j=0; j<c; j++)
  {
   scanf("%d",&arr1[i][j]);
  }
 }

 printf("Enter elements of the second matrix:\n");
 for(i=0; i<r; i++)
 {
  for(j=0; j<c; j++)
```

```
  {
   scanf("%d",&arr2[i][j]);
  }
 }

 for(i=0; i<r; i++)
 {
  for(j=0; j<c; j++)
  {
   arr3[i][j] = arr1[i][j] - arr2[i][j];
  }
 }

 printf("The resultant matrix is:\n");
 for(i=0; i<r; i++)
 {
  for(j=0; j<c; j++)
  {
   printf("%d\t",arr3[i][j]);
  }
  printf("\n");
 }
}
```

OUTPUT:

Enter the number of rows:

3

Enter the number of columns:

2

Enter elements of the first matrix:

2

4

6

8

6

8

Enter elements of the second matrix:

1

2

3

1

1

3

The resultant matrix is:

1 2

3 7

5 5