

## UNIT-III

# ARRAYS, STRINGS

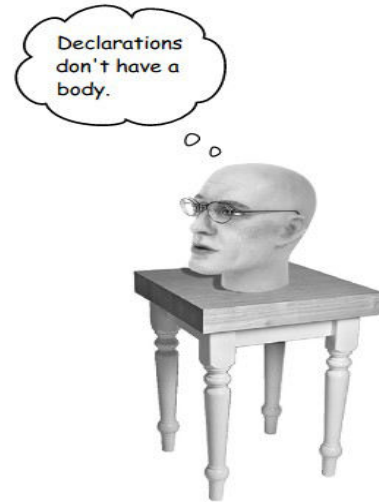
### INDEX

#### Arrays:

Array Concepts,  
Using Arrays in C,  
Array Applications,  
Two- Dimensional Arrays,  
Multidimensional Arrays;

#### Strings:

Declaring, Initializing, Printing and reading strings,  
String input and output functions,  
string manipulation functions,  
array of strings



# ARRAYS

## Definition of an array:

An array is a kind of variable that refers to a collection of data items which all have same name and same data type. The elements of an array can be any data type. All the elements in an array must be of same type. There are two types of arrays.

1. One dimensional array
2. Multidimensional array

The general form of declaring one dimensional array is

```
datatype array_name[size];
```

where array\_name is the name of the array, size is the number of elements that array contains.

The general form of declaring multidimensional array is

```
datatype array_name[size1][size2]....[size N]
```

where array\_name is the name of the array

size1,size2....sizeN refers to the minimum size of the each array locations.

## Merits of Arrays in C:

1. We can easily access each element of array
2. Array elements are stored in continuous memory locations.
3. It is used to represent multiple data items of same type by using only single name
4. It can be used to implement other data structures like linked lists, stacks, queues, trees, graphs etc.,
5. 2D arrays are used to represent matrices.

## Demerits of Arrays in C:

1. We must know in advance that how many elements are to be stored in array.
2. Array is static structure. It means that array is of fixed size. The memory which is allocated to array can not be increased or reduced.
3. Since array is of fixed size, if we allocate more memory than requirement then the memory space will be wasted. And if we allocate less memory than requirement, then it will create problem.
4. The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.
5. Wastage of memory space. We cannot change size of array at the run time.
6. It can store only type of data.
7. There is a certain chance of memory wastage/shortage.

## ONE DIMENSIONAL ARRAYS:

**Definition:** A list of items can be given one variable name using only one subscript and such a variable is called a single – subscripted variable or a one dimensional array.

### 1. Declaration of one-dimensional arrays:

Like any other variable, arrays must be declared before they are used so that the compiler can allocate space for them in memory. The general form of array declaration is

**Type variable\_name[size];**

The type specifies the type of element that will be contained in the array, such as **int, float or char** and the size indicates the maximum number of elements that can be stored inside the array. For example,

**int a[5];**

<b>a[0]</b>	<b>a[1]</b>	<b>a[2]</b>	<b>a[3]</b>	<b>a[4]</b>

## **2. Initialization of one-dimensional arrays:**

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared. The general form of initialization of arrays is

**Type array\_name[size]={list of values};**

The values in the list are separated by commas. For example, the statement

1) **int marks[5]={90,45,67,85,78}** will declare the variable marks as an array of size 5 and will initialize the elements 90,45,67,85 and 78

90	45	67	85	78
<b>[0]</b>	<b>[1]</b>	<b>[2]</b>	<b>[3]</b>	<b>[4]</b>

2) If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically. For example the statement

**int marks[5]={90,45};**

90	45	0	0	0
<b>[0]</b>	<b>[1]</b>	<b>[2]</b>	<b>[3]</b>	<b>[4]</b>

3) The size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. for example the statement.

**int counter[]={10,20,30,40};**

will declare the counter array to contain four elements with initial values 10,20,30 and 40

10	20	30	40
<b>[0]</b>	<b>[1]</b>	<b>[2]</b>	<b>[3]</b>

## **3. Accessing elements of the one-dimensional array:**

For accessing an individual element of the array, the array subscript must be used. For example, to access the fourth element of the array, we must write `arr[3]`. The subscript index must be index value.

**Example of 1D array:**

```
//program to read and display n numbers using an array
#include<stdio.h>
void main()
{
    int i,n,arr[20];
    printf("Enter the number of elements:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("The array elements are:\n");
    for(i=0;i<n;i++)
    {
        printf("arr[%d]=%d\n",i,arr[i]);
    }
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc arr.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter the number of elements:5
1
2
3
4
5
The array elements are:
arr[0]=1
arr[1]=2
arr[2]=3
arr[3]=4
arr[4]=5
```

## **TWO DIMENSIONAL ARRAYS:**

The concept of one-dimensional array is extended to incorporate two dimensional (2D) arrays. A 2D array is specified using two subscripts where one subscript denotes row and the other denotes column. C considers the 2D array as an array which can be viewed as an array of arrays.

### **1. Declaration of Two-dimensional arrays:**

Similar to one-dimensional arrays, two-dimensional arrays 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];
```

For example, if we want to store the marks obtained by 3 students in 3 different subjects, then we can declare a two-dimensional array as

```
int marks[3][3];
```

A two-dimensional array called marks is declared that has 3 rows and 3 columns.

### **2. Initializing Two-dimensional Arrays:**

i) Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. For example,

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

initializes the elements of the first row to zero and the second row to one.

ii) The initialization is done row by row. The above statement can be equivalently written as

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

by surrounding the elements of the each row by braces.

iii) We can also initialize a two-dimensional array in the form of a matrix as shown below:

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

iv) When the array is completely initialized with all values, explicitly, we need not specify the size of the first dimension. That is, the statement

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

is permitted.

v) When all the elements are to be initialized to zero, the following short-cut method may be used.

```
int m[3][5]={{0},{0},{0}};
```

The first element of each row is explicitly initialized to zero while other elements are automatically initialized to zero.

### 3. Accessing the elements:

2D array contains two subscripts, we will use **two** for loops to access the elements. The first for loop will loop for each row in the 2D array and the second for loop will access individual columns for every row in the array.

**Example for 2D array:**

```
//A Program to print the elements of a 2D array.
#include<stdio.h>
void main()
{
    int arr[2][2]={12,34,56,32};
    int i,j;
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            printf("%d\t",arr[i][j]);
        }
        printf("\n");
    }
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc array.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
12      34
56      32
```

## TWO DIMENSIONAL ARRAY:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Row 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Row 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]

## ACCEPTABLE STYLES OF DECLARING 2-D ARRAYS:

### 1) **int a[4][5];**

This declares a 2-D array of 4 rows and 5 columns in each column.


### 2) **int a[4][5]={{1,7,1},{7,2,1},{3,1,3}};**

This is also an acceptable style of declaration. Here, a matrix of 4x5 is declared and values for each row is specified. That is 1,7,1 is stored in first row; 7,2,1 is stored in second row; and 3,1,1 is stored in third row

1	7	1		
7	2	1		
3	1	1		

### 3) **int a[ ][5]={{1,7,1},{7,2,1},{3,1,3}};**

This style of 2-D array declaration is also acceptable. Here, we will not be specifying number of rows. From the values(right hand side), number of rows are calculated. In this case it is 3. Thus, a 3 x 5 matrix is allocated and in it 1,7,1 is stored in first row, 7,2,1 is stored in second row; and 3,1,1 is stored in third row.

1	7	1		
7	2	1		
3	1	1		

### 4) **int a[4][5]={1,2,2,2,2,1,1,1};**

This declares a 2-D array of 4 rows and 5 columns . In each row the data will be stored as follows.

1	2	2	2	2
2	1	1	1	

5) **int a[][5]={1,2,2,2,2,1,1,1};**

This declares a 2-D array of 2 rows and 5-columns. In each row the data will be stored as follows.

1	2	2	2	2
2	1	1	1	



## MULTIDIMENSIONAL ARRAYS IN C

- ◆ A multidimensional array in simple terms is an **array of arrays**.
- ◆ Like we have **one index** in a 1D array, **two indices** in a 2D array, in the same way we have **n indices** in an n-dimensional array or multi dimensional array.
- ◆ Conversely, an n-dimensional collection  **$m_1 \times m_2 \times m_3 \times \dots m_n$**  array is a collection  **$m_1 * m_2 * m_3 * \dots m_n$**  elements.
- ◆ In a multidimensional array, a particular element is specified by using n subscripts as  **$A[I_1][I_2][I_3] \dots [I_n]$**
  
- ◆ A multidimensional array can contain as many indices as needed and the requirement of the memory increases with the number of indices used.
- ◆ Figure below shows a three dimensional(3D) array. The array has three pages, four rows, and two columns.

### ACCEPTABLE STYLES OF DECLARING AND INITIALIZING 3-D ARRAYS:

1) **int a[2][4][5];** This statement declares a 3-D array with 2 planes of 4x5 size.

Plane 0:


Plane 1:


2) **int a[2][4][5]={{1,2},{1,4}},{3,4},{4,5}};** This statement declares a 3-D array with 2 planes of 4x5 size. Also the given data is stored as shown below

Plane 0:

1	2			
1	4			

Plane 1:

3	4			
4	5			

3) **int a[][4][5]={{1,2},{1,4}},{3,4},{4,5}};** This statement declares a 3-D array with 2 planes of 4x5 size. Also, the given data is stored as shown below

Plane 0:

1	2			
1	4			

Plane 1:

3	4			
4	5			

4) **int a[2][4][5]={1,2,3,4,4,4,4,4,4,4,4,4,5,5,5,5,5,6,6,6,6,9,4,5};** The following style is also acceptable style of declaring 3-D array. The data will be stored in the array as shown below.

Plane 0:

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

Plane 1:

6	6	6	6	9
4	5			

5) **int a[ ][4][5]={1,2,3,4,4,4,4,4,4,4,4,5,5,5,5,5,6,6,6,6,9,4,5};** The following style is also acceptable style of declaring 3-D array. The data will be stored in the array as shown below. Based on the data given, it understands two planes are needed.

Plane 0:

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

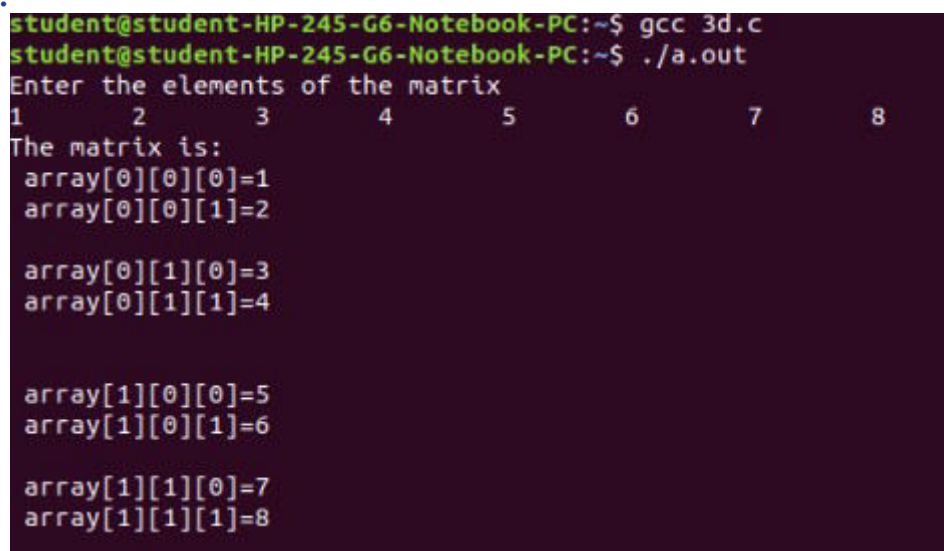
Plane 1:

6	6	6	6	9
4	5			

### PROGRAM USING 3D ARRAY:

```
//A program to read and display a 2x2x2 array
#include<stdio.h>
void main()
{
    int array[3][3][3],i,j,k;
    printf("Enter the elements of the matrix\n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            for(k=0;k<2;k++)
            {
                scanf("%d",&array[i][j][k]);
            }
        }
    }
    printf("The matrix is:");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            for(k=0;k<2;k++)
            {
                printf("\n array[%d][%d][%d]=%d",i,j,k,array[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
}
```

### OUTPUT:



```
student@student-HP-245-G6-Notebook-PC:~$ gcc 3d.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter the elements of the matrix
1      2      3      4      5      6      7      8
The matrix is:
array[0][0][0]=1
array[0][0][1]=2

array[0][1][0]=3
array[0][1][1]=4

array[1][0][0]=5
array[1][0][1]=6

array[1][1][0]=7
array[1][1][1]=8
```

## VARIOUS OPERATIONS ON ARRAYS:

There are a number of operations that can be performed on arrays. These operations include:

- 1) Traversing an array
- 2) Inserting an element in an array
- 3) Searching an element in an array
- 4) Deleting an element from an array
- 5) Merging two arrays
- 6) Sorting an array in ascending or descending order

### **Traversing an Array**

Traversing an array means accessing each and every element of the array for a specific purpose.

### Algorithm to insert an element at a particular index in an array:

The algorithm INSERT will be declared as INSERT (A, N, POS, VAL) . The arguments are

- (a) A , the array in which the element has to be inserted
- (b) N , the number of elements in the array
- (c) POS , the position at which the element has to be inserted
- (d) VAL , the value that has to be inserted

In the algorithm,

```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:     SET A[I + 1] = A[I]
Step 4:     SET I = I - 1
           [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

In **Step 1**, we first initialize I with the total number of elements in the array.

In **Step 2**, a while loop is executed which will move all the elements having an index greater than POS one position towards right to create space for the new element.

In **Step 5**, we increment the total number of elements in the array by 1

In **Step 6**, the new value is inserted at the desired position.

Now, let us visualize this algorithm by taking an example.

Initial Data[ ] is given as below.

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

The process of insertion in an array is as follows:

45	23	34	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	100	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

### **Program to insert an element at a particular index in an array:**

```
//PROGRAM TO INSERT AN ELEMENT IN AN ARRAY
#include<stdio.h>
void main()
{
    int i,n,num,pos,arr[10];

    printf("no. of elements in the array:");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("\narr[%d]=",i);
        scanf("%d",&arr[i]);
    }

    printf("\n enter the number to be inserted");
    scanf("%d",&num);

    printf("\n enter the position at which number has to be added");
    scanf("%d",&pos);

    for(i=n;i>=pos;i--)
    {
        arr[i+1]=arr[i];
    }
    arr[pos]=num;
    n=n+1;
    printf("the array after insertion of %d:",num);
    for(i=0;i<n;i++)
    {
        printf("\n arr[%d]=%d",i,arr[i]);
    }
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc insert.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
no. of elements in the array:5

arr[0]=10

arr[1]=11

arr[2]=13

arr[3]=14

arr[4]=15

    enter the number to be inserted12

    enter the position at which number has to be added2
the array after insertion of 12:
arr[0]=10
arr[1]=11
arr[2]=12
arr[3]=13
arr[4]=14
arr[5]=15
```

**Algorithm to delete an element from an array:**

The algorithm DELETE will be declared as DELETE(A, N, POS). The arguments are:

- (a) A, The array from which the element has to be deleted.
- (b) N , the number of elements in the array.
- (c) POS , the position from which the element has to be deleted.

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:     SET A[I] = A[I + 1]
Step 4:     SET I = I + 1
           [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

The above figure shows the algorithm in which

In **Step 1**, we first initialize I with the position from which the element has to be deleted.

In **Step 2**, a while loop is executed which will move all the elements having an index greater than POS one space towards left to occupy the space vacated by the deleted element. When we say that we are deleting an element, actually we are overwriting the element with the value of its successive element.

In **Step 5**, we decrement the total number of elements in the array by 1.

Now, let us visualize this algorithm by taking an example given below.

Calling DELETE (Data, 6, 2) will lead to the following processing in the array.

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]

**Program:**



```

//PROGRAM TO DELETE AN ELEMENT IN AN ARRAY
#include<stdio.h>
void main()
{
    int i,n,pos,arr[10];

    printf("enter number of elements");
    scanf("%d",&n);

    printf("enter the elements of an array");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }

    printf("Enter the position from which number has to be deleted");
    scanf("%d",&pos);

    for(i=pos;i<n;i++)
    {
        arr[i]=arr[i+1];
    }
    n=n-1;

    printf("The array after deletion:");
    for(i=0;i<n;i++)
    {
        printf("\n arr[%d]=%d",i,arr[i]);
    }
}

```

**Output:**

```

student@student-HP-245-G6-Notebook-PC:~$ gcc delete.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter number of elements5
enter the elements of an array10
90
20
30
40
Enter the position from which number has to be deleted1
The array after deletion:
arr[0]=10
arr[1]=20
arr[2]=30
arr[3]=40

```

**Write a C program to search an element in an array using linear search?**

```
//PROGRAM TO SEARCH AN ELEMENT IN AN ARRAY USING LINEAR SEARCH
#include<stdio.h>
void main()
{
    int i,n,num,pos=-1,arr[10],found=0;

    printf("enter the no. of elements");
    scanf("%d",&n);

    printf("enter the elements of the array");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }

    printf(" enter number that has to be searched:");
    scanf("%d",&num);

    for(i=0;i<n;i++)
    {
        if(arr[i]==num)
        {
            found=1;
            pos=i;
            printf("%d is found in the array at position %d",num,pos);
            break;
        }
    }
    if(found==0)
        printf("\n the desired element is not found in the array");
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc linearsearch.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter the no. of elements5
enter the elements of the array
10
20
30
40
50
enter number that has to be searched:20
20 is found in the array at position 1
```

## ARRAY PROGRAMMING EXAMPLES

1) Write a C program to print sum of two matrices?

Program:

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

### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc matrix.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter the number of rows in first matrix2
enter the number of cols in first matrix2
enter the number of rows in second matrix2
enter the number of cols in second matrix2

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
```

## 2. Write a C program for multiplication of two matrices?

### Program:

```
//multiplication of two matrices
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i,j,k;
    int mat1[5][5],mat2[5][5],res[5][5];
    int rows1,cols1,rows2,cols2,res_rows,res_cols;
    printf("enter the number of rows in first matrix");
    scanf("%d",&rows1);
    printf("enter the number of cols in first matrix");
    scanf("%d",&cols1);
    printf("enter the number of rows in second matrix");
    scanf("%d",&rows2);
    printf("enter the number of cols in 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");
        exit(0);
    }
    res_rows=rows1;
    res_cols=cols2;
    printf("Enter the elements of the first matrix\n");
    for(i=0;i<rows1;i++)
    {
        for(j=0;j<cols1;j++)
            scanf("%d",&mat1[i][j]);
    }
    printf("Enter the elements of the second matrix\n");
    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]=res[i][j]+mat1[i][k]*mat2[k][j];
        }
    }

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



## Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc product.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter the number of rows in first matrix2
enter the number of cols in first matrix2
enter the number of rows in second matrix2
enter the number of cols in second matrix2
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
```

### 3. Write a C program for matrix addition using functions?

#### Program:

```
//matrix addition using functions
#include<stdio.h>
void read_arr(int a[10][10],int row,int col);
void add_arr(int m1[10][10],int m2[10][10],int m3[10][10],int row, int col);
void print_arr(int m[10][10],int row,int col);
void main()
{
    int m1[10][10],m2[10][10],m3[10][10],row,col;

    printf("enter no. of rows:");
    scanf("%d",&row);

    printf("enter no. of columns");
    scanf("%d",&col);

    read_arr(m1,row,col);
    read_arr(m2,row,col);

    add_arr(m1,m2,m3,row,col);
    print_arr(m3,row,col);
}

void read_arr(int a[10][10],int row,int col)
{
    int i,j;
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            printf("enter element %d%d:",i,j);
            scanf("%d",&a[i][j]);
        }
    }
}

void add_arr(int m1[10][10],int m2[10][10],int m3[10][10],int row,int col)
{
    int i,j;
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            m3[i][j]=m1[i][j]+m2[i][j];
        }
    }
}
```

```
void print_arr(int m[10][10],int row,int col)
{
    int i,j;
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            printf("\t%d",m[i][j]);
        }
        printf("\n");
    }
}
```

### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc matrixaddition.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter no. of rows:2
enter no. of columns2
enter element 00:1
enter element 01:2
enter element 10:3
enter element 11:4
enter element 00:1
enter element 01:2
enter element 10:3
enter element 11:4
      2      4
      6      8
```



#### 4. Write a C program to do multiplication of two matrices using functions?

##### Program:

```
//matrix multiplication using functions
#include<stdio.h>
#include<stdlib.h>
void read_data(int a[10][10],int b[10][10],int r1,int c1,int r2,int c2);
void multiplication(int a[10][10],int b[10][10],int mult[10][10],int r1,int c1,int r2,int c2);
void display(int mult[10][10],int r1,int c2);
void main()
{
    int a[10][10],b[10][10],mult[10][10],r1,c1,r2,c2,i,j,k;
    printf("Enter rows and columns for first matrix:");
    scanf("%d%d",&r1,&c1);
    printf("Enter rows and column for second matrix:");
    scanf("%d%d",&r2,&c2);
    /*If column of first matrix in not equal to row of second matrix*/
    if(c1!=r2)
    {
        printf("Error! column of first matrix not equal to row of second\n");
        exit(0);
    }
    // Function to take matrices data
    read_data(a,b,r1,c1,r2,c2);
    // Function to multiply two matrices.
    multiplication(a,b,mult,r1,c1,r2,c2);
    // Function to display resultant matrix after multiplication.
    display(mult,r1,c2);
}

void read_data(int a[10][10],int b[10][10],int r1,int c1,int r2,int c2)
{
    int i,j;
    printf("Enter elements of matrix1:\n");
    for(i=0;i<r1;i++)
    {
        for(j=0;j<c1;j++)
        {
            printf("Enter elementsa%d%d:",i+1,j+1);
            scanf("%d",&a[i][j]);
        }
    }
    printf("Enter elements of matrix2:\n");
    for(i=0;i<r2;i++)
    {
        for(j=0;j<c2;j++)
        {
            printf("Enter elementsa%d%d:",i+1,j+1);
            scanf("%d",&b[i][j]);
        }
    }
}

void multiplication(int a[10][10],int b[10][10],int mult[10][10],int r1,int c1,int r2,int c2)
{
    for(int i=0;i<r1;i++)
    {
        for(int j=0;j<c2;j++)
        {
            // Initializing elements of matrix mult to 0.
            mult[i][j]=0;
            // Multiplying matrix firstMatrix and secondMatrix and storing in array mult.
            for(int k=0;k<r2;k++)
            {
                mult[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
}
```

```

void display(int mult[10][10],int r1,int c2)
{
    printf("Output matrix:\n");
    for(int i=0;i<r1;i++)
    {
        for(int j=0;j<c2;j++)
        {
            printf("%d\t",mult[i][j]);
        }
        printf("\n");
    }
}

```

### Output:

```

student@student-HP-245-G6-Notebook-PC:~$ gcc matrixmultiplication.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter rows and columns for first matrix:2
2
Enter rows and column for second matrix:2
2
Enter elements of matrix1:
Enter elementsa11:1
Enter elementsa12:2
Enter elementsa21:3
Enter elementsa22:4
Enter elements of matrix2:
Enter elementsa11:5
Enter elementsa12:6
Enter elementsa21:7
Enter elementsa22:8
Output matrix:
19    22
43    50

```

## 5. Write a C program to print transpose of matrix?

### Program:

```
//transpose of a matrix
#include<stdio.h>
void main()
{
    int i,j,mat[3][3],tmat[3][3];
    printf("enter the elements of the matrix");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d",&mat[i][j]);
        }
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\t mat[%d][%d]=%d",i,j,mat[i][j]);
        }
        printf("\n");
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            tmat[i][j]=mat[j][i];
        }
    }
    printf("\n The elements of the transposed matrix are:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\t\t%d",tmat[i][j]);
        }
        printf("\n");
    }
}
```

### Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc transpose.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter the elements of the matrix1 2 3 4 5 6 7 8 9
    mat[0][0]=1    mat[0][1]=2    mat[0][2]=3
    mat[1][0]=4    mat[1][1]=5    mat[1][2]=6
    mat[2][0]=7    mat[2][1]=8    mat[2][2]=9

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

# STRINGS

A string is a sequence of characters that is treated as a single data item. In the C language, a string is nothing but a null-terminated character array. This means that after the last character, a null-terminated character array. This means that after the last character, a **null character('0')** is stored to signify the end of the character array. For example, if we write,

```
char str[] = "HELLO";
```

H	E	L	L	O	\0
---	---	---	---	---	----

Beginning of string

End of string

**Declaring string variables:** C does not support strings as a data type. However, it allows us to represent strings as character arrays. In C, therefore, a string variable is any valid C variable and is always declared as an array of characters. The general form of declaration of a string variable is:

```
char string_name[size];
```

The size determines the number of characters in the string\_name. Some examples are:

```
char city[10];  
char name[30];
```

When the compiler assigns a character string to a character array, it automatically supplies a *null* character ('\0') at the end of the string. Therefore, the size should be equal to the maximum number of characters in the string *plus* one.

## Initializing string variables:

C also permits us to initialize a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements initialized. For example, the statement

```
char string[] = {'G', 'O', 'O', 'D', '\0'};
```

defines the array **string** as a five element array.

We can also declare the much larger than the string size in the initializer. That is, the statement

```
char str[10] = "GOOD";
```

is permitted. In this case, the computer creates a character array of size 10, places the value "GOOD" in it, terminates with the null character, elements to NULL. The storage will look like:

G	O	O	D	\0	\0	\0	\0	\0	\0
---	---	---	---	----	----	----	----	----	----

is not allowed.

### **String Taxonomy:**

In C, we can store a string either in fixed length format or in variable format.

**1. Fixed length string:** When storing a string in a fixed length format, we need to specify an appropriate size for the string variable. If the size is too small, then we will not be able to store all the elements in the string. On the other hand, if the string size is large, then unnecessarily memory space will be wasted.

**2. Variable length string:** A better option is to use a variable length format in which the string can be expanded or contracted to accommodate the elements in it. For example, if we declare a string variable to store the name of a student. If a student has a long name of say 20 characters, then the string can be expanded to accommodate 20 characters. On the other hand, a student name has only 5 characters, then the string variable can be contracted to store only 5 characters.

Variable length technique indicate the end of elements by using length controlled string or a delimiter.

**i) Length-controlled string:** In a length-controlled string, we need to specify the number of characters in the string. This count is used by string manipulation functions to determine the actual length of the string variable.

**ii) Delimited string:** In this format, the string is ended with a delimiter. The delimiter is then used to identify the end of the string. In C we can use any character such as comma, semicolon, colon, dash, null character, etc, as the delimiter of a string. However, the null character is the most commonly used string delimiter in the C language.

### **Different functions used to perform string input operation:**

Strings can be read from the user by using 3 ways.

1. Using scanf( ) function
2. Using gets( ) function
3. Using getchar( ), getch( ) or getche( ) functions

#### **1. Using scanf() function:**

The familiar input function **scanf** can be used with %s format specification to read in a string of characters. Example:

```
char address[10];  
scanf("%s",address);
```

The problem with the scanf function is that it terminates its input on the first white space it finds. A white space includes blanks, tabs, carriage returns, form feeds, and new lines. Therefore, if the following line of text is typed in at the terminal.

**NEW YORK**

then only the string "NEW" will be read into the array **address**, since the blank space after the word "NEW" will terminate the reading of string.

**Example:**

```
#include<stdio.h>
void main( )
{
    char str[10];
    printf("Enter any string:\n");
    scanf("%s",str);
    printf("Entered string=%s\n",str);
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc scanf.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter any string:
NEW YORK
Entered string=NEW
```

### **Reading a Line of Text:**

C supports a format specifications known as the *edit set conversion code* %[..] that can be used to read a line containing a variety of characters, including whitespaces.

**Example:**

```
#include<stdio.h>
void main( )
{
    char line[80];
    scanf("%[^\n]",line);
    printf("%s\n",line);
}
```

will read a line of input from the keyboard and display the same on the screen.

```
student@student-HP-245-G6-Notebook-PC:~$ gcc scanfline.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Once upon a time, there was a cow in krishna house
Once upon a time, there was a cow in krishna house
```

## **2. Using gets( ) function:**

Reading a string can be done by using gets( ) function. The string can be read by writing

```
gets(str);
```

gets( ) is a simple function that overcomes the drawbacks of scanf(). The gets( ) function takes the starting address of the string which will hold the input.

**Example:**

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str[20];
    printf("enter any string:\n");
    gets(str);
    printf("enter string=%s\n",str);
}
```



### 3. Using getchar() function:

The string can also be read by calling the getchar() repeatedly to read a sequence of single characters and simultaneously storing it in a character array.

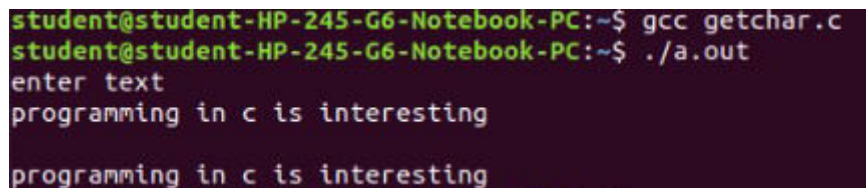
In general syntax :

```
character_variable=getchar();
```

Program:

```
#include<stdio.h>
void main()
{
    char line[81],character;
    int c=0;
    printf("enter text\n");
    do
    {
        character=getchar();
        line[c]=character;
        c=c+1;
    }while(character!='\n');
    c=c-1;
    line[c]='\0';
    printf("\n%s\n",line);
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ gcc getchar.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter text
programming in c is interesting
programming in c is interesting
```

### Functions used to display string on screen:

The strings can be displayed on screen using 3 ways.

1. Using printf( ) function
2. Using puts( ) function
3. Using putchar( ) function repeatedly

#### 1. Using printf() function:

We have used extensively the **printf** function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character. For example, the statement

```
printf("%s",name);
```

can be used to display the entire contents of the array **name**.

We can also specify the precision with which the array is displayed. For instance, the specification

**%10.4**

indicates that the first four characters are to be printed in a field width of 10 columns.

However, if we include the minus sign in the specifications (e.g., %-10.4s), the string will be printed left-justified. The example illustrates the effect of various %s specifications.

**Program:**

```
#include<stdio.h>
#include<string.h>
void main()
{
    char country[15]="United Kingdom";
    printf("%15s\n",country);
    printf("%5s\n",country);
    printf("%15.6s\n",country);
    printf("%-15.6s\n",country);
    printf("%15.0s\n",country);
    printf("%.3s\n",country);
    printf("%s\n",country);
    printf("----\n",country);
}
```

**Output:**



```
United Kingdom
United Kingdom
      United
United
Uni
United Kingdom
----
```

## 2. Using puts( ) function:

This function enables to print/display the string. The general format is

```
puts(string_name);
```

where string\_name should be a valid variable name and it should have been declared a string.

**Program:**

```
#include<stdio.h>
void main()
{
    char str[10]="Hello";
    puts("str value=");
    puts(str);
}
```



**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc puts.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
str value=
Hello
```

### 3. Using putchar() function:

The string can also be written by calling the putchar() repeatedly to print a sequence of single characters.

**Program:**

```
#include<stdio.h>
void main()
{
    char str[10]="Hello ";
    int i=0;
    while(str[i]!='\0')
    {
        putchar(str[i]);
        printf("\n");
        i=i+1;
    }
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc putchar.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
H
e
l
l
o
```

## String Handling Functions

There are several library functions used to manipulate strings. The prototypes for these functions are in header file **string.h**.

### 1. strlen( )

- ◆ This function returns the length of the string i.e. the number of characters in the string excluding the terminating null character.
- ◆ It accepts a single argument, which is pointer to the first character of the string.
- ◆ For example **strlen("suresh")** returns the value 6.
- ◆ Similarly if s1 is an array that contains the name "**deepali**" then strlen(s1) returns the value 7.

**Program:**

```
/*Example of strlen function*/
#include<stdio.h>
#include<string.h>
void main()
{
    char name[]="Sai Aitika";
    int count;
    count=strlen(name);
    printf("string is %s\n",name);
    printf("Its length is %d\n",count);
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc strlen1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
string is Sai Aitika
Its length is 10
student@student-HP-245-G6-Notebook-PC:~$
```

### 2. strcmp( )

- ◆ This function is used for comparison of two strings.
- ◆ If the two strings match, strcmp() returns value 0, otherwise it returns a non-zero value.
- ◆ This function compares the strings character by character.
- ◆ The comparison stops when either the end of string is reached or the corresponding characters in the two strings are not same.
- ◆ The non-zero value returned on mismatch is the difference of the ASCII value of the non-matching characters of the two strings-

**strcmp( s1, s2)** returns a value-

< 0    when s1 < s2  
= 0    when s1==s2  
> 0    when s1 > s2

Generally we don't use the exact non-zero value returned in case of mismatch. We only need to know its sign to compare the alphabetical positions of the two strings. We can use this function to sort strings alphabetically.

**Program:**

```

/*Example of strcmp function*/
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[]="Ramana";
    char s2[]="Sai Aitika";
    int i,j,k;
    i=strcmp(s1,"Ramana");
    j=strcmp(s1,s2);
    k=strcmp(s1,"Ramana V3");
    printf("strcmp example\n");
    printf("%d\n%d\n%d\n",i,j,k);
}

```

Output:

```

student@student-HP-245-G6-Notebook-PC:~$ gcc strcmp.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
strcmp example
0
-1
-32

```

### 3.strcpy()

- ◆ This function is used for copying one string to another string.
- ◆ **strcpy( str1, str2 )** copies **str2** to **str1**.
- ◆ Here str2 is the source string and str1 is destination string.

Program:

```

/*Example of strcpy function*/
#include<stdio.h>
#include<string.h>
void main()
{
    char name[]="copy cat";
    char namecpy[26];
    strcpy(namecpy,name);
    printf("source string is name=%s\n",name);
    printf("Target is namecpy=%s\n",namecpy);
}

```

Output:

```

student@student-HP-245-G6-Notebook-PC:~$ gcc strcpy1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
source string is name=copy cat
Target is namecpy=copy cat

```

### 4. strcat()

- ◆ This function is used for concatenation of two strings.
- ◆ If first string is "King" and second string is "size", then after using this function the first string becomes "Kingsize".
- ◆ **strcat(str1, str2);** /\*concatenates str2 at the end of str1 \*/
- ◆ The null character from the first string is removed, and the second, string is added at the end of first string. The second string remains unaffected.

Program:

```

/* example of strcat function*/
#include<stdio.h>
#include<string.h>
void main()
{
    char x1[]="friends!";
    char x2[25]="Hello!";
    printf(" original strings\n");
    printf("string is x1=%s",x1);
    printf("string is x2=%s",x2);
    strcat(x2,x1);
    printf("After cat string2 is x2=%s",x2);
}

```

**Output:**

```

student@student-HP-245-G6-Notebook-PC:~$ gcc strcat1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
original strings
string is x1=friends!
string is x2=Hello!
After cat string2 is x2=Hello!friends!

```

## 5. strncpy():

It is used to copy only the left most n characters from source to destination. The destination string should be a variable and source string can either be a string constant or a variable.

**Example:**

```

#include<stdio.h>
#include<string.h>
void main()
{
    char source[20]="hello india";
    char destination[20];
    strncpy(destination,source,5);
    destination[5]='\0';
    printf("destination string after strncpy()=%s\n",destination);
}

```

**Output:**

```

student@student-HP-245-G6-Notebook-PC:~$ gcc strncpy.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
destination string after strncpy()=hello

```

## 6. strncat():

It is used to concatenate only the leftmost n characters from source with the destination string.

**Example:**

```

#include<stdio.h>
#include<string.h>
void main()
{
    char source[15]="hello";
    char target[15]="india";
    strncat(source,target,3);
    printf("Target string after strncat()=%s\n",source);
}

```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc strncat1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Target string after strncat()=helloind
```

## 7. strncmp()

- ◆ This function compares the first n characters of the strings s1 and s2.
- ◆ If the first n characters of s1 and s2 are identical then the function will return 0.
- ◆ If the first n characters of s1 are alphabetically lower than s2 then a negative value is returned.
- ◆ If the first n characters of s1 are alphabetically higher than s2 then a positive value is returned.

**Example:**

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[20]="hello";
    char s2[20]="hello world";
    int result;
    result=strncmp(s1,s2,3);
    if(result==0)
    {
        printf("first 3 characters in strings are same\n");
    }
    else
    {
        printf("first 3 characters in strings are not same\n");
    }
}
```

**Output:**

```
student@student-HP-245-G6-Notebook-PC:~$ gcc strncmp1.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
first 3 characters in strings are same
```

## ARRAY OF STRINGS

Suppose that there are 20 students in a class and we need a string that stores names of all the 20 students. We need an array of strings. Such an array of strings would store 20 individual strings. An array of string is declared as,

```
char names[20][30];
```

Here the first index will specify how many strings are needed and the second index specifies the length of every individual string. So here we allocate space for 20 names where each name can be a maximum of 30 characters long. Hence the general syntax for declaring a 2D array of strings can be given as,

```
<data type> <array_name> [<row_size>][<column_size>];
```

Let us see the memory representation of an array of strings. If we have an array declared as,

```
char name[5][10]={“RAM”, “MOHAN”, “SHYAM”, “HARI”, “GOPAL”};
```

Then in memory the array is stored as shown in figure below

Name[0]	R	A	M	\0					
Name[1]	M	O	H	A	N	\0			
Name[2]	S	H	Y	A	M	\0			
Name[3]	H	A	R	I	\0				
Name[4]	G	O	P	A	L	\0			

**Write a program to read and print names of n students of a class**

```
//to read and print names of n students of a class
#include<stdio.h>
#include<string.h>
void main()
{
    char names[5][10];
    int i,n;
    printf("no. of students:");
    scanf("%d",&n);
    printf("Enter the name of students:");
    for(i=0;i<=n;i++)
    {
        gets(names[i]);
    }

    printf("\n the names of students are:\n");
    for(i=0;i<=n;i++)
    {
        puts(names[i]);
    }
}
```

**Output:**

**no.of students: 3**

**Enter the name of students:**

**Vani**  
**Rohith**  
**Rohan**

**The names of the students are:**  
**Vani   Rohith       Rohan**



# ARRAYS, POINTERS

## POINTERS:

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

```
type *var_name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var\_name** is the name of the pointer variable. The asterisk \* used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Some of the valid pointer declarations –

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch;    /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Let us see the relation between arrays and pointers. let us see an example below:

```
//array with pointers
#include<stdio.h>
void main()
{
    int j=3,*a;
    float k=1.5,*b;
    char i='c',*c;

    printf("value of j=%d\n",j);
    printf("value of k=%f\n",k);
    printf("value of i=%c\n",i);

    a=&j;
    b=&k;
    c=&i;

    printf("address in a=%u\n",a);
    printf("address in b=%u\n",b);
    printf("address in c=%u\n",c);

    a++;
    b++;
    c++;

    printf("new address in a=%u\n",a);
    printf("new address in b=%u\n",b);
    printf("new address in c=%u\n",c);
}
```



For example j,k,i are the variables located at 1001,2005 , 4005. If we run the above program, output is as follows.

**Value of j=3**

**value of k=1.500000**

**value of i=c**

**address in a=1001**

**address in b=2005**

**address in c=4005**

**new address in a=1005**

**new address in b=2009**

**new address in c=4006**

## ADDING VALUES TO THE POINTERS:

We can add value to the pointer.

**Example:**

```
int j=9,*k,*l;  
k=&j;  
k=k+1;  
k=k+8;  
l=k+13;
```

## SUBTRACTING VALUES FROM THE POINTERS:

We can subtract values from the pointers .

**Example:**

```
int j=9,*k,*l;  
k=&j;  
k=k-2;  
k=k-3;  
l=k-9;
```

## SUBTRACTIONS IN POINTERS:

Not only a value is subtracted from pointers. We can also subtract one pointer from another pointer. However the pointers should relate to the same array.

**Example:**

```
//subtracting of pointers  
#include<stdio.h>  
void main()  
{  
    int mats[]={11,15,37,28,10,19,18};  
    int *m,*n;  
    m=&mats[1]; //m=mats+1  
    printf("%u\n",m);  
    n=&mats[6]; //n=mats+6  
    printf("%u\n",n);  
    printf("%u\n",*n-*m); //18-15=3  
    printf("%u\n",n-m);  
}
```

## COMPARING POINTERS:

We can compare 2 pointer variables. The pointer variables should relate to the same array. We can compare a pointer to zero also. This can be called as comparing with NULL value.

**Example:**

```
//comparing the pointers
#include<stdio.h>
void main()
{
    int mats[]={11,25,63,27,54,31};
    int *i,*j;
    i=&mats[7];
    j=mats+7;
    printf("%u\n",i);
    printf("%u\n",j);
    if(i==j)
        printf("both pointers point to the same location");
    else
        printf("both pointers are different");
}
```

## Character functions

### List of inbuilt Character functions in ctype.h file:

- “ctype.h” header file support all the below functions in C language.

Functions	Description
<a href="#">isalpha()</a>	checks whether character is alphabetic
<a href="#">isdigit()</a>	checks whether character is digit
<a href="#">isalnum()</a>	Checks whether character is alphanumeric
<a href="#">isspace()</a>	Checks whether character is space
<a href="#">islower()</a>	Checks whether character is lower case
<a href="#">isupper()</a>	Checks whether character is upper case
<a href="#">isxdigit()</a>	Checks whether character is hexadecimal
<a href="#">iscntrl()</a>	Checks whether character is a control character
<a href="#">isprint()</a>	Checks whether character is a printable character
<a href="#">ispunct()</a>	Checks whether character is a punctuation
<a href="#">isgraph()</a>	Checks whether character is a graphical character
<a href="#">tolower()</a>	Checks whether character is alphabetic & converts to lower case
<a href="#">toupper()</a>	Checks whether character is alphabetic & converts to upper case

**Write a C program to read a line of text, and count the number of vowels, consonants, digits, special characters, spaces and words**

*//program to read a line of text, count no. of vowels, consonants, digits, special characters, spaces and words*

```
#include<stdio.h>
#include<ctype.h>
void main()
{
    char text[80],ch;
    int vowel=0,cons=0,digit=0,word=0,space=0,special=0,i=0;

    printf("enter the text:");
    gets(text);

    while((ch=tolower(text[i++]))!='\0')
    {
        if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
            vowel++;
        else if(ch>='a' && ch<='z')
            cons++;
        else if(ch>='0' && ch<='9')
            digit++;
        else if(ch==' ' || ch=='\t')
            space++;
        else
            special++;
    }
    word=space+1;
    printf("\n the text contains:\n");
    printf("no. of vowels=%d\n",vowel);
    printf("no. of consonents=%d\n",cons);
    printf("no. of digits=%d\n",digit);
    printf("no. of special characters=%d\n",special);
    printf("no. of spaces=%d\n",space);
    printf("no. of words=%d\n",word);
}
```

**Output:**

**enter the text:2 cows 3 sheep 4 dogs**

**the text contains:**

**no. of vowels=4**

**no. of consonents=9**

**no. of digits=3**

**no. of special characters=0**

**no. of spaces=5**

**no. of words=6**

## PROGRAMMING EXAMPLES

1. Write a C program to find length of a string.

Program:

```
//length of the string without using string function
#include<stdio.h>
void main()
{
    char str[100];
    int length=0;
    printf("enter the string");
    gets(str);
    for(int i=0;str[i]!='\0';i++)
    {
        length=length+1;
    }
    printf("the length of the string is %d",length);
}
```

Output:

enter the string KRISHNA  
the length of the string is 7

2. Write a C program to reverse a string without using string functions.

Program:

```
//to reverse a string without using string functions
#include<stdio.h>
void main()
{
    char str[100],temp;
    int i=0,j=0,length=0;
    printf("enter a string\n");
    gets(str);
    while(str[i]!='\0')
    {
        length=length+1;
        i=i+1;
    }
    i=0;
    j=length-1;
    while(i<j)
    {
        temp=str[j];
        str[j]=str[i];
        str[i]=temp;
        i++;
        j--;
    }
    printf("\n the reversed string=");
    puts(str);
}
```

Output:

enter a string

**KRISHNA**

the reversed string = ANHSIRK

**3. Write a C program to find whether a given string is palindrome or not?**

**Program:**

```
//to check whether a string is palendrome or not without using string functions
#include<stdio.h>
void main()
{
    char str[100];
    int i=0,j,length=0;
    printf("enter the string:");
    gets(str);
    while(str[i]!='\0')
    {
        length++;
        i++;
    }
    i=0;
    j=length-1;
    while(i<=length/2)
    {
        if(str[i]==str[j])
        {
            i++;
            j--;
        }
        else
            break;
    }
    if(i>=j)
        printf("it is palendrome");
    else
        printf("it is not palendrome");
}
```

**Output:**

enter the string: MADAM

it is palendrome

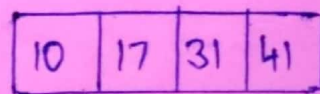
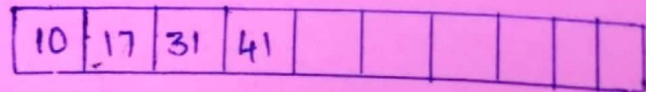


## Not Acceptable Array Declarations

1. `int a[1.7116];` //Not acceptable as size is float
2. `int a[-71];` //Not acceptable as size is negative integer.
3. `int N=10;`  
`int a[N];` //Not acceptable as size is a variable.
4. `int a[];` //Not acceptable as size is unknown.
5. `int a[10] = {1, 2, 9, ..., 12, 2};` //Not acceptable as assignment is not valid.
6. `int a[] = {1, 2, 1, ..., 22, 12};` //Not acceptable as assignment is not valid.
7. `#define N 10`  
`int a[N] = {1, 2, 1, ..., 22, 12};` //Not acceptable as assignment is not valid.
8. `int a["RAM"];` //Not acceptable as size is a string.

## Acceptable Array Declarations:

1. `int a[10];`
2. `#define N 20`  
`int a[N];`
3. `int a[10] = {10, 17, 31, 41};`
4. `int a[] = {10, 17, 31, 41};`
5. `#define N 10`  
`int a[N] = {10, 17, 31, 41};`



## Acceptable styles of declaring 2-D arrays.

1. `int a[4][5];`

2. `int a[4][5] = { { 1, 7, 1 }, { 7, 2, 1 }, { 3, 1, 3 } };`

3. `int a[ ][5] = { { 1, 7, 1 }, { 7, 2, 1 }, { 3, 1, 3 } };`

4. `int a[4][5] = { 1, 2, 2, 2, 2, 2, 1, 1, 1 };`

1	2	2	2	2
2	1	1	1	

5. `int a[ ][5] = { 1, 2, 2, 2, 2, 2, 1, 1, 1 };`

1	2	2	2	2
2	1	1	1	

6. `int a[2][4][5];`

plane 0:


plane 1:


7. `a[2][4][5] = { { { 1, 2 }, { 1, 4 } }, { { 3, 4 }, { 4, 5 } } };`

1	2			
1	4			

3	4			
4	5			



## Acceptable Declarations at 2-D character Arrays

1. `char X[10][20];`

2. `char X[5][20] = { "RAM", "RAVI", "ABHI" };`

R	A	M	\0																
R	A	V	I	\0															
A	B	H	I	\0															

3. `char X[ ][20] = { "RAM", "RAVI", "ABHI" };`

R	A	M	\0																
R	A	V	I	\0															
A	B	H	I	\0															

4. `#define M=10`

`#define N=20`

`char X[M][N] = { "RAM", "RAVI", "ABHI" };`

R	A	M	\0																
R	A	V	I	\0															
A	B	H	I	\0															

## Un-Acceptable ways of Declaring String Variables.

1. `char X[-170];` /\* Negative size \*/
2. `char X[1.780];` /\* Float Array Size \*/
3. `char X["RAM"];` /\* size cannot be String \*/
4. `int n=20; char X[n];` /\* size cannot be a variable \*/
5. `char X[]; X="RAM";` /\* default size is not acceptable \*/