Arrays:

- Array is a collection of similar type of elements which are in contiguous memory location.

Syntax:

datatype array_name[number of elements];

Examples:

```
int   a[5] ;

char  ch[10] ;

float   f[5] ;
```

NOTE: The elements of array share the same variable name but each element has a different index number known as subscript.

- **Types of Arrays.**

1. One dimensional array having one subscript.
2. Two dimensional array having two subscripts.
3. Multidimensional array having more than two subscript.

❖ One Dimensional Array

- ### Declaration of 1-D array:

Like other simple variables, arrays should also be declared before they are used in program.

Syntax:

datatype array_name[number of elements];

Examples:

```
int   a[5] ;

char  ch[10] ;

float   f[size] ;       /* not valid */
```

- **Accessing 1-D array elements:**

- Specifying the array name followed by subscript in brackets can access the elements of an array.
- Array subscript starts from 0.
- If there is an array of size 5 then the valid subscripts will be from 0 to 4.
- The last valid subscript is one less than the size of array.

Example:

```
int   a[5] ;       /* Size of array a is 5,can hold five integer elements */
```

The elements of this array are:

a [0], a[1], a[2], a[3], a[4]

NOTE: The subscript can be any expression that yields an integer value. It can be any integer constant, integer variable, integer expression or return value(int) from a function call.

Example:

If i and j are integer variables then then these are some valid subscripted array elements:

a[3] a[i] a[i+j] a[2*j] a[i++]

NOTE: A subscripted array element is treated as any other variable in the program.

we can store values in them, print their values or perform operation that is valid for any simple variable of the same data type.

NOTE: If we have an array a of size 5,the valid subscript are only 0,1,2,3,4.

if someone tries to access element beyond these subscripts, like a[5],a[10], the compiler will not show any error message but this may lead to run time errors.

So it is the responsibility of programmer to provide array bounds checking whenever needed.

- ## Processing 1-D array:

For processing array we generally use for loop and the variable is used at the place of subscript.

Example:

Suppose a[10] is an array of int type,

- **Reading elements in array**

```
for(i=0;i<10;i++)
        scanf("%d", &a[i] ) ;
```

- **Displaying elements of array**

```
for(i=0;i<10;i++)
        printf("%d", a[i] ) ;
```

- ## Initialization 1-D array:

Syntax:

```
datatype    a[size]={value1,value2……..valueN};
```

Example:

```
int a[5] = {10,20,30,40,50};
```

NOTE: After declaration, the elements of a local array have garbage value while the elements of global and static arrays are automatically initialized to zero.

NOTE: In 1-D array, it is optional to specify the size while initializing .

NOTE: During If elements are less all remaining elements will be zero. If elements are more compiler will generate error.

NOTE: we can't copy all the element of an array to another array by simply assigning it to other array. We have to copy all the element of array one by one using for loop.

Example:

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

int  b[5];

b=a;     /* not valid  */
```

Some important points you must know:

- We can pass individual array elements to a function.

NOTE: you must and should have knowledge of function to understand below statements.

Example:

```
Void check(int num);    //function declaration.

Check(a[i]);     //function calling.
```

- We can pass whole 1-D array to a function.

Example:

```
int  a[5] ;

check(a) ;      //function calling.
```

NOTE: While passing whole array actual argument are affected by changes made on formal arguments.

NOTE: It is optional to specify size of array in formal argument in function definition so we can write general function that can work on arrays of same type but different sizes.

❖ Two Dimensional Array

- **Declaration and Accessing individual elements of 2-D array:**

Syntax:

datatype array_name[rowsize] [columnsize];

Example:

int a[4][5];

NOTE: the total number of elements in the array are rowsize*columnsize.

- The total number of elements in this array is 4 * 5=20.

	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
Row1	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
Row 3	a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

- **Processing 2-D array:**

For processing 2-D arrays, we use two nested for loops. The outer for loop corresponds to the row and the inner for loop corresponds to the column.

Example:

int a[4][5];

- **Reading elements in array**

```
for(i=0;i<4;i++)
    for(j=0;j<5;j++)
        scanf("%d", &a[i][j] ) ;
```

- **Displaying elements of array**

```
for(i=0;i<4;i++)

        for(j=0;j<5;j++)

                printf("%d", a[i][j] );
```

- ## Initialization 2-D array:

2-D arrays can be initialized in a way similar to that of 1-D arrays.

Example:

```
int  a[4][3] = {11,12,13,14,15,16,17,18,19,20,21,22};


int  a[4][3] = {
                {11,12,13},          // Row 0
                {14,15,16},          // Row 1
                {17,18,19},          // Row 2
                {20,21,22}           // Row 3
                };
```

NOTE: In 2-D array, it is optional to specify the first dimension while initializing but the second dimension should always be present.

NOTE: we can consider above 2-D array as four 1-D array and each 1-D array is having three elements in it.

❖ Multi-Dimensional Array

Example:

```
int  a[2] [4] [3];
```

- The total number of elements in this array is 2 * 4 * 3=24.

NOTE: We can consider above 3-D array as two 2-D array and each of those 2-D array has 4 rows and 3 columns.

Strings:

In C strings are treated as arrays of type char and are terminated by a null character('\0').This null character has ASCII value zero.

- These are the two forms of initialization of a string.

Example:

```
char str[10] = {'v' ,'e' ,'c' ,'t', 'o' ,'r' ,'\0'};

char str[10] = "vector";      /* here the null character is automatically placed at end*/
```