Arrays

Concept:

- An array is a group of data items of the same data type that share a common name.
- An array should be of a single type, comprising of integers or strings and so on.
- Only one name is assigned to array and individual elements are referenced by specifying a subscript. A subscript is also called an index. In C, subscripts start at 0(zero) and cannot be negative.
- There are two types of arrays :
    1. One dimensional array
    2. Multi-dimensional array

Definition:

- An Array is collection of similar Elements.
- The C Language provides capability that enables user to design a set of similar data types

Why To Create an array :

Suppose we want to arrange percentage marks of 100 students in ascending order.

Two Solutions:

1. Construct 100 variables to store percentage marks obtained by 100 different students. i.e. each containing one student's marks.

2. Construct one variable (called array or subscripted variable) capable of storing or holding all the hundred values.

Second solution is better than first .

Declaration of one dimensional array :

Syntax: data type arrayname[size];

Where,

Data type : the type of the data stored in the array

Arrayname: Name of the array (name variable name)

Size: Maximum number of elements that an array can hold.

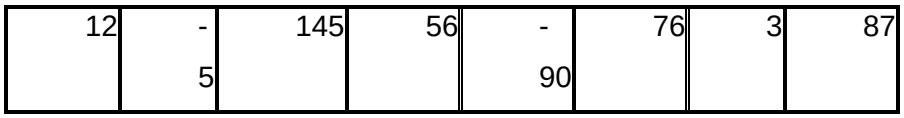
Example :

```
int marks[8];
```

marks is array name which can hold 8 integers.

Memory representation :

Marks[0] marks[1]……..marks[7]



All array elements would always be present in contiguous memory locations.

16 bytes immediately gets reserved in memory.

All elements will get initialized with garbage values.

Initialization of one dimensional array:

1. You can initialize the array elements one by one :

   Syntax:

   Arrayname[index] = value;

   Example:

   Marks[0] = 34;

   Marks[1] = 77;

   …….

   Marks[7]=99;

2. You can initialize the complete array directly at the time of declaration.

   Syntax:

   DataType arrayname[] = {list of values};

   Examples:

   Integer array

   int arr[3]={100,200,300};

   int n[] = {4,6,5,7,8};

   Float array

   float price[3]={1.2,2.3,5.6};

   float price[] ={29.5,56.80,7.5}; /// size can be omitted during declaration

   Character array

   char name[6] = {‘h’,’e’,’l’,’l’,’o’,’\0’};

char name[] = “book”;

when the compiler sees a character array , it terminates it with an additional null character . Thus , when declaring character arrays, we must always allow one extra element space for the null terminator.

3.

```
#define   qty  10
int  item[qty];
```

declares item as an array of 10 elements.

4.

```
int   x[5] = {2,5};
```

The elements which are not explicitly initialized are automatically set to zero.

Means in this case:

X[0] = 2; x[1] = 5; x[2] = 0; x[3] = 0; x[4] = 0;

Accessing and Displaying array elements :

- To Access an individual element of an array , use the name of the array name followed by the index of the element in square brackets. Array indices start at 0 and end at size-1

  Arrayname[index] ;

  Accesses the index’th element of array_name starting at zero. If index is 0 ,then the first element is accessed.

Examples:

```
int num[10];
num[0] = 5;   //sets  the first element of num to 5.
```

Note that it is not valid to attempt to access num[10] -- this is outside the bounds of the array.

One can access num[0] to num[9] --- 10 values.

```
Scanf(“%d”,&num[i]);     //  accessing an array element
Printf(“%d”,num[i]);      //   displaying an array element
```

Arrays and Function :

There are two methods to pass array to function.

1. Passing entire array at a time
2. Passing array elements one-by-one.

1. Passing entire array

A complete array can be passed to a called function by just giving the name of the array as the parameter for that called function- since array name specifies the base address(address of the first element) of the array in the memory, we are sending the entire array.

Example:

```
int  num[5];            //array with 5 elements
int sum(int []);                //function prototype
int sum(num);           //passing the address of 1st element of the array i.e passing //entire array
```

In the above example num is array with 5 elements and name num is equivalent to &num[0] . sum is function and num i.e array name is parameter of that function. (i.e we are passing entire array at a time to called function sum).

2.Passing array elements one-by-one

Array elements can also be passed to the called function one-by-one. Thus the function can access only one element at a time.

Example:

```
int num[5];       //array with  5 elements
int sum(int);     //function prototype

for(i=0;i<5;i++)
        sum(num[i]);  //passing one-by-one elements to function.
```

Multi-dimensional Arrays:

Arrays in C may have more than one dimension. Such arrays are called multi-dimensional arrays.

Two dimensional Arrays:

We need two dimensional array to store a table of values. Thus it is referred to as a matrix or a table.

A matrix has two subscripts.- 1. number of rows 2. number of columns

Representation of 2D array is as follows:

Arrayname [row][column]

Syntax:

Data type arrayname[row][column];

Where,

Data type : the type of the data stored in the array

Arrayname: Name of the array (name variable name)

Row: Maximum number of rows in the array .

Column Maximum number of columns in the array .

Example:

int matrix[2][3]; // implies 2 rows and 3 columns

Initialization:

int matix[2][3]={1,2,3,4,5,6};

This 2-D array is thought as:

	Col 0	Col 1	Col 2
Row 0	1	2	3
Row 1	4	5	6

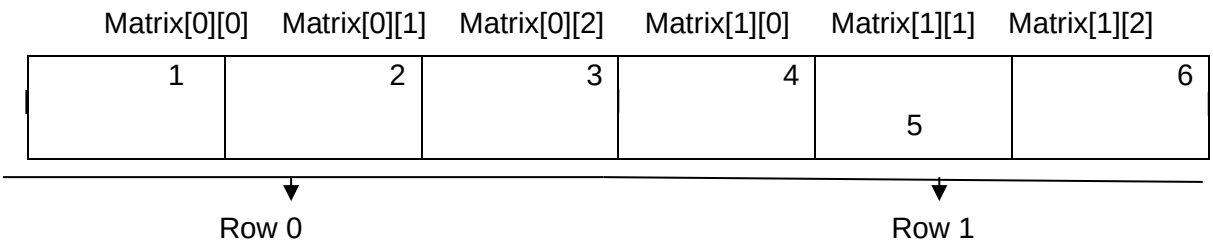
The memory of a computer is linear and not a matrix like 2-D array. So, the elements of the array are stored either by row , called “row major”, or by column called “Column major”.

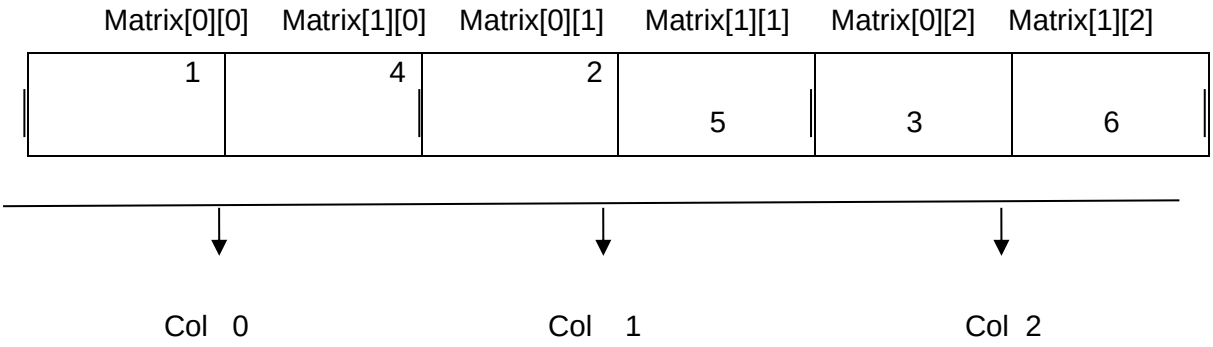
1. Row major: Elements of row 0 are stored first and the row 1 is stored. Thus, the elements of each row are stored in order.

   Row order major is used mostly in C and C++.

2. Column major: Elements of column 0 are stored 1st and then column1 and next column 2. Thus, the elements of each column are stored in order.

   Used mostly in Fortran.





Three Dimensional array:

It takes 3 subscripts:

int mat[3][2][3];

In this example mat consists of 3 2-D arrays having 2 rows and 3 columns.

2D Array 0

1	2	3
4	5	6

2D Array 1

7	8	9
10	11	12

2D Array 2

13	14	15
16	17	18

Advantages Of Array:

- Arrays are very useful while working with sequence of the same kind of data.
- We can use same name for similar objects and save them with same name but different indexes.
- Random access – Any element in the array can be accessed immediately if its exact position in the array is known.

Limitations Of Array:

- Constant size - To get an array of different size, you must explicitly deal with memory using realloc, malloc etc.
- Constant Data type.

- Large free sequential memory block to store large arrays.
- Arrays do not support copy assignment .
- Elements can not be inserted in between into arrays- Inserting an element into an array means shifting down by one place of all the elements after the correct position of inserted element.
- Elements can not be deleted from arrays- deleting an element from an array means shifting up by one place of all the elements.
- Boundary checking is not done in C, its programmer's duty to check the boundaries. E.g. int arr[10] ; suppose user is trying to access arr[34] ,user will not get syntax error but will get unpredictable output.

Dynamic Memory Allocation (DMA):

- The Process of allocating memory at run time is known as Dynamic Memory Allocation.
- Consider an array declaration : int arr[20];
  This reserves 40 bytes (20elements * 2 bytes each) for arr in the memory. But we may not use all the storage space. In such case there is wastage of the memory space. On the other hand, if we want to store 30 elements, then there is no way to increase the array size during program execution. (i.e. at run time)
- Such Problems can be solved by the use of dynamic memory allocation technique.
- One can declare pointer instead of declaring array to avoid above problem. After declaring pointer memory can be allocated at run time as per user requirement using predefined memory allocation functions available in <alloc.h>

Memory Allocation Functions :

Function	Meaning
Malloc	Allocates requested size of bytes and returns the pointer to 1st byte
Calloc	Allocates space for an array of elements, Initializes them to zero and then return the pointer to the memory
Free	Frees the previously allocated space.
Realloc	Changes (modifies) the size of previously allocated space.

malloc() :

Syntax:

ptr = (cast_type *) malloc(byte_size);

where , ptr is a pointer of type cast_type . Type casting is required because malloc() returns void pointer.

Example:

```
int *ptr;
 ptr = (int *)malloc(sizeof(int) * 10);
```

This statement reserves a block of memory which can store 10 integers i.e. 20 bytes.

```
char *ch;
ch = (char *)malloc(sizeof(char) * 10);
```

This statement reserves a block of memory which can store 10 characters i.e. 10 bytes.

How to allocate memory to pointer to pointer

```
  e.g.  int **p;
p  = (int **)malloc(sizeof(int) * no_of_rows);
for(i=0;i<no_of_rows)
    *p = (int *)malloc(sizeof(int) * no_of_columns);
```

calloc() :

syntax:

ptr = (cast_type *) calloc(nitems,byte_size);

where , ptr is a pointer of type cast_type . Type casting is required because calloc() returns void pointer.

Example:

```
int *ptr;
ptr = (int *)calloc(10, sizeof(int));
```

This statement reserves a block of memory which can store 10 integers i.e. 20 bytes and all integers initialized with zero.

free() :

syntax:

free(ptr);

releases the memory pointed by the ptr.

Where, ptr is a pointer to memory which has already been created by malloc or calloc. When we no longer need the data we stored in the block of memory and also don't want to use that block any more for storing any other information, we may release that storage using free().

realloc :

syntax:

pointer = realloc(pointer,newsize);

Example:

ptr = realloc(ptr,5);

It allocates new memory size of newsize to the pointer variable ptr and returns the pointer to the 1st byte of the new memory block.

Advantages of Dynamic memory allocation(DMA) over static memory allocation:

OR

Advantages of Run time memory allocation over compile time memory allocation:

OR

Advantages of Pointers over Array:

1. Allocates memory at runtime.
2. Neither wastage nor shortage of memory space.
3. It can either grow or shrink during execution of the program.
4. Data can be rearranged efficiently through DMA.
5. Speed of DMA is faster than static allocation.
6. Freeing of unnecessary memory is possible through DMA.