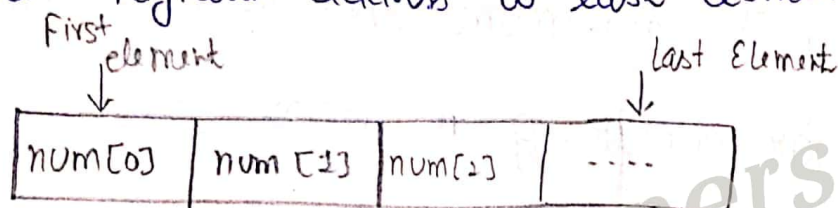


# Array

- An array is a group of similar data items sharing a common name. Each element of an array has a unique Index number.
- Array are always stored at contiguous (one after the other) memory location.
- The lowest address corresponds to the first element and the highest address to last element.



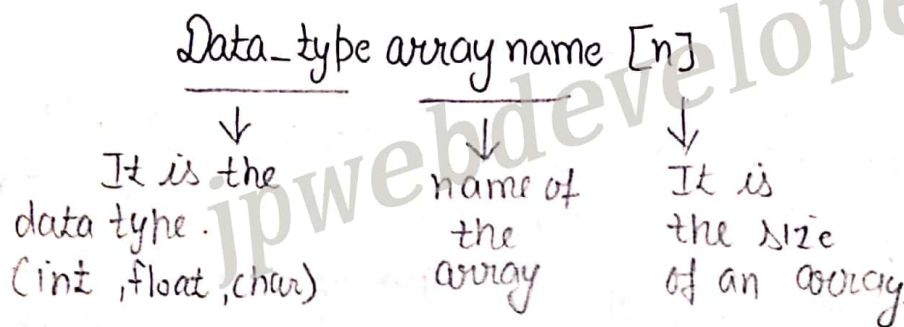
- Array is a derived data type.
- It is a collection of same data elements. Each element of an array has a unique index number. Each element of an array can be accessed with the combination of array name and the Index number.
- An array with a single index number is called single dimensional array. and the array with two subscripts for a single element is called two dimensional array.
- The index is enclosed in brackets after array name.  
 e.g.  $\text{int } a[10]$ ,  
 ↓                      ↓                      → subscript or index  
 data type            name of array

# Array Declaration:-

Like other variables, an array has to be declared before it is used. To declare an array, we must provide the following information:

1. Data type of an array.
2. The name of the array.
3. The number of index (subscripts) in an array.
4. The maximum value of each subscript.

## Syntax for declaring an array



e.g.:-

int a [10]; (one-dimensional array because of one subscript)

→ int specifies the type of variable.

→ a is the name of array variable.

→ [10] is the subscript

10 elements stored in this array.

e.g. (II):

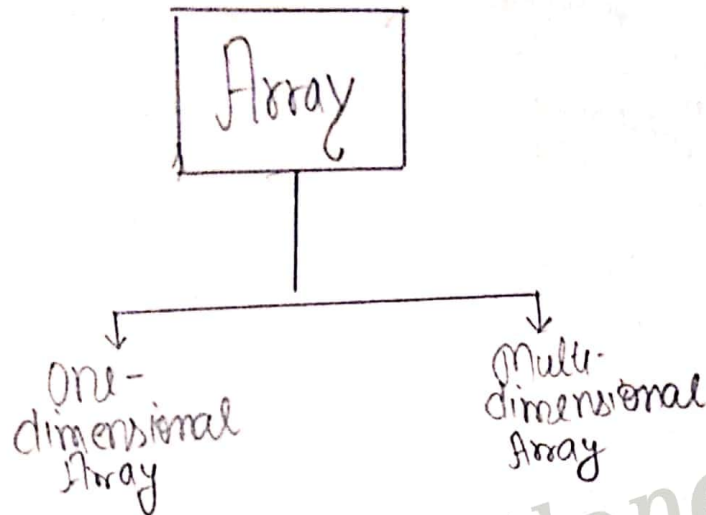
int a [5] [6]; (two dimensional array because of two subscripts)

↓  
Index 1

↓  
Index 2

# Types of Array!

1. Single dimensional Array. Or One dimensional array.
2. Two " "
3. Multi " "



## 1. Single dimensional array:

A Single dimensional array has only a single subscript. The subscript number can identify the number of individual elements in the array. They take contiguous memory location either column wise or row wise.

for example

int arr [10];

↓            ↓            ↓

data type   name of the array   size of an array

Syntax:-

<datatype><arrayname>[size of an array];

# Memory Representation of Single Dimensional Array

`int ar[10] = {20, 30, 40, 50, 60, 70, 80, 90, 100, 110};`

Row wise

Storage	Index
20	ar[0]
30	ar[1]
40	ar[2]
50	ar[3]
60	ar[4]
70	ar[5]
80	ar[6]
90	ar[7]
100	ar[8]
110	ar[9]

`ar[0] = 20`

`ar[1] = 30`

⋮

`ar[9] = 110`

Column wise

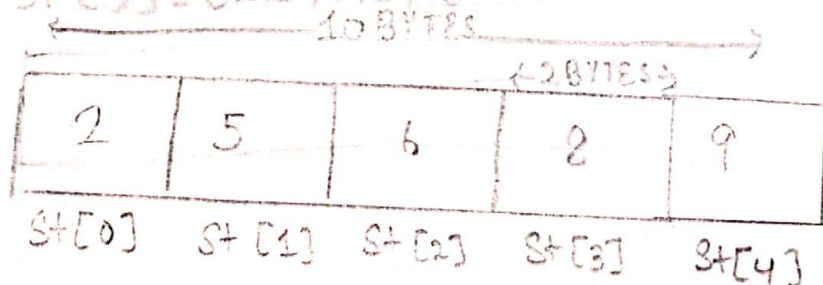
Index	0	1	2	3	4	5	6	7	8	9
Storage	20	30	40	50	60	70	80	90	100	110

## Initialization of One-dimensional Array

One dimensional array can be declared and initialized at the same place following are the different ways to initialize array:-

Syntax :- `<datatype> <name of the array> <size is optional>`  
`= {assigned values}`

Ex 1: `int s[5] = {2, 5, 6, 8, 9}`



Ex 2: `int i[10] = {10, 20, 30, 40, 50}`

It will take first five elements as it is and next five as the Garbage values.

# Two Dimensional Array:-

Two-dimensional array identified by two subscripts -  
A 2-D array is also ~~not~~ referred as a matrix or grid. A matrix has two subscript. one subscript tells us number of rows and the second subscript tells about the columns.

for example

```
int m[10][20];
      ↓  ↓   ↓   ↓
datatype matrix rows columns
```

⇒ Here m is declared as a matrix having 10 rows and 20 columns

## Syntax

<datatype> \* name of array > [no. of rows] [no. of columns];

## \* Memory Representation of Two-dimensional Array

Memory does not have any concept of rows and columns. It stores the elements in one continuous chain wheather it is one-dimensional array or more than one-d array.

```
int a[2][3] = {10, 20, 30, 40, 50, 60}
```

memory map

	a[0]	a[1]	a[2]
Row 0	10	20	30
Row 1	40	50	60

memory map in actual Memory

10	a[0][0]
20	a[0][1]
30	a[0][2]
40	a[1][0]
50	a[1][1]
60	a[1][2]

# Initialization of Two dimensional Array

The General form of initializing 2-D array is -

data\_type array\_name [row size] [column size] =  
{ { list of row values }, { list of col values } };

Ex 1: int st [2][3] = { 0, 0, 0, 1, 1, 1 };

OR

int st [2][3] = { 0, 0, 0 }, { 1, 1, 1 }.

Note When we Initializing 2-d array, we have to mention the second dimension. Where as first dimension is optional. Example

int [2][ ] = { 10, 20, 30 } // Never work

int [ ] [3] = { 10, 20, 30 } // Yes

## Example of 1-D Array

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int arr[5];
    int i;
    for(i=0; i<5; i++)
    {
        printf("Enter the array elements");
        scanf("%d", &arr[i]);
    }
    printf("\n Printing elements of array");
    for(i=0; i<5; i++)
    {
        printf("%d |t", arr[i]);
    }
    getch();
}
```

## Example of 2-D array

```
void main()
{
    int arr[2][2];
    int i, j;
    for(i=0; i<2; i++)
    {
        for(j=0; j<2; j++)
        {
            printf("Enter array element");
            scanf("%d", &arr[i][j]);
        }
        printf("\n");
    }
    printf("In Printing array elements");
    for(i=0; i<2; i++)
    {
        for(j=0; j<2; j++)
        {
            printf("%d |t", arr[i][j]);
        }
        printf("\n");
    }
    getch();
}
```

# Multidimensional Array

A multidimensional array has more than one subscript.  
A two-dimensional array has two subscripts, a three dimensional array has three subscripts and so on.  
There is no limit to the number of dimensions a C array can have.

⇒ Two dimensional array and more than two dimensional arrays are under the category of multidimensional array.

⇒ C allows more than 2-d arrays but in practice more than two dimensions are rarely used.

Multidimensional array can be used in the same manner as two-dimensional arrays.

The General form of a Multidimensional Array :-

type arrayname [s1] [s2] ... [sn];

The number of size specifiers depend upon the dimension of the array.

for example

int a [3] [5] [12];

float num [5] [4] [5] [3];

a is three dimensional array } Mult  
num is four dimensional array }

# WAP to find the sum of two Matrices

Imp.

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
void main()
```

```
{
```

```
int a[2][2], b[2][2], i, j;
```

```
printf("Enter the elements of A matrix \n");
```

```
for (i=0; i<2; i++)
```

```
{
```

```
for (j=0; j<2; j++)
```

```
{
```

```
printf("Enter the number ");
```

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

```
}
```

```
}
```

```
printf("Enter the elements of B matrix \n");
```

```
for (i=0; i<2; i++)
```

```
{
```

```
for (j=0; j<2; j++)
```

```
{
```

```
printf("Enter the number");
```

```
scanf("%d", &b[i][j]);
```

```
}
```

```
}
```

```
printf("Addition of A and B Matrix \n");
```

```
for (i=0; i<2; i++)
```

```
{
```

```
for (j=0; j<2; j++)
```

```
{
```

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

```
}
```

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

```
}
```

```
getch();
```

```
}
```

## Advantages of Array :-

- ⇒ It is the simplest kind of data structure.
- ⇒ The calculation of matrices become easy with the use of arrays.
- ⇒ Array make loops very effective.
- ⇒ We can refer to a group of elements by a single name.
- ⇒ The method of Accessing array element is very simple.
- ⇒ Easy to declare array element
- ⇒ All array element can store and accessed with a single variable name. `int a [2] [3] [4];`

## Disadvantages of Array :-

- ⇒ Sometimes It's not easy to work with many Index arrays.
- ⇒ The size of the array should be known in advance.
- ⇒ Wastage of Memory.
- ⇒ Array will always hold similar kind of information so it is impossible to store unrelated information to the array.
- ⇒ Array are largely static.

## \* Passing Array to function

We Pass array to function when we need to Pass a list of values to a given function.

The array elements can be Passed to a function by two methods.

1. Calling the function by value.
2. Calling the function by reference.

1 Calling the function by value:- In call by value method, We Pass the values of array elements to the function.

Example :-

```
#include <stdio.h>
#include <conio.h>
void show(int a)
{
    printf ("-d %d", a);
}
void main()
{
    clrscr();
    int a int a[] = {10, 20, 30, 40, 50};
    printf ("The elements of array : \n");
    for (int i = 0; i < 5; i++)
    {
        show (a[i]);
    }
    getch();
}
```

Calling the function by Reference:- In call by Reference, we Pass address of array elements to the function.

Example:

```
#include <stdio.h>
#include <conio.h>
void show (int *a)
{
    printf ("-ld\t", a);
}
void main()
{
    clrscr();
    int a[] = {10, 20, 30, 40, 50};
    printf ("The elements of Array:\n");
    for (int i=0; i<5; i++)
    {
        show (&a[i]);
    }
    getch();
}
```

## \* Pass Entire Array to the Function

In place of passing individual elements of an array one by one to one a function, we can also pass an entire array to a function.

### Example

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int arr[] = {10, 20, 30, 40, 50};
    clrscr();
    show(arr, 5);
    getch();
}

show(int *a, int size)
{
    int i;
    for(i=0; i < size; i++)
    {
        printf(" %d", *a);
        a++;
    }
}
```

