:
:
:

data-type member ;
};

Advantages of structure :-
- It can hold variables of different data types.
- We can create objects containing different types of attributes.
- It allows us to re-use the data layout across programs.
- It is used to implement other data structure like linked list, queues, trees and graphs.

Program :-
how to use structure in program →

```
#include <stdio.h>
#include <conio.h>
Void main ()
{
struct employee
{
int id ;
float salary ;
int mobile ;
};
```

```
struct employee e1, e2, e3;
printf ("\n Enter ids, salary & mobile no. \n");
scanf ("%d %f %d", &e1.id, &e1.salary, &e1.mobile);
scanf ("%d %f %d", &e2.id, &e2.salary, &e2.mobile);
printf (%d %f %d", &e3.id, &e3.salary, &e3.mobile);
printf ("\n Entered result");
printf ("\n %d %f %d", e1.id, e1.salary, e1.mobile);
printf ("\n %d %f %d", e2.id, e2.salary, e2.mobile);
printf ("\n %d %f %d", e3.id, e3.salary, e3.mobile);
getch ();
}
```

output →

guess the output

And write it here....

Array :- Arrays are defined as collection of similar type of data items stored at contigous memory locations.

Array is the simplest data structure where each data element can be randomly accessed by using its index number.

Array declaration :-

```
int arr[10] ; char arr[10] ; float arr[5]
```

Program without Array :-

```
#include <stdio.h>
void main ()
{
 int marks-1 = 56 ; marks-2 = 78, marks-3 = 89 ;
 float avg = (marks-1 + marks-2 + marks-3)/3 ;
 print (avg) ;
}
```

Program by using Array :-

```
#include <stdio.h>
void main
 {
 int marks [3] = { 56, 78, 89} ;
 int i ;
 float avg ;
 for (i=0 ; i<3 ; i++)
```

```
    {
    avg = avg + marks[i];
    }
  printf(avg);
}
```

Complexity of Array operations :-

1). Time complexity :-

Algorithm	Average case	worst case
Access	O(1)	O(1)
search	O(n)	O(n)
insertion	O(n)	O(n)
Deletion	O(n)	O(n)

2). Space complexity :-

In Array space complexity for worst case is O(n)

Memory Allocation of the Array :-

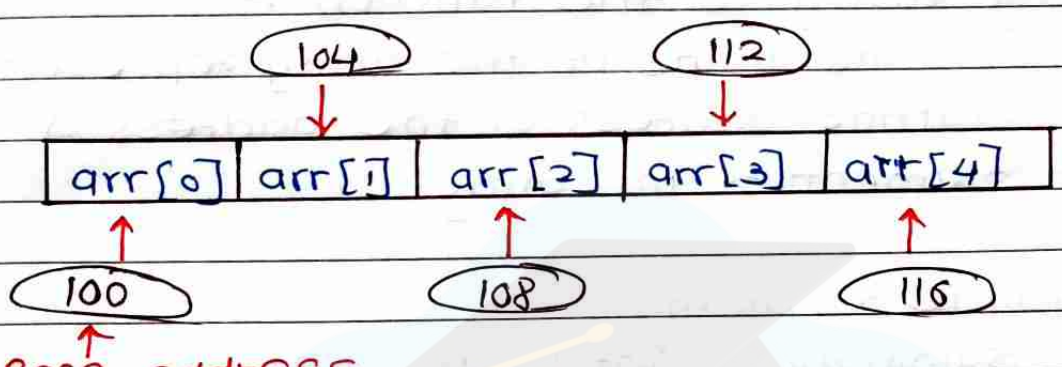
Each element in Array represented by indexing Indexing of array can be defined in three ways:

1. O (Zero Based indexing) :-

The first element of the array will be arr[0].

2. 1 (one-based indexing) :-
The first element of array will be arr[1].

3. n (n-based indexing) :-
The first element of array can reside at any random index number.



(fig : int arr[5])

Accessing elements of an Array :-
To access any random element of an array we need the following information :
1. Base address of the array
2. size of an element in bytes.
3. which type of indexing, array follows.

Address of any element of 1D array can be calculate

Byte address of element A[i] = base address + size * (first - index)

Example: In an array, A[-10 .... +2] Base address (BA) = ggg, size of an element = 2 bytes, find location of A[-1].

Solution: $L(A[-1]) = ggg + [(-1) - (-10)] \times 2$

$= ggg + 18$

$= 1017.$

$\therefore$ location of $A[-1] = 1017.$

Passing array to the function :-

The name of the array represents the starting address or the address of the first element of the array.

Program:

```
#include <stdio.h>
int summation (int []);
void main ()
{
 int arr [5] = {0,1,2,3,4};
 int sum = summation (arr);
 printf ("%d", sum);
}
int summation (int arr[])
 {
  int sum =0, i;
  for(i=0 ; i<5 ; i++)
     {
      sum = sum + arr [i];
     }
  return sum ;
}
```

2D Array :- 2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as collection of rows and columns.

How to declare 2D Array :-

The syntax for declaration of two dimensional array is as follows :

```
int arr[max-rows][max-columns];
```

However, it produces the data structure which looks like following :

	0	1	2	...	n-1
0	a[0][0]	a[0][1]	a[0][2]	.....	a[0][n-1]
1	a[1][0]	a[1][1]	a[1][2]	....	a[1][n-1]
2	a[2][0]	a[2][1]	a[2][2]	...	a[2][n-1]
:	:	:	:	...	:
n-1	a[n-1][0]	a[n-1][1]	a[n-1][2]		a[n-1][n-1]

a[n][n]

(Fig : a[n][n])

How to access data in 2D-array :-

Due to fact that elements of 2D arrays can be random accessed.

```
int x = a[i][j];
```

where i, j are the rows and coloumns respectively.

Initializing 2D arrays :-

The syntax to declare and initialize the 2D array is given as follows :

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

number of elements in 2D arrays
= number of rows * number of coloumns.

Mapping 2D array to 1D array :-

The size of a two dimensional array is equal to the multiplication of number of rows and number of coloumns present in the array.

A 3x3 two dimensional array is as shown:-

	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2)

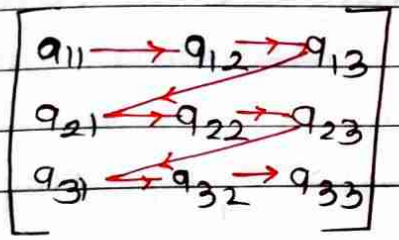
column index

row index

There are two main techniques of storing 2D array elements into memory.

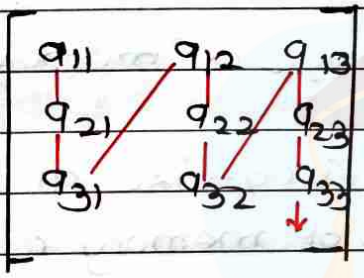
1. Row major ordering :—

In row major ordering, all the rows of 2D array are stored into memory contiguously.



2. Column major ordering :—

According to column major ordering, all the columns of 2D array are stored into the memory contigously.



Calculating address of random element of a 2D array:—

1). By row major order :—

If array is declared a[m][n] where m is the number of rows while n is number of columns. then address of an element a[i][j] is calculated as,

Address (a[i][j]) = B.A + (i * n + j) * size

B.A → Base Address

2). By column major order :—

Address (a[i][i] = (j * m) + i) * size + B.A.

Linked list :-

✓ Why there is a need of linked list ?

If we declare an array of size 3. As we know that all the values of an array are stored in a continous manner, so all three values of an array are stored in a sequential fashion.

Then, total memory space occupied by array would be $3 * 4 = 12$ bytes.

Drawbacks of using array :-

- we cannot insert more than 3 elements in above example because only 3 spaces are allocated by 3 elements.
- In case of array, the wastage of memory can occur.
- In array, we are providing fixed-size at compile time, due to which wastage of memory occurs.

The solution to this problem is to use linked list.

What is Linked List ?

A linked list is also a collection of elements, but the elements are not stored in a consecutive location. or linked list is a collection of the nodes in which one node is connected to another node and node consists of two parts i.e. one is data part and second one is the address part.