**C Arrays [1-D]**

Array is a linear collection of similar elements and it is also known as subscript variable.

Syntax :-

// If We not Initialize The array element during the declaration of array we have to maintain the size of array.

* **Data\_type Array\_Name[Size Of Array] ;**

// If We Initialize The array element during the declaration of array we have a choice we can or cannot maintain the size of array.

* **Data\_type Array\_Name[] = {Array Elements};**

**Declaration of array**

// we can’t declaration of array without mention the size of array

* Int array[ ];

// we can declaration of array with the size of array. When we declare with size it contains the memory block in a sequence according to the length of size. These memory block contain space in memory in the form of byte. And these byte depend on data type and architecture of operating system.

* Int array[ 7 ];

Example Array

These are the index number

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Block1  (4Byte) | Block2  (4Byte) | Block3  (4Byte) | Block4  (4Byte) | Block5  (4Byte) | Block6  (4Byte) | Block7  (4Byte) |

28 byte memory

Note

* 0 to 6 is a index number and we read as index of array 0, array 1 so on.
* We can access any value of array through array index number.

Main(){

Arr [7]= {20,40,60,80,90,35,55};

Array

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 20 | 40 | 60 | 80 | 90 | 35 | 55 |

Arr[4]; index number

Printf(“%d”, Arr[]);

}

//If we declare the array with the size and without initialized the element of array in that case all block contains garbage value.

* Int array[ 7 ];

Array

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Garbage  value | Garbage  value | Garbage  value | Garbage  value | Garbage  value | Garbage  value | Garbage  value |

Example

void main()

{

   int arr[5];

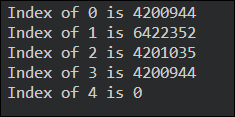
   for(int i=0;i<5;i++) {

      printf("Index of %d is %d",i, arr[i]);

      printf("\n");

   }

Output



**Declaration with initialization of array**

* **Int arr[5]={1,2,3,4,5,6,};**
* **Int arr[ ]={1,2,3,4,5,6,};**
* If We Initialize The array element during the declaration of array we have a choice we can or cannot maintain the size of array.
* We can’t Initialize The array elements more then the size of array.
* If we Initialize The array elements more then the size of array. It is also called index out of bound error.
* If we Initialize The array elements less then the size of array. In that case rest of the array block automatic contain null value(0) .

Arr [7]= {20,40};

Array

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 20 | 40 | 0 | 0 | 0 | 0 | 0 |

Example

Main()

{

   int array[5]={5,9};

   for(int i=0;i<5;i++){

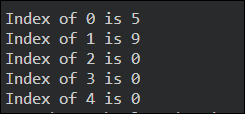
      printf("Index of %d is %d",i, array[i]);

      printf("\n");

}

}

Output



**Memory Allocation in array**

In array memory’s are allocated in continuous form.

Int Arr[3]={5,6,7};

Arr[0] Arr[0] Arr[0]

4 Byte 4 Byte 4 Byte

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

If we declare random variables . it allocate random address of each variable .

Int a,b,c;

4 Byte 4 Byte 4 Byte

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**C Arrays [2-D]**

A 2D array is a array of array it is also known as a matrix (a table of rows and columns).

1-D array:- it contain memory physically and logically in linear way.

Arr [7]= {20,40,60,80,90,35,55};

Array

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 20 | 40 | 60 | 80 | 90 | 35 | 55 |

It contain 6 variable and value linearly .

2-D array:- it contain memory physically in linear way but 2-d array contain logically in the form of rows and columns .

Int arr[2][3]

Array

0 (index) 1(index)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 1 2   |  |  |  | | --- | --- | --- | |  |  |  | | 0 1 2   |  |  |  | | --- | --- | --- | |  |  |  | |

We can say that 2-D array is a array of array.

We call 2-d array like.

Array(0,0) , Array(0,0) , Array(0,0) , Array(0,0) , Array(0,0) , Array(0,0) , Array(0,0),

We can draw a 2-d array like that

Int Array[2][3]; (2 is row and 3 is coloum)

Array

0

1 1(index)

0 1 2

|  |  |  |
| --- | --- | --- |
|  |  |  |

0 1 2

|  |  |  |
| --- | --- | --- |
|  |  |  |

Why and When we use 2-d array.

Real life example

Suppose we have to store 5 student of 5 different classes in that case we use 2-d array.

Int arr[5][5];

Passing array to function

* Pass by value
* Pass by reference

Array always pass by reference without using pointer. In array we can pass the array value with reference (&).

**Declaration of array**

* Int arr[2][3]; we can
* Int arr[2][3]; we can
* Int arr[2][ ]; we can not
* Int arr[ ][ ]; we can not

**Declaration with initialization of 2-d array**

While Initialize a 2-d array it is necessary to maintain the second (column) dimension , whereas the first dimension (row) is optional.

Int arr[row][column];

* Int arr[2][3]={(0,1), (0,2), (0,3), (1,1), (1,2)}, (1,3)};
* Int arr[2][3]={1,2,3,4,5,6};
* Int arr[ ][ 3]= {12,23,34,46,56,68};

Solve some question

Find the average of given number using array through user input?