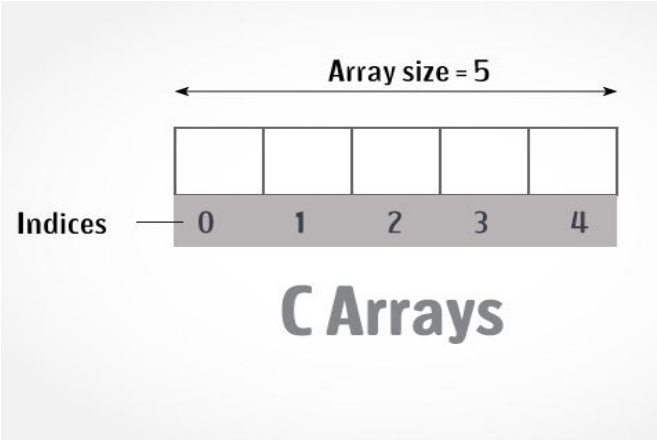
Arrays

An array is a collection of fixed number of values of a single type. For example: if you want to store 100 integers in sequence, you can create an array for it.

```
Syntex:    int data[100];
```



The size and type of arrays cannot be changed after its declaration.

Arrays are of two types:

1. One-dimensional arrays
2. Multidimensional arrays

How to declare arrays?

```
data_type array_name[array_size];
```

example

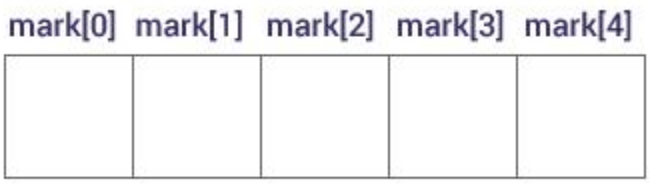
```
float mark[5];
```

Here, we declared an array, *mark*, of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

Elements of an Array and How to access them?

You can access elements of an array by indices.

Suppose you declared an array *mark* as above. The first element is *mark[0]*, second element is *mark[1]* and so on.



Few key notes:

- Arrays have 0 as the first index not 1. In this example, *mark[0]*
- If the size of an array is *n*, to access the last element, `(n-1)` index is used. In this example, *mark[4]*
- Suppose the starting address of `mark[0]` is 2120d. Then, the next address, `a[1]`, will be 2124d, address of `a[2]` will be 2128d and so on. It's because the size of a float is 4 bytes.

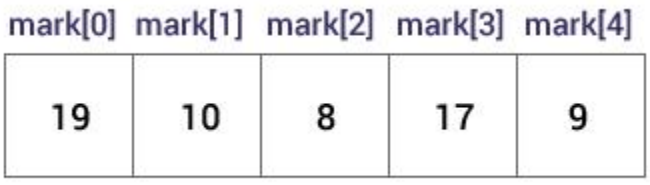
How to initialize an array?

It's possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

Another method to initialize array during declaration:

```
int mark[] = {19, 10, 8, 17, 9};
```



Here,

```
mark[0] is equal to ??
mark[1] is equal to ??
mark[2] is equal to ??
mark[3] is equal to ??
mark[4] is equal to ??
```

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// insert different value to third element
mark[3] = 9;

// take input from the user and insert in third element
scanf("%d", &mark[2]);

// take input from the user and insert in (i+1)th element
scanf("%d", &mark[i]);

// print first element of an array
printf("%d", mark[0]);

// print ith element of an array
printf("%d", mark[i-1]);
```

Important thing to remember when working with C arrays

Suppose you declared an array of 10 elements. Let's say,

```
int testArray[10];
```

You can use the array members from `testArray[0]` to `testArray[9]`.

If you try to access array elements outside of its bound, let's say `testArray[12]`, the compiler may not show any error. However, this may cause unexpected output (undefined behavior).

Multidimensional Arrays

In C programming, you can create array of an array known as multidimensional array. For example,

```
float x[3][4];
```

Here, *x* is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as table with 3 row and each row has 4 column.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Initialization of a two dimensional array

```
// Different ways to initialize two dimensional array

int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};

int c[][3] = {{1, 3, 0}, {-1, 5, 9}};

int c[2][3] = {1, 3, 0, -1, 5, 9};
```

Above code are three different ways to initialize a two dimensional arrays

Accessing Elements of an array

o 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 a 1D array can be calculated by using the following formula:

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

Example :

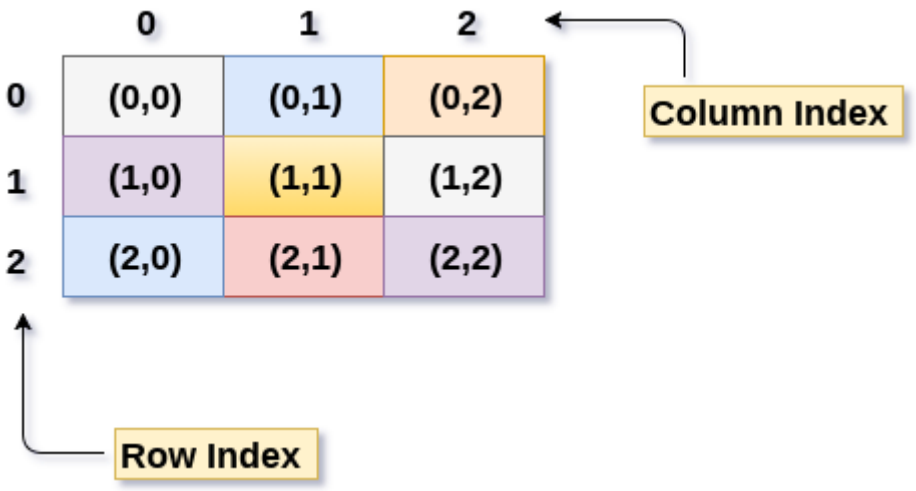
1. In an array, A[-10 ..... +2], Base address (BA) = 999, size of an element = 2 bytes,
2. find the location of A[-1].
3. L(A[-1]) = 999 + [(-1) - (-10)] x 2
4. = 999 + 18
5. = 1017

Mapping 2D array to 1D array

When it comes to map a 2 dimensional array, most of us might think that why this mapping is required. However, 2 D arrays exists from the user point of view. 2D arrays are created to implement a relational database table lookalike data structure, in computer memory, the storage technique for 2D array is similar to that of an one dimensional array.

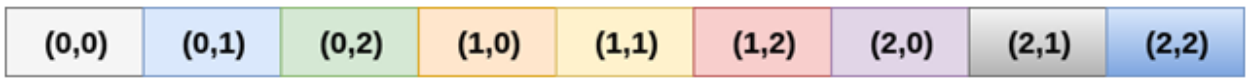
The size of a two dimensional array is equal to the multiplication of number of rows and the number of columns present in the array. We do need to map two dimensional array to the one dimensional array in order to store them in the memory.

A 3 X 3 two dimensional array is shown in the following image. However, this array needs to be mapped to a one dimensional array in order to store it into the memory.

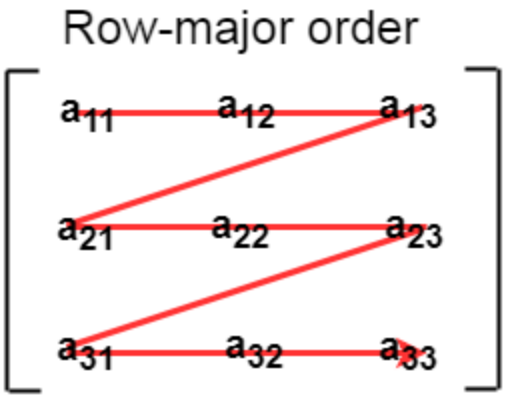


1. Row Major ordering

In row major ordering, all the rows of the 2D array are stored into the memory contiguously. Considering the array shown in the above image, its memory allocation according to row major order is shown as follows.

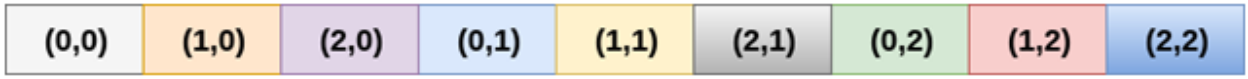


irst, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.

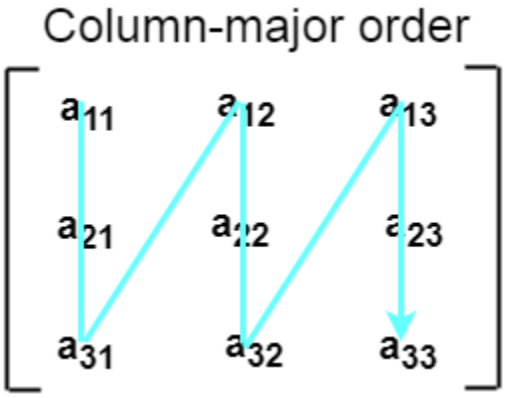


2. Column Major ordering

According to the column major ordering, all the columns of the 2D array are stored into the memory contiguously. The memory allocation of the array which is shown in in the above image is given as follows.



first, the 1st column of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last column of the array.



Calculating the Address of the random element of a 2D array

Due to the fact that, there are two different techniques of storing the two dimensional array into the memory, there are two different formulas to calculate the address of a random element of the 2D array.

By Row Major Order

If array is declared by a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,

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

where, B. A. is the base address or the address of the first element of the array a[0][0] .

Example :

1. a[10...30, 55...75], base address of the array (BA) = 0, size of an element = 4 bytes .
2. Find the location of a[15][68].
3.
4. Address(a[15][68]) = 0 +
5. ((15 ? 10) x (68 ? 55 + 1) + (68 ? 55)) x 4
6.
7. = (5 x 14 + 13) x 4
8. = 83 x 4
9. = 332 answer

By Column major order

If array is declared by a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,

1. Address(a[i][j]) = ((j*m)+i)*Size + BA

where BA is the base address of the array.

Example:

1. A [-5 ... +20][20 ... 70], BA = 1020, Size of element = 8 bytes. Find the location of a[0][30].
2.
3. Address [A[0][30]) = ((30-20) x 24 + 5) x 8 + 1020 = 245 x 8 + 1020 = 2980 bytes

Sum of two matrices using Two dimensional arrays---------???

Three Dimensional Array---------???

Pass arrays to a function

You can pass a single array element of an array or an entire array to a function.

Passing One-dimensional Array to a Function

******Passing a single element of an array to a function is similar to passing variable to a function

- **Passing single element of an array to function**
- **Passing an entire array to a function2*****

2*****To pass an entire array to a function, only the name of the array is passed as an argument. However, notice the use of `[]` after argument name in `float average(float age[])`. This informs the compiler that you are passing a one-dimensional array to the function.

Passing Multi-dimensional Arrays to Function

To pass multidimensional arrays to a function, only the name of the array is passed (similar to one dimensional array).