Arrays in C++

An array is a collection of elements of the same data type stored in contiguous memory locations. It allows multiple values to be stored under a single name and accessed using an index.

- All elements in an array must be of the same data type.
- Array indexing start from 0, enabling fast and direct access to elements.

Example: Iterating an array element using a for loop

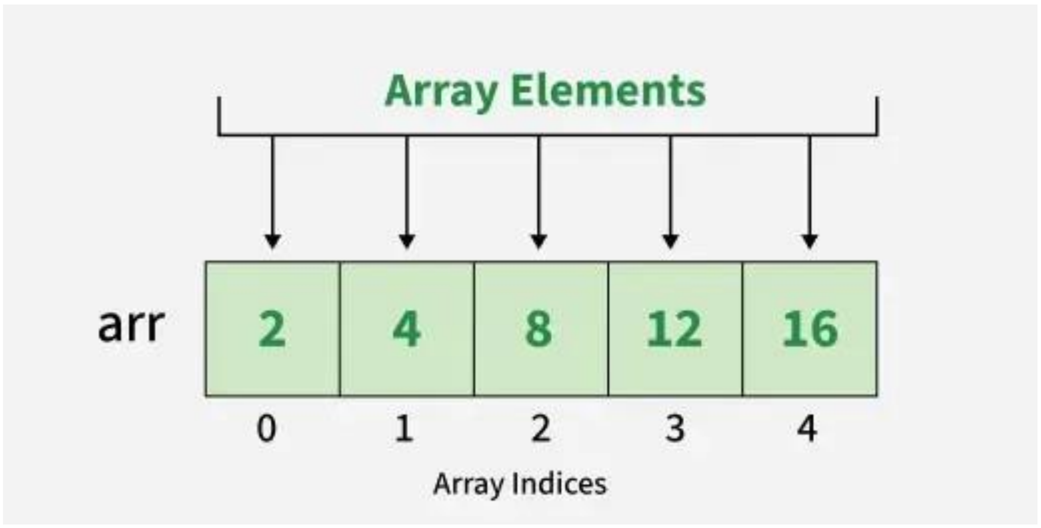
```cpp
#include <iostream>
using namespace std;

int main() {
    // declaring and initializing an array of size 5
    int arr[5] = {2, 4, 8, 12, 16};

    // printing array elements
    for (int i = 0; i < 5; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Output

```
2 4 8 12 16
```

Explanation:

- int arr[5] declares an array of 5 integers.
- The elements are initialized with {2, 4, 8, 12, 16}.
- The for loop is used to iterate over the array and print each element.
- Array indices in C++ start from **0**, so arr[0] refers to the first element, and arr[4] refers to the last one in this case.



Declaration of an Array

We can create/declare an array by simply specifying the data type first and then the name of the array with its size inside [] square brackets(better known as array subscript operator).

Syntax:

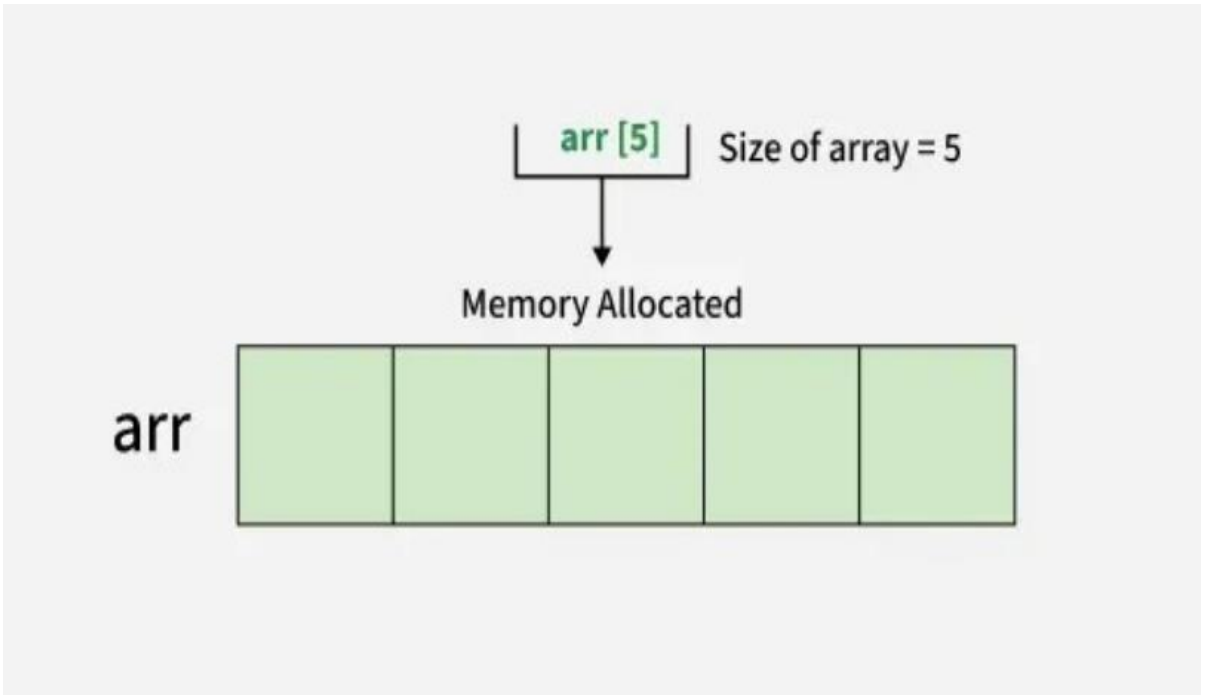
data_type array_name [size]

This statement will create an array with name array_name that can store size elements of given data_type. Once the array is declared, its size cannot be changed.

Example:

int arr[5]

This will create the array with name arr to store 5 integers.

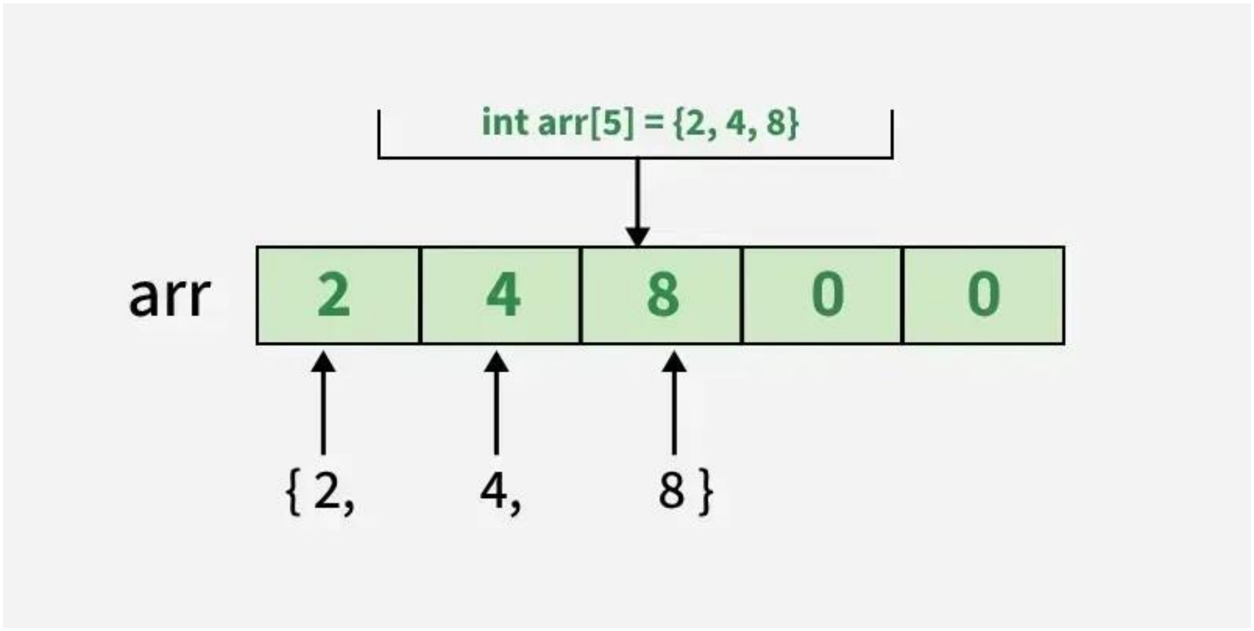


When we declared an array, the elements of array do not contain any valid value.

Initialize the Array

Initialization means assigning initial values to array elements. We can initialize the array with values enclosed in **curly braces '{}'** are assigned to the array.
Example:
int arr[5] = {2, 4, 8, 12, 16};
These values will be assigned sequentially. It means that the first element (index 0) will be 2, second will be 4, and so on. The number of values in the list cannot be more than the size of the array. But they can be less that the size. This is called partial initialization.

int arr[5] = {2, 4, 8};

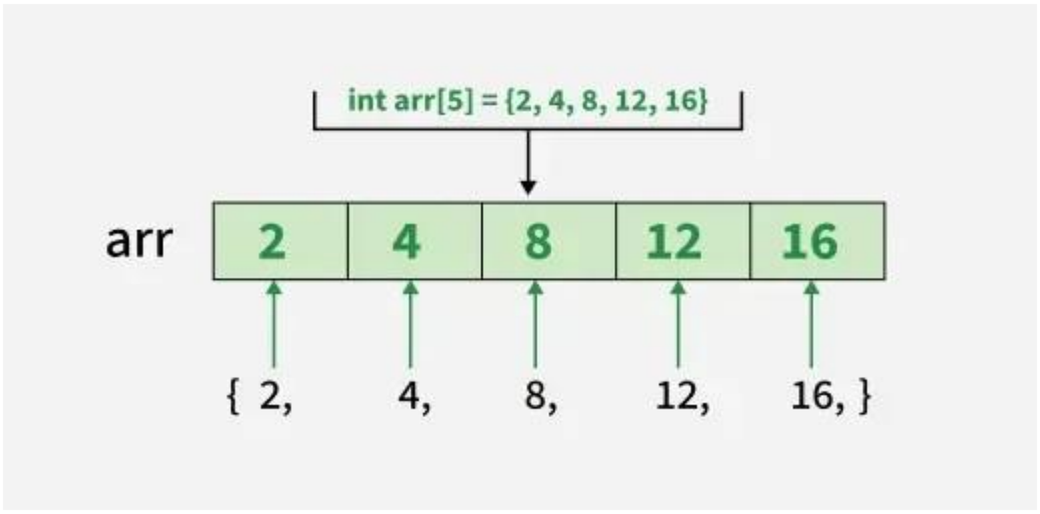


Unused array elements become 0 automatically.

The size of the array can be skipped if the size should be same as the number of values.

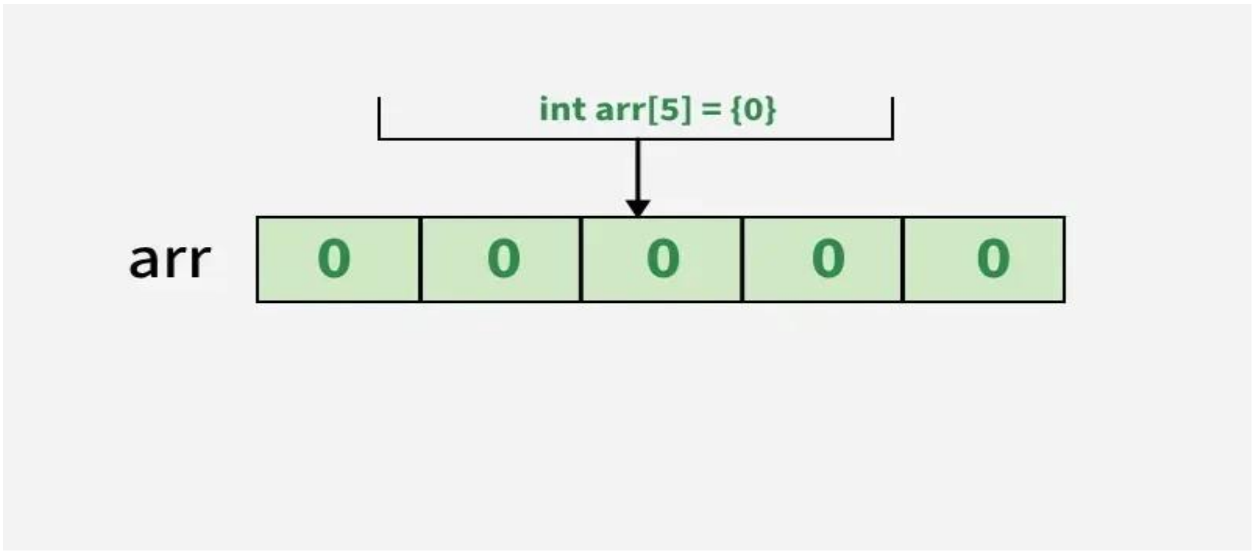
int arr[] = {2, 4, 8, 12, 16}

Array Initialization



Moreover, all the elements can be easily initialized to 0 as shown below:

int arr[5] = {0};
This method only works for 0, but not for any other value.



{0} initializes all array elements to zero.

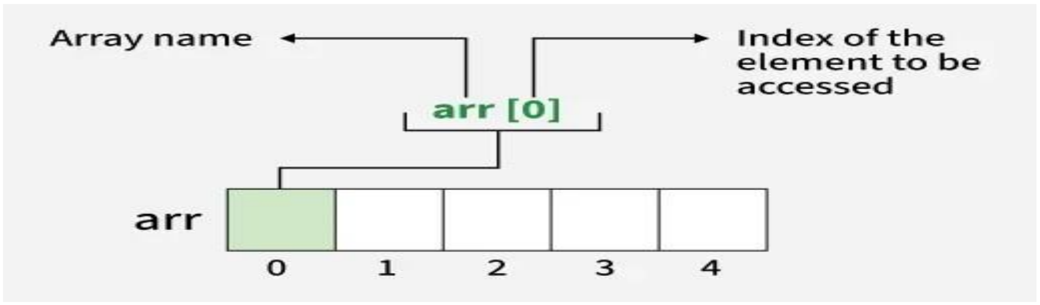
Note: *The value assigned should be of the same type of the array elements specified in the declaration*

Operations on Array Elements

These are the common actions performed on arrays to store, access, modify, and manage data efficiently. let's discuss one by one

1. Access Array Elements

Elements of an array can be accessed by their position (called index) in the sequence. In C++, indexes of an array starts from 0 instead of 1. We just have to pass this index inside the **[] square brackets** with the array name as shown: *array_name[index];*



It is important to note that **index** cannot be negative or greater than size of the array minus 1. **(0 ≤ index ≤ size - 1).** Also, it can also be any expression that results in valid index value.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int arr[] = {2, 4, 8, 12, 16};

    // Accessing fourth element
    cout << arr[3] << " ";

    // Accessing first element
    cout << arr[0];

    return 0;
}
```

Output

```
12 2
```

2. Update Array Elements

To change the element at a particular index in an array, just use the = assignment operator with new value as right hand expression while accessing the array element.

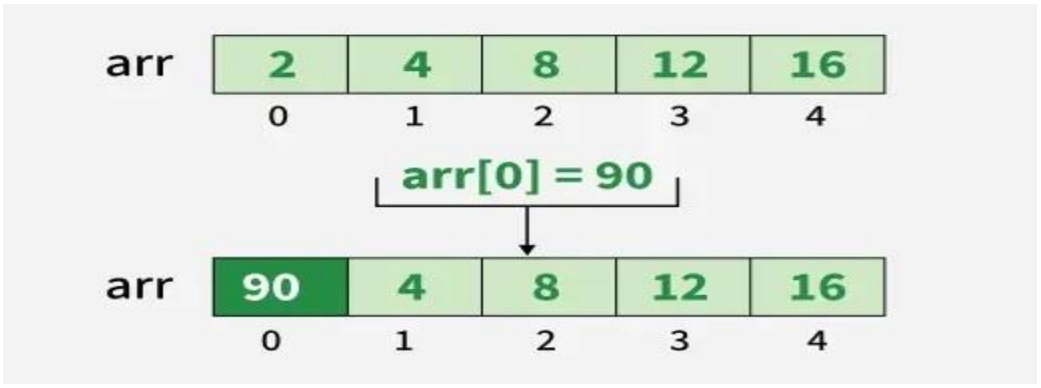
array_name[index] = value;

Example:

```
#include <iostream>
using namespace std;

int main() {
    int arr[] = {2, 4, 8, 12, 16};

    // Updating first element
    arr[0] = 90;
    cout << arr[0] << endl;

    return 0;
}
```

Output

```
90
```



Updating Array Element

3. Traverse Array

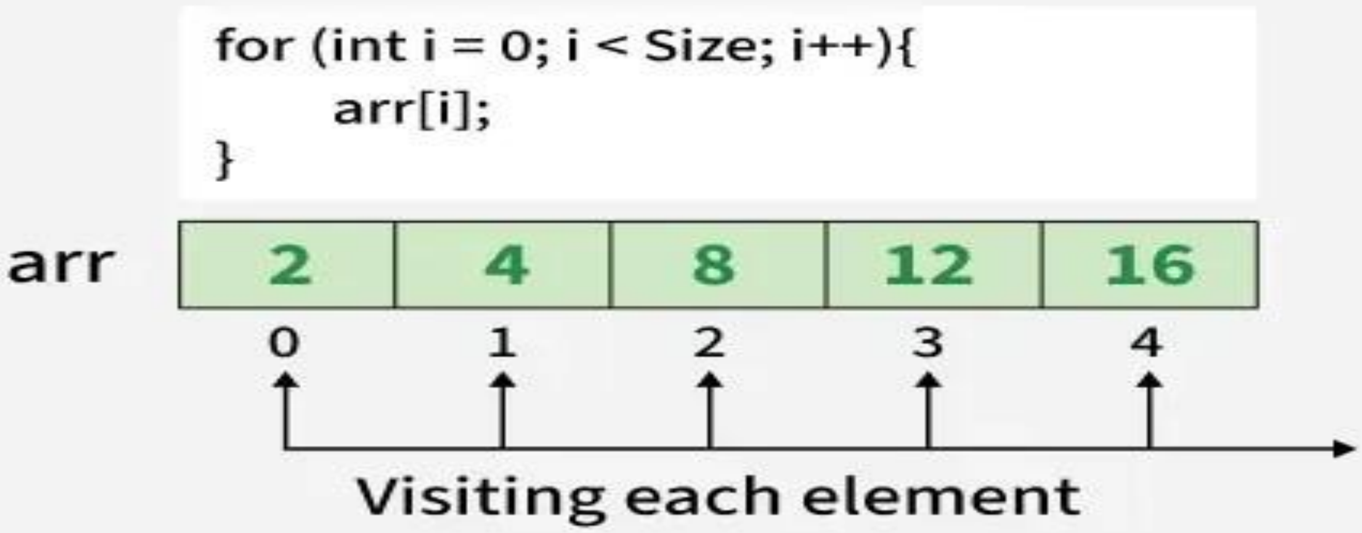
Traversing means visiting each element one by one. The advantage of array is that it can be easily traversed by using a loop with loop variable that runs from 0 to size - 1. We use this loop variable as index of the array and access each element one by one sequentially.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {2, 4, 8, 12, 16};

    // Traversing and printing arr
    for (int i = 0; i < 5; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output

```
2 4 8 12 16
```



4. Size of Array

The size of the array refers to the number of elements that can be stored in the array. The array does not contain the information about its size but we can extract the size using sizeof() operator.

```
#include <iostream>
using namespace std;

int main() {
    char arr[] = {'a', 'b', 'c', 'd', 'f'};

    // Size of one element of an array
    cout << "Size of arr[0]: " << sizeof(arr[0])
    << endl;

    // Size of  'arr'
    cout << "Size of arr: " << sizeof(arr) << endl;

    // Length of an array
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Length of an array: " << n << endl;

    return 0;
}
```

Output

```
Size of arr[0]: 1

Size of arr: 5

Length of an array: 5
```

Arrays and Pointers

In C++, arrays and pointersare closely related to each other. The array name can be treated as a constant pointer that stored the memory address of the first element of the array.

```
#include <iostream>
using namespace std;

int main() {
    int arr[5];

    // Printing array name
    cout << arr << endl;

    // Printing address of first element
    cout << &arr[0];

    return 0;
}
```

Output

```
0x7ffd57920530

0x7ffd57920530
```

Internally, arrays operators are performed using pointer arithmetic. So, we can do almost any array operation using by using pointer to the first element.For example, we can access all the elements of an array using pointer to the first element.

```
#include <iostream>
using namespace std;

int main() {
    int arr[] = {2, 4, 8, 12, 16};
    // Define a pointer to first element
    int* ptr = arr;

    for (int i = 0; i < 5; i++)
        cout << *(ptr + i) << " ";

    return 0;
}
```

Output

```
2 4 8 12 16
```

Pass Array to Function

Arrays are passed to functions using pointers, as the array name decays to a pointer to the first element. So, we also need to pass the size of the array to the function.

```
#include <iostream>
using namespace std;

// Function that takes array as argument
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

int main() {
    int arr[] = {2, 4, 8, 12, 16};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Passing array
    printArray(arr, n);
    return 0;
}
```

Output

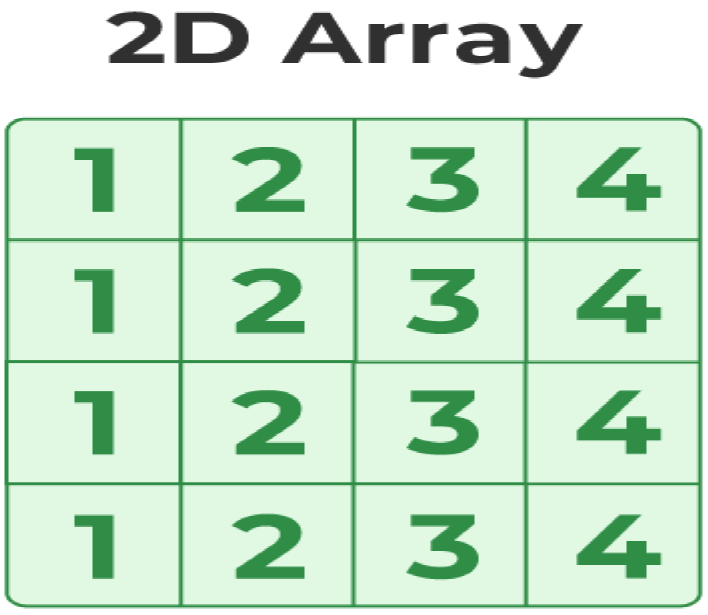
```
2 4 8 12 16
```

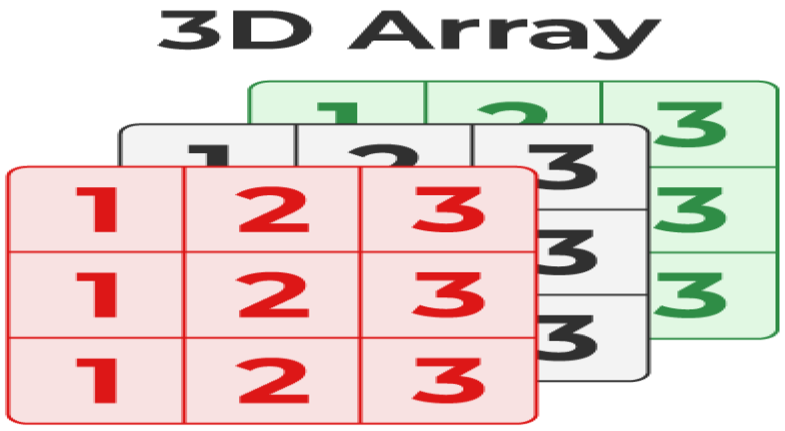
Multidimensional Arrays

In the above examples, we saw 1D (one dimensional) array. This array's size can only increase in a single direction (called dimension). C++ provides the

feature to have as many dimensions as desired for an array. Arrays declared with more than one dimension are called multidimensional arrays.
Syntax:
data_type array_name [size1][size2]...
where **size1, size2 ...** are the sizes of each dimension.
The complexity of the array operations increases exponentially with increase in dimensions.
Some commonly used multidimensional arrays are:

- **Two-Dimensional Array**: It is an array that has exactly two dimensions. It can be visualized in the form of rows and columns organized in a two-dimensional plane.
- **Three-Dimensional Array**: A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.





Properties of Arrays

- An array is a collection of data of the same data type, stored at a contiguous memory location.
- Indexing of an array starts from **0.** It means the first element is stored at the 0th index, the second at 1st, and so on.
- Elements of an array can be accessed using their indices.
- Once an array is declared its size remains constant throughout the program.
- An array can have multiple dimensions.
- The size of the array in bytes can be determined by the sizeof operator using which we can also find the number of elements in the array.
- We can find the size of the type of elements stored in an array by subtracting