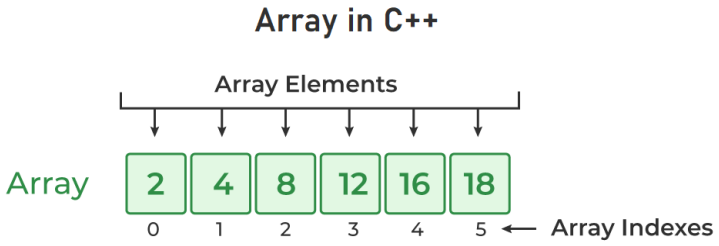
Arrays
In C++, an array is a data structure that is used to store multiple values of similar data types in a contiguous memory location.

For example, if we have to store the marks of 4 or 5 students then we can easily store them by creating 5 different variables but what if we want to store marks of 100 students or say 500 students then it becomes very challenging to create that numbers of variable and manage them. Now, arrays come into the picture that can do it easily by just creating an array of the required size.

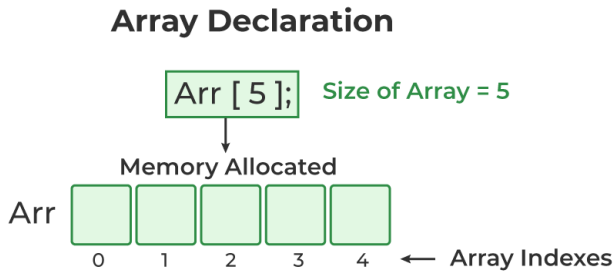


Properties of Arrays in C++

- 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 number of elements in an array can be determined using the sizeof operator.
- We can find the size of the type of elements stored in an array by subtracting adjacent addresses.

Array Declaration in C++
In C++, we can declare an array by simply specifying the data type first and then the name of an array with its size.
data_type array_name[Size_of_array];
Example: int arr[5];

Here,
int: It is the type of data to be stored in the array. We can also use other data types such as char, float, and double.
arr: It is the name of the array.
5: It is the size of the array which means only 5 elements can be stored in the array.



Initialization of Array in C++
In C++, we can initialize an array in many ways but we will discuss some most common ways to initialize an array. We can initialize an array at the time of declaration or after declaration.

1. Initialize Array with Values in C++
We have initialized the array with values. The values enclosed in curly braces '{}' are assigned to the array. Here, 1 is stored in arr[0], 2 in arr[1], and so on. Here the size of the array is 5.
int arr[5] = {1, 2, 3, 4, 5};

2. Initialize Array with Values and without Size in C++
We have initialized the array with values but we have not declared the length of the array, therefore, the length of an array is equal to the number of elements inside curly braces.
int arr[] = {1, 2, 3, 4, 5};

3. Initialize Array after Declaration (Using Loops)
We have initialized the array using a loop after declaring the array. This method is generally used when we want to take input from the user or we cant to assign elements one by one to each index of the array. We can modify the loop conditions or change the initialization values according to requirements.

```
for (int i = 0; i < N; i++) {
    arr[i] = value;
}
```

4. Initialize an array partially in C++
Here, we have declared an array 'partialArray' with size '5' and with values '1' and '2' only. So, these values are stored at the first two indices, and at the rest of the indices '0' is stored.
int partialArray[5] = {1, 2};

5. Initialize the array with zero in C++
We can initialize the array with all elements as '0' by specifying '0' inside the curly braces. This will happen in case of zero only if we try to initialize the array with a different value say '2' using this method then '2' is stored at the 0th index only.
int zero_array[5] = {0};

Accessing an Element of an Array in C++

Elements of an array can be accessed by specifying the name of the array, then the index of the element enclosed in the array subscript operator []. For example, arr[i].

Example 1:

```
//  C++ Program to Illustrate How to Access Array Elements
#include <iostream>
using namespace std;
int main()
{
    int arr[3];
    // Inserting elements in an array
    arr[0] = 10;
    arr[1] = 20;
    arr[2] = 30;

    // Accessing and printing elements of the array
    cout << "arr[0]: " << arr[0] << endl;
    cout << "arr[1]: " << arr[1] << endl;
    cout << "arr[2]: " << arr[2] << endl;
    return 0;
}
```

Output

```
arr[0]: 10
arr[1]: 20
arr[2]: 30
```

Update Array Element

To update an element in an array, we can use the index which we want to update enclosed within the array subscript operator and assign the new value.

```
arr[i] = new_value;
```

Traverse an Array in C++

We can traverse over the array with the help of a loop using indexing in C++. First, we have initialized an array ‘table_of_two’ with a multiple of 2. After that, we run a for loop from 0 to 9 because in an array indexing starts from zero. Therefore, using the indices we print all values stored in an array.

Example 2:

```
// C++ Program to Illustrate How to Traverse an Array
#include <iostream>
using namespace std;
int main()
{
    // Initialize the array
    int table_of_two[10]
```

```
        = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
    // Traverse the array using for loop
    for (int i = 0; i < 10; i++) {
        // Print the array elements using indexing
        cout << table_of_two[i] << " ";
    }
    return 0;
}
```

Output

```
2 4 6 8 10 12 14 16 18 20
```

Size of an Array in C++

In C++, we do not have the length function as in Java to find array size but we can calculate the size of an array using sizeof() operator trick. First, we find the size occupied by the whole array in the memory and then divide it by the size of the type of element stored in the array. This will give us the number of elements stored in the array.

```
data_type size = sizeof(Array_name) / sizeof(Array_name[index]);
```

Example 3:

```
// C++ Program to Illustrate How to Find the Size of an Array
#include <iostream>
using namespace std;
int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    // Size of one element of an array
    cout << "Size of arr[0]: " << sizeof(arr[0]) << endl;
    // Size of array '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]: 4
Size of arr: 20
Length of an array: 5
```

Relation between Arrays and Pointers in C++

In C++, arrays and pointers are closely related to each other. The array name is treated as a pointer that stored the memory address of the first element of the array. As we have discussed

earlier, In array elements are stored at contiguous memory locations that's why we can access all the elements of an array using the array name.

Example 4: Illustrating the Relationship between Array and Pointers

```
//Program to Illustrate that Array Name is a Pointer that Points to First Element of the Array
#include <iostream>
using namespace std;
int main()
{
    // Defining an array
    int arr[] = { 1, 2, 3, 4 };
    // Define a pointer
    int* ptr = arr;
    // Printing address of the arrary using array name
    cout << "Memory address of arr: " << &arr << endl;
    // Printing address of the array using ptr
    cout << "Memory address of arr: " << ptr << endl;
   return 0;
}
```

Output

```
Memory address of arr: 0x7fff2f2cabb0
Memory address of arr: 0x7fff2f2cabb0
```

Explanation:

In the above code, we first define an array "arr" and then declare a pointer "ptr" and assign the array "arr" to it. We are able to assign arr to ptr because arr is also a pointer. After that, we print the memory address of arr using reference operator (&) and also print the address stored in pointer ptr and we can see arr and ptr, both stores the same memory address.

Example 5: Printing Array Elements without Indexing in C++

We generally access and print the array elements using indexing. For example to access the first element we use array_name[0]. We have discussed above that the array name is a pointer that stored the address of the first element and array elements are stored at contiguous locations. Now, we are going to access the elements of an array using the array name only.

```
// C++ Program to Print Array Elements without Indexing
#include <iostream>
using namespace std;

int main()
{

    // Define an array
```

```
    int arr[] = { 11, 22, 33, 44 };

    // Print elements of an array
    cout << "first element: " << *arr << endl;
    cout << "Second element: " << *(arr + 1) << endl;
    cout << "Third element: " << *(arr + 2) << endl;
    cout << "fourth element: " << *(arr + 3) << endl;

    return 0;
}
```

Output

```
first element: 11
Second element: 22
Third element: 33
fourth element: 44
```

Explanation

In the above code, we first declared an array “arr” with four elements. After that, we are printing the array elements. Let’s discuss how we do it. We discussed that the array name is a pointer that stores the address of the first element of an array so, to print the first element we have dereferenced that pointer (*arr) using dereferencing operator (*) which prints the data stored at that address.

To print the second element of an array we first add 1 to arr which is equivalent to (address of arr + size_of_one_element *1) that takes the pointer to the address just after the first one and after that, we dereference that pointer to print the second element. Similarly, we print rest of the elements of an array without using indexing.

Passing Array to Function in C++

To use arrays efficiently we should know how to pass arrays to function. We can pass arrays to functions as an argument same as we pass variables to functions but we know that the array name is treated as a pointer using this concept we can pass the array to functions as an argument and then access all elements of that array using pointer.

So ultimately, arrays are always passed as pointers to the function. Let’s see 3 ways to pass an array to a function that are majorly used.

1. Passing Array as a Pointer

In this method, we simply pass the array name in function call which means we pass the address to the first element of the array. In this method, we can modify the array elements within the function.

Syntax

```
return_type function_name ( data_type *array_name ) {
    // set of statements
}
```

2. Passing Array as an Unsized Array

In this method, the function accepts the array using a simple array declaration with no size as an argument.

Syntax

```
return_type function_name ( data_type array_name[] ) {
    // set of statements
}
```

3. Passing Array as a Sized Array

In this method, the function accepts the array using a simple array declaration with size as an argument. We use this method by sizing an array just to indicate the size of an array.

Syntax

```
return_type function_name(data_type array_name[size_of_array]){
    // set of statements
}
```

Note: Array will be treated as a pointer in the passed function no matter what method we use. As the array are passed as pointers, they will loose the information about its size leading to a phenomenon named as Array Decay.

Example: Illustrating Different Ways to Pass Arrays to a Function

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

// passing array as a sized array argument
void printArraySized(int arr[3], int n)
{
   cout << "Array as Sized Array Argument: ";
   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
}

// passing array as an unsized array argument
void printArrayUnsized(int arr[], int n)
{
```

```
    cout << "Array as Unsized Array Argument: ";
    for (int i = 0; i < n; i++) {
        cout << *(arr + i) << " ";
    }
    cout << endl;
}

// Passing array as a pointer argument
void printArrayPointer(int* ptr, int n)
{
    // Print array elements using pointer ptr
    // that store the address of array passed
    cout << "Array as Pointer Argument: ";
    for (int i = 0; i < n; i++) {
        cout << ptr[i] << " ";
    }
}

// driver code
int main()
{

    int arr[] = { 10, 20, 30 };

    // Call function printArray and pass
    // array and its size to it.
    printArraySized(arr, 3);
    printArrayUnsized(arr, 3);
    printArrayPointer(arr, 3);

    return 0;
}
```

Output

```
Array as Sized Array Argument: 10 20 30
Array as Unsized Array Argument: 10 20 30
Array as Pointer Argument: 10 20 30
```

Multidimensional Arrays in C++

Arrays declared with more than one dimension are called multidimensional arrays. The most widely used multidimensional arrays are 2D arrays and 3D arrays. These arrays are generally represented in the form of rows and columns.

Multidimensional Array Declaration

```
Data_Type Array_Name[Size1][Size2]...[SizeN];
```

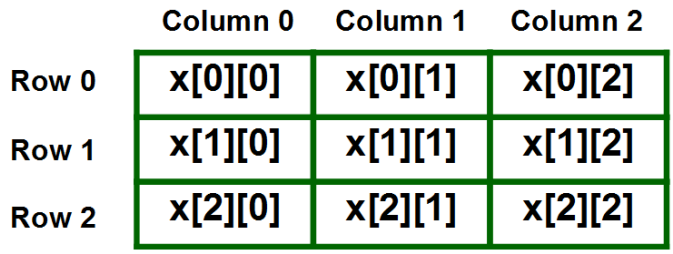
where,

Data_Type: Type of data to be stored in the array.
Array_Name: Name of the array.
Size1, Size2,…, SizeN: Size of each dimension.
Two Dimensional Array in C++
In C++, a two-dimensional array is a grouping of elements arranged in rows and columns. Each element is accessed using two indices: one for the row and one for the column, which makes it easy to visualize as a table or grid.

Syntax of 2D array

data_Type array_name[n][m];
Where,

n: Number of rows.
m: Number of columns.



Example: The C++ Program to Illustrate the Two-Dimensional Array

```
// c++ program to illustrate the two dimensional array
#include <iostream>
using namespace std;

int main()
{
    // Declaring 2D array
    int arr[4][4];

    // Initialize 2D array using loop
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            arr[i][j] = i + j;
        }
    }

    // Printing the element of 2D array
```

```
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output

```
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
```

Explanation

In the above code we have declared a 2D array with 4 rows and 4 columns after that we initialized the array with the value of (i+j) in every iteration of the loop. Then we are printing the 2D array using a nested loop and we can see in the below output that there are 4 rows and 4 columns.