**C++ Arrays**

In this tutorial, we will learn to work with arrays. We will learn to declare, initialize, and access array elements in C++ programming with the help of examples.

In C++, an array is a variable that can store multiple values of the same type. For example,

Suppose a class has 27 students, and we need to store the grades of all of them. Instead of creating 27 separate variables, we can simply create an array:

double grade[27];

Here, grade is an array that can hold a maximum of 27 elements of double type.

In C++, the size and type of arrays cannot be changed after its declaration.

**C++ Array Declaration**

dataType arrayName[arraySize];

For example,

int x[6];

Here,

* int - type of element to be stored
* x - name of the array
* 6 - size of the array

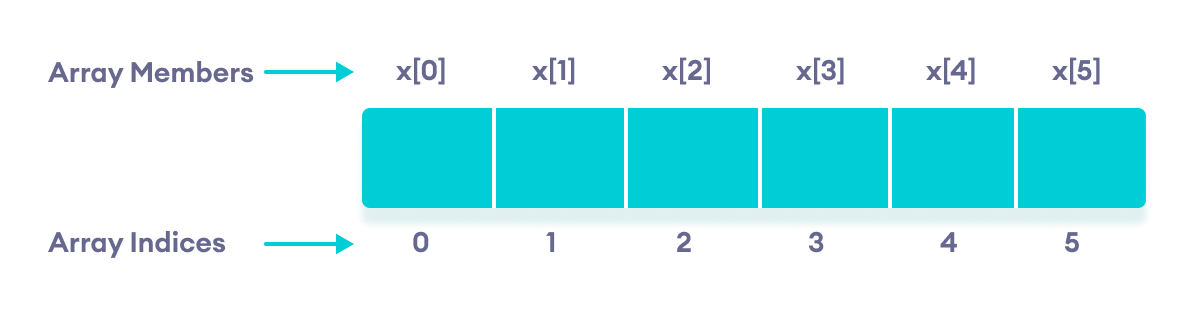
**Access Elements in C++ Array**

In C++, each element in an array is associated with a number. The number is known as an array index. We can access elements of an array by using those indices.

// syntax to access array elements

array[index];

Consider the array x we have seen above.

Elements of an array in C++

**Few Things to Remember:**

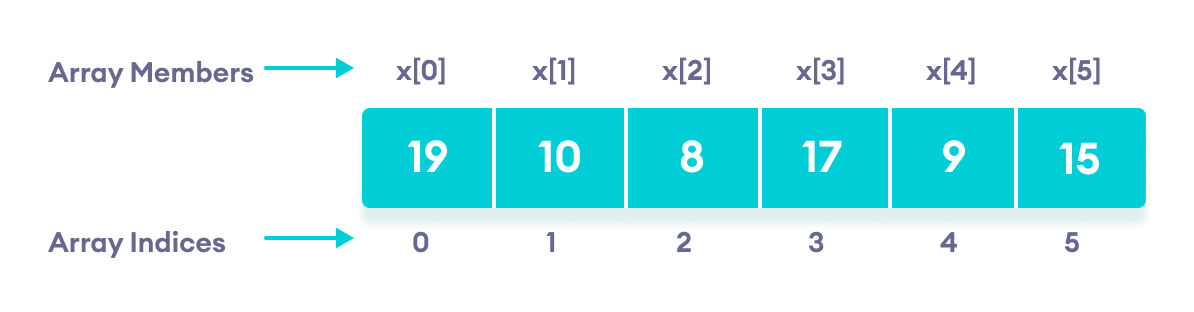
* The array indices start with 0. Meaning x[0] is the first element stored at index 0.
* If the size of an array is n, the last element is stored at index (n-1). In this example, x[5] is the last element.
* Elements of an array have consecutive addresses. For example, suppose the starting address of x[0] is 2120d. Then, the address of the next element x[1] will be 2124d, the address of x[2] will be 2128d and so on.  
    
  Here, the size of each element is increased by 4. This is because the size of int is 4 bytes.

**C++ Array Initialization**

In C++, it's possible to initialize an array during declaration. For example,

// declare and initialize and array

int x[6] = {19, 10, 8, 17, 9, 15};

C++ Array elements and their data

Another method to initialize array during declaration:

// declare and initialize an array

int x[] = {19, 10, 8, 17, 9, 15};

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

**C++ Array With Empty Members**

In C++, if an array has a size n, we can store upto n number of elements in the array. However, what will happen if we store less than n number of elements.

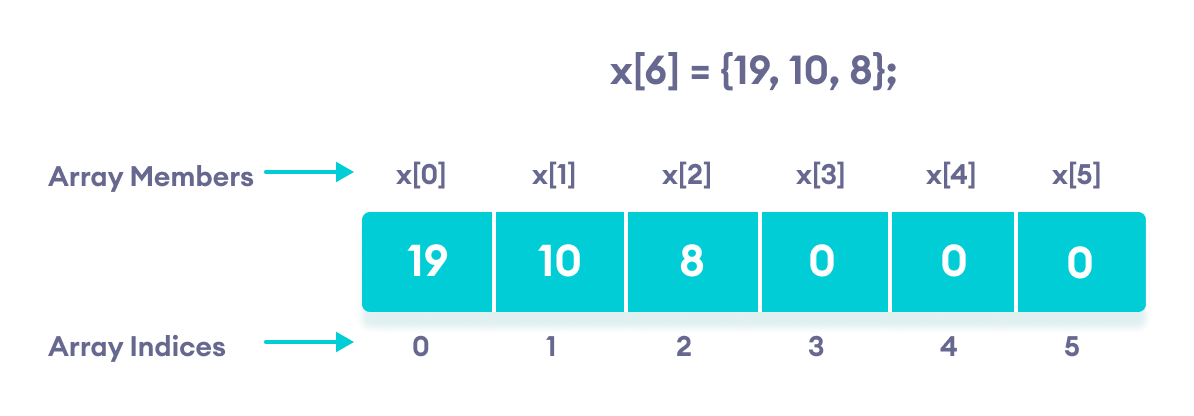
For example,

// store only 3 elements in the array

int x[6] = {19, 10, 8};

Here, the array x has a size of 6. However, we have initialized it with only 3 elements.

In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply 0.

Empty array members are automatically assigned the value 0

**How to insert and print array elements?**

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

// change 4th element to 9

mark[3] = 9;

// take input from the user

// store the value at third position

cin >> mark[2];

// take input from the user

// insert at ith position

cin >> mark[i-1];

// print first element of the array

cout << mark[0];

// print ith element of the array

cout >> mark[i-1];

**Example 1: Displaying Array Elements**

#include <iostream>

using namespace std;

int main() {

int numbers[5] = {7, 5, 6, 12, 35};

cout << "The numbers are: ";

// Printing array elements

// using range based for loop

for (const int &n : numbers) {

cout << n << " ";

}

cout << "\nThe numbers are: ";

// Printing array elements

// using traditional for loop

for (int i = 0; i < 5; ++i) {

cout << numbers[i] << " ";

}

return 0;

}

**Output**

The numbers are: 7 5 6 12 35

The numbers are: 7 5 6 12 35

Here, we have used a for loop to iterate from i = 0 to i = 4. In each iteration, we have printed numbers[i].

We again used a range based for loop to print out the elements of the array. To learn more about this loop, check [C++ Ranged for Loop](https://www.programiz.com/cpp-programming/ranged-for-loop).

**Note:** In our range based loop, we have used the code const int &n instead of int n as the range declaration. However, the const int &n is more preferred because:

1. Using int n simply copies the array elements to the variable n during each iteration. This is not memory-efficient.  
     
   &n, however, uses the memory address of the array elements to access their data without copying them to a new variable. This is memory-efficient.
2. We are simply printing the array elements, not modifying them. Therefore, we use const so as not to accidentally change the values of the array.

**Example 2: Take Inputs from User and Store Them in an Array**

#include <iostream>

using namespace std;

int main() {

int numbers[5];

cout << "Enter 5 numbers: " << endl;

// store input from user to array

for (int i = 0; i < 5; ++i) {

cin >> numbers[i];

}

cout << "The numbers are: ";

// print array elements

for (int n = 0; n < 5; ++n) {

cout << numbers[n] << " ";

}

return 0;

}

**Output**

Enter 5 numbers:

11

12

13

14

15

The numbers are: 11 12 13 14 15

Once again, we have used a for loop to iterate from i = 0 to i = 4. In each iteration, we took an input from the user and stored it in numbers[i].

Then, we used another for loop to print all the array elements.

**Example 3: Display Sum and Average of Array Elements Using for Loop**

#include <iostream>

using namespace std;

int main() {

// initialize an array without specifying size

double numbers[] = {7, 5, 6, 12, 35, 27};

double sum = 0;

double count = 0;

double average;

cout << "The numbers are: ";

// print array elements

// use of range-based for loop

for (const double &n : numbers) {

cout << n << " ";

// calculate the sum

sum += n;

// count the no. of array elements

++count;

}

// print the sum

cout << "\nTheir Sum = " << sum << endl;

// find the average

average = sum / count;

cout << "Their Average = " << average << endl;

return 0;

}

**Output**

The numbers are: 7 5 6 12 35 27

Their Sum = 92

Their Average = 15.3333

In this program:

1. We have initialized a double array named numbers but without specifying its size. We also declared three double variables sum, count, and average.  
     
   Here, sum =0 and count = 0.
2. Then we used a range based for loop to print the array elements. In each iteration of the loop, we add the current array element to sum.
3. We also increase the value of count by 1 in each iteration, so that we can get the size of the array by the end of the for loop.
4. After printing all the elements, we print the sum and the average of all the numbers. The average of the numbers is given by average = sum / count;

**Note:** We used a ranged for loop instead of a normal for loop.

A normal for loop requires us to specify the number of iterations, which is given by the size of the array.

But a ranged for loop does not require such specifications.

**C++ Array Out of Bounds**

If we declare an array of size 10, then the array will contain elements from index 0 to 9.

However, if we try to access the element at index 10 or more than 10, it will result in Undefined Behaviour.

**C++ Multidimensional Arrays**

In this tutorial, we'll learn about multi-dimensional arrays in C++. More specifically, how to declare them, access them, and use them efficiently in our program.

In C++, we can create an [array](https://www.programiz.com/cpp-programming/arrays) of an array, known as a multidimensional array. For example:

int x[3][4];

Here, x is a two-dimensional array. It can hold a maximum of 12 elements.

We can think of this array as a table with 3 rows and each row has 4 columns as shown below.

Elements in two-dimensional array in C++ Programming

Three-dimensional arrays also work in a similar way. For example:

float x[2][4][3];

This array x can hold a maximum of 24 elements.

We can find out the total number of elements in the array simply by multiplying its dimensions:

2 x 4 x 3 = 24

**Multidimensional Array Initialization**

Like a normal array, we can initialize a multidimensional array in more than one way.

**1. Initialization of two-dimensional array**

int test[2][3] = {2, 4, 5, 9, 0, 19};

The above method is not preferred. A better way to initialize this array with the same array elements is given below:

int test[2][3] = { {2, 4, 5}, {9, 0, 19}};

This array has 2 rows and 3 columns, which is why we have two rows of elements with 3 elements each.

Initializing a two-dimensional array in C++

**2. Initialization of three-dimensional array**

int test[2][3][4] = {3, 4, 2, 3, 0, -3, 9, 11, 23, 12, 23,

2, 13, 4, 56, 3, 5, 9, 3, 5, 5, 1, 4, 9};

This is not a good way of initializing a three-dimensional array. A better way to initialize this array is:

int test[2][3][4] = {

{ {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },

{ {13, 4, 56, 3}, {5, 9, 3, 5}, {5, 1, 4, 9} }

};

Notice the dimensions of this three-dimensional array.

The first dimension has the value 2. So, the two elements comprising the first dimension are:

Element 1 = { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} }

Element 2 = { {13, 4, 56, 3}, {5, 9, 3, 5}, {5, 1, 4, 9} }

The second dimension has the value 3. Notice that each of the elements of the first dimension has three elements each:

{3, 4, 2, 3}, {0, -3, 9, 11} and {23, 12, 23, 2} for Element 1.

{13, 4, 56, 3}, {5, 9, 3, 5} and {5, 1, 4, 9} for Element 2.

Finally, there are four int numbers inside each of the elements of the second dimension:

{3, 4, 2, 3}

{0, -3, 9, 11}

... .. ...

... .. ...

**Example 1: Two Dimensional Array**

// C++ Program to display all elements

// of an initialised two dimensional array

#include <iostream>

using namespace std;

int main() {

int test[3][2] = {{2, -5},

{4, 0},

{9, 1}};

// use of nested for loop

// access rows of the array

for (int i = 0; i < 3; ++i) {

// access columns of the array

for (int j = 0; j < 2; ++j) {

cout << "test[" << i << "][" << j << "] = " << test[i][j] << endl;

}

}

return 0;

}

**Output**

test[0][0] = 2

test[0][1] = -5

test[1][0] = 4

test[1][1] = 0

test[2][0] = 9

test[2][1] = 1

In the above example, we have initialized a two-dimensional int array named test that has 3 "rows" and 2 "columns".

Here, we have used the nested for loop to display the array elements.

* the outer loop from i == 0 to i == 2 access the rows of the array
* the inner loop from j == 0 to j == 1 access the columns of the array

Finally, we print the array elements in each iteration.

**Example 2: Taking Input for Two Dimensional Array**

#include <iostream>

using namespace std;

int main() {

int numbers[2][3];

cout << "Enter 6 numbers: " << endl;

// Storing user input in the array

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 3; ++j) {

cin >> numbers[i][j];

}

}

cout << "The numbers are: " << endl;

// Printing array elements

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 3; ++j) {

cout << "numbers[" << i << "][" << j << "]: " << numbers[i][j] << endl;

}

}

return 0;

}

**Output**

Enter 6 numbers:

1

2

3

4

5

6

The numbers are:

numbers[0][0]: 1

numbers[0][1]: 2

numbers[0][2]: 3

numbers[1][0]: 4

numbers[1][1]: 5

numbers[1][2]: 6

Here, we have used a nested for loop to take the input of the 2d array. Once all the input has been taken, we have used another nested for loop to print the array members.

**Example 3: Three Dimensional Array**

// C++ Program to Store value entered by user in

// three dimensional array and display it.

#include <iostream>

using namespace std;

int main() {

// This array can store upto 12 elements (2x3x2)

int test[2][3][2] = {

{

{1, 2},

{3, 4},

{5, 6}

},

{

{7, 8},

{9, 10},

{11, 12}

}

};

// Displaying the values with proper index.

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 3; ++j) {

for (int k = 0; k < 2; ++k) {

cout << "test[" << i << "][" << j << "][" << k << "] = " << test[i][j][k] << endl;

}

}

}

return 0;

}

**Output**

test[0][0][0] = 1

test[0][0][1] = 2

test[0][1][0] = 3

test[0][1][1] = 4

test[0][2][0] = 5

test[0][2][1] = 6

test[1][0][0] = 7

test[1][0][1] = 8

test[1][1][0] = 9

test[1][1][1] = 10

test[1][2][0] = 11

test[1][2][1] = 12

The basic concept of printing elements of a 3d array is similar to that of a 2d array.

However, since we are manipulating 3 dimensions, we use a nested for loop with 3 total loops instead of just 2:

* the outer loop from i == 0 to i == 1 accesses the first dimension of the array
* the middle loop from j == 0 to j == 2 accesses the second dimension of the array
* the innermost loop from k == 0 to k == 1 accesses the third dimension of the array

As we can see, the complexity of the array increases exponentially with the increase in dimensions.

**Passing Array to a Function in C++ Programming**

In this tutorial, we will learn how to pass a single-dimensional and multidimensional array as a function parameter in C++ with the help of examples.

In C++, we can pass arrays as an argument to a function. And, also we can return arrays from a function.

Before you learn about passing arrays as a function argument, make sure you know about [C++ Arrays](https://www.programiz.com/cpp-programming/arrays) and [C++ Functions](https://www.programiz.com/cpp-programming/function).

**Syntax for Passing Arrays as Function Parameters**

The syntax for passing an array to a function is:

returnType functionName(dataType arrayName[arraySize]) {

// code

}

Let's see an example,

int total(int marks[5]) {

// code

}

Here, we have passed an int type array named marks to the function total(). The size of the array is 5.

**Example 1: Passing One-dimensional Array to a Function**

// C++ Program to display marks of 5 students

#include <iostream>

using namespace std;

// declare function to display marks

// take a 1d array as parameter

void display(int m[5]) {

cout << "Displaying marks: " << endl;

// display array elements

for (int i = 0; i < 5; ++i) {

cout << "Student " << i + 1 << ": " << m[i] << endl;

}

}

int main() {

// declare and initialize an array

int marks[5] = {88, 76, 90, 61, 69};

// call display function

// pass array as argument

display(marks);

return 0;

}

**Output**

Displaying marks:

Student 1: 88

Student 2: 76

Student 3: 90

Student 4: 61

Student 5: 69

Here,

1. When we call a function by passing an array as the argument, only the name of the array is used.

display(marks);

Here, the argument marks represent the memory address of the first element of array marks[5].

1. However, notice the parameter of the display() function.

void display(int m[5])

Here, we use the full declaration of the array in the function parameter, including the square braces [].

1. The function parameter int m[5] converts to int\* m;. This points to the same address pointed by the array marks. This means that when we manipulate m[5] in the function body, we are actually manipulating the original array marks.  
     
   C++ handles passing an array to a function in this way to save memory and time.

**Passing Multidimensional Array to a Function**

We can also pass [Multidimensional arrays](https://www.programiz.com/cpp-programming/multidimensional-arrays) as an argument to the function. For example,

**Example 2: Passing Multidimensional Array to a Function**

// C++ Program to display the elements of two

// dimensional array by passing it to a function

#include <iostream>

using namespace std;

// define a function

// pass a 2d array as a parameter

void display(int n[][2]) {

cout << "Displaying Values: " << endl;

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 2; ++j) {

cout << "num[" << i << "][" << j << "]: " << n[i][j] << endl;

}

}

}

int main() {

// initialize 2d array

int num[3][2] = {

{3, 4},

{9, 5},

{7, 1}

};

// call the function

// pass a 2d array as an argument

display(num);

return 0;

}

**Output**

Displaying Values:

num[0][0]: 3

num[0][1]: 4

num[1][0]: 9

num[1][1]: 5

num[2][0]: 7

num[2][1]: 1

In the above program, we have defined a function named display(). The function takes a two dimensional array, int n[][2] as its argument and prints the elements of the array.

While calling the function, we only pass the name of the two dimensional array as the function argument display(num).

**Note**: It is not mandatory to specify the number of rows in the array. However, the number of columns should always be specified. This is why we have used int n[][2].

We can also pass arrays with more than 2 dimensions as a function argument.

**C++ Returning an Array From a Function**

We can also return an array from the function. However, the actual array is not returned. Instead the address of the first element of the array is returned with the help of [pointers](https://www.programiz.com/cpp-programming/pointers).

We will learn about returning arrays from a function in the coming tutorials.

**C++ Strings**

In this article, you'll learn to handle strings in C. You'll learn to declare them, initialize them and use them for various input/output operations.

String is a collection of characters. There are two types of strings commonly used in C++ programming language:

* Strings that are objects of string class (The Standard C++ Library string class)
* C-strings (C-style Strings)

**C-strings**

In C programming, the collection of characters is stored in the form of arrays, this is also supported in C++ programming. Hence it's called C-strings.

C-strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

**How to define a C-string?**

char str[] = "C++";

In the above code, str is a string and it holds 4 characters.

Although, "C++" has 3 character, the null character \0 is added to the end of the string automatically.

**Alternative ways of defining a string**

char str[4] = "C++";

char str[] = {'C','+','+','\0'};

char str[4] = {'C','+','+','\0'};

Like arrays, it is not necessary to use all the space allocated for the string. For example:

char str[100] = "C++";

**Example 1: C++ String to read a word**

**C++ program to display a string entered by user.**

#include <iostream>

using namespace std;

int main()

{

char str[100];

cout << "Enter a string: ";

cin >> str;

cout << "You entered: " << str << endl;

cout << "\nEnter another string: ";

cin >> str;

cout << "You entered: "<<str<<endl;

return 0;

}

**Output**

Enter a string: C++

You entered: C++

Enter another string: Programming is fun.

You entered: Programming

Notice that, in the second example only "Programming" is displayed instead of "Programming is fun".

This is because the extraction operator >> works as scanf() in C and considers a space " " has a terminating character.

**Example 2: C++ String to read a line of text**

**C++ program to read and display an entire line entered by user.**

#include <iostream>

using namespace std;

int main()

{

char str[100];

cout << "Enter a string: ";

cin.get(str, 100);

cout << "You entered: " << str << endl;

return 0;

}

**Output**

Enter a string: Programming is fun.

You entered: Programming is fun.

To read the text containing blank space, cin.get function can be used. This function takes two arguments.

First argument is the name of the string (address of first element of string) and second argument is the maximum size of the array.

In the above program, str is the name of the string and 100 is the maximum size of the array.

**string Object**

In C++, you can also create a string object for holding strings.

Unlike using char arrays, string objects has no fixed length, and can be extended as per your requirement.

**Example 3: C++ string using string data type**

#include <iostream>

using namespace std;

int main()

{

// Declaring a string object

string str;

cout << "Enter a string: ";

getline(cin, str);

cout << "You entered: " << str << endl;

return 0;

}

**Output**

Enter a string: Programming is fun.

You entered: Programming is fun.

In this program, a string str is declared. Then the string is asked from the user.

Instead of using cin>> or cin.get() function, you can get the entered line of text using getline().

getline() function takes the input stream as the first parameter which is cin and str as the location of the line to be stored.

**Passing String to a Function**

Strings are passed to a function in a similar way [arrays are passed to a function](https://www.programiz.com/cpp-programming/passing-arrays-function).

#include <iostream>

using namespace std;

void display(char \*);

void display(string);

int main()

{

string str1;

char str[100];

cout << "Enter a string: ";

getline(cin, str1);

cout << "Enter another string: ";

cin.get(str, 100, '\n');

display(str1);

display(str);

return 0;

}

void display(char s[])

{

cout << "Entered char array is: " << s << endl;

}

void display(string s)

{

cout << "Entered string is: " << s << endl;

}

**Output**

Enter a string: Programming is fun.

Enter another string: Really?

Entered string is: Programming is fun.

Entered char array is: Really?

In the above program, two strings are asked to enter. These are stored in str and str1 respectively, where str is a char array and str1 is a string object.

Then, we have two functions display() that outputs the string onto the string.

The only difference between the two functions is the parameter. The first display() function takes char array as a parameter, while the second takes string as a parameter.

This process is known as function overloading. Learn more about [Function Overloading](https://www.programiz.com/cpp-programming/function-overloading).