Passing Array to a Function in C++ Programming

programiz.com/cpp-programming/passing-arrays-function

Join our newsletter for the latest updates.

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 and C++ Functions.

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]*.
2. 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 `[]`.

3. 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 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.

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