

Make your voice heard. [Take the 2020 Developer Survey now.](#)

# Passing Arrays to Function in C++

Asked 7 years ago · Active 1 month ago · Viewed 145k times



64



36



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

void printarray (int arg[], int length) {
    for (int n = 0; n < length; n++) {
        cout << arg[n] << " ";
        cout << "\n";
    }
}

int main ()
{
    int firstarray[] = {5, 10, 15};
    int secondarray[] = {2, 4, 6, 8, 10};
    printarray(firstarray, 3);
    printarray(secondarray, 5);

    return 0;
}
```

This code works, but I want to understand how is the array being passed.

When a call is made to the `printarray` function from the `main` function, the name of the array is being passed. The name of the array refers to the address of the first element of the array. How does this equate to `int arg[]` ?

C++

edited Jan 13 '13 at 23:04



Seth Carnegie

66.8k · 17 · 159 · 229

asked Jan 13 '13 at 22:49



Jay K

835 · 1 · 8 · 12

Just to be specific, the name of the array refers to the array. It can be converted to a pointer to the first element, which is what happens in most cases. – [Joseph Mansfield](#) Jan 13 '13 at 22:54

I propose making knatten's answer the accepted one. – [wally](#) Nov 22 '19 at 19:25

## 5 Answers



The syntaxes

40



```
int[]
```

and

```
int[X] // Where X is a compile-time positive integer
```

are exactly the same as

```
int*
```

when in a function parameter list (I left out the optional names).

Additionally, an array name decays to a pointer to the first element when passed to a function (and not passed by reference) so both `int firstarray[3]` and `int secondarray[5]` decay to `int* s`.

It also happens that both an array dereference and a pointer dereference with subscript syntax (subscript syntax is `x[y]`) yield an lvalue to the same element when you use the same index.

These three rules combine to make the code legal and work how you expect; it just passes pointers to the function, along with the length of the arrays which you cannot know after the arrays decay to pointers.

edited Dec 26 '19 at 18:25



Baum mit Augen ♦

44.2k ● 12 ● 126 ● 164

answered Jan 13 '13 at 22:49



Seth Carnegie

66.8k ● 17 ● 159 ● 229

I just wanna add this, when you access the position of the array like

```
arg[n]
```

is the same as

```
*(arg + n)
```

than means an offset of n starting from the arg address.

so `arg[0]` will be `*arg`

edited Dec 26 '19 at 17:43



HolyBlackCat

28.8k ● 4 ● 54 ● 86

answered Jan 13 '13 at 22:52



DGomez

1,350 ● 7 ● 22

The question has already been answered, but I thought I'd add an answer with more precise terminology and references to the C++ standard.

Two things are going on here, *array parameters being adjusted to pointer parameters*, and *array arguments being converted to pointer arguments*. These are two quite different mechanisms, the first is an adjustment to the actual type of the parameter, whereas the other is a standard conversion which introduces a temporary pointer to the first element.

**Adjustments to your function declaration:**

[dcl.fct#5:](#)

After determining the type of each parameter, any parameter of type “array of T” (...) is adjusted to be “pointer to T”.

So `int arg[]` is adjusted to be `int* arg`.

### Conversion of your function argument:

[conv.array#1](#)

An lvalue or rvalue of type “array of N T” or “array of unknown bound of T” can be converted to a prvalue of type “pointer to T”. The temporary materialization conversion is applied. The result is a pointer to the first element of the array.

So in `printarray(firstarray, 3);`, the lvalue `firstarray` of type “array of 3 int” is converted to a prvalue (temporary) of type “pointer to int”, pointing to the first element.

answered Mar 10 '18 at 22:29



[knatten](#)

4,694 ● 3 ● 17 ● 24

`firstarray` and `secondarray` are converted to a pointer to int, when passed to `printarray()`.

9

`printarray(int arg[], ...)` is equivalent to `printarray(int *arg, ...)`

However, this is not specific to C++. C has the same rules for passing array names to a function.

answered Jan 13 '13 at 22:51



[Olaf Dietsche](#)

60.6k ● 5 ● 77 ● 159

The simple answer is that arrays are ALWAYS passed by reference and the `int arg[]` simply lets the compiler know to expect an array

0

answered Nov 24 '19 at 4:09



[Bob Warner](#)

79 ● 1 ● 1