# Why C or C++ does not allow passing array by values to function

C and C++ allows passing of structure and objects by value to function, although prevents passing arrays by values, why?

c++ c

asked Mar 22 '09 at 14:17

Kazoom

1,624 ● 8 ● 36 ● 57

## 6 Answers

In C/C++, internally, an array is passed as a pointer to some location, and basically, it *is* passed by value. The thing is, that copied value represents a memory address to the *same location*.

In C++, a `vector<T>` is copied and passed to another function, by the way.

edited Mar 22 '09 at 16:58

answered Mar 22 '09 at 14:22

Mehrdad Afshari

234k ● 54 ● 623 ● 682

Best answer! I think though it should emphasis also that C/C++ does not *prevent* passing the array by value. – hasen Mar 22 '09 at 14:39

3 This answer confuses arrays with pointers. It may sound simple, but it indeed causes confusion i think. It's important to keep the concepts of arrays and pointers apart, to understand their fundamental difference in the first place. – Johannes Schaub - litb Mar 22 '09 at 14:46

@litb: Well, yeah, it's pretty important to know that it's not precisely the same as passing a pointer from the compiler's point of view (e.g. multidimensional arrays), but it's equally important to emphasis that the actual array address is *copied*, not passed by reference. – Mehrdad Afshari Mar 22 '09 at 14:55

An array is indeed more than a pointer, otherwise for char a[1]; sizeof(a) would equal sizeof(char*) instead of being 1. – anon Mar 22 '09 at 15:22

1 @Mehrdad: I know what you're trying to say, but I think your current explanation is very close to wrong. The problem is that it blurs the line -- *all* cases where pointer semantics are used instead of value semantics can be explained as "well, it's really value semantics on the underlying address". – j_random_hacker Mar 23 '09 at 3:59

You can pass an array by value, but you have to first wrap it in a struct or class. Or simply use a type like std::vector.

I think the decision was for the sake of efficiency. One wouldn't want to do this most of the time. It's the same reasoning as why there are no unsigned doubles. There is no associated CPU instruction, so you have to make what's not efficient very hard to do in a language like C++.

As @litb mentioned: "C++1x and boost both have wrapped native arrays into structs providing std::array and boost::array which i would always prefer because it allows passing and returning of arrays within structs"

An array is a pointer to the memory that holds that array and the size. Note it is not the exact same as a pointer to the first element of the array.

Most people think that you have to pass an array as a pointer and specify the size as a separate parameter, but this is not needed. You can pass a reference to the actual array itself while maintaining it's sizeof() status.

```
//Here you need the size because you have reduced
// your array to an int* pointing to the first element.
void test1(int *x, int size)
{
  assert(sizeof(x) == 4);
```

```
}

//This function can take in an array of size 10
void test2(int (&x)[10])
{
  assert(sizeof(x) == 40);
}

//Same as test2 but by pointer
void test3(int (*x)[10])
{
  assert(sizeof(*x) == 40);
  //Note to access elements you need to do: (*x)[i]
}
```

Some people may say that the size of an array is not known. This is not true.

```
int x[10];
assert(sizeof(x) == 40);
```

But what about allocations on the heap? Allocations on the heap do not return an array. They return a pointer to the first element of an array. So new is not type safe. If you do indeed have an array variable, then you will know the size of what it holds.

edited Mar 22 '09 at 16:23

answered Mar 22 '09 at 14:18

Brian R. Bondy

163k ● 68 ● 414 ● 537

1 You see that [10] in the function parameter? That's where the size is compimg from. The reference has nothing to do with it. – anon Mar 22 '09 at 14:35

it's more type safe than the test1 – Brian R. Bondy Mar 22 '09 at 14:37

@Neil Butterworth: The size of an array is part of the array type itself. If you wanted variable size you'd be better off with an std::vector. – Brian R. Bondy Mar 22 '09 at 14:39

1 You said that "people think you need to specify the size". People are right, as your needlessly complex code demonstrates. – anon Mar 22 '09 at 14:46

It has everything to do with it. Had you said void test2(int x[10]); you would pass nothing more than a simple pointer, and the "10" there is completely ignored and useless (even dangerous). Now the reference accepts the parameter by-reference and avoids conversion of the argument to pointer. – Johannes Schaub - litb Mar 22 '09 at 14:49

EDIT: I've left the original answer below, but I believe most of the value is now in the comments. I've made it community wiki, so if anyone involved in the subsequent conversation wants to edit the answer to reflect that information, feel free.

**Original answer**

For one thing, how would it know how much stack to allocate? That's fixed for structures and objects (I believe) but with an array it would depend on how big the array is, which isn't known until execution time. (Even if each caller knew at compile-time, there could be different callers with different array sizes.) You could force a particular array size in the parameter declaration, but that seems a bit strange.

Beyond that, as Brian says there's the matter of efficiency.

What would you want to achieve through all of this? Is it a matter of wanting to make sure that the contents of the original array aren't changed?

edited Mar 22 '09 at 20:46

community wiki

2 revs

Jon Skeet

char x[10]; assert(sizeof(x) == 10); – Brian R. Bondy Mar 22 '09 at 14:23

Brian is right. arrays in C++ and C89 always have sizes known at compile time :) but i think your answer still has to do with it. there are many situations where an array decays into a pointer. and making arrays pass to functions would for one dangerous (precisely because of that decay) ... – Johannes Schaub - litb Mar 22 '09 at 14:29

and useless to a certain degree, because we would be limited for passing one size (i.e int[42] - but not int[41]). – Johannes Schaub - litb Mar 22 '09 at 14:30

i believe your answer points out exactly that, just from a point of view which expects to make it work to accept different array sizes but which would not work in C++/C because it would need to accept array sizes that have different sizes. so i +1 anyway :D – Johannes Schaub - litb Mar 22 '09 at 14:33

The size of the array is part of the type itself. If you wanted variable size you'd be better off with an std::vector. – Brian R. Bondy Mar 22 '09 at 14:38

I think that there 3 main reasons why arrays are passed as pointers in C instead of by value. The first 2 are mentioned in other answers:

- efficiency
- because there's no size information for arrays in general (if you include dynamically allocated arrays)

However, I think a third reason is due to:

- the evolution of C from earlier languages like B and BCPL, where arrays were actually implemented as a pointer to the array data

Dennis Ritchie talks about the early evolution of C from languages like BCPL and B and in particular how arrays are implemented and how they were influenced by BCPL and B arrays and how and why they are different (while remaining very similar in expressions because array names decay into pointers in expressions).

- http://plan9.bell-labs.com/cm/cs/who/dmr/chist.html

answered Mar 22 '09 at 17:41

Michael Burr
**205k** ● 25 ● 292 ● 519

I'm not actually aware of **any** languages that support passing **naked** arrays by value. To do so would not be particularly useful and would quickly chomp up the call stack.

**Edit:** To downvoters - if you know better, please let us all know.

edited Mar 22 '09 at 15:17

answered Mar 22 '09 at 14:27

anon

C++ vector<...> objects, when passed by value, are copied. And although this wastes (heap-)memory, it won't "quickly chomp up the call stack". – vog Mar 22 '09 at 14:58

The native array inside a vector is passed by reference - it's a pointer. C++ native arrays are exactly the same as C arrays. – anon Mar 22 '09 at 15:02

MATLAB has value semantics even for arrays. This is one reason why non-trivial matlab programs often use a huge amount of memory, though current versions of the interpreter use copy-on-write to reduce the memory usage. – janneb Mar 22 '09 at 15:04

Copy on write is passing by ref (R does the same). The fact the array may grow later (depending on use) is not the issue. – anon Mar 22 '09 at 15:07

i agree with Neil. i know no language which can pass arrays by value. C# and Java pass them not at all (not even by reference), C++ can pass them by reference only. And C can't pass them at all either, only passing a pointer to their first element. – Johannes Schaub - litb Mar 22 '09 at 15:10

This is one of those "just because" answers. C++ inherited it from C, and had to follow it to keep compatibility. It was done that way in C for efficiency. You would rarely want to make a copy of a large array (remember, think PDP-11 here) on the stack to pass it to a function.

answered Mar 22 '09 at 16:42

Brian Neal
**15.8k** ● 4 ● 35 ● 52