Pass an array to a function by value

Below is a snippet from the book *C Programming Just the FAQs*. Isn't this wrong as Arrays can never be passed by reference?

> **VIII.6: How can you pass an array to a function by value?**
>
> **Answer:** An array can be passed to a function by value by declaring in the called function the array name with square brackets (`[` and `]`) attached to the end. When calling the function, simply pass the address of the array (that is, the array's name) to the called function. For instance, the following program passes the array `x[]` to the function named `byval_func()` by value:
>
> The `int[]` parameter tells the compiler that the `byval_func()` function will take one argument—an array of integers. When the `byval_func()` function is called, you pass the address of the array to `byval_func()` :
>
> ```
> byval_func(x);
> ```
>
> Because the array is being passed by value, an exact copy of the array is made and placed on the stack. The called function then receives this copy of the array and can print it. Because the array passed to `byval_func()` is a copy of the original array, modifying the array within the `byval_func()` function has no effect on the original array.

`c`

edited Jan 23 '11 at 15:02

Tom Medley
11k ● 15 ● 67 ● 148

asked Jan 23 '11 at 14:58

Zuzu
665 ● 4 ● 9 ● 16

This is true for structs - you can pass them by value (over the stack) or by reference but not for arrays. – sinelaw Jan 23 '11 at 15:07

11 This is in a book about C, really? I see it dates from 1995, but this was already plain wrong then. – Jens Gustedt Jan 23 '11 at 16:11

15 If that is what it says in the book, you need to chuck the book away - it contains serious misinformation on a basic issue, and who knows what other misinformation on what other issues. The last quoted paragraph is nonsense - if you modify the array in the called function, you are modifying the array in the calling function too, because arrays are not passed by value in C. – Jonathan Leffler Jan 23 '11 at 16:17

4 Answers

> Because the array is being passed by value, an exact copy of the array is made and placed on the stack.

This is incorrect: **the array itself is not being copied**, only a copy of the pointer to its address is passed to the callee (placed on the stack). (Regardless of whether you declare the parameter as `int[]` or `int*` , it *decays into a pointer*.) **This allows you to modify the contents of the array from within the called function**. Thus, this

> Because the array passed to byval_func() is a copy of the original array, modifying the array within the byval_func() function has no effect on the original array.

is plain wrong (kudos to @Jonathan Leffler for his comment below). However, reassigning the pointer inside the function will not change the pointer to the original array outside the function.

edited Jan 23 '11 at 16:50

answered Jan 23 '11 at 15:03

Péter Török
78.6k ● 14 ● 168 ● 249

I think you're right. – sinelaw Jan 23 '11 at 15:06

He indeed is. In C++ you can of course have array objects passed by value, but in C, an array is just a pointer. – Stefan H Singer Jan 23 '11 at 15:07

3 StefanHållén: You can't pass (or return) arrays by value in either C or C++, you can pass arrays by reference in C++ because C++ has reference types. – Charles Bailey Jan 23 '11 at 15:09

I meant that you could use for instance std::array. Instances of that can be passed and returned by value. – Stefan H Singer Jan 23 '11 at 15:11

2 I would go further; although the sentence you quote is mildly ambiguous and could charitably be construed as correct, the last sentence is simply wrong within the normal meaning of the term 'changing the array'. If the called function writes to `array[0]`, it modifies `array[0]` in the calling function - under the obvious call notation. – Jonathan Leffler Jan 23 '11 at 16:25

Burn that book. If you want a real C FAQ that wasn't written by a beginner programmer, use this one: http://c-faq.com/aryptr/index.html.

Syntax-wise, strictly speaking you *cannot* pass an array by value in C.

```
void func (int* x); /* this is a pointer */

void func (int x[]); /* this is a pointer */

void func (int x[10]); /* this is a pointer */
```

However, for the record there is a dirty trick in C that does allow you to pass an array by value in C. Don't try this at home! Because despite this trick, there is still never a reason to pass an array by value.

```
typedef struct
{
  int my_array[10];
} Array_by_val;

void func (Array_by_val x);
```

edited Jun 10 '13 at 13:14

answered Jan 23 '11 at 20:07

Lundin
30.3k ● 5 ● 39 ● 87

3 Cosmetics for 3rd example means you can do a `sizeof(array)/sizeof(array[0])` in the function – Ulterior Jun 29 '12 at 3:17

@Ulterior Wouldn't that be nice if it worked? But alas you are wrong, the C standard isn't that smart. Try this: `void func (int arr[10]) { printf("No, I am still a pointer, my size is %d.", sizeof(arr)); }`. It will print 4 or 8 on a PC, not 40 as you would expect. – Lundin Dec 11 '12 at 10:01

"Because despite this trick, there is still never a reason to pass an array by value." -- why? – pbs Jan 23 '13 at 10:50

@pbs Lets say you have a 32-bit system and an array of 10 ints. You can pass a pointer, occupy 4 bytes of memory and have the program run fast, or you can use the above dirty trick, occupy 40 bytes of memory and have the program run slow. Furthermore, if you pass an array by value you can't alter it so that the caller is aware of the change. – Lundin Jan 24 '13 at 17:47

1 @Lundin I agree that this is better if it can be done. But what if an algorithm requires different copies of the array at each level of a tree. I have written an algorithm in which a copy of a number of points is required for every node of a tree called, and there is no avoiding the amount of memory used, for otherwise the points would not be distinct, which is required. The only possible drawback is that it might be faster in such cases to allocate all memory in one go at the beginning... Then again if the required data is dynamic this may not be possible in general. – pbs Jan 25 '13 at 13:20

In C and C++ it is NOT possible to pass a complete block of memory by value as a parameter to a function, but we are allowed to pass its address. In practice this has almost the same effect and it is a much faster and more efficient operation.

To be safe, you can pass the array size or put const qualifier before the pointer to make sure the callee won't change it.

answered Jun 13 '12 at 15:11

Rongkai Xu
31 ● 1

6 Structs and unions can be passed by value. – Keith Thompson Jun 10 '13 at 6:35

```
#include<stdio.h>
void  fun(int a[],int n);
int main()
{
    int a[5]={1,2,3,4,5};
    fun(a,5);
}
void  fun(int a[],int n)
{
    int i;
    for(i=0;i<=n-1;i++)
        printf("value=%d\n",a[i]);
}
```

By this method we can pass the array by value, but actually the array is accessing through the its base address which actually copying in the stack.

edited Jun 10 '13 at 6:20

Rob
3,255 ● 10 ● 19 ● 33

answered Jun 10 '13 at 6:04

Varun Chhangani
361 ● 4 ● 12

1 No, that does *not* pass the array by value. The parameter is a pointer. – Keith Thompson Jun 10 '13 at 6:34

And if `fun` changed the value of `a[0]`, that change would be visible in `main`. – Keith Thompson Jun 10 '13 at 14:49

so how we pass the array by value. In this there are simple passing array and array also behave like constant pointer so when you will pass array by value it's simply means you are passing it by pointer. If you have any example please tell me. – Varun Chhangani Jun 11 '13 at 6:03

You cannot directly pass an array as a parameter in C. (You can pass a structure that has an array as a member, but that's not all that useful, since the array then has to have a fixed size known at compile time.) Suggested reading: Section 6 of the comp.lang.c FAQ. – Keith Thompson Jun 11 '13 at 6:36