Passing an array as a function parameter in C++

Asked 15 years, 11 months ago Modified 10 years, 6 months ago Viewed 114k times

24

In C++, arrays cannot be passed simply as parameters. Meaning if I create a function like so:

```
void doSomething(char charArray[])
{
    // if I want the array size
    int size = sizeof(charArray);
    // NO GOOD, will always get 4 (as in 4 bytes in the pointer)
}
```

I have no way of knowing how big the array is, since I have only a pointer to the array.

Which way do I have, without changing the method signature, to get the size of the array and iterate over it's data?

EDIT: just an addition regarding the solution. If the char array, specifically, was initialized like so:

```
char charArray[] = "i am a string";
```

then the `\0` is already appended to the end of the array. In this case the answer (marked as accepted) works out of the box, so to speak.

c++ arrays pointers parameters arguments

Share

Improve this question

Follow

edited May 24, 2009 at 6:22

Shog9

159k ● 36 ● 235 ● 240

asked Jan 14, 2009 at 21:24

Yuval Adam

165k ● 95 ● 315 ● 404

2 Actually there's a fairly big misunderstanding here. The syntax in your example makes no sense: function taking an unknown quantity of data on the stack. The compiler actually ignores your original syntax and gives the function the signature `void doSomething(char* charArray)`. None of the arrayness is preserved if you call it with `char array[10]; doSomething(array);`. See c-faq.com/aryptr/aryptrparam.html – kfsone Jun 2, 2013 at 17:53

12 Answers

Sorted by: Highest score (default)

47

Use templates. This technically doesn't fit your criteria, because it changes the signature, but calling code does not need to be modified.

```
void doSomething(char charArray[], size_t size)
{
   // do stuff here
}

template<size_t N>
inline void doSomething(char (&charArray)[N])
{
    doSomething(charArray, N);
}
```

This technique is used by Microsoft's Secure CRT functions and by STLSoft's array_proxy class template.

Share Improve this answer Follow

answered Jan 14, 2009 at 21:47

Josh Kelley

58.3k • 20 • 162 • 256

27

Without changing the signature? Append a sentinel element. For char arrays specifically, it could be the null-terminating `'\0'` which is used for standard C strings.

```
void doSomething(char charArray[])
{
    char* p = charArray;
    for (; *p != '\0'; ++p)
    {
         // if '\0' happens to be valid data for your app,
         // then you can (maybe) use some other value as
         // sentinel
    }
    int arraySize = p - charArray;

    // now we know the array size, so we can do some thing
}
```

Of course, then your array itself cannot contain the sentinel element as content. For other kinds of (i.e., non-char) arrays, it could be any value which is not legal data. If no such value exists, then this method does not work.

Moreover, this requires co-operation on the caller side. You really have to make sure that the caller reserves an array of `arraySize + 1` elements, and *always* sets the sentinel element.

However, if you really cannot change the signature, your options are rather limited.

Share

Improve this answer

Follow

edited Jan 14, 2009 at 21:48

answered Jan 14, 2009 at 21:25

Reunanen

8,001 ● 2 ● 37 ● 58

> The sentinel element technique works even with data with no invalid elements, though in that case you'll need to escape the data when it comes in and unescape it when it comes out. But at that point it's probably time to use a class instead. – Brian Jan 15, 2009 at 14:16

6

In general when working with C or low-level C++, you might consider retraining your brain to never consider writing array parameters to a function, because the C compiler will always treat them as pointers anyway. In essence, by typing those square brackets you are fooling yourself in thinking that a real array is being passed, complete with size information. In reality, in C you can only pass pointers. The function

```
void foo(char a[])
{
    // Do something...
}
```

is, from the point of view of the C compiler, exactly equivalent to:

```
void foo(char * a)
{
    // Do something
}
```

and obviously that nekkid char pointer contains no length information.

If you're stuck in a corner and can't change the function signature, consider using a length prefix as suggested above. A non-portable but compatible hack is to specify the array length in an size_t field located *before* the array, something like this:

```
void foo(char * a)
{
    int cplusplus_len = reinterpret_cast<std::size_t *>(a)[-1];
    int c_len = ((size_t *)a)[-1];
}
```

Obviously your caller needs to create the arrays in the appropriate way before passing them to foo.

Needless to say this is a horrible hack, but this trick can get out of trouble in a pinch.

Share Improve this answer Follow

answered Jan 14, 2009 at 22:44

John Källén
7,923 ● 35 ● 67

Isn't that undefnied behaviour? – José D. Jun 20, 2015 at 18:54

6

It actually used to be a quite common solution to pass the length in the first element of the array. This kind of structure is often called `BSTR` (for “BASIC string”), even though this also denoted different (but similar) types.

The advantage over the accepted solution is that determining the length using a sentinel is slow for large strings. The disadvantage is obviously that this is a rather low-level hack that respects neither types nor structure.

In the form given below it also only works for strings of length <= 255. However, this can easily be expanded by storing the length in more than one byte.

```
void doSomething(char* charArray)
{
    // Cast unnecessary but I prefer explicit type conversions.
    std::size_t length = static_cast<std::size_t>(static_cast<unsigned char>
(charArray[0]));
    // … do something.
}
```

Share

Improve this answer

Follow

edited Jan 18, 2009 at 11:51

answered Jan 14, 2009 at 22:03

Konrad Rudolph
545k ● 139 ● 956 ● 1.2k

3 it has a slight issue 'cause of the strange C++ and C handling of char. if it is signed, values >127 could result in huge size_t values (wrap-around) when converted to size_t. you can fix it by converting to unsigned char first: static_cast<std::size_t>((unsigned char) charArray[0]); – Johannes Schaub - litb Jan 18, 2009 at 5:24

4

if it's nullterminated, strlen() would work.

Share Improve this answer Follow

answered Jan 14, 2009 at 21:26

Jimmy
91.3k ● 18 ● 121 ● 139

1 Yes, this is the perfect answer for char arrays as in the question, but would not really work for other data types. – Reunanen Jan 14, 2009 at 21:35

Aside from the fact that it is perfectly legal to write `doSomething(NULL);`. The **compiler** changes the OPs fingerprint to `char* charArray` :) c-faq.com/aryptr/aryptrparam.html – kfsone Jun 2, 2013 at 18:01

2

You can't determine the size from `charArray` alone. That information is not automatically passed to the function.

Of course if it's a null-terminated string you can use `strlen()`, but you have probably considered that already!

Consider passing a `std::vector<char>` & parameter, or a pair of pointers, or a pointer plus a size parameter.

Share Improve this answer Follow

answered Jan 14, 2009 at 21:26

finnw
48.6k ● 24 ● 148 ● 223

I think Yuval's constrained on the function signature. – Calyth Jan 14, 2009 at 21:45

@finnw That's because it doesn't pass the array at all, it just passes a pointer to it. – kfsone Jun 2, 2013 at 18:00

2

This is actually more C than C++, in C++ you'd probably rather use a std::vector. However, in C there's no way to know the size of an array. The compile will allow you to do a sizeof if the array was declared in the current scope, and only if it was explicitly declared with a size (**EDIT:** and "with a size", I mean that it was either declared with an integer size or initialized at declaration, as opposed to being passed as a parameter, thanks for the downvote).

The common solution in C is to pass a second parameter describing the number of elements in the array.

EDIT:
Sorry, missed the part about not wanting to change the method signature. Then there's no solution except as described by others as well, if there's some data that is not allowed within the array, it can be used as a terminator (0 in C-strings, -1 is also fairly common, but it depends on your actual data-type, assuming the char array is hypothetical)

Share edited Jan 15, 2009 at 14:05 answered Jan 14, 2009 at 21:28

Improve this answer

Follow

falstro

35.6k ● 10 ● 76 ● 88

1 -1. (a) Arrays declared without sizes infer their sizes from their initialisers and this size can be accessed using sizeof(). (b) Using a function template instead (as per Josh Kelley's answer) *does* allow you to infer the size of an array passed to a function. – j_random_hacker Jan 15, 2009 at 12:53

1

In order for a function to know the number of items in an array that has been passed to it, you must do one of two things:

1. Pass in a size parameter
2. Put the size information in the array somehow.

You can do the latter in a few ways:

- Terminate it with a NULL or some other sentinel that won't occur in normal data.
- store the item count in the first entry if the array holds numbers
- store a pointer to the last entry if the array contains pointers

Share Improve this answer Follow

answered Jan 14, 2009 at 21:34

AShelly

35.5k ● 16 ● 95 ● 157

Even the compiler agrees he needs to change the fingerprint... c-faq.com/aryptr/aryptrparam.html – kfsone Jun 2, 2013 at 18:00

0

try using strlen(charArray); using the cstring header file. this will produce the number of characters including spaces till it reaches the closing ".

Share Improve this answer Follow

answered Jun 7, 2014 at 6:03

Percy Stainwall

1

-1

You are guarranteed to receive 4 in a 32-bit PC and that's the correct answer. because of the reason explained here and here. The short answer is, you are actually testing the sizeof a pointer rather than an array, because "the array is implicitly

converted, or decays, into a pointer. The pointer, alas, doesn't store the array's dimension; it doesn't even tell you that the variable in question is an array."

Now that you are using C++, boost::array is a better choice than raw arrays. Because it's an object, you won't loose the dimention info now.

Share Improve this answer Follow

answered Mar 18, 2009 at 8:52

t.g.
1,739 ● 2 ● 14 ● 25

-2

I think you can do this:

```
size_t size = sizeof(array)/sizeof(array[0]);
```

PS: I think that the title of this topic isn't correct, too.

Share Improve this answer Follow

answered Nov 19, 2012 at 17:49

01codebit
1 ● 2

2 No, all the statements in the OP are correct. The big gotcha is that in the definition `void doSomething(char charArray[]) {/*...*/}`, the parameter `charArray` is not actually an array. – aschepler Nov 19, 2012 at 18:02

-3

Dude you can have a global variable to store the size of the array which will be accessible throughout the program. At least you can pass the size of the array from the main() function to the global variable and you will not even have to change the method signature as the size will be available globally.

Please see example:

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

int size; //global variable

//your code

void doSomething(char charArray[])
{
    //size available

}
```

Share

Improve this answer

Follow

edited Jan 16, 2013 at 9:00

cleg

5,022 ● 5 ● 37 ● 52

answered Jan 16, 2013 at 8:36

ScaryJoker

11 ● 3