Exceptions to array decaying into a pointer?

Asked 8 years, 10 months ago Modified 3 years, 6 months ago Viewed 34k times

80

I have seen in many posts that "in most of the cases array names decay into pointers".
Can I know in what cases/expressions the array name doesn't decay into a pointer to its first elements?

c arrays pointers

61

Share Edit Follow Flag

edited Oct 21, 2018 at 3:20
Swordfish
12.7k ● 3 ● 19 ● 43

asked Jul 19, 2013 at 18:21

nj-ath
2,820 ● 2 ● 22 ● 41

2 More context is required: Are you working in a specific language? Do you have an example? – abiessu Jul 19, 2013 at 18:23

consider C language. And i'm looking for an example where array names doesn't decay into pointer. – nj-ath Jul 19, 2013 at 18:25

1 @TheJoker I given here an answer in which I show this cases – Grijesh Chauhan Jul 19, 2013 at 18:57

Re H2Co3's second point, i.e. with sizeof, I'm reading Head First C, and it first illustrates pointer decay using sizeof(msg) inside a function where msg was passed in as an argument. They had a little box explaining that an array variable decays to a pointer when it's passed into a function as an argument (paraphrasing) so you get 4 or 8 (bytes), not array size. I got confused because in the next chapter on the string library, they introduce strlen() and use it the same way they'd used sizeof(). I came here to straighten my head out and now you twisted it up a little more. :P – punstress Aug 6, 2013 at 0:36

This answer has all the exceptions with examples. – legends2k Jul 9, 2014 at 8:27

1 Answer

Sorted by: Highest score (default)

69

Sure.

In C99 there are three fundamental cases, namely:

1. when it's the argument of the `&` (address-of) operator.
2. when it's the argument of the `sizeof` operator.
3. When it's a string literal of type `char [N + 1]` or a wide string literal of type `wchar_t [N + 1]` (`N` is the length of the string) which is used to initialize an array, as in `char str[] = "foo";` or `wchar_t wstr[] = L"foo";`.

Furthermore, in C11, the newly introduced `alignof` operator doesn't let its array argument decay into a pointer either.

In C++, there are additional rules, for example, when it's passed by reference.

Share Edit Follow Flag

edited Jul 19, 2013 at 18:37

answered Jul 19, 2013 at 18:26

user529758

2 I'm sorry, but could you explain the 3rd case more clearly? I mean isn't the *str* in the third case used to refer to the memory location of the string? Hence a pointer again? – nj-ath Jul 19, 2013 at 18:38

3 @TheJoker No, that would be `const char *str_ptr = "literal";`. In my example, `str` is declared as an array, hence it **is** an array. – user529758 Jul 19, 2013 at 18:39

@GrijeshChauhan Thanks :) – user529758 Jul 19, 2013 at 21:08

@H2CO3 did you get me? actually I refer your answer in my linked answer. :) – Grijesh Chauhan Jul 19, 2013 at 21:09

1 3rd case is not a case. It's just one of the cases for array initialization syntax when it's an array of `char`. The right side of an array initialization statement doesn't even take an expression, so it doesn't make sense as an answer to this question. – newacct Jul 21, 2013 at 9:25

@newacct I suggest you read the relevant section of the C standard - literally these cases are enumerated. Furthermore, in an array initialization, if and only if the RHS is a string literal, then it **is** an expression (of array type), not an initializer list. – user529758 Jul 21, 2013 at 9:53

6 @H2CO3: First of all, for initialization it specifically says "string literal". It does not say anything about "array expression" or "expression" at all. So the string literal syntax just happens to be used in two places: as the right side of an initialization, and also as an expression. That does not mean the right side of initialization is an expression. Plus, the question specifically asked about "array name", and a string literal is not an array name, and you cannot put an array name (or any other array expression except a string literal) on the right side of an initialization for array. – newacct Jul 22, 2013 at 3:12

I agree with @newacct, it doesn't make sense to initialize and array with an expression, since array initialization is essentially about initializing all the elements of the array that you are trying to initialize, and that's why we use brace-enclosed initializer or in the case of an array of char a string literal syntax. So the string literal for example "hello" is a syntax that is used for two different things depending on the context: 1) as unnamed static array of constant char. 2) as a syntax in the right side of an array of char initialization. – AlexDan Jan 17, 2014 at 5:34

@AlexDan possibly. nevertheless, string literals **are** of array type. – user529758 Jan 17, 2014 at 6:25

@H2CO3 If string literals were of array type, then this should fail to compile, char c[10]="hi"; because you are assigning char[3] to char[10], but in reality it does compile just fine. With that being said, when I was referring to 'string literals' in my last comment, I meant by it a syntax by which it's meaning change depending on its context. so string literals are not always of array type. – AlexDan Jan 17, 2014 at 6:34

@AlexDan I know it compiles, and I accept that array initialization with string literals is a special syntax. However, if we accept the presence of special syntax and exceptions, then we can as well accept that `const char foo[] = <another array>;` is fine -- only if the array is a string literal. – user529758 Jan 17, 2014 at 6:50

@AlexDan By the way, the standard is ambiguous at best as to what a string literal really is. It says "A character string literal is a sequence of zero or more multibyte characters enclosed in double-quotes", and that it "is then used to initialize an array of static storage duration and length just sufficient to contain the sequence.". Furthermore, later it says "It is unspecified whether these arrays are distinct". So once string literals are arrays, once they are not... then who knows what they are? – user529758 Jan 17, 2014 at 6:53

@H2CO3 that's a different thing, but it is also possible if the standard choosed to do so, but it's not the case. I will explain why it is different, when you you do `char c[]="hello";` the exception here is about how to implement the string literal `"hello"`. But to do something like ` const char foo[] = <another array>; ` the compiler must have an implementation of how to initialize an array with another array which was not possible before, after all array are kind of second class-citizens. – AlexDan Jan 17, 2014 at 7:04

This answer misses the `_Alignof` operator that has been added in C11. – Jens Gustedt Feb 23, 2014 at 16:43

@JensGustedt; Nope. It shouldn't be added there! – haccks Aug 19, 2015 at 16:13