

Re: A question on string literals

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2005-08/msg02144.html

- *From:* Chris Torek <nospam@xxxxxxxx>
 - *Date:* 20 Aug 2005 19:26:12 GMT
-

>> Chris Torek wrote:
>>> &foo – object context
>>> sizeof foo – object (or maybe even "sizeof") context
[snippage]
>>> and so on.

>Joe Wright <jwright@xxxxxxxx> writes:
>> I know you Chris and I love you like a brother but given..
>> int foo = 2, bar = 3;
>>
>> &foo – is the address of foo, a 'value' of type (int*)
>> sizeof foo – is a value (4 at my house) of type (size_t)
>> ++foo – is a value (now 3) of type (int)
>>
>> None of these have 'object' context as I see it. What are you trying
>> to tell us here?

In article <ln64u5p4hp.fsf@xxxxxxxxxxxxxxxx>
Keith Thompson <kst-u@xxxxxx> wrote:
>In "&foo", he's talking about the context in which the name "foo"
>appears, not the entire expression. The argument to the unary "&"
>operator must be an lvalue, so foo is in an "object context". Similarly,
>the argument of unary "++" must be an lvalue.

Keith is right about this.

>He's mistaken about sizeof, which doesn't require an lvalue.

This is true, and is why I said "or maybe even `sizeof context`".
sizeof is even more exceptional than unary-&.

Finally, I would like to note that -- while it was never added to
C99 -- there have been various proposals, over the years, to
implement array assignment in C. I suspect that C1X or C2X (if
these ever come about) may eventually do so -- and if so, I believe
the "most natural" way to handle array assignment is to add a rule
that says, in:

```
a = b
```

Re: A question on string literals

if "a" has type "array N of T", b is converted to a value of type "pointer to T" (and if it does not match a diagnostic is required); then the code acts much like a call of the form:

```
memcpy(&a[0], b, sizeof a)
```

except that the type of the result is "pointer to T" (rather than "pointer to void").

If this *were* added to C, the left side of a simple assignment operator would join the ranks of the "exceptions" for the array-to-pointer conversion rule.

Of course, there are also "obvious" and "natural" (to me anyway) interpretations for:

```
arr += arithmetic_type;
arr -= arithmetic_type;
arr++; /* same as arr += 1 */
arr--; /* same as arr -= 1 */
/* and so on for all arithmetic and logical operators */
```

which involve replicated arithmetic applying the right-hand-side value (or implied 1, for ++ and --) to each element of the array, with the value of the expression as a whole again being "pointer to first element of array". (Some might object that this renders identical values for arr++ and ++arr, to which I say: "so what?" :-)

All of these interpretations arise from one single central idea: naming an array always just names the entire array, but the "value" of an array (when a value is needed) is computed by finding a pointer to its first element. If a value is *not* needed, the array still names the entire array.

Again, this is not quite what the actual C standards (C89 and C99 both) say, but it delivers the same result, in what I think is a simpler, yet ultimately more powerful, way.

--

In-Real-Life: Chris Torek, Wind River Systems
Salt Lake City, UT, USA (40°39.22'N, 111°50.29'W) +1 801 277 2603
email: forget about it <http://web.torek.net/torek/index.html>
Reading email is like searching for food in the garbage, thanks to spammers.

.

-
- *Follow-Ups:*
 - ◆ *Re: A question on string literals*
 - ◇ *From:* Tim Rentsch

- **References:**
 - ◆ **[A question on string literals](#)**
 - ◇ From: junky_fellow
 - ◆ **[Re: A question on string literals](#)**
 - ◇ From: Chris Torek
 - ◆ **[Re: A question on string literals](#)**
 - ◇ From: Joe Wright
 - ◆ **[Re: A question on string literals](#)**
 - ◇ From: Keith Thompson
- Prev by Date: **[Re: duff's device / loop unriolling](#)**
- Next by Date: **[Re: Array in struct](#)**
- Previous by thread: **[Re: A question on string literals](#)**
- Next by thread: **[Re: A question on string literals](#)**
- Index(es):
 - ◆ **[Date](#)**
 - ◆ **[Thread](#)**