# Disambiguate the storage class of some compound literals

Jens Gustedt, Martin Uecker

2021-9-15

# Disambiguate the storage class of some compound literals

**change request for C23**

Jens Gustedt[1] and Martin Uecker[2]

[1] INRIA and ICube, Université de Strasbourg, France
[2] University Medical Center Göttingen, Germany

**Versions:** This is a split off of N2739, namely its aspect to clarify the the storage class of compound literals that appear in function parameters. The main issue of blocks and scopes is continued as N2818.

## 1. INTRODUCTION

When digging into the grammar for the definition of blocks, we noticed that there is a need for clarification about the storage class of compound literals that are evaluated in the type expressions of function parameters. It turned out that `clang` has them as static storage duration while `gcc` has them as automatic. In N2739 we asked WG14 to clarify this situation and consensus (13-0-8) has been reached that it should be specified as automatic.

## 2. AMBIGUITIES

The current text has an unfortunate ambiguity by not clarifying through the grammar that function parameter declarations are contained in the same block as the function body. Whereas for the parameters themselves it is made more or less clear that they somehow live inside the body of the function, the situation is not clear for compound literals. Their definition refers to a grammatical term "*occurs outside the body of a function*" and not to the moment they are evaluated.
Consider the following contrived example:

```
1  int f(int*);
2  int g(char* para[f((int[27]){ 0, })]) {
3    return 0;
4  }
```

Here a strict reading of the wording would imply that the compound literal occurs outside of the function body. With that reading it would have static storage duration and a lifetime that covers the whole execution.
Another interpretation would place the evaluation of the parameter declaration within the function body and thus the result would be an object of automatic storage duration that is bound to the corresponding function call.
Common implementations currently do not agree on these interpretations. Whereas `clang` follows the first and creates a file scope static object without linkage, `gcc` follows the second and has an automatic object accessible only during a call to `g`.
Although this is admittedly a marginal problem (such function definitions with compound literals are not much heard of) we thought that a clarification is in order and WG14 followed us with our proposal to fix the case in question to impose automatic storage duration for such compound literals.

## 3. IMPACT

The proposed changes are meant for clarification and should have no impact on the validity of user code, as long as it doesn't use compound literals in parameter declarations. (But if they do, they are not portable in the current situation, anyhow.)

For the conformance of implementations, WG14 clearly decided to have the compound literals in question as having automatic storage duration and to associate them to the function body, much as parameters. Implementations such as clang would have to change to be conforming to C23.

## 4. INTERACTION WITH LAMBDAS

The proposed specification is chosen to be compatible with the possible adoption of lambdas into C, see N2738 and follow-ups. They do not need lambdas themselves and the addition of lambdas would not necessitate changes to text as proposed here. It ensures the following:

(1) A compound literal that appears in the size expressions of VLA or VM parameter of a lambda will be instantiated for each call to the lambda and have a lifetime that spans the call.
(2) A compound literal that appears in the value expression of a capture will have a lifetime that starts at the evaluation (of the capture expression) and ends with the end of the surrounding scope. If the surrounding scope of the lambda is a block (in particular a function) the storage class will be automatic. If the lambda expression lives in file scope the storage duration of such a compound literal is static. By this it is guaranteed that such compound literals live at least as long as any lambda object that is derived from the lambda expression.

## 5. PROPOSED CHANGES

We propose a change that is independent of the acceptance (or not) of the other sequel of N2739, namely N2818 that is proposed to redesign the syntax of blocks.

CHANGE 1. *Change* `6.5.2.5 p6` *as follows:*

> *The value of the compound literal is that of an unnamed object initialized by the initializer list. If the compound literal* ~~occurs~~ is evaluated *outside the body of a function* and outside of any parameter list*, the object has static storage duration; otherwise, it has automatic storage duration associated with the enclosing block,* if any*. If it is evaluated for a function call to determine the variably modified type of a parameter it has a lifetime that starts with the evaluation and that ends with the end of the execution of the call.*

CHANGE 2. *Add the following example (place at the discretion of the project editors):*

**Example**: Consider the following two functions

```
1  int f(int*);
2  int g(char* para[f((int[27]){ 0, })]) {
3    ...
4    return 0;
5  }
```

Here, each call to `g` creates an unnamed object of type `int[27]` to determine the variably modified type of `para` for duration of the call. During that determination, a pointer to the object is passed into a call to the function `f`. If a pointer to the object is kept by `f`, access to that object is possible during the whole execution of the call to `g`. The lifetime of the object ends with the end of the call to `g`; for any access after that, the behavior is undefined.

## 6. QUESTION FOR WG14

QUESTION 1. *Shall the indicated changes of N2819 be integrated as such into C23?*