ISO/IEC JTC 1/SC 22/WG14 **N2739 v1**
WG 21, SG 22 **P2378R0**

2021-5-15

Properly define blocks as part of the grammar
clarification request and proposal for C23

Jens Gustedt[1] and Martin Uecker[2]

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

1. INTRODUCTION

Blocks are a fundamental concept in C for the definition of visibility scopes of identifiers and for the lifetime of objects. Currently, there is no closed definition what a block is and the different definitions that compose the term have to be collected in different places that spread over several clauses. In particular, the fact that dependent statements of iteration or selection statements form blocks of their own is easily overlooked and leads to misunderstandings for example concerning the lifetime of compound literals.
We propose to change that situation by introducing terms *primary block* and *secondary block* in the syntax and by referring to the other definitions of blocks, namely functions definitions and lambda expressions (if added to C23), in a summary definition.
Additionally, when digging into this 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. We ask WG14 to clarify this situation, but nevertheless for our proposal we assume that the answer is to go for automatic storage duration because it fits better with the overall semantics of compound literals.

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:

```c
int f(int*);
int g(int argc, char* argv[f((int[27]){ 0, })]) {
  return 0;
}
```

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 think that a clarification is in order. Tightening the

grammar for blocks would give a good opportunity to have a parameter and a compound literal that was evaluated during its declaration with a similar life span.
When introducing lambdas into C this will become a little bit more important, because such compound literals found in parameter declarations of lambdas otherwise would have a storage duration that depends on the context in which the lambda expression is found.

3. IMPACT

The proposed changes are meant for clarification and should have not much 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, we would push the ballance on the side where compound literals in parameter declarations have automatic storage duration. If WG14 wants such compound literals to have static storage duration, a reformulation of the proposed text would be necessary.

4. PROPOSED CHANGES

The changes that are proposed here suppose that the lambda feature is integrated into C23. If not, they can editorially be adapted to remove the reference to lambdas.

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~~a block, the object has static storage duration; otherwise, it has automatic storage duration associated with the enclosing block.*

The following two are the principal changes proposed to the grammar.

CHANGE 2. *Replace the content of clause* `6.8 p1` *by*

> *statement:*
> *labeled-statement*
> *expression-statement*
> *attribute-specifier-sequence*$_{opt}$ *primary-block*
> *attribute-specifier-sequence*$_{opt}$ *jump-statement*
>
> *primary-block:*
> *compound-statement*
> *selection-statement*
> *iteration-statement*
>
> *secondary-block:*
> *statement*

CHANGE 3. *Add a two new sentences to the start of* `6.8 p3`

> *A* block *is a primary block, a secondary block, or the block associated with a function definition or a lambda expression*
>
> *Whenever a* block *B appears in the syntax production as part of the definition of an enclosing block A, scopes of identifiers and lifetimes of objects that are associated with B do not extend to the parts of A that are outside of B.*

Then, we also need to change the grammar of the three different forms of primary blocks.

CHANGE 4. *In* `6.8.2` *(Compound statement) change* `p2` *as follows:*

> *A* compound statement *that is a function body together with the capture list, if any, the parameter type list and the optional attribute specifier sequence of the function declarator forms the block associated with the function definition or lambda expression in which it appears. Otherwise, it* is a block *that is different from any other block*.

CHANGE 5. *In* `6.8.4 p1` *(Selection statements) replace the syntax term* ~~*statement*~~ *by* *secondary-block*.

CHANGE 6. *Remove* `6.8.4 p4`.

> ~~*A selection statement is a block whose scope is a strict subset of the scope of its enclosing block. Each associated substatement is also a block whose scope is a strict subset of the scope of the selection statement.*~~

CHANGE 7. *In* `6.8.5 p1` *(Iteration statements) replace the syntax term* ~~*statement*~~ *by* *secondary-block*.

CHANGE 8. *Remove* `6.8.4 p5`.

> ~~*An iteration statement is a block whose scope is a strict subset of the scope of its enclosing block. The loop body is also a block whose scope is a strict subset of the scope of the iteration statement.*~~

And finally, we make the association of the parameter type list and the function body into a single block explicit.

CHANGE 9. *Change* `6.9.1 p9` *(Function definitions) as follows*

> *The parameter type list and the optional attribute specifier sequence of the declarator together with the compound statement of the function body forms a single block.*[FNT1)] *Each parameter and each compound literal that is evaluated in a parameter declaration is associated with this block and* has automatic storage *duration.* ~~*its*~~*The* identifier *of a parameter*, *is an lvalue.*

and add the footnote

> [FN1)] *The visibility scope of parameters starts immediately after their declaration and extends to parameter declarations and possible attributes that follow and then to the entire function body.*

CHANGE 10. *Apply the analogous changes to the language syntax summary in Annex* `A.2.3` *and* `A.2.4`.

5. QUESTIONS FOR WG14

QUESTION 1. *Is a compound literal that occurs in the parameter list of a function definition associated with the block of the function?*

QUESTION 2. *Shall the indicated changes be integrated into C23?*