N3235: Initializing static and thread objects with compound literals

4 February, 2025

Document
    N3235
Target Audience
    C developers concerned with writing portable code

Abstract:
C permits sub-objects of objects with `static` storage duration to be initialized with a braced-initializer, but does not permit them be initialized with a compound literal. This paper proposes remedying this apparent incongruity by allowing `static` object initialization with a compound literal.

Introduction and Rationale

Document N496 included in the C99 standard introduced compound literals, "to provide important additional flexibilty to literals in expressions". Together with braced initialization, initialization by compound literal supports a minimal, but useful, declarative mode of programming in C that allows programmers to specify the entire initial state of an aggregate object and its nested sub-objects.

When used for an object with `static` storage duration, braced initializers also support an optimization whereby programmers can specify entire aggregate objects to be initialized once and used for the entire duration of a program. This increases the representational power of C by allowing programmers to create constants that behave similarly to string literals in terms of their uses and ease of creation, but may be of any aggregate type.

However, this benefit does not extend to compound literals. Since compound literals are not listed among the permissible types of constant expression, an object of `static` or `thread` storage duration cannot be initialized by a compound literal, even if that compound literal is itself declared with `static` storage duration and/or `const` qualification.

Subsequent examples use the following structure definitions:

```
struct s {
    int len;
    const char* str;
};

struct z {
    struct s name;
};
```

To initialize an object of type `struct z` with `static` storage duration, it is permissible to use a braced initializer list to initialize the subobject `name`:

```
static const struct z val = { { 6, "dragon" } };
```

However, it is currently a constraint violation to use a compound literal of type `struct s`, even if that compound literal is itself declared with `static` storage specifier and/or `const` qualification:

```
static const struct z val2 = { (static const struct s){ 6, "dragon" } };
```

To a programmer unfamiliar with the standard, it's unclear why the initialization of `val` is permitted, but `val2` is not. Both a braced initializer and a compound literal seem to be providing in-line, compile-time known information for initialization.

This paper proposes that both kinds of expressions be permitted in initialization of an object with `static` or `thread` storage duration.

Alternatives

Braced initializers

As already mentioned, braced initializers can already be used to initialize `static const` variables:

```
static const struct z val = { { 6, "dragon" } };
```

However, compound literals provide an additional affordance that braced initializers do not: specifying the type and storage duration of a literal value inline:

```
struct a {
    int c;
    const int* v;
};

struct a get_data() {
    return (struct a){
        .c = 3,
        .v = (static const int[]){4, 5, 6},
    };
}
```

This can be used when calling functions or returning data from a function to define static constant data, inline, that would otherwise be stack-allocated. The equivalent in C23 requires a separate variable declaration before the compound literal expression:

```
struct a get_data() {
    static const int v[] = {4, 5, 6};
    return (struct a){
        .c = 3,
        .v = v,
    };
}
```

CONSTEXPR

In C23, compound literals are permitted in static and thread object initializers as long as they are *compound literal constants*, i.e. compound literals with `constexpr` storage class:

```
static const struct z val3 = { (constexpr struct s){ 0, nullptr } };
```

However, this precludes the use of string literals in the initializer, as string literals are not valid in `constexpr` context:

```
static const struct z val4 = { (constexpr struct s){ 6, "dragon" } }; // Constraint vi
```

Prior art

POPULAR COMPILER IMPLEMENTATIONS

Initialization of a `static` object by compound literal is already supported by `gcc` in C23 standard mode. The above example compiles successfully with `gcc -std=c23` and only produces a warning if `-Wpedantic` is used:

```
<source>:12:32: warning: initializer element is not constant [-Wpedantic]
   19 | static const struct z val2 = { (static const struct s){ 6, "dragon" } };
      |                                ^
<source>:12:32: note: (near initialization for 'val2.name')
```

Compiling the above example with `clang` generates an error and terminates compilation:

```
<source>:12:33: error: expected expression
   16 | static const struct z val2 = { (static const struct s){ 6, "dragon" } };
      |
```

Popular minimal C compiler `tcc` compiles the above example successfully and produces no warnings at all.

C++

The C++ standard does not permit the same compound-literal syntax in initializer lists, but does permit a similar construction:

```
static const struct z val2 = { s{ 6, "dragon" } };
```

Both `clang` and `gcc` accept and successfully compile a modified version of the earlier example when compiling in C++ mode, despite the C++ standard's initializer list grammar:

```
static const struct z val2 = { (const struct s){ 6, "dragon" } };
```

Proposed wording

This proposed wording is relative to the most recently available (as of March 2024) public draft of N3435.

Modify §6.7.11 ¶8:

- All the expressions in an initializer for an object that has static or thread storage duration shall be constant expressions, string literals, or compound literals initialized with constant expressions, string literals, or nested compound literals initialized with the same. ~~or is declared with~~. Objects declared with the `constexpr` storage-class specifier shall be constant expressions (including compound literal constants) or string literals.

Modify §6.7.11 footnote 169:

- If the object being initialized does not have automatic storage duration, this case violates a constraint unless the expression is a named constant or compound literal constant. If the object being initialized has static or thread storage duration, the expression may also be a compound literal initialized with constant expressions, string literals, or nested compound literals initialized with the same.

Add an example to the end of the Semantics section in §6.7.11:

- EXAMPLE 15 An object with static or thread storage duration is valid to initialize with a compound literal so long as the initializer of the compound literal contains only constants or string literals:

```
    struct s {
        int len;
        const char* str;
    };

    struct z {
        struct s name;
    };
    static const struct z val2 = { (struct s){ 6, "dragon" } };
```

References

- 1: Example on Godbolt: https://godbolt.org/z/3eqW99zKf