WG14 N3451
Author: Stephen Heumann (stephenheumann@gmail.com)
Date: 2025-01-10

Initialization of anonymous structures and unions (v2)

Revision history:

N3314
- Initial version

N3451
- Add examples
- Rebase on N3435 working draft

Summary:

The C standard contains wording that appears to prohibit an anonymous structure or union from participating in initialization as a structure or union when the containing structure or union is initialized with a brace-enclosed initializer list. However, many implementations (including widely-used ones) actually allow this, suggesting that their implementors may have understood this standard wording differently, or that they chose not to follow it. Since at least one implementation does behave largely as the standard appears to specify, there is disagreement between the behavior of actual implementations regarding the initialization of anonymous structures and unions.

This document proposes changes to clarify and standardize the behavior in this area. It proposes to specify that anonymous structures and unions do participate in initialization, consistent with the behavior of those implementations that already allow this.

Behavior specified by existing standard wording:

N3435 6.7.11 p6 says “The initializer for an object that has structure or union type shall be either a single expression that has compatible type or a brace-enclosed list of initializers for the elements or named members.” The initialization of structures or unions with a brace-enclosed initializer list is therefore explicitly limited to named members. This is reinforced by 6.7.11 p13, which says “Except where explicitly stated otherwise, for the purposes of this subclause unnamed members of objects of structure and union type do not participate in initialization.”

In 6.7.3.2 p15, anonymous structures and unions are defined as being (certain kinds of) unnamed members of the containing structure or union. The wording mentioned above therefore prohibits them from directly being initialized when the containing structure or union is initialized with a brace-enclosed initialized list. For example, it means the following is invalid:

```
union {struct {int a,b;}; long c;} u = {{1,2}}; // Example 1
```

Members of an anonymous structure or union can be initialized, since they "are members of the containing structure or union" (6.7.3.2 p15). However, this leads to potentially surprising behavior when there is an anonymous structure in a union, because the rules for union initialization will apply (only the first named member can be explicitly initialized when not using designators). Similarly, when there is an anonymous union in a structure, the rules for structure initialization will apply (initializers can be provided for all named members without using designators).

Accordingly, the following should be allowed, and should initialize s.b to 2 and s.c to 3:

```
struct {union {int a; float b;}; int c;} s = {1,2,3}; // Example 2
```

Similarly, this should initialize s.b to 2 and (implicitly) s.c to 0:

```
struct {union {int a; float b;}; int c;} s = {1,2}; // Example 3
```

This should produce a diagnostic, because a union initializer without designators can only initialize the first named member of the union (in this case u.a):

```
union {struct {int a,b;}; long c;} u = {1,2}; // Example 4
```

In this document, the behavior described above will be referred to as "Behavior A."

(References above are to the latest available draft, but C11 through C23 contain similar wording and also specify Behavior A.)

Existing implementations that allow anonymous structure/union initialization:

Many implementations (including GCC, Clang, MSVC, and others listed below) do not actually behave as described above. For purposes of initialization, these implementations treat anonymous structures or unions the same as named structure or union members, except that they cannot be described by a designator. They treat the members of an anonymous structure as members of a structure, even if it is contained in a union, and treat the members of an anonymous union as members of a union, even if it is contained in a structure.

These implementations handle the above examples as follows:

1. They will accept Examples 1 and 4, treating them as initializing u.a to 1 and u.b to 2.
2. They give a diagnostic for Example 2 (reporting "too many initializers" or similar).
3. In Example 3, they initialize s.a to 1 and s.c to 2.

In this document, the behavior described above will be referred to as "Behavior B."

Behavior of existing implementations:

Here is a list of the implementations I have tested and their behavior with regard to initialization of anonymous structures and unions:

Follows Behavior A, except that it accepts Example 1 with no diagnostic:
SDCC 4.4.0

Follows Behavior B:
zig cc 0.13.0
CompCert 3.12
icx 2024.0.0
icc 2021.10.0
gcc 14.1
clang 18.1.0
msvc v19.40
cproc
TCC 0.9.27
Apple clang 15.0.0
Calypsi C 5.4
ORCA/C 2.2.0
Open Watcom 2.0
Oracle Developer Studio 12.6
TI cl430 v21.6.1.LTS
TI cl2000 v22.6.1.LTS
TI cl6x v8.3.13
TI armcl v20.2.7.LTS
TI clpru v2.3.3

Follows Behavior B, except that it accepts Example 2 with no diagnostic:
Chibicc 2020-12-07

SDCC is the only implementation I am aware of that largely follows Behavior A. SDCC differs from Behavior A in that it accepts Example 1 with no diagnostic, but that seems to be because it generally ignores extra braces and extra elements in initializers; it generates the same code as if the initializer was just `{1}`.

In light of the above results, Behavior A is not standardized among C implementations in practice, in spite of the wording in the standard that appears to specify it. In fact, it appears that Behavior B is substantially more prevalent among C implementations, although I acknowledge that there may be other implementations I have not tested that follow Behavior A.

Since numerous implementations do not behave in accordance with the interpretation of the standard wording described previously, it seems that their implementors may have interpreted that wording differently, or that it may have been unclear to them. Alternatively, those implementors may have chosen to deviate from the standard with regard to the initialization of anonymous structures and unions, perhaps to maintain compatibility with existing implementations of anonymous structures and unions that predate their standardization in C11.

Historical background:

Anonymous structures and unions were added to Standard C in C11, based on the proposal in WG14 N1406 (with some wording changes). That proposal does not mention anything

specifically about initialization of anonymous structures and unions, but it does cite both anonymous unions in C++ and the support for anonymous structures and unions provided as an extension in pre-C11 versions of GCC and Microsoft C compilers. All of these prior implementations followed Behavior B.

In light of the references to those prior implementations and the lack of any statement that the initialization behavior in Standard C was intended to be different from them, it is possible that the difference caused by the current C standard wording may have been an oversight, rather than the deliberate intention of WG14. The wording mentioned above concerning initialization of unnamed members dates back to the C99 or earlier standards, in which unnamed bit-fields were the only possible unnamed members. When anonymous structures and unions were added in C11, the impact of that wording on them may not have been fully considered.

Rationale for standardizing Behavior B:

As described above, there is currently divergent behavior between different implementations with regard to the initialization of anonymous structures and unions, and numerous widely-used implementations behave in a way that does not seem to follow the wording in the standard. I believe it would be in the interest of C users to clarify and standardize the behavior in this area so that it is consistent across C implementations, enabling portable and standard-conforming code to rely on it.

Clearly standardizing either Behavior A or Behavior B would be preferable to the status quo, but I believe it is preferable to standardize Behavior B, i.e. to allow anonymous structures and unions to participate in initialization. This will bring the standard in line with the existing behavior of many implementations, including widely used ones. There may well be existing code written for those implementations that relies on Behavior B, and changing them to follow Behavior A could break such code. In some cases (e.g. Example 3), the code would remain valid, but its semantics would change, potentially leading to silent breakage.

I believe Behavior B is also likely to be less confusing and less error-prone for C programmers in general. It is more consistent with the initialization behavior for structures or unions with named members, it permits the use of fully-braced initializers for anonymous structures and unions, and it avoids the potentially surprising situations discussed above with regard to the initialization of anonymous unions within structures or anonymous structures within unions. It is also more consistent with the initialization behavior of anonymous unions in C++.

Standardizing Behavior B will require changes to SDCC and any other implementations that may follow Behavior A. The maintainer of SDCC has expressed willingness to make such changes if Behavior B is standardized. There may be existing code for SDCC or other implementations that relies on Behavior A, which would also have to be changed. However, given that many widely-used implementations already follow Behavior B, the portability of any code relying on Behavior A is already limited in practice, and I suspect there is likely to be significantly less existing code relying on Behavior A than on Behavior B.

Question for WG14:

Does WG14 want to standardize Behavior B, allowing anonymous structures and unions to participate in initialization?

If the answer is yes, proposed wording to do so is provided below.

If the answer is no, I believe the standard should be revised to more clearly indicate that Behavior A is intended, e.g. by adding examples. I have not proposed wording for this, but I would be willing to work with the committee to do so if this is its preferred direction.

Explanation of proposed wording:

The wording proposed below is intended to standardize Behavior B. It is meant to be consistent with the existing behavior of those implementations listed above as following Behavior B, and as such should not require any changes to those implementations.

Wording is adjusted in several places to only restrict unnamed bit-fields from participating in initialization, rather than all unnamed members of structures and unions. The wording in question dates back to C99 or earlier, when unnamed bit-fields were the only possible unnamed members. Thus, the changes restore the meaning that this wording had in C99, while making clear that it does not apply to anonymous structures and unions.

A paragraph is also added to explicitly describe how anonymous structures and unions participate in initialization. In particular, this makes clear that members of anonymous structures and unions participate in initialization as members of the anonymous structure or union, not as members of the structure or union that contains it. This is necessary to ensure that anonymous structures within a union are initialized according to the rules for initialization of structures (not unions), and vice versa.

Examples are also provided, both for the newly-specified behavior and for the existing initialization behavior of unnamed bit-fields (which previously was not covered by an example).

Proposed wording:

This shows proposed **additions** and ~~removals~~ relative to WG14 N3435.

Change 6.7.11 paragraph 6 as follows:

The initializer for an object that has structure or union type shall be either a single expression that has compatible type or a brace-enclosed list of initializers for the ~~elements or named~~ members **other than unnamed bit-fields**.

Change 6.7.11 paragraph 13 as follows:

Except where explicitly stated otherwise, for the purposes of this subclause unnamed ~~members of objects of structure and union type~~ **bit-fields** do not participate in initialization. Unnamed **bit-field** members of structure objects have indeterminate representation even after initialization.

Change the last item in 6.7.11 paragraph 14 as follows:

— if it is a union, the first ~~named~~ member **that is not an unnamed bit-field** is initialized (recursively) according to these rules, and any padding is initialized to zero bits.

Change 6.7.11 paragraph 17 as follows:

If the initializer for a struct or a union is a single expression, the initial value of the object, including unnamed ~~members~~**bit-fields**, is that of the expression.[169)]

Add a new paragraph after 6.7.11 paragraph 19:

When a structure or union object is initialized using a brace-enclosed initializer list, any anonymous structure or union members of the object participate in initialization in the same way as named members, except that they cannot be described by designators. For the purposes of initialization, the members of an anonymous structure or union are treated as being members of the anonymous structure or union object rather than of the structure or union that contains it, but they may be described by designators as if they are members of the containing structure or union.

Change 6.7.11 paragraph 20 (now paragraph 21) as follows:

Each brace-enclosed initializer list has an associated *current object*. When no designations are present, subobjects of the current object are initialized in order according to the type of the current object: array elements in increasing subscript order, structure members **(other than unnamed bit-fields)** in declaration order, and the first ~~named~~ member of a union **that is not an unnamed bit-field**.[170)] In contrast, a designation causes the following initializer to begin initialization of the subobject described by the designator. Initialization then continues forward in order, beginning with the next subobject after that described by the designator.[171)]

Add the following additional examples in 6.7.11:

EXAMPLE *N* The declarations

```
struct {
      int a:10;
      int :12;
      long b;
} s = {1, 2};

union {
      int :16;
      char c;
} u = {3};
```

initialize `s.a` to 1, `s.b` to 2, and `u.c` to 3. The unnamed bit-field in `s` has indeterminate representation even after initialization.

EXAMPLE *N+1* The declaration

```
struct {
      union {
            float a;
            int b;
            void *p;
      };
      char c;
} s = {{.b = 1}, 2};
```

initializes `s.b` to 1 and `s.c` to 2. Members of the anonymous union can also be described by designators as if they are members of the containing structure, so the same initialization result can be achieved by:

```
struct {
      union {
            float a;
            int b;
            void *p;
      };
      char c;
} s = {.b = 1, 2};
```

Acknowledgments

Thank you to JeanHeyd Meneide, Jens Gustedt, Philipp Klaus Krause, and Joseph Myers for feedback on earlier versions of this proposal.

References

GCC Manual. Unnamed Structure and Union Fields.
https://gcc.gnu.org/onlinedocs/gcc/Unnamed-Fields.html

ISO/IEC 14882, Programming Languages—C++ (all editions).

Keaton, David. Anonymous Member-Structures and -Unions. WG14 N1406.
https://www.open-std.org/JTC1/SC22/WG14/www/docs/n1406.pdf

Microsoft. C language reference. Union Declarations.
https://learn.microsoft.com/en-us/cpp/c-language/union-declarations