Anonymous Member-Structures and -Unions

David Keaton

2009-09-28

1. Introduction

1.1 Purpose

This proposal specifies the form and interpretation of a pure extension to the language portion of the C standard to permit structures and unions to contain members that are anonymous structures and unions.

1.2 Scope

This document, although extending the C standard, still falls within the scope of that standard, and thus follows all rules and guidelines of that standard except where explicitly noted herein. All proposed changes are relative to WG14/N1362.

1.3 References

1. ISO/IEC 9899:1999(E), *Programming Languages—C.*
2. WG14/N1362, Committee Draft of C1X, 2009-03-01.
3. ISO/IEC 14882:1998(E), *Programming Languages—C++.*
4. Re: Anonymous structs in C++, <http://gcc.gnu.org/ml/gcc/2000-10/msg00067.html>.

1.4 Rationale

C++ allows two kinds of anonymous unions: those that define objects similar to variables, and those that are members of structures. The latter has become a widely-implemented extension to C that would benefit from standardization within the C language. Ken Thompson has suggested that anonymous member-unions become part of C since before C99.

Anonymous member-structures have become widely implemented as extensions to both C and C++.

Any compiler that must compile applications for the Win32 API, such as the gcc and Microsoft compilers, must permit anonymous member-structures and -unions in C. In addition, Sun has implemented anonymous member-structures in at least C++.

Some implementations have permitted anonymous member-structures and -unions in extended C to contain tags, which allows tricks such as the following.

```
struct point { float x, y, z; };

struct location {
     char *name;
     struct point;  // inheritance in extended C, but
                    // forward declaration in C++
};
```

This proposal does not support that practice, for two reasons. First, it introduces a gratuitous difference between C and C++, since C++ implementations must treat the declaration of **`point`** within **`location`** as a forward reference to the type **`location::point`** rather than a definition of an unnamed member. Second, this feature does not seem to be used widely in applications, perhaps because it compiles differently in extended C vs. C++.

Parts of this proposal were adapted from C++ subclause 9.5 [class.union].

1.5 Impact

There is no impact to existing conforming code. If nonconforming code depends on an extension that allows structures and unions to contain type declarations, then some useless declarations could become member definitions and increase the size of the containing structure or union. This is illustrated as follows.

```
struct date {
     struct { int hour, minute; }; // useless declaration → member
     int year;
     int day;
};
```

This situation is not expected to occur in code that otherwise works. The code would also have to depend on the size of **`struct date`** for there to be an impact.

There is no link-time or run-time impact to existing implementations. Only a parser change is required.

2. Language

Changed text surrounded by unchanged text is underlined in the following sections.

2.1 Changes to subclause 6.7.2.1

In paragraph 1, change the definition of struct-declaration to make the struct-declarator-list optional.

struct-declaration:

specifier-qualifier-list struct-declarator-list$_{opt}$ **;**

static_assert-declaration

Add the following paragraph to the constraints section after paragraph 4.

A struct-declaration that does not define an anonymous structure or anonymous union shall contain a struct-declarator-list.

In paragraph 7, change the third sentence to the following.

If the struct-declaration-list contains no named members, and no member names are brought into its scope via anonymous structures or anonymous unions, the behavior is undefined.

Add the following two paragraphs to the semantics section after paragraph 16.

An unnamed member of structure type with no tag is called an *anonymous structure*. An unnamed member of union type with no tag is called an *anonymous union*.

For the purpose of name lookup, after the definition of an anonymous structure or anonymous union, its members are considered to have been defined in the same scope in which the anonymous structure or anonymous union is declared. If the anonymous structure or anonymous union contains members that are anonymous structures or anonymous unions, this applies recursively to those contained members.

Add the following example after paragraph 20.

The following illustrates anonymous structures and unions.

```
struct v {
     union {   // anonymous union
          struct { int i, j; };    // anonymous structure
          struct { long k, l; } w;
     };
     int m;
} v1;

v1.i = 2;      // valid
v1.k = 3;      // invalid:  inner structure is not anonymous
v1.w.k = 5;    // valid
```

2.2 Changes to subclause A.2.2

Change the definition of struct-declaration to make the struct-declarator-list optional.

(6.7.2.1) struct-declaration:

specifier-qualifier-list struct-declarator-list$_{opt}$ **;**

static_assert-declaration

2.3 Changes to subclause J.2

In paragraph 1, change the first bullet point for 6.7.2.1 to the following.

— A structure or union is defined as containing no named members, and has no named members brought into the scope of its struct-declaration-list via anonymous structures or anonymous unions (6.7.2.1).