N2007

ISO/IEC 9899: Potential defect report and proposed change for C2X
Re: named child struct/union, that doesn't declare a member

Introduction

Some weeks ago I posted the following code to the reflector after an argument at work as to whether it was legal, and if so, what was it meant to do?

```
struct  S1
     {  union U11
             {int     m11;
              float   m12;
             };

        int  m13;
     } s1;
```

The issue is that U11 isn't an anonymous union (because it has a name "U11") but doesn't declare a member of S1. When tried with a number of compilers the result was either that it was rejected as a constraint error or it was treated as an anonymous union, with m11 and m12 accessible as though they were members of S1 (e.g. s1.m11 = 42;). The declaration of union U11 was also added at file scope, so could be used in other structs.

After some discussion (thanks to Doug and Roberto), it was concluded that the code was intended to be a constraint error because it can be argued that it violates the first constraint in 6.7.2.1 para 2 "A *struct-declaration* that does not declare an anonymous structure or anonymous union shall contain a *struct-declarator-list*". The syntax fragment its referring to is:

struct-declaration:
    specifier-qualifier-list struct-declarator-list$_{opt}$;
    static_assert-declaration

In parsing the above code `"union U11 {int m11; float m12;} ;"` is a *struct-declaration.*
It is believed that the intended reading is:

- `"union U11 {int m11; float m12;}"` is a *specifier-qualifier-list*
- There is no *struct-declarator-list*

Hence, as U11 isn't an anonymous union and doesn't have a *struct-declarator-list,* it violates the quoted constraint.

However, there would appear to be a different reading that says that this *struct-declaration* does "contain a *struct-declarator-list*". So it shouldn't be a constraint error – hence the potential need for a DR to clarify the intent (discussed in the next section).

The second issue that came up was 'why is that constraint there?' Hence the proposal to remove the constraint in C2X (discussed in the final section)

1) DR to clarify the legality or otherwise of the above code

The alternative reading of the constraint requirement comes about because the *specifier-qualifier-list* `"union U11 {int m11; float m12;}"` is interpreted by recursively entering the same part of the syntax tree. As its interpreted, "int m11;" and "float m12;" are *struct-declarations,* where "m11" and "m12" are *struct-declarator-lists.* So the *struct-declaration* for U11 does contain a *struct-declarator-lists*, so shouldn't be a constraint error.

The response to that argument on the reflector was that 'whenever a constraint refers to elements of the syntax tree, it means those elements in the term currently being processed, and not any terms that may be found by recursively traversing the tree'. However, I cannot see this principle stated anywhere in the standard.

Hence, I'd argue that whether this code is legal or not is ambiguous and a DR is required, either to:

- Establish the principle that "whenever a constraint refers to elements of the syntax tree, it means those elements in the term currently being processed, and not any terms that may be found by recursively traversing the tree", or
- Reword the constraint in 6.7.2.1 para 2 to clarify that the above code is intended to be a constraint error, for example by adding 'shall contain a *struct-declarator-list,* other than any that may be found in the interpretation of the *specifier-qualifier-list'*

2) Proposal to remove this constraint in C2X

There doesn't seem to be any compelling reason for this constraint, as its not protecting the user from any unspecified or undefined behaviour.

For example, the struct S1 may have been declared with an anonymous union, and then during subsequent maintenance a need for the same union arises elsewhere in the program. The easiest solution would be to add a name to the anonymous union (now U11), and reuse union U11.

The proposed semantics of a named child struct/union that doesn't declare a member are:

- The struct/union is added as a new type to the translation unit, as 6.7.2.1 para 8 "*The presence of a struct-declaration-list in a struct-or-union-specifier declares a new type, within a translation unit*"
- In the style of 6.7.2.1 para 13, a name is required for struct/union declarations such as the above. Then 'the members of <a named child struct/union declaration that doesn't declare a member variable in the containing structure or union> are considered to be members of the containing structure or union'