**WG14 N2905**

**Title: Safer Flexible Array Members**
**Author: Martin Uecker**
**Date: 2021-12-29**

This paper follows up on a specific proposal presented already in N2660. We propose to make a structure with flexible array member (FAM) an incomplete type. This is a breaking change, but we believe this change is desirable as the use of structs with FAM in situations where the FAM is silently ignored is inherently dangerous. With the proposed change such uses will then become constraint violations and then have to be diagnosed. In particular, this change affects the following constraints where incomplete types are constraint violations:

Modifiable lvalue (6.3.2.1p1) and assignment (6.5.12p2)
Function parameters (6.7.6.3p3)
Generic selection (6.5.1.p2)
Array subscripting (6.5.2.1p1)
Return types in calls and definitions (6.5.2.2p1,6.9.1p3)
Compound literals (6.5.2.5p1)
Pointer arithmetic (6.5.6p2,3;6.5.16.2p2)
Array elements (6.7.6.2p1)
Initialization (6.7.9p3)
Sizeof and _Alignof (6.5.3.4p1)

Only for sizeof and _Alignof we add an exception to not break existing idiomatic code. Nevertheless, we propose to make using sizeof on a struct with FAM an obsolescent feature and propose to add a new macro that can be safely used instead.

Incomplete types can also cause undefined behavior:

Address constants (6.6p7)
Lvalue conversion (6.3.2.1p2)
Tentative definition with internal linkage (6.9.2p3)

**Example:**

```
struct foo {
  size_t len;
  char buf[];
} f, g;

f = g;      // becomes a constraint violation
```

In the future, we could consider allowing other incomplete types as last elements of a struct (or as element type for the FAM) or allow dynamic length specifiers for bounds checking.

**Proposed Change 1:**

**6.2.5 Types**

24 An array type of unknown size is an incomplete type. It is completed, for an identifier of that type, by specifying the size in a later declaration (with internal or external linkage). A structure or union type of unknown content (as described in 6.7.2.3) is an incomplete type. It is completed, for all declarations of that type, by declaring the same structure or union tag with its defining content later in the same scope. **A structure with a flexible array member is an incomplete type that can not be completed.**

**6.5.3.4 The sizeof and _Alignof operators**

Constraints

1 The sizeof operator shall not be applied to an expression that has function type or an incomplete type **except if the type is a structure with flexible array member**, to the parenthesized name of such a type, or to an expression that designates a bit-field member. The _Alignof operator shall not be applied to a function type or an incomplete type **except if the type is a structure with flexible array member**.

Semantics

4 When sizeof is applied to an operand that has type char , unsigned char , or signed char , (or a qualified version thereof) the result is 1. When applied to an operand that has array type, the result is the total number of bytes in the array. 113) When applied to an operand that has structure or union type, the result is the total number of bytes in such an object, including internal and trailing padding. **When the operand is a structure with flexible array member, the size is as if the flexible array member were omitted except that it may have more trailing padding than the omission would imply.**

**6.7.2.1 Structure and union specifiers**

**Constraints**

3 A structure or union shall not contain a member with incomplete or function type (hence, a structure shall not contain an instance of itself, but may contain a pointer to an instance of itself), except that the last member of a structure with more than one named member may have incomplete array type; ~~**such a structure (and any union containing, possibly recursively, a member that is such a structure) shall not be a member of a structure or an element of an array.**~~

**Semantics**

20 As a special case, the last member of a structure with more than one named member may have an incomplete array type; this is called a flexible array member.

**A structure with a flexible array member is an incomplete type.** ~~**In most situations, the flexible array member is ignored. In particular, the size of the structure is as if the flexible array member were omitted except that it may have more trailing padding than the omission would imply. However,**~~ When a . (or -> ) operator has a left operand that is (a pointer to) a structure with a flexible array member and the right operand names that member, it behaves as if that member were replaced with the longest array (with the same element type) that would not make the structure larger than the object being accessed; the offset of the array shall remain that of the flexible array member, even if this would differ from that of the replacement array. If this array would have no elements, it behaves as if it had one element but the behavior is undefined if any attempt is made to access that element or to generate a pointer one past it.

**6.11 Future language directions**

**6.11.XX Using the sizeof operator on a structure with flexible array member is an obsolescent feature.**

**Proposed Change 2:**

**7.19 Common definitions <stddef.h>**

3 which expands to an integer constant expression that has type size_t , the value of which is the offset in bytes, to the subobject (designated by *member-designator*), from the beginning of any object of type *type;* **and**

**sizeof_struct_fam(type, member-designator, count)**

**which can be applied to a type *type* with flexible array member (designated by *member-designator*), and which expands to an integer expression that has type size_t, the value of which is the size of the structure with flexible array member with *count* elements. The expression is an integer constant expression if *count* is an integer constant expression. If the member-designator does not designate a flexible array member, the behavior is undefined. In these macros, t**he type and member designator shall be such that given