DCL38-C. Use the correct syntax when declaring a flexible array member

Flexible array members are a special type of array in which the last element of a structure with more than one named member has an incomplete array type; that is, the size of the array is not specified explicitly within the structure. This "struct hack" was widely used in practice and supported by a variety of compilers. Consequently, a variety of different syntaxes have been used for declaring flexible array members. For conforming C implementations, use the syntax guaranteed to be valid by the C Standard.

Flexible array members are defined in the C Standard, 6.7.2.1, paragraph 18 [ISO/IEC 9899:2011], as follows:

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

Structures with a flexible array member can be used to produce code with defined behavior. However, some restrictions apply:

1. The incomplete array type must be the last element within the structure.
2. There cannot be an array of structures that contain a flexible array member.
3. Structures that contain a flexible array member cannot be used as a member of another structure.
4. The structure must contain at least one named member in addition to the flexible array member.

Noncompliant Code Example

Before the introduction of flexible array members in the C Standard, structures with a one-element array as the final member were used to achieve similar functionality. This noncompliant code example illustrates how struct flexArrayStruct is declared in this case.

This noncompliant code example attempts to allocate a flexible array-like member with a one-element array as the final member. When the structure is instantiated, the size computed for malloc() is modified to account for the actual size of the dynamic array.

```c
#include <stdlib.h>

struct flexArrayStruct {
    int num;
    int data[1];
};

void func(size_t array_size) {
    /* Space is allocated for the struct */
    struct flexArrayStruct *structP = (struct flexArrayStruct *) malloc(sizeof(struct flexArrayStruct) + sizeof(int) * (array_size - 1));
    if (structP == NULL) {
        /* Handle malloc failure */
    }
    structP->num = array_size;
    /*
    * Access data[] as if it had been allocated
    * as data[array_size].
    */
    for (size_t i = 0; i < array_size; ++i) {
        structP->data[i] = 1;
    }
}
```

This example has undefined behavior when accessing any element other than the first element of the data array. (See the C Standard, 6.5.6.) Consequently, the compiler can generate code that does not return the expected value when accessing the second element of data.

This approach may be the only alternative for compilers that do not yet implement the standard C syntax.
**Compliant Solution**

This compliant solution uses a flexible array member to achieve a dynamically sized structure:

```c
#include <stdlib.h>

struct flexArrayStruct{
    int num;
    int data[];
};

void func(size_t array_size) {
    /* Space is allocated for the struct */
    struct flexArrayStruct *structP = (struct flexArrayStruct *)
        malloc(sizeof(struct flexArrayStruct) + sizeof(int) * array_size);
    if (structP == NULL) {
        /* Handle malloc failure */
    }
    structP->num = array_size;

    /*
    * Access data[] as if it had been allocated
    * as data[array_size].
    */
    for (size_t i = 0; i < array_size; ++i) {
        structP->data[i] = 1;
    }
}
```

This compliant solution allows the structure to be treated as if its member `data[]` was declared to be `data[array_size]` in a manner that conforms to the C Standard.

**Risk Assessment**

Failing to use the correct syntax when declaring a flexible array member can result in undefined behavior, although the incorrect syntax will work on most implementations.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCL38-C</td>
<td>Low</td>
<td>Unlikely</td>
<td>Low</td>
<td>P3</td>
<td>L3</td>
</tr>
</tbody>
</table>

**Automated Detection**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrée</td>
<td>19.04</td>
<td>array_out_of_boun ds</td>
<td>Supported Astrée reports all out-of-bounds array access.</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-DCL38</td>
<td>Detects if the final member of struct which is declared as an array of small bound, is used as a flexible array member.</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td>Can detect some violations of this rule. In particular, it warns if the last element of a struct is an array with a small index (0 or 1)</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>648 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-DCL38-a</td>
<td>The final member of a structure should not be an array of size '0' or '1'</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule DCL38-C</td>
<td>Checks for incorrect syntax of flexible array member size (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>1037 1039</td>
<td></td>
</tr>
<tr>
<td>TrustInSoft Analyzer</td>
<td>1.38</td>
<td>index_bound</td>
<td>Exhaustively detects out-of-bounds array access (see the compliant and the non-compliant example).</td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**
Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

This rule supplements MEM33-C. Allocate and copy structures containing a flexible array member dynamically.

Bibliography

| ISO/IEC 9899:2011 | 6.5.6, "Additive Operators"
| ISO/IEC 9899:2011 | 6.7.2.1, "Structure and Union Specifiers"
| McCluskey 2001 | "Flexible Array Members and Designators in C9X" |