Clarification of Composite Type

David Keaton

2008-08-25

1. Introduction

1.1 Purpose

This proposal specifies a wording change to the C standard to clarify existing intent regarding variable-length array composite types. It provides a response to Defect Report #340 that can be used as C1X wording.

1.2 Scope

This document, although augmenting 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/N1256.

1.3 References

1. ISO/IEC 9899:1999, *Programming Languages—C*. (C99)
2. WG14/N1256, Committee Draft of ISO/IEC 9899:1999+TC1+TC2+TC3.
3. ISO/IEC 9899:1990, *Programming Languages—C*. (C90)
4. WG14/DR340, Myers. "Composite types for variable-length arrays."
5. WG14/DR342, Myers. "VLAs and conditional expressions."
6. WG14 e-mail 11145, Myers. "VLA composite types."
7. WG14/N1238, Myers. "Proposed wording for DR340."
8. WG14/N1267, Hedquist. "Final Minutes for London, April, 2007."
9. WG14/N1270, Hedquist. "Draft Minutes for [Kona,] October, 2007."
10. WG14/N1300, Stoughton. "Draft Minutes for [Delft,] April, 2008."

1.4 Rationale

Defect Report #340 (DR 340) points out a mistake that appeared in the C99 wording when variable-length arrays (VLAs) were added.

The intended behavior when constructing a composite type can be seen from the example in subclause 6.2.7, paragraph 5. This leaves no doubt. Information is gathered from both source types, so that the composite type may contain more information than either source type individually.

In C90, the first point in subclause 6.1.2.6, Compatible type and composite type, paragraph 3 said “If one type is an array of known size, the composite type is an array of that size.” C99 was meant to be extended analogously for VLAs. The first point of subclause 6.2.7, Compatible type and composite type, paragraph 3 says (with italics added to indicate the C99 changes) “If one type is an array of known *constant* size, the composite type is an array of that size*; otherwise, if one type is a variable length array, the composite type is that type*.”

Unfortunately, if this is read literally, the appearance of a VLA stops the recursive determination of composite type, as pointed out in WG14 e-mail 11145. However, note that if the literal interpretation were chosen for the example in e-mail 11145, the code would compile but could then exhibit undefined behavior at run time by calling a parameter-taking function without passing it a parameter.

The intended behavior was clear. The problem was clear. Wordsmithing the solution has been more elusive. The suggested Technical Corrigendum (TC) in DR 340 was “close but not quite right” (WG14/N1267). In the next iteration, WG14/N1238, “The proposed wording has some problems” (WG14/N1270).

In Delft (WG14/N1300), we discussed the DRs in reverse numerical order. We decided that DR 342, a subcase of DR 340, should have undefined behavior. The next DR to be discussed was DR 340, and it was mentioned that perhaps this also should have undefined behavior. I believe this was an error caused by discussing the specific case before the general case, since consistency with the rest of that subclause makes the intent clear.

I have provided wording below in an attempt to clarify the intended behavior, while continuing to make the DR 342 subcase undefined.

1.5 Impact

This proposal clarifies existing intent.

Some compilers have interpreted the C99 wording in the intended way, some have interpreted it literally (not in the intended way), and some terminate abnormally upon seeing code that exercises it. Therefore, there is impact to some existing translators. This is balanced by the benefits of making the behavior consistent.

Code that compiled under a literal interpretation, but would fail to compile under the intended interpretation, would have caused undefined behavior at run time. Therefore, the impact on existing code is to move bug detection from run time (if ever) to compile time, which is a positive result.

1.6 Acknowledgments

Thanks to Tom Plum for serving as a sounding board for this proposal.

2. Language

In subclause 6.2.7, paragraph 3, change the first bullet to the following.

– If both types are array types, the following rules are applied:

> If one type is an array of known constant size, the composite type is an array of that size.
>
> Otherwise, if one type is a variable length array whose size is specified by an expression that is not evaluated, a composite type cannot be constructed.
>
> Otherwise, if one type is a variable length array whose size is specified, the composite type is a variable length array of that size.
>
> Otherwise, if one type is a variable length array of unspecified size, the composite type is a variable length array of unspecified size.
>
> The element type of the composite type is the composite type of the two element types.