Clarification of Expressions

David Keaton

2008-03-02

1. Introduction

1.1 Purpose

This proposal specifies a wording addition to the C standard to clarify existing intent regarding the ordering of subexpressions within a full expression.

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/DR087, Feather. "Order of Evaluation."
5. WG14/DR117, Guilmette. "Abstract semantics, sequence points, and expression evaluation."
6. ISO/IEC 14882:1998, *Programming Languages—C++*. (C++98)

1.4 Rationale

There has been an understanding within the committee that an expression exhibits undefined behavior if any valid ordering of subexpressions would cause undefined behavior. This means, for example, that a programmer cannot claim "I can find an ordering that would not trigger the undefined behavior, therefore the expression is not undefined." However, the understanding has not yet been communicated in either the standard or the rationale. As a result, there has sometimes been confusion about it.

Subclause 6.5, Expressions, paragraph 2 currently reads as follows.

> Between the previous and next sequence point, an object shall have its stored value modified at most once by the evaluation of an expression.[72)] Furthermore, the prior value shall be read only to determine the value to be stored.[73)]

The same paragraph existed in C90. DR087 quoted the first sentence of this paragraph and asked how

to interpret the standard when applying it to the marked lines of the following program.

```
int g;

int main(void)
{
     int x;

     x = (10, g = 1, 20) + (30, g = 2, 40); /* Line A */
     x = (10, f(1), 20) + (30, f(2), 40); /* Line B */
     x = (g = 1) + (g = 2); /* Line C */
     return 0;
}

int f(int i)
{
     g = i;
     return 0;
}
```

The committee's response to DR087 was a definite reference to the understanding, but did not state it explicitly.

> In line B, the expression does not exhibit undefined behavior, but because the order of evaluation of the operands of the addition operator is not specified and function calls do not overlap, it is unspecified whether **g** will attain the value 1 or 2. Lines C and A violate the quoted restriction from subclause 6.3 [6.5 in C99], so the behavior is undefined.

This was the first time that evidence of the principle was placed in writing.

DR117 then presented the same quote and a program with an equivalent problem, along with the more specific question “Does the following code involve usage which renders the code itself not strictly conforming?” It drew the following response.

> The C Standard does not forbid an implementation from interleaving the subexpressions in the given example as specified above. Similarly, there is no requirement that an implementation use this particular interleaving. It is irrelevant that one particular interleaving yields code that properly delimits multiple modifications of the same object with sequence points. Any program that depends on this particular interleaving is depending on unspecified behavior, and is therefore not strictly conforming.

This adds some information, but bases its rejection of the program on unspecified behavior rather than undefined behavior. Since DR117's question was more specific, the committee was not required to go beyond this explanation. The missing link is the principle that if it is unspecified whether undefined behavior occurs, then the behavior is undefined. In other words, the more severe problem wins. That principle leads to the DR087 response.

C++98 clause 5, Expressions, paragraph 4 (known as 5p4) adopted the C wording and followed the reasoning through to the ultimate conclusion from DR087, adding the following sentence to document the understanding.

> The requirements of this paragraph shall be met for each allowable ordering of the subexpressions of a full expression; otherwise the behavior is undefined.

The proposal here is to add the above sentence to the C standard as well.

There are two benefits. First, it clarifies the committee's intent so that the implementor and user communities have unambiguous documentation of the state of the language. Second, it provides a solid base from which to discuss further C1X proposals, ensuring that committee members remember the status quo before attempting to make changes to it.

This second benefit is important as a prelude to discussing potential changes to the sequence point model. If sequence point changes are adopted and render this paragraph obsolete, the clarification will still have served its purpose in the interim. If sequence point changes are not adopted, the benefit to implementors and users applies as well.

1.5 Impact

This proposal does not impact existing code or implementations. It clarifies existing intent.

2. Language

The following sentence is added to the end of subclause 6.5, paragraph 2.

> The requirements of this paragraph shall be met for each allowable ordering of the subexpressions of a full expression; otherwise the behavior is undefined.