N3503 - Integer and arithmetic constant expressions

Author: Javier A. Múgica

The problem and other issues

The standard uses the sentence *the result is an integer constant expression*, applied to **`sizeof`** and **`alignof`** expressions. This is not right, since what is an integer constant expression is the expression itself, as the term conveys; the result in those cases is an integer constant; it is a property of the value of the expression, or of the evaluation of the expression, and exists only if the expression is evaluated. *constant expression* is something which, among other requirements, has the form of a conditional expression, so it cannot be referred to as the result of an evaluation.

"Integer constant" was the term used until recently in those places, which was changed by the adoption of the proposal N3239 *Some constants are literally literals*. As that proposal argues, the use of "integer constant" there was incorrect because this term is defined as a certain category of tokens. That paper changed the term to "integer literal" and changed the sentence *the result is an integer constant* to append *expression* after it. In other places, the proposal changed this to "constants of integer type".

We like the choice of N3239 of talking about **`sizeof`** and **`alignof`** expressions as integer constant expressions, but has to be expressed in some other form. For this, a wording is proposed in "First change".

The second change proposes a simplification of the wording for arithmetic constant expressions. The third change proposes a reelaboration of the wording for those two kinds of consant expressions, which are defined by recursion.

The recursion of the third change needs the definition of *candidate integer constant expression* (CICE) and *candidate arithmetic constant expression*. The paper "Resolved & discarded" also proposes changes to the wording of these two kinds of expressions. The wordings in that paper are in substance: they enlarge the kind of expressions allowed by permitting any expressions with an adequate type in subexpressions of the integer / arithmetic constant expression that are not evaluated. The changes in this paper are in form. The combination of both papers would result in a form like this one with the enlargement of the other proposal. This makes the wording in the form in this paper *simpler*. The reason is that currently the kind of expressions allowed in not evaluated subexpressions of an integer constant expressions are very limited; precisely the CICE. With the changes in "Resolved & discarded", the need for this term disappear. Analogously for arithmetic constant expressions.

Wording

First change

In 6.5.4.5 The **`sizeof`**, **`_Lengthof`** and **`alignof`** operators, pp. 2, 3 and 5, change

the result is an integer constant expression *to* the expression is an integer constant expression

In 6.6 Constant expressions, 6.6.1 General, pp. 8 and 10 and in appendix J.2, items 50 and 52, change

whose results are integer constant expressions *to* which are integer constant expressions

In 6.9 External definitions, in the constraints,

— part of the operand of a **`sizeof`** operator whose result is an integer constant expression;

— part of the operand of a **_Lengthof** operator whose result is an integer constant expression;

— part of the operand of an **alignof** operator whose result is an integer constant expression;

change those items to

— part of the operand of a **sizeof** expression which is an integer constant expression;

— part of the operand of a **_Lengthof** expression which is an integer constant expression;

— part of the operand of an **alignof** operator;

Second change

In the paragraph on arithmetic constant expressions, the operands "integer literals", "character literals" **sizeof**, **_Lengthof** and **alignof** expressions can be included in "integer constant expressions":

10 An *arithmetic constant expression* shall have arithmetic type and shall only have operands that are floating literals, named or compound literal constants of arithmetic type and integer constant expressions. Cast operators in an arithmetic constant expression shall only convert arithmetic types to arithmetic types, except as part of an operand to the typeof operators, **sizeof** operator, **_Lengthof** operator or **alignof** operator.

Third change

Finally, we understand that the current wording does not allow an expression like `3` or `'a'` to be an ICE by itself, for example in `int a[3];` or `case 'a':`, since there `3` and `'a'` are not operands of anything. The opposite interpretation; namely, that becuase they are not operands none of those *shall*'s apply to them, would make an ICE of any identifier with integer type, which is obviously not. Even if it is understood that it does allow those expressions, we believe it is clearer to state that literals of the appropriate type are integer/arithmetic constant expressions.

Furthermore, it is not clear that **_Generic** selections may include other kinds of expressions in the generic associations which are not taken. Also, if the generic association which is taken is a literal, it should be allowed where the literal is.

We propose the following wording in accordance to this:

8 An *outermost operand* of an expression or typeof specifier is an operand of the same for which there does not exist another expression or type name lying strictly between the operand and the expression or typeof specifier.

9 *candiate integer constant expressions* and *integer constant expressions* are defined recursively as follows:

10 Every integer constant expressions is a candiate integer constant expression. A candiate integer constant expression that satisfies the constraints for constant expressions is an integer constant expression.

11 The following are integer constant expressions:

— Integer literals, named and compound literal constants of integer type and character literals.

— **alignof** expressions and some **sizeof** and **_Lengthof** expressions, as specified in 6.5.4.5.

12 For the purposes of these definitions, a *castable float* is defined recursively as any of:

— A floating literal.

— A named or compound literal constant of floating type.

— A parenthesized castable float.

— A generic selection where the result expression is a castable float.

13 The following are candiate integer constant expressions (CICE):

— A cast expressions where the type of the cast is an integer type and the operand is a castable float.

— A parethesized CICE

— An expression of integer type other than a compound literal or generic selection whose outermost operands are CICE.

— A generic selection where the result expression is a CICE.

15 The following are *candidate arithmetic constant expressions*: candiate integer constant expressions, floating literals and named or compound literal constants of arithmetic type. Other candiate arithmetic constant expressions (CACE) are defined recursively:

— a parethesized CACE.

— An expression of arithmetic type other than a compound literal or generic selection whose outermost operands are CACE.

— A generic selection where the result expression is a CACE.

16 An *arithmetic constant expression* is a candiate arithmetic constant expressions that satisfies the constraints for contant expressions.

Notes on the third change

1. "castable float" has been restricted to proper floats (not integers) to avoid some expressions being CICE through two different productions, as `(int)true`.

2. The definition of certain **`sizeof`** and **`_Lengthof`** expressions as integer constant expressions at the place these operands are defined are part of, and at the same time need, the recursive definition, for if the operand to any of those operators is the type name of an array type, the expression is or is not an i.c.e. according to whether the size of the array is given by an i.c.e. or not (and that of the element's type, if it is itself an array, etc.). The same is true for named and compound literal constants, where the initializer must be an integer constant expression.

3. The definition of *outermost operand* has been introduced in the section on constant expressions. We believe it would be better to define that concept in the first subsection of 6.5 Expressions, based on the concept, also to be defined there, of *proper* kind *expression*. This one would be so that, for example, both

```
a * b                  a + b
```

are additive expressions but only the second one is a proper additive expression.

We think the definition of this concept is not difficult. The specification should note that the syntax defines a grading of the diferent kinds of expressions, so that

$$E_{\text{primary}} \subset E_{\text{postfix}} \subset E_{\text{unary}} \subset \text{etc.}$$

where all inclusions are proper. Then, a certain expression is a proper expression of the first kind in this list in which it is included.

We prefer to introduce this concept in a separate paper.