Author: Javier A. Múgica

Purpose: Definition of terms

Date: 2025 - may - 12

This paper proposes the definition of *to evaluate* when applied to a type name. Currently, the term is only defined for expressions but is used both for expressions and type names.

To evaluate a type

What does "to valuate a type" mean? The standard defines *evaluation* as something that applies to expressions:

> *Evaluation* of an expression in general includes both value computations and initiation of side effects. Value computation for an lvalue expression includes determining the identity of the designated object.

When applied to a type name it can only mean evaluation of the subexpressions contained therein. Hence "not to evaluate" would mean exactly what it says. But that is no what is understood:

```
sizeof(int[3+1])
```

Here, `3+1` needs to be evaluated. But according to the standard, the operand `int[3+1]` is not evaluated.

The justification to this is that the expression is part of the type name. However, what is part of the type name is *the result* of the evaluation of the expression. The expression is evaluated, unambiguously.

This evaluation goes to the point that it is performed even when the value is not needed:

```
sizeof(int(*)[1-3])    //Error
```

The reason being, again, that the translator needs to evaluate the expression because it needs to resolve the number of elements of the array.

This behaviour is well understood, but it is not what the standard says. The document must make clear that integer constant expressions that are part of a type name are evaluated as part of the resolution of the type: the determination of the type that the type name names. Thence, the value of the expression becomes part of the type, and the expression "disappears", as though it were no more, for the purposes of "not to evaluate" when applied to the type name.

Proposed wording

5.2.2.4 Program semantics

3 *Evaluation* of an expression in general includes both value computations and initiation of side effects. Value computation for an lvalue expression includes determining the identity of the designated object. Evaluation of a type name is is defined in (6.7.8).

6.7.8 Type names

4 Integer constant expressions that are part of a type name are evaluated during translation. The resulting value becomes part of the type and the expression thereby disappears (it is not

considered part of any expression including the type name). Expressions that are part of the type name and are not integer constant expressions are evaluated at runtime whenever the type name is reached and is evaluated. The term "to evaluate", when applied to a type name, refers to this runtime evaluation and the retrieval of values of other non-integer constant expressions on which the type name depends. What said in this paragraph applies also to the type name implicit in a declaration.

EXAMPLE 2

```
extern int n;

void func(void)
{
      int A[n];
      sizeof(int(*)[3+1][n+1]);
      sizeof(typeof(A)[n+2]);
}
```

The operand of the first **sizeof** is not evaluated. This does not apply to the expression `3+1`, that is evaluated as part of the resolution of the type; the expression `n+1` is not evaluated. The operand of the second **sizeof** is evaluated. The array length of `A` is retrieved and the expression `n+2` is evaluated.