**Author:** Javier A. Múgica

**Purpose:** Clarification

**Date:** 2025 - may - 8

This paper intends to make clear the status of generic selections and parentheses surrounding expressions of a certain kind, when at some points the standard mandates an expression of that kind.

# Analysis

A literal is a kind of token at translation phase 7, endowed with a type and a value. Its definition is very precise, so that when a literal is mandated it does not follow that a parenthesised literal is allowed, neither a generic selection where the selected expression is a literal. However, it is tacitly agreed that these are allowed, unless the mandate comes from a syntax rule. Here follow cases where one or the other happens.

## Cases

### Literals in constant expressions

Integer constant expressions allow literals of arithmetic type as operands to casts. The following code is accepted by compilers without warnings:

```
enum A{a=(int)(_Generic(0, int: (2.5)))};
```

Here, parentheses surround the literal `2.5` and the generic selection.

More generally, constant expressions of different kinds allow only operands of certain types, but any operand in _Generic which is not the selected expressions is not considered for this:

```
constexpr int a= _Generic((float*)0, default:0);
```

### String literals as initializers

Here the status of parentheses is made explicit by the standard: ... a string literal, optionally enclosed in braces, ...

```
const char s[]= ("word");
const char s[]= _Generic(0, default: "word");
```

Gcc warns on "Array initialized from parenthesis", while the other compilers tested give no warning (clang, MSVC and ICX). No compiler warns because of **`_Generic`**.

### Null pointer constant

The standard currently defines

> An integer constant expression with the value 0, such an expression cast to type **`void`***, or the predefined constant **`nullptr`** is called a *null pointer* constant.

The following is accepted by compilers with no warning:

```
float (*f)(void) = (_Generic(1, default: (void*)0));
```

### static_assert

Here, neither parentheses nor _Generic are allowed:

```
static_assert(1, ("Error message")); // Error
```

and similarly for **_Generic**.

### attribute messages

The situation here is the same as for **static_assert**:

```
[[nodiscard(("Do not discard this"))]] int important_func(void); // Error
```

In both attributes and static_assert the string literal is part of the syntax of the feature.

## Solutions considered

### Quasi-literal

Our first choice was the definition of *quasi-literal,* to be placed in the section on primary expressions:

> The following are *quasi-literals:*
>
> — A literal.
>
> — A generic selection where the selected expression is a quasi-literal.
>
> — A parenthesized quasi-literal.
>
> A quasi-literal is of the same kind as the literal on which it is based: integer quasi-literal, string quasi-literal, etc.

And use quasi-literal instead of literal in a few places.

### Generic replacement

The term quasi-literal is of no use for null pointer constants. It is for this reason that the concept *generic replacement* was conceived:

> *Generic replacement* refers to the process of replacing a generic selection by its result expression, enclosed in parentheses if the expression is not a primary expression. (Therefore, the result of generic replacement is always a primary expression).

Using this term, the definition of null pointer constant would become

> A *null pointer* constant is an expression that, after generic replacement and removal of all surrounding parentheses, is an integer constant expression with the value 0 or such an expression cast to type **void** *, or the predefined constant **nullptr**.

### Quasi-constant

While the term *generic replacement* serves well for null pointer constants and string literals as initializers, the wording for floating operands in integer constant expressions remains very verbose, in part because it is not only literals that are allowed, but “floating, named, or

compound literal constants of arithmetic type". All this must be subject to generic replacement and parentheses removal, resulting in the wording seen in the proposal.

To simplify that wording, the use of a term, *quasi-constant* or other, is proposed. Its definition is not placed under "primary expressions", where we intended that of quasi-literals to be, but in the section for constant expressions, because it is only applied there. We propose it separate from the main re-wording proposal.

Our first thought for the name of the term was *quasi-constant*. We are not satisfied with it. Contrary to quasi-literals, where the term emphasizes that they are not literals, we do not want that bias in the term for constants. *atomic constant* was considered, since they are the atomic pieces out of which integer or arithmetic constant expressions are build. But *atomic* already has a very different meaning. Continue with the idea that they are the atomic pieces, *primary constant* was also considered. It hints that they are primary expressions, which is true except for the compound literal constant. Other terms that came to mind are *root, seed, idecomposable*. The latter can lead to confusion. The other two follow the argument that they are the atomic pieces for constant expressions. Along the same lines, the term *immediate constant* seems better. The term *immediate* is only used by the standard in the construction *immediate operand*, and only twice: once in an example and the other one precisely hrer, in *immediate operands of casts*.

Finally, we propose to the committee two choices: *quasi-constant* and immediate constant.

## Not addressed

The wording we propose for integer constant expressions takes care of generic selections and surrounding parentheses for the literals and constants of arithmetic type that are allowed as operands to casts. We do not take care of any operand that may be in that situation, for integers or for arithmetic constant expressions, as for example

```
_Generic(sqrt(2.0), default: sizeof(float))
```

In order to handle this in the wording, an "**after generic replacement**" would have to be inserted preceding the enumeration of all possible operands.

We do not do that because we believe that this is better achieved by a deeper change of the wording for these two kinds of expressions, that would list the atomic ones and then a point saying that an expression is an ICE if its operands are either discarded by the expression or ICEs, and similarly for ACE. That change is the subject of another proposal.

This notwithstanding, the introduction of the term *quasi-constant*, in addition of simplifying the wording, handles generic selections for all cases when the result expression is a constant or literal.

# Wording

## Main proposal

Add at the end of the semantics of generic selections, the following:

*Generic replacement* refers to the process of replacing a generic selection by its result expression, enclosed in parentheses if the expression is not a primary expression. (Therefore, the result of generic replacement is always a primary expression). When the term is applied to

an expression, it refers to the application of generic replacement in it till no generic selections remain.

Change also, in "6.3.3.3 Pointers":

A *null pointer* constant is an expression that, after generic replacement and removal of all surrounding parentheses, is an integer constant expression with the value 0 or such an expression cast to type **`void`** `*`, or the predefined constant **`nullptr`**.

And maybe add a forward reference to generic selection.

In 6.6 Constant expressions,

An *integer constant expression* shall have integer type and shall only have operands that are integer literals, named and compound literal constants of integer type, character literals, **`sizeof`** or **`_Countof`** expressions which are integer constant expressions, **`alignof`** expressions, and floating, named, or compound literal constants of arithmetic typethat are the immediate operands of casts.; when these operands have floating type, they shall only appear in cast expressions of integer type where the operand, after generic replacement, is such an operand optionally enclosed in an arbitrary number of parentheses. Cast operators in an integer constant expression shall only convert arithmetic types to integer types, except as part of an operand to the typeof operators, **`sizeof`** operator, **`_Countof`** operator, or **`alignof`** operator.

In 6.7.11 Initialization,

The initializer for an array shall be either a string literal optionally enclosed in braces , possibly after generic replacement, or a brace-enclosed list of initializers for the elements.

## Quasi-constant

With the introduction of a term here, the text on integer and arithmetic constant expressions reduces to the following. We have used *immediate constant*, but propose also *quasi-constant* in its place.

8 An *immediate constant* is an expression that, after generic replacement and removal of surrounding parentheses, is a literal, a compound literal constant or a named constant.

9 An *integer constant expression* shall have integer type and shall only have operands that are immediate constants of integer type, **`sizeof`** or **`_Countof`** expressions which are integer constant expressions, **`alignof`** expressions, and immediate constants of arithmetic type that are the immediate operands of casts. Cast operators in an integer constant expression shall only convert arithmetic types to integer types, except as part of an operand to the typeof operators, **`sizeof`** operator, **`_Countof`** operator, or **`alignof`** operator.

11 An *arithmetic constant expression* shall have arithmetic type and shall only have operands that are immediate 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, **`_Countof`** operator, or **`alignof`** operator.