**Proposal for C2y**
**WG14**

**Document Number**: N3809
**Author:** Abdulmalek Almkainzi <aalmkainzi@gmail.com>
**Title:** Unselected _Generic branches should be ignored
**Proposal category:** New Features
**Target Audience:** General Developers

**Abstract:**
This paper proposes a solution to the problem of unselected branches in _Generic expressions still needing to be valid expressions.

# Table of Contents

Prior Art....2
Introduction and Rationale....2
Proposal....3
Proposed Wording....3
Limitations....4
References....4

# Prior Art

The proposal N3785 [1] aims to solve the same problem as this one. This paper demonstrates an alternative approach, which can work even when using a type as the controlling operand in a _Generic expression. Also, it is generally easier to understand (as it is how most people thought _Generic originally worked [2] [3]).

The TCC compiler already implements _Generic the way this paper proposes [4].

# Introduction and Rationale

A common problem when using _Generic is that all branches must be valid expressions, even if unselected. For example:

```
struct MyString {
    char *chars;
    size_t len;
};

#define string_length(s) \
_Generic(s, \
char*:  strlen(s), \
struct MyString: s.len \
)
```

The macro `string_length` may looks simple enough, but it actually will never work. This is because if `s` is `char*` then the second branch's expression is not valid, and if it's a `struct MyString` then the first branch's expression is not valid.

Achieving the desired behavior requires using a technique often called "type coercion" or "contrav". The way it works is, instead of using `s` directly, turn it into the current branch's type if it isn't already of that type:

```
#define coerce_type(s, T) \
_Generic(s, T: s, default: (T){})

#define string_length(s) \
_Generic(s, \
char*:  strlen(coerce_type(s, char*)), \
struct MyString: coerce_type(s, struct MyString).len \
)
```

Godbolt link.

# Proposal

This paper proposes that unselected _Generic branches should be parsed as outer-balanced-token-sequence, which are identical to balanced-token-sequence, but without commas. Balanced-token-sequence is defined in the standard as:

```
balanced-token-sequence:
    balanced-token
    balanced-token-sequence balanced-token

balanced-token:
    ( balanced-token-sequence-opt )
    [ balanced-token-sequence-opt ]
    { balanced-token-sequence-opt }
    any token other than a parenthesis, a bracket, or a brace
```

The proposed outer-balanced-token-sequence would produce outer-balanced-token instead of balanced-token. The difference being that outer-balanced-token cannot have a comma outside of parenthesis, a brackets, or braces.

The one _Generic branch that matches the type of the operand would be parsed as an assignment-expression, while every other branch would be parsed as outer-balanced-token-sequence.

With that, the previous example that needed `coerce_type` to work doesn't need it anymore, because unselected branches are allowed to be invalid expressions:

```
#define string_length(s) \
_Generic(s, \
char*: strlen(s), \
struct MyString: s.len \
)
```

This macro would work as expected. If a `char*` is passed, then `s.len` is parsed as an outer-balanced-token-sequence. And if a `struct MyString` is passed, then `strlen(s)` is parsed as outer-balanced-token-sequence.

# Proposed Wording

Two new productions for `generic-association`:

```
(6.5.2.1) generic-association:
    type-name : assignment-expression
    default : assignment-expression
    type-name : outer-balanced-token-sequence
    default : outer-balanced-token-sequence
```

```
outer-balanced-token-sequence:
    outer-balanced-token
    outer-balanced-token-sequence outer-balanced-token

outer-balanced-token:
    ( balanced-token-sequence-opt )
    [ balanced-token-sequence-opt ]
    { balanced-token-sequence-opt }
    any token other than a parenthesis, a bracket, a brace, or a comma
```

# Limitations

In this proposal, _Generic branches can contain errors which are not caught by the compiler, unlike N3785 , which still requires that all branches must be valid expressions. This may lead to mistakes going unnoticed. The upside is that it will make usage of _Generic simpler overall.

# References


[1]: N3785: Expression Evaluation and Access in _Generic

[2]: Problem with _Generic : r/C_Programming

[3]: Workarounds for C11 _Generic

[4]: https://godbolt.org/z/E6rGPdhx7