ISO/IEC JTC1 SC22 WG14 N2023

ISO/IEC JTC1 SC22 WG21 D00306r0
Date: 2016-03-07
To: EWG, CWG, WG14 liaison
Thomas Köppe <tkoeppe@google.com>

Comma omission and comma deletion

Contents

Revision history

D00306r0: Initial proposal.

Summary

This is a proposal to make variadic macros easier to use with no arguments.

Details

Function-style macros that can have variable arguments work well if at least one actual argument is provided in the macro invocation. For example, consider the following macro definition:

```
#define F(X, ...) f(10, X, __VA_ARGS__)
```

The invocation `F(a, b, c)` is replaced by `f(10, a, b, c)`. However, invocations without *any* arguments matching the ellipsis work less well. The invocation `F(a)` is not allowed by the preprocessor rules, and the allowed invocation `F(a, )` results in the replacement `f(10, a, )`, which is a syntax error.

However, it is natural for a macro invocation with variable arguments to degenerate to the case where there are no arguments at all. In the example, we would like `F(a)` to be replaced with `f(10, a)`. A more realistic example is a custom diagnostic facility such as the following:

```
#define ERROR(msg, ...) std::printf("[" __FILE__ ":%d] " msg,
__LINE__, __VA_ARGS__)

ERROR("%d errors.\n", 0);  // OK, std::printf("[" "file.cpp"
":%d] " "%d errors.\n", 7, 0);
ERROR("No errors.\n");     // Error
```

The proposal is to allow the macro invocation without any variable arguments. This requires two changes to the core language:

1. Allow the omission of the comma before the variable arguments in the invocation (i.e. allow `F(a)` rather than requiring `F(a, )`). This behaviour is already supported by many popular compilers as a non-conforming extension.
2. Provide a mechanism to delete an existing comma from the replacement text (i.e. delete the last comma from `f(10, a, )` to produce `f(10, a)`). This behaviour should not occur automatically, but require opt-in. Many platforms already offer a non-conforming extension that enables this behaviour, namely the syntax `, ## __VA_ARGS__` in the replacement text. This syntax reuses the concatenation operator `##`, which cannot appear in that position in conforming code.

We propose to standardise both of the described extensions. With this proposal, we can define:

```
#define F(X, ...) f(10, X, ## __VA_ARGS__)
```

Now `F(a)` is replaced by `f(10, a)`.

Impact

The proposal is a pure extension of the preprocessor. Syntax that was previously not allowed becomes admissible under the proposed changes.

Implementation experience

The proposed extensions are already implemented in GCC and Clang and are widely used throughout many code bases. Many real-world macros would not work without these extensions.

Proposed wording

Comma omission

Change paragraph 16.3p4 as follows.

> If the *identifier-list* in the macro definition does not end with an ellipsis, the number of arguments (including those arguments consisting of no preprocessing tokens) in an invocation of a function-like macro shall equal the number of parameters in the macro definition. Otherwise, there shall be ~~more~~at least as many arguments in the invocation ~~than~~as there are parameters in the macro definition (excluding the `...`). There shall exist a `)` preprocessing token that terminates the invocation.

Change paragraph 16.3p12 as follows.

> If there is a `...` immediately preceding the `)` in the function-like macro definition, then the trailing arguments (if any), including any separating comma preprocessing tokens, are merged to form a single item: the variable arguments. The number of arguments so combined is such that, following merger, the number of arguments is either equal to or one more than the number of parameters in the macro definition (excluding the `...`).

Comma deletion

Insert a new paragraph into subsection 16.3.3 between existing paragraphs 2 and 3.

> If the replacement list of a function-like macro contains the token sequence `, ## __VA_ARGS__`, and if the variable arguments consist of no tokens, and if the number of arguments in the macro invocation is equal to the number of parameters in the macro definition (excluding the `...`), the comma is removed from the resulting replacement. [Example: In the following fragment:

```
#define F(x, ...) f(0, x, ## __VA_ARGS__)

F(a, b, c)  // more arguments than parameters
F(a, )      // more arguments than parameters
F(a)        // equal number of arguments and parameters
```

The expansion produces:

```
f(0, a, b, c)
f(0, a,)
f(0, a)
```

– end example]

Compatibility with C

The entire proposal (rationale, implementation experience and wording) applies almost verbatim to the C language as well. (For the wording changes, the C++ section 16.3 corresponds to the C section 6.10.3.) We would like to ask the WG14 liaison to discuss this proposal with WG14 and provide feedback, and we would like to encourage WG14 to adopt the same extension for C in the interest of future compatibility.