Variadic-argument introspection: `__VA_COUNT__` and `__VA_SLICE__`

Kamila Szewczyk, Universität des Saarlandes

2026-01-15

Document number:	**N3792**
Date:	2026-01-15
Project:	ISO/IEC 9899:202y
Target:	Integration into the next revision (“C2y”)
Proposal category:	New feature (preprocessing)
Audience:	WG14
Reply-to:	`k@iczelia.net`
License:	CC BY 4.0

Abstract

This paper proposes two new predefined function-like macros for the C preprocessor:

- `__VA_COUNT__(...)` expands to the number of macro arguments in its invocation, including empty arguments, as a single decimal preprocessing-number token.
- `__VA_SLICE__(start, len, ...)` expands to a comma-separated subsequence of the variadic argument list, preserving empty arguments, suitable for forwarding as `__VA_ARGS__`.

Together these features provide direct, portable support for common “variadic macro metaprogramming” patterns (counting, selection, head/tail, forwarding) without the complexity and fixed maximum arity typical of macro-library workarounds.

Contents

1 Introduction

C23 standardized `__VA_OPT__` to improve usability of variadic macros with zero arguments. However, the standard still provides no direct facility to (1) determine the number of arguments in a variadic macro invocation, or (2) forward a subsequence of a variadic list while preserving empty arguments.

These capabilities are routinely emulated in macro libraries using intricate techniques with hard-coded maximum arities, and are widely regarded as "nasty to program" despite being straightforward for an implementation. [2]

This paper proposes two predefined function-like macros to address these needs.

1.1 Previous work

- **N3190** (*Extensions to the preprocessor for C2Y*) discusses several candidate predefined function-like macros, including two argument-counting facilities, `__COMMAS__` and `__NARGS__`. [2]
- **Deniau (comp.std.c, 2006)**: a long-cited posting titled `__VA_NARG__` argues that counting the number of arguments in `__VA_ARGS__` is "rather easy to compute for cpp" and that the standard should provide such a facility; it also archives a representative macro workaround with an explicit maximum arity. [4]
- **GCC PR33877 (2007–2008)**: a GCC enhancement request asks for a predefined `__VA_ARGC__` to expose the number of variadic arguments, citing difficulty of doing so portably without double-evaluating arguments; follow-up discussion notes this avoids "busy-work clutter prone to unchecked error" when forwarding to real variadic functions. [5, 6]
- **N3307** (*Tail recursion for preprocessor macros*) motivates further variadic-list processing following standardization of `__VA_OPT__` and uses an operation named `__VA_TAIL__` in examples of scalable list processing. [7]

2 Motivation

2.1 Portability and simplicity

Many macro packages need the arity of `__VA_ARGS__` to:

- select among overload-like macro definitions (*arity dispatch*);
- provide a count argument to a varargs function wrapper.

Extracting the n-th element or taking the tail of a variadic list is similarly common. Both operations are feasible for a preprocessor implementation once arguments are parsed, yet are difficult to express portably in standard macro language without imposing a fixed maximum.

2.2 Preserving empty arguments

C macro argument parsing permits empty arguments. For forwarding and transformation, it is often necessary to preserve empties rather than "compress" them away. A slicing primitive that preserves empty arguments is a direct building block for such forwarding.

3 Proposed features

3.1 `__VA_COUNT__(...)`

Synopsis

```
__VA_COUNT__( /* zero or more arguments */ )
```

Semantics

Let the invocation be:

$$\texttt{\_\_VA\_COUNT\_\_(}A_1\texttt{, }A_2\texttt{, ..., }A_m\texttt{)}$$

where m is the number of arguments as determined by the preprocessor's *function-like macro argument parsing* rules (top-level commas separate arguments; parentheses nesting is respected; empty arguments are permitted).

The expansion of `__VA_COUNT__` is a single *preprocessing-number* token denoting the decimal integer value m.

Notable cases

- `__VA_COUNT__()` expands to `0`.
- `__VA_COUNT__(x)` expands to `1`.
- Empty arguments count: `__VA_COUNT__(x, , y)` expands to `3`.
- Parenthesized commas do not split arguments: `__VA_COUNT__((a,b), c)` expands to `2`.

Suggested Constraints

- The value m shall be representable as `signed long`.
- The expansion shall not contain digit separators.

3.2 `__VA_SLICE__(start, len, ...)`

Synopsis

```
__VA_SLICE__(start, len, /* zero or more arguments */ )
```

Intended use

Produces a comma-separated subsequence of the variadic argument list, preserving empty arguments, suitable for forwarding as `__VA_ARGS__` to another macro.

Parsing and evaluation of `start` and `len`

After macro expansion, each of `start` and `len` is evaluated as if it were the controlling expression of a `#if` group (i.e., with preprocessing-expression evaluation rules). The resulting integer values are:

$$s = \text{start} \qquad \text{and} \qquad \ell = \text{len}.$$

(Equivalently: `__VA_SLICE__` accepts arithmetic expressions for `start` and `len` as in `#if` contexts, and uses the resulting integer values.)

Constraints

- s shall satisfy $s \geq 1$.
- ℓ shall satisfy $\ell \geq 0$.
- s and ℓ shall be representable as `signed long`.

Semantics

Let the invocation be:

$$\texttt{\_\_VA\_SLICE\_\_(s, } \ell\texttt{, A}_1\texttt{, A}_2\texttt{, ..., A}_\texttt{m}\texttt{)}$$

where m is the number of *variadic* arguments following the first two arguments, determined by ordinary macro-argument parsing rules (including empty arguments). Define:

- If $\ell = 0$ or $s > m$, the expansion is the empty preprocessing-token sequence.
- Otherwise let $t = \min(m, s + \ell - 1)$.
  Then the expansion is:

$$\texttt{A}_\texttt{s}\texttt{, A}_{\texttt{s+1}}\texttt{, ..., A}_\texttt{t}$$

with a comma preprocessing-token between each adjacent pair. Each $\texttt{A}_\texttt{k}$ is the original preprocessing-token sequence of that argument (including the possibility that it is empty). Selected arguments are macro-expanded as ordinary macro arguments are expanded when substituted (i.e., fully macro replaced prior to emission), while unselected arguments are not required to be macro-expanded.

Examples

```
__VA_SLICE__(1, 3, a, b, c, d)     // -> a, b, c
__VA_SLICE__(3, 99, a, b, c, d)    // -> c, d
__VA_SLICE__(2, 2, a, , c, d)      // -> , c    (first selected argument is empty)
__VA_SLICE__(5, 1, a, b, c)        // ->        (empty)
__VA_SLICE__(1, 0, a, b)           // ->        (empty)
```

3.3 Derived operations

Using `__VA_SLICE__` and `__VA_COUNT__`:

```
#define __VA_PICK__(n, ...)  __VA_SLICE__(n, 1, __VA_ARGS__)
#define __VA_HEAD__(...)     __VA_SLICE__(1, 1, __VA_ARGS__)
#define __VA_TAIL__(...)     __VA_SLICE__(2, __VA_COUNT__(__VA_ARGS__) - 1,
    __VA_ARGS__)
```

4 Design considerations

4.1 One-based indexing

`__VA_SLICE__` indexes the variadic list with $s \geq 1$ to align with typical "first argument is 1" conventions for macro-argument selection and to avoid ambiguity with "no selection" in the presence of $\ell = 0$.

4.2 Empty arguments and commas

Because empty arguments are preserved and commas are inserted between selected elements, the expansion of `__VA_SLICE__` may begin with a comma (if the first selected argument is empty and at least one additional argument is selected) or may end with a comma (if the last selected argument is empty and at least one preceding argument is selected). This is intentional and matches the semantics of forwarding empty macro arguments.

4.3 Expression evaluation for `start` and `len`

Allowing `start` and `len` to be preprocessing expressions (as in `#if`) enables idioms such as:

```
#define DROP1(...)  __VA_SLICE__(2, __VA_COUNT__(__VA_ARGS__) - 1, __VA_ARGS__)
```

without requiring users to precompute lengths through additional macro layers.

5 Suggested wording

This wording is presented in terms of C23 clause numbering for the preprocessor (Clause 6.10). Editors may need to adjust numbering for the target working draft.

5.1 Clause 6.10.5.1

Suggested new wording of Paragraph 5:

> 5 The identifiers `__VA_ARGS__`, `__VA_COUNT__`, `__VA_SLICE__` and `__VA_OPT__` shall occur only in the replacement-list of a function-like macro that uses the ellipsis notation in the parameters

5.2 New clause 6.10.5.x — The `__VA_COUNT__` predefined macro

Add a new subclause:

> **6.10.5.x The `__VA_COUNT__` predefined macro**
>
> **Description**
>
> 1 The expansion of the function-like macro `__VA_COUNT__(...)` produces a single preprocessing-number token not containing any digit separators. When interpreted in translation phase 7, its value is the number of arguments in the macro invocation (including empty arguments), as determined by the macro argument parsing rules of 6.10.5, and is representable with type `signed long`.
>
> **Constraints**
>
> 2 The number of arguments shall be representable as `signed long`.
>
> **Semantics**
>
> 3 Let the invocation be `__VA_COUNT__(A1, A2, \dots , Am)`, where m is the number of arguments determined by function-like macro argument parsing. The result of macro expansion is a single preprocessing-number token denoting the decimal integer value m.
>
> 4 The preprocessing tokens of the arguments are not required to be macro-expanded in order to determine m.
>
> 5 Example 1

```
__VA_COUNT__()           // -> 0
__VA_COUNT__(x)          // -> 1
__VA_COUNT__(x, , y)     // -> 3
__VA_COUNT__((a,b), c)   // -> 2
```

5.3 New clause 6.10.5.y — The `__VA_SLICE__` predefined macro

Add a new subclause:

6.10.5.y The `__VA_SLICE__` predefined macro

Description

1 The expansion of the function-like macro `__VA_SLICE__(start, len, ...)` produces a (possibly empty) preprocessing-token sequence consisting of selected macro arguments, separated by comma preprocessing-tokens. The selection is from the variadic argument list (the arguments following the first two).

Constraints

2 After macro expansion, the preprocessing-token sequences of `start` and `len` shall be valid preprocessing expressions as in 6.10.1 and shall evaluate to integer values s and ℓ representable as `signed long`. The value s shall satisfy $s \geq 1$ and ℓ shall satisfy $\ell \geq 0$.

Semantics

3 Let the invocation be `__VA_SLICE__(s, ℓ, A1, A2, ..., Am)`, where m is the number of variadic arguments following the first two arguments, determined by function-like macro argument parsing rules, including empty arguments.

4 If $\ell = 0$ or $s > m$, the result of macro expansion is the empty preprocessing-token sequence.

5 Otherwise, let $t = \min(m, s + \ell - 1)$. The result of macro expansion is the token sequence `As, As+1, ..., At` with a comma preprocessing-token between each adjacent pair.

6 For each selected argument `Ak` (for $s \leq k \leq t$), the preprocessing tokens emitted for `Ak` are the preprocessing tokens of that argument after all macros contained therein have been expanded as in 6.10.5.2 for an ordinary macro argument substitution not involving `#` or `##`. Unselected arguments are not required to be macro-expanded.

7 Example 1

```
__VA_SLICE__(1, 3, a, b, c, d)     // -> a, b, c
__VA_SLICE__(3, 99, a, b, c, d)    // -> c, d
__VA_SLICE__(2, 2, a, , c, d)      // -> , c    (first selected
   argument is empty)
__VA_SLICE__(5, 1, a, b, c)        // ->        (empty)
__VA_SLICE__(1, 0, a, b)           // ->        (empty)
__VA_SLICE__(1, 1+1+1, x, y, z)    // -> x, y, z
__VA_SLICE__(2, 2, a, , c+1, d)    // -> , c+1
```

5.4 Clause 7.1.3, Reserved identifiers (optional editorial note)

No additional wording is required beyond the existing rules for reserved identifiers and predefined macro names; `__VA_COUNT__` and `__VA_SLICE__` are reserved by virtue of being predefined macros.

Acknowledgments

Thanks to reviewers and WG14 participants for discussion of preprocessing facilities.

References

[1] J. Gustedt, J. Rifkin. *The `__COUNTER__` predefined macro.* WG14 N3457, 2025-01-25. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3457.htm

[2] J. Gustedt. *Extensions to the preprocessor for C2Y.* WG14 N3190, 2023-12-13. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3190.htm

[3] ISO/IEC JTC1/SC22/WG14. *N841 (public comments archive).* (Contains: "Add a `__VA_COUNT__` facility for varargs macros"). https://www.open-std.org/jtc1/sc22/wg14/www/docs/n841.htm

[4] L. Deniau. *`__VA_NARG__`* (comp.std.c posting). 2006-01-16. https://groups.google.com/g/comp.std.c/c/d-6Mj5Lko_s

[5] GCC Bugs mailing list archive. *[Bug c/33877] New: Request for `__VA_ARGC__`.* 2007-10-24. https://gcc.gnu.org/legacy-ml/gcc-bugs/2007-10/msg02127.html

[6] GCC Bugs mailing list archive. *[Bug c/33877] Request for `__VA_ARGC__`* (comment emphasizing elimination of "busywork" count arguments). 2008-07-23. https://gcc.gnu.org/pipermail/gcc-bugs/2008-July/277400.html

[7] J. Gustedt. *Tail recursion for preprocessor macros.* WG14 N3307, 2024-08-05. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3307.htm