Proposal for C2Y

WG14 N 3262

Title:	Usability of a byte-wise copy of va_list
Author, affiliation:	Robert C. Seacord, Woven by Toyota, United States rcseacord@gmail.com
Date:	2024-05-10
Proposal category:	Feature
Target audience:	Implementers
Abstract:	Assign a disposition to the usability of a byte-wise copy of a va_list object
Prior art:	C23

Usability of a byte-wise copy of va_list

Reply-to: Robert C. Seacord (rcseacord@gmail.com)

Document No: **N 3262**

Reference Document: **N 3220**

Date: 2024-04-01

Proposal to allow arrays of non-atomic character type (coined byte arrays) to be accessed as other object types.

Change Log

2024-5-10:

- Initial version

Table of Contents

1 Problem Description

It is unclear in C23 if certain objects are usable after being byte-wise copied by calls to functions such as `memcpy` or `memset`. A `va_list` object is usable after initialization by the `va_start` macro, but the standard is silent about whether an initialized `va_list` object is usable after a byte-wise copy.

A `va_list` can be just a pointer and copying it should have similar consequences to copying any other object types as provided by a library. On the other hand, all complete structure types can have members that deal with offsets from their own address or contain pointers or indices into some global structure (such as `FILE`) that would prevent them from being copied or result in dangling pointers after the original copy has been destroyed. Having a complete type only guarantees that such an object can be declared, not that you can be used without calling a setup function. C23 doesn't disallow

implementations of `va_list` that cannot be copied, so passing a byte-wise copy to a standard function is implicitly undefined behavior.

C23 specifies the `va_copy` macro for copying a `va_list` object, suggesting that functions such as `memcpy` or `memset` should not be used for this purpose. A `va_list` function parameter can be a `va_list` or, if implemented as an array, a pointer. Consequently, taking the address of such a parameter to pass to `memcpy` is unreliable.

The solution proposed by this paper is to eliminate the implicit undefined behavior by making the usability of a byte copy of a **`va_list`** implementation-defined behavior.

2 Proposed Text

Text in green is added to the N3220 working draft. ~~Text in red~~ that has been struck through is removed from the N3220 working draft.

Make the following modification to Subclause 7.16 Variable arguments **`<stdarg.h>`**, paragraph 4:

The type declared is

`va_list`

which is a complete object type suitable for holding information needed by the macros **`va_start`**, **`va_arg`**, **`va_end`**, and **`va_copy`**. If access to the varying arguments is desired, the called function shall declare an object (generally referred to as **`ap`** in this subclause) having type **`va_list`**. The object **`ap`** may be passed as an argument to another function; if that function invokes the **`va_arg`** macro with parameter **`ap`**, the representation of **`ap`** in the calling function is indeterminate and shall be passed to the **`va_end`** macro prior to any further reference to **`ap`**.[295)] Whether a byte copy of **`va_list`** can be used in place of the original is implementation-defined.

3 Acknowledgements

I would like to recognize the following people for contributing to this work: Jens Gustedt, Alex Celeste, and Joseph Myers.