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

<table>
<thead>
<tr>
<th>Proposal for C2Y</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>WG14 N 3262</td>
<td>1</td>
</tr>
<tr>
<td>Change Log</td>
<td>2</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>1 Problem Description</td>
<td>2</td>
</tr>
<tr>
<td>2 Proposed Text</td>
<td>3</td>
</tr>
<tr>
<td>3 Acknowledgements</td>
<td>3</td>
</tr>
</tbody>
</table>

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 \texttt{va_list} that cannot be copied, so passing a byte-wise copy to a standard function is implicitly undefined behavior.

C23 specifies the \texttt{va_copy} macro for copying a \texttt{va_list} object, suggesting that functions such as \texttt{memcpy} or \texttt{memset} should not be used for this purpose. A \texttt{va_list} function parameter can be a \texttt{va_list} or, if implemented as an array, a pointer. Consequently, taking the address of such a parameter to pass to \texttt{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 \texttt{va_list} implementation-defined behavior.

\section*{2 Proposed Text}

Text in \textcolor{green}{green} is added to the N3220 working draft. Text in \textcolor{red}{red} that has been struck through is removed from the N3220 working draft.

Make the following modification to Subclause 7.16 Variable arguments \texttt{<stdarg.h>}, paragraph 4:

The type declared is

\begin{verbatim}
va_list
\end{verbatim}

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

\section*{3 Acknowledgements}

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