N3285: `stdarg.h` wording...

`stdarg.h`, especially in C2x, is byzantine. Modernising the language can alleviate this.

наб, seb
**N3285: stdarg.h wording...**

*stdarg.h*, especially in C2x, is byzantine. Modernising the language can alleviate this.

---

Document #: 3285  
Date: 2024-06-21  
Project: Programming Language C  
Reply-to: nab <nabijaczleweli@nabijaczleweli.xyz>

1. *Casus belli*

seb <@sebastian@jittr.click> had identified a series of inconsistencies both in the wording of *stdarg.h* in the current draft C2X standard *N3220* and in compilers’ interpretations thereof. These have been refined in subsequent discussion, this paper presents a summary of diffs, along with rationales.

2. **Proposed wording**

2.1. 7.16.1

The **va_start** and **va_arg** macros described in this subclause shall be implemented as macros, not functions. It is unspecified whether **va_copy** and **va_end** are macros or identifiers declared with external linkage. If a macro definition is suppressed to access an actual function, or a program defines an external identifier with the same name, the behavior is undefined. Each invocation of the **va_start** and **va_copy** macros shall be matched by a corresponding invocation of the **va_end** macro in the same function.

Append or add footnote:

For conciseness only, this section refers to **va_copy** and **va_end** just as “macros”. This is to be understood as a short-hand, not as constraining only one of the possible implementations.

2.1.1. **Rationale**

Kinda odd that it says these can be macros or symbols but then it calls them macros, innit. If it said “the **va_end** macro or symbol” then that would be worse though.

2.2. 7.16

The type declared is

```c
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**.²⁵⁵

Replace

```c
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**.
with

```c
va_arg, va_end, and va_copy to access the varying arguments. Objects of type va_list are generally referred to as ap in this subclause.
```

and replace

```c
The object
```

```c
ap may be passed as an argument to another function; if that function invokes the va_arg macro
```

```c
with parameter ap, the representation of ap in the calling function is indeterminate and shall be
```

```c
passed to the va_end macro prior to any further reference to ap.\(^{295}\)
```

with

```c
If an initialised-with-va_start ap object is passed as an argument to another function and that function invokes the va_arg macro on ap then the representation of ap in the calling function is indeterminate and
```

```c
ap must be passed to the va_end macro before being passed to any other va__... macros.
```

2.2.1. Rationale

Beside updating the ancient-style wording ("if ... is desired, the function ... shall"), it hinted at a restrixion of

```
where va_lists may be created. There are none such.
```

"reference to" is clarified to be w.r.t. the other va__... macros exclusively. It’s still a valid object.

If ap was never initialised with va_start, then mandating the use of va_end is obviously incorrect.

2.3. 7.16.1.4

2 The va_start macro shall be invoked before any access to the unnamed arguments.

replace with

2 The va_start macro may only be invoked in the function scope of a function whose parameter type list ends with an ellipsis.

2.3.1. Rationale

There is no other way to access the unnamed arguments (pt. 3 defines the way va_start facilitates this) anyway, so this can be deleted.

Currently, the way this limits where the standard allows va_start to be invoked is strictly by domain error of

the counterfactual (if there are no unnamed arguments). Can you use va_start if there is an ellipsis but no
unnamed arguments were given? Yes. Does the current wording allow it? No, for the same reason.

Even then, this allows

```c
void f(va_list ap, int [(va_start(ap), 1)], ...) { va_end(ap); }
```

which makes little sense, and yet GCC allows it, while Clang refuses it (‘va_start’ cannot be used outside a function). This limits va_start to the scopes where it’s meaningful.

3. Further issues

The section refers to the same concept ad lib as “varying arguments” and “unnamed arguments”, i.a. compound nouns thereof; similarly with functions that accept such. It would benefit from globally normalising to a single spelling.

4. References

The seminal post: https://jittr.click/@sebastian/statuses/01HYYTSHPDNAFDNQSTXVXYSAAY2
## Contents

1. Casus belli ................................................................. 1

2. Proposed wording .................................................. 1
   2.1. 7.16.1 ............................................................... 1
   2.1.1. Rationale ....................................................... 1
   2.2. 7.16 ................................................................. 1
   2.2.1. Rationale ....................................................... 2
   2.3. 7.16.1.4 ............................................................ 2
   2.3.1. Rationale ....................................................... 2

3. Further issues ....................................................... 2

4. References ........................................................... 2