JTC1/SC22/WG14 - N2778

Title: Variably-Modified Types

Author: Martin Uecker, University of Göttingen

Date: 2021-07-11

This is a follow up paper to N2660. As discussed there, array types with static or dynamic bound can be used instead of pointers for safe programming because compilers can use the length information encoded in the type to detect errors. In fact, a pointer to an array is nothing else than a bounded pointer type and existing compilers can already add run-time checks to detect out-of-bounds accesses [Keaton 2014]. From a theoretical point of view, a variably-modified type is a dependent type, i.e. a type that depends on a value. Such types were recognized previously as a valuable tool for type-safe low-level programming (e.g. see [Condit 2007]). For these reasons, we propose to make variably-modified types mandatory in C23. VLAs with automatic storage duration remain an optional language feature due to their higher implementation overhead and security concerns on some implementations (i.e. when allocated on the stack and not using stack probing).

Example:

```
void foo(int n, double (*x)[n])
{
  (*x)[n] = 1;     // invalid access can be detected at run-time
                   // (and possibly at compile-time with stronger analysis)
}
```

References:

[Keaton 2014] Keaton D., and Seacord. R. (2014) Performance of Compiler-Assisted Memory Safety Checking. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Note CMU/SEI-2014-TN-014, 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=299175

[Condit 2007] Condit J., Harren M., Anderson Z., Gay D., Necula G.C. (2007) Dependent Types for Low-Level Programming. In: De Nicola R. (eds) Programming Languages and Systems. ESOP 2007. Lecture Notes in Computer Science, vol 4421. Springer, Berlin, Heidelberg. DOI:10.1007/978-3-540-71316-6_35

Proposed Wording (relative to N2596)

6.10.8.3 Conditional feature macros
__STDC_NO_VLA__ The integer constant 1 , intended to indicate that the implementation does not support variable length arrays ~~or variably modified types~~ **with automatic storage duration.**

4 If the size is not present, the array type is an incomplete type. If the size is * instead of being an expression, the array type is a variable length array type of unspecified size, which can only be used in declarations or type names with function prototype scope; 146) such arrays are nonetheless complete types. If the size is an integer constant expression and the element type has a known constant size, the array type is not a variable length array type; otherwise, the array type is a variable length array type. (Variable length arrays **with automatic storage duration** are a conditional feature that implementations need not support; see 6.10.8.3.)

Acknowledgements: Aaron Ballman and Robert Seacord for feedback.