WG 14: Towards Attributes for C

Nick Stoughton, John Benito, Tom Plum, Arjun Bijanki, Jeff Muller

Introduction

During the WG 14 Kona meeting, the group considered several papers relating to attributes, including:

- N1229 (previously reviewed in London), which includes interesting attributes from GCC
- N1226 (arising from WG 21 paper N2418)
- N1264, which includes interesting attributes from MSVC
- N1259, which describes concerns with the C++ syntax proposed in N1226 (N2418)

WG 14 decided to distinguish between the syntax relating to attributes and the semantics.

An overarching goal of WG 14, described in N1250, is to strive for compatibility with existing implementations, and to avoid invention.

Concerns were voiced at the meeting, and in N1259, about both the inventiveness of the [[...]] syntax proposed by WG 21 as well as its disrespect of existing practice, although there was sympathy for the cleanliness of the proposed syntax. WG 14 decided to examine the problem from another point-of-view: instead of inventing a syntax and then fitting the semantics to it, the WG expressed a desire to understand the semantics, and then to attempt to fit a syntax to that, accommodating as far as possible existing practice and minimizing any invention.

Attribute Semantics

WG 14 considered the proposed attributes listed in N1229, N1226 (those listed as “good” examples), and N1264, and agreed on the following list (strikeouts indicate items where there was no consent to move forward):

N1229	**N1262**	**N1264**
FUNCTION		
noreturn	noreturn	noreturn

~~returns_twice~~		
pure		pure
warn_unused_result		
nonnull		
	deprecated	deprecated
VARIABLE		
aligned	align(unsigned int)	align(alignment)
cleanup		*has related proposal in try{}finally{}*
unused	unused	
gcc has similar mechanism in __builtin_expect	probably(unsigned int)	
	~~noalias~~	
	~~register~~	
	~~owner~~	
has similar proposal in __thread		thread
TYPE		
aligned		
packed		
~~transparent_union~~		
~~unused~~		

It is proposed that WG 14 will move forward with the union of these attributes. The semantics of each

is discussed further in this paper.

Attribute Syntax

WG 14 could not agree on a syntax at the Kona meeting. However, the group agreed that:

- whatever syntax is used, it must be compatible with existing practice, to the extent that simple #define macros could be used to move from pre-existing practice syntax (e.g. __attribute__((x)) or __declspec(x)) to the new syntax and vice versa. Attributes must be permitted in the position of a storage-class specifier, i.e. an attribute applies to all declarators.
- the existing C _Pragma keyword goes a long way toward solving the generalized issue, and might well serve as the ultimate way of expressing attributes at levels not currently recognized in existing practice (e,g, at block or translation unit level).

One further syntactic point was discussed, relating to namespace reservation. It was generally agreed that the attributes themselves have names, and individual implementations should be free to extend that namespace. Existing practice has many more names than described in the table above, and it was agreed that any solution proposed by the committee that outlawed existing practice would not be well received. Therefore, it is proposed that the names used for standardized C attributes be prepended with "`stdc_`"; the remainder of the attribute namespace should remain as it is now: open to vendor invention. Does this prefix make the name too long? For example "`stdc_warn_unused_result`". Any other suitable prefix would be acceptable.

Detailed Attribute Semantics

The remainder of this paper either proposes actual standardized wording for the semantics of a given attribute where multiple existing implementations overlap in attributes, or surveys existing practice where there are conflicting semantics.

Function Level Attributes

The following attributes describe functions:

stdc_noreturn

Syntax: to be defined. Associated with a function.

Constraints:The return type of the associated function shall be `void.` [[NB gcc does not make this a constraint without additional compiler flags. MSVC does enforce this]]

Semantics: The `stdc_noreturn` attribute indicates that the associated function does not return.

[Note -- control may still return from a `stdc_noreturn`-marked function via a non-local goto. --end

note]
Any path that terminates in a call to a `stdc_noreturn`-marked function need not be expected to result in any value.
The behavior is undefined if the function returns normally.

stdc_pure

Syntax: to be defined. Associated with a function.

Constraints: None.

Semantics: Any function marked as `stdc_pure` has no side effects and depends only on the values of the arguments passed; any call to the function with the same arguments has the same results as a previous call*. [*Footnote: an implementation is free to call a pure function fewer times than the program might suggest if it is able to show that subsequent calls have the same arguments as an earlier call. Some common examples of pure functions are `strlen` and `imaxabs`)–end footnote].

stdc_warn_unused_result

Syntax: to be defined. Associated with a function.

Constraints: The return value from a function marked as `stdc_warn_unused_result` shall be either stored or examined.

Semantics: A diagnostic message shall be issued if the value returned from any function marked as `stdc_warn_unused_result` is ignored.

(NB Annex I may need updating) [[not sure about the name]]

stdc_deprecated

Syntax: to be defined. Associated with a function. A message may be provided as `message`.

Constraints: None.

Semantics: A diagnostic message shall be issued if the associated function is referenced. If the `message` argument is provided, the diagnostic shall contain `message`.

stdc_nonnull

Syntax: to be defined. Associated with the argument to a function. To be discussed. SAL associates this with the argument itself, gcc associates it with the function.

Constraints: The argument shall not be a null pointer. [[at least as far as the implementation can tell]]

Semantics: To be discussed[[*the implementation may/it is implementation defined* add assertions to

ensure the runtime values of the variables are not null pointers]]

stdc_align

Syntax: to be defined. Associated with any variable. Must include a power-of-two alignment value, `alignment`. Existing practice varies subtly on the name of the attribute ("`aligned`" versus "`align`"). Gcc also permits alignment to be associated with types.

Constraints: `alignment` shall be a power of two.

Semantics: The implementation shall arrange that the associated variable has an address that is an exact multiple of `alignment`. [[some details remain to be worked out with WG 21. Possible upper boundaty, ALIGN_MAX or similar]].

~~***stdc_cleanup***~~

~~*Syntax:* to be defined. Associated with any block scoped variable. Takes a pointer to a function that receives a pointer to the variable in question, called `fptr`.~~

~~*Constraints*: `fptr` shall be a pointer to a function. [[need to say something about the arg type]]~~

~~*Semantics:* Immediately before the associated variable goes out of scope, the function referenced by `fptr` shall be called with a pointer to the variable.~~ Move this to a separate paper.

stdc_unused

Syntax: to be defined. Associated with a function argument. [[gcc also allow it to be associated with a function itself]]

Constraints: [[Should there be a constraint forbidding the use of the variable? Probably yes.]]

Semantics: No diagnostic shall be issued if the associated variable is not used. [An unused parameter to a function need not have a name.]

stdc_probably

Syntax: to be defined. Associated with *any(?)* branch statement. [[gcc has alternative syntax]]

Constraints:

Semantics:

stdc_thread

Syntax: to be defined. [[gcc uses keyword `__thread`]]

Constraints:

Semantics: The `stdc_thread` attribute may be used alone, with the `extern` or `static` specifiers, but with no other storage class specifier. When used with `extern` or `static`, `stdc_thread` must appear immediately after the other storage class specifier.

The `stdc_thread` attribute may be applied to any global, file-scoped static, or function-scoped static. It may not be applied to block-scoped automatic.

When the address-of operator is applied to a thread-local variable, it is evaluated at run-time and returns the address of the current thread's instance of that variable. An address so obtained may be used by any thread. When a thread terminates, any pointers to thread-local variables in that thread become invalid.

No static initialization may refer to the address of a thread-local variable.

stdc_packed

Syntax: to be defined. Applies to any enum, structure or union type

Constraints: None.

*Semantics:*This attribute, attached to `struct` or `union` type definition, specifies that each member (other than zero-width bitfields) of the structure or union is placed to minimize the memory required. When attached to an `enum` definition, it indicates that the smallest integral type should be used.