C and C++ Compatibility Study Group Meeting Minutes (Apr 2022)

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2959
SG Meeting Date: 2022-04-01

Fri Apr 1, 2022 at 11:00am EST

Attendees

Aaron Ballman	WG14 WG21	chair
Hubert Tong	(14) WG21	
Michael Wong	(14) WG21	
JeanHeyd Meneide	WG14 WG21	co-chair
Jens Gustedt	WG14 (21)	scribe
Steve Downey	WG21	
Corentin Jabot	WG21 (14)	
Timur Doumler	WG21	
Tom Honermann	WG21	
Zhihao Yuan	WG21	

Code of Conduct: follows ISO, IEC, and WG21 CoCs (no current WG14-specific CoC)

Agenda

Discussing the following papers:

P2552R0 (https://wg21.link/p2552r0) On the ignorability of standard attributes
P2174R0 (https://wg21.link/p2174r0) Compound Literals

P2552R0 On the ignorability of standard attributes

TD: he doesn't know what the purpose of the meeting would be, in particular he has no polls

AB: possible feedback from WG14

TD: one benefit would be to ensure that there is no conflict with C

TD Presents

TD: He has no experience with attributes in C.

TD: There is currently no consensus, yet, what it means that an attribute is ignorable. So here is a proposal for a text in the paper.

TD: the new text does not require the exact behavior when the attribute is removed, but it can have another behavior, as long as that behavior is valid

TD: the new text requires that standard attributes are syntactically correct

TD: the removal of an attribute does not add UB

TD: Is the proposal consistent with what C has currently?

TD: Does C++ want to constrain themselves for future standard attributes?

TD: thinks that, yes.

JG: Effectively we don't have as much attributes in C as in C++. I think that the C standard is stricter than what is proposed in the paper.

TD: what does strictly conforming mean in C?

AB: explains by referring to the text in the C standard.

TD: This does not say anything about behavior. In C++ there are attributes where removal changes behavior, but that is still valid.

AB: We (clang) can't diagnose easily whether attributes are in the correct position.

TD: "any" is interpreted differently: you must just issue a diagnostic if you are ignoring an attribute that you don't know

AB: contracts could be difficult

TH: contracts will not be attributes

TD: This similarity with contracts exactly triggered him to write this paper. Because it would not be clear.

HT: scribe was not able to follow

TH: syntactically correct,

HT: for the no-unique-address attribute, if someone attempts to apply an argument to the attribute then any diagnostic is valid

TH: diagnosing correctly what attribute appertains to is a real implementation burden

AB: in clang, only done for attributes that they know nothing about

HT: going back to wording and similarity with C, no-unique-address. Well-defined is not a term used in C++, probably it should be differentiated for well-formed etc

CJ: rules do not apply to "no unique address" because e.g. of size changes

TD: yes that is an interesting example, but could still argue that there still is a semantics that is valid with and without attribute

HT: different parties involved, programmer, implementation, and committee. The C rule would allow "no unique address". Does the C rule allow things that we would not like?

AB: C rule based on definition of strict conformance; I think it's correct with that in mind

TD: You claim that there currently is no requirement to diagnose a wrongly appertained attribute?

AB: Yes, currently there is no such requirement.

TD: The text for individual attributes states that an attribute "shall" appertain to a position

AB: Yes, the text also says that unrecognized attributes are ignored; standard is in conflict, implementations picked differing interpretations

TD: ??

AB: Intent was not written clear enough in the original paper that introduced attributes, but nowadays practice has evolved in a different direction.

TH: Are you saying that the C++ standard not reflect the original intent?

AB: Yes, ignoring attributes in the wrong position should be allowed.

TD: I want this to be settled, one way or another.

HT: There could be another paper that would change C++ to "you can ignore standard attributes"

TD: Split this issue into two different papers or separate sections of the same paper, then

AB: related topic is __has_cpp_attribute has different interpretations; standard says you must return a value from the table, but that doesn't do what users want for attributes that aren't actually supported by the implementation (with their recommended effects)

HT: For the C wording we want to understand what it says concerning semantics.

TD: Wants to hear back from WG14 if there would be changes

P2174R0 Compound Literals

ZY Presents

ZY: Braced initializer and compound literal have different semantics in C++ and C: one is a prvalue, the other an lvalue.

ZY: So in C you can take the address, in C++ you can't.

ZY: Compound literal extension in C++ have different semantics, depending e.g if the class has a destructor.

ZY: Among the C++ compilers, MSVC does not implement compound literals.

ZY: Real world apps, need the possibility to take the address of a compound literal, to avoid naming uninteresting temporaries.

ZY: Maybe hypothetical demand for C++ also, but currently not implemented as that.

ZY: Paper proposes different semantics than in C.

ZY: The first need of the feature are direct assignment of values, not lvalues.

AB: you got it correct how it is actually

JG: uncomfortable with having semantic differences between the original C feature and the current version for C++

HT: you can to address by reference binding. Does array to pointer conversion happen in C++ extensions?

ZY: clang does allow it, gcc doesn't

HT: passing inwards is fine, but you can't pass it out of a function. Extending lifetime could be an issue for C++

JG: In C the lifetime is usually up to the end of the current block.

HT: scribe didn't get all of it Leans towards behavior of clang to transform array compound literal into a pointer

CJ: Model of lifetime in C++ is complicated enough, it is bound to the expression, changing that would be hard for C++.

AB: also expresses discomfort with the semantic difference

JG: what is the lifetime in clang of an array compound literal when its address is passed into a function?

ZY: If this is return of the function, the pointer would be dangling outside the expression that contains the call.

CJ: There is no win-win situation here, because we will be incompatible in one way or the other

JG: C model extends lifetime so this wouldn't break code when put into C++.

AB: The WG21 polls are not so unique or not in one way or another.

ZY: We should talk to EWG people why they implement this in a different way.

JM: At least some C++ compiler has to change, even now. If you use the C feature all C++ would have to change. Just having another way to specify prvalues is not such an interesting feature. Somebody has to go to change something. Least worse is to use the C semantics.

HT: The idea that there is a difference between C-like types C++-like types. The problem is that the syntax is too similar to existing syntax in C++

JM: This idea would be mildly confusing.

HT: C++ already decided not to do this. The real problem is the storage duration, not the type category. C++ also previously kept the lifetime for C-like structures for the block, but that changed.

JM: No suggesting that would resolve this in a nice way

HT: Suggests that we perhaps we want the C semantics but not the syntax? or restrict the feature to C-like types

ZY: strongest consensus in EWG was against that

AB: If this would be attached to C-like types, would be difficult to teach

ZY: usually the places where this is used there are no destructors

JM: would it be sufficient to restrict the feature to "trivially destructible"?

HT: we should poll here if we want something along the lines

POLL: Does SG22 prefer C semantics (lifetime, storage duration, value category) for C types (is trivially destructible) for compound literals in P2174R0?

Committee	For	Against	Abstain	Notes
WG14	4	0	0	Unanimous consent
WG21	5	1	0	Consensus
Overall: Consensus, but participation is weak				

HT: C++ also would have to think if they want to impose constant initialization in global scope

Wrapup

Aaron: I'll schedule the meeting for May shortly

End at 1:06 pm EST