P1847R0

EWG
2018-08-14

Reply-to: Balog, Pal (pasa@lib.hu)
Target: C++23

Make declaration order layout mandated

Abstract

The current rules allow implementations freedom to reorder members in the layout if they have different access control. To our knowledge no implementation actually used that freedom. We propose to fix this established industry practice in the standard as mandatory.

Motivation

The idea emerged in discussion on P1112. (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1112r1.pdf). It proposes to empower programmers with better control over the layout created for classes. The attribute-based framework allows selecting a strategy to relax or strengthen the ordering compared to the current default. It shows examples why asking for strict declaration order can be desired.

The review discovered that the declorder strategy would render the attribute not conforming with EWG guidelines. However, based on observation that this strategy really does not alter anything in known existing practice, it could me made the default, thus resolving the problem with the attribute, simplifying it and also improving the coherence of the standard with the real world.

EWGI discussion in Cologne suggested to split off this change in a separate paper with a 3/5/10/0/0 vote.

Proposal

We propose to remove the implementation's license of member reordering in case access control is mixed. In this paper we do not propose anything else. Probably easiest to look directly at the wording section.

Impact on standard layout

What counts as standard layout for C++ does not change from this paper, as [class.prop] bullet (3.3) does not change. This is deliberate as observable impact of that would bring in extra risk that is best handled separately. That supposed to happen in paper P1848 dedicated to this narrow topic. Or remains with P1112 as EWGI or EWG decides.

This means that is_standard_layout<T> still reports false (as did before, avoiding behavior change in code that issued the test). Also related benefits like offsetof or common initial sequence do not apply. But for interfacing with other languages the actual layout will be fine.

Impact on implementation that did not use the license

It is conforming without any change. The users who wanted declorder layout and this far only relied on hope or ABI documentation now can rely on the standard directly. And run no further risk whether the implementation decides to change its mind later on.

Impact if an implementation did use the license after all

Without change it becomes non-conforming. If it doesn't track the standard, will likely not change the behavior at all, and the users will not notice any change. A more interesting risk for this case if new code starts to rely on the new rule and gets migrated to such a platform. To help avoiding this risk the change comes with a feature-test macro and we suggest the programmers inspect it.

If it does track the standard, it will most likely offer mode for old compatibility. As the actual layout for the case was "unspecified", the users had not much to rely on beyond ABI specification and will see changes there.

In case we think this case is realistic and users actually liked the current rule, we can offer a related strategy in P1112, say "c++20". We think that rearrangeability tied to access control provides a very weak and hardly useful way to do better (while it has a significant cost in ABI-stability), so we do not suggest it, just mention the possibility as extra risk mitigation.

What happens if this proposal is rejected

P1112 will revert to its original approach having the opt-in for the declorder strategy. Likely it will need a different framework.

Benefits of accepting this change

The main benefit is creating a cleaner state that is 1/ aligned with existing practice and 2/ no longer carry a rule that looks arbitrary and lacks a good known motivation. The standard carries plenty of old baggage that is in the way, but we keep it to save backward compatibility. For this case we can make the change while remaining backward compatible.

Wording

(Delta against N4820)

In 11.3 [class.mem] change p19 by deleting text:

19 Non-static data members of a (non-union) ~~class with the same access control (11.8)~~ are allocated so that later members have higher addresses within a class object. ~~The order of allocation of non-static data members with different access control is unspecified (11.8).~~ Implementation alignment requirements might cause two adjacent members not to be allocated immediately after each other; so might requirements for space for managing virtual functions (11.6.2) and virtual base classes (11.6.1).

Add __cpp_declorder_layout *to table 17 in 15.10 [cpp.predefined]*

Acknowledgements

Thanks to Jens Maurer for the idea of this alternative path and for reviewing this paper.