A trait for implicit lifetime types

Timur Doumler (papers@timur.audio)
Vittorio Romeo (vittorio.romeo@outlook.com)

Document #:	P2674R0
Date:	2022-10-14
Project:	Programming Language C++
Audience:	Library Evolution Working Group, Library Working Group

Abstract

C++20 introduced the notion of implicit-lifetime types. There are certain operations that are only valid for such types. We therefore need a way to check whether a type is implicit-lifetime, and constrain on this property. This paper proposes a new type trait `std::is_implicit_lifetime` to achieve this.

1 Motivation

C++20 introduced the notion of implicit-lifetime types [P0593R6]. This notion is defined in [basic.types.general]/9:

> Scalar types, implicit-lifetime class types ([class.prop]), array types, and cv-qualified versions of these types are collectively called *implicit-lifetime types*.

and in [class.prop]/9:

> A class `S` is an *implicit-lifetime class* if
>
> — it is an aggregate or
>
> — it has at least one trivial eligible constructor and a trivial non-deleted destructor.

There are certain operations that are only valid for implicit-lifetime types. In particular, in certain situations the lifetime of an object of implicit-lifetime type can be implicitly started by operations such as `malloc`, where otherwise the code would be undefined behaviour due to a violation of the C++ object lifetime rules. Additionally, in C++23 we are adding the possibility to start the lifetime of such objects explicitly with `std::start_lifetime_as<T>` [P2590R2].

Unfortunately, C++ lacks the ability to programmatically check whether a type is an implicit-lifetime type, and to constrain functions using such operations to only be valid for such types. This is particularly important to help catch a possible regression: if a type that was once implicit-lifetime mistakenly loses that property as a part of a change, this silently turns previously correct code into undefined behaviour.

To fix this issue, this paper proposes to add the type trait `std::is_implicit_lifetime<T>`. Because this is necessary to avoid a class of bugs, we consider this proposal a bugfix, rather than a new feature request, and propose to adopt it in the C++23 timeframe. This fixes a submitted NB comment.

In addition, the proposed type trait will be very useful for adding container support for implicit lifetime types [P1010R1].

2 Implementation considerations

With the definition of *implicit-lifetime type* that is currently in the C++ working paper [N4917], it is possible to implement such a type trait as follows:

```
template<typename T>
struct is_implicit_lifetime : std::disjunction<
    std::is_scalar<T>,
    std::is_array<T>,
    std::is_aggregate<T>,
    std::conjunction<
        std::is_trivially_destructible<T>,
        std::disjunction<
            std::is_trivially_default_constructible<T>,
            std::is_trivially_copy_constructible<T>,
            std::is_trivially_move_constructible<T>>>> {};
```

However, with the planned resolution of [CWG2605], which is necessary to fix an issue with implicit-lifetime aggregates, the definition of implicit-lifetime class will change as follows in C++23:

A class `S` is an *implicit-lifetime class* if

— it is an aggregate whose destructor is not user-provided or

— it has at least one trivial eligible constructor and a trivial non-deleted destructor.

With this new definition, it will be impossible for users to implement this type trait themselves, because it is impossible to programmatically check whether the destructor of an aggregate class is user-provided. Therefore, the only option we see is to add the type trait to the C++ standard library, and to implement it there as a "magic" metafunction.

3 Proposed wording

The proposed changes are relative to the C++ working paper [N4917].

In [meta.type.synop], add:

```
// [meta.unary.prop], type properties
template<class T> struct is_implicit_lifetime;

template<class T>
  inline constexpr bool is_implicit_lifetime_v = is_implicit_lifetime<T>::value;
```

In [meta.unary.prop], add a row to Table 48:

Template	Condition	Preconditions
`template<class T>` `struct is_implicit_lifetime;`	`T` is an implicit-lifetime type ([basic.types.general]).	`remove_all_extents_t<T>` shall be a complete type or *cv* `void`.

References

[CWG2605] Davis Herring. Core Issue 2605: Implicit-lifetime aggregates. https://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html#2605, 2022-06-27.

[N4917] Thomas Köppe. Working Draft, Standard for Programming Language C++. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/n4917.pdf, 2022-09-05.

[P0593R6] Richard Smith. Implicit creation of objects for low-level object manipulation. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p0593r6.html, 2020-02-14.

[P1010R1] Mark Zeren and Chris Kennelly. Container support for implicit lifetime types. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1010r1.html, 2018-10-08.

[P2590R2] Timur Doumler and Richard Smith. Explicit lifetime management. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2590r2.pdf, 2022-07-15.