1 Abstract

The member function `polymorphic_allocator::destroy` was deprecated by C++23 as it defines the same semantics that would be synthesized automatically by `std::allocator_traits`. However, some common use cases for `std::pmr::polymorphic_allocator` do not involve generic code and thus do not necessarily use `std::allocator_traits` to call on the services of such allocators. This paper recommends undeprecating that function and restoring its wording to the main Standard clause.

2 Revision history

2.1 R0: Varna 2023

Initial draft of this paper.

3 Introduction

At the start of the C++23 cycle, [P2139R2] tried to review each deprecated feature of C++, to see which we would benefit from actively removing, and which might now be better undeprecated. Consolidating all this
analysis into one place was intended to ease the (L)EWG review process, but in return gave the author so much feedback that the next revision of that paper was not completed.

For the C++26 cycle there will be a concise paper tracking the overall review process, [P2863R0], but all changes to the standard will be pursued through specific papers, decoupling progress from the larger paper so that delays on a single feature do not hold up progress on all.

This paper takes up the deprecated member function std::polymorphic_allocator::destroy, D.18 [depr.mem.poly.allocator.mem].

4 Issue History

This feature was deprecated by [#LWG3036].

4.1 LWG Poll, 2019 Kona meeting

Are we in favor of deprecation, pending on paper [P0339R6]

<table>
<thead>
<tr>
<th>F</th>
<th>N</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

4.2 2020-10-11 Reflector poll

Moved to Tentatively Ready after seven votes in favour.

5 Analysis

std::pmr::polymorphic_allocator is an allocator that will be used in non-generic circumstances, unlike std::allocator, so this member function that could otherwise be synthesized should still be part of its public interface. Hence, the recommendation is to undeprecate the destroy member function, as the natural and expected analog paired with construct.

6 Proposed wording

All changes are relative to [N4944].

20.4.3.1 [mem.poly.allocator.class.general] General

A specialization of class template pmr::polymorphic_allocator meets the allocator completeness requirements (16.4.4.6.2 [allocator.requirements.completeness]) if its template argument is a cv-unqualified object type.

```cpp
namespace std::pmr {
    template<class Tp = byte> class polymorphic_allocator {
        memory_resource* memory_rsrc; // exposition only

    public:
        using value_type = Tp;

        // 20.4.3.2[mem.poly.allocator.ctor], constructors
        polymorphic_allocator() noexcept;
        polymorphic_allocator(memory_resource* r);

        polymorphic_allocator(const polymorphic_allocator& other) = default;

    template<class U>
```
polymorphic_allocator(const polymorphic_allocator<U>& other) noexcept;

polymorphic_allocator& operator=(const polymorphic_allocator&) = delete;

// 20.4.3.3 [mem.poly.allocator.mem], member functions
[[nodiscard]] Tp* allocate(size_t n);
void deallocate(Tp* p, size_t n);

[[nodiscard]] void* allocate_bytes(size_t nbytes, size_t alignment = alignof(max_align_t));
void deallocate_bytes(void* p, size_t nbytes, size_t alignment = alignof(max_align_t));
template<class T> [[nodiscard]] T* allocate_object(size_t n = 1);
template<class T> void deallocate_object(T* p, size_t n = 1);
template<class T, class... CtorArgs> [[nodiscard]] T* new_object(CtorArgs&&... ctor_args);
template<class T> void delete_object(T* p);

// friends
friend bool operator==(const polymorphic_allocator& a,
    const polymorphic_allocator& b) noexcept {
    return *a.resource() == *b.resource();
}
};

20.4.3.3 [mem.poly.allocator.mem] Member functions

template<class T, class... Args>
void construct(T* p, Args&&... args);

template<class T>
void destroy(T* p);

polymorphic_allocator select_on_container_copy_construction() const;

memory_resource* resource() const;

// friends
friend bool operator==(const polymorphic_allocator& a,
    const polymorphic_allocator& b) noexcept {
    return *a.resource() == *b.resource();
}
};

14 Mandates: Uses-allocator construction of T with allocator *this (see 20.2.8.2 [allocator.uses.construction]) and constructor arguments std::forward<Args>(args) is well-formed.

15 Effects: Construct a T object in the storage whose address is represented by p by uses-allocator construction with allocator *this and constructor arguments std::forward<Args>(args)....

16 Throws: Nothing unless the constructor for T throws.

template<class T>
void destroy(T* p);

X Effects: As if by p->~T().

polymorphic_allocator select_on_container_copy_construction() const;

17 Returns: polymorphic_allocator().

18 [Note 4: The memory resource is not propagated. — end note]

D.18 [depr.mem.poly.allocator.mem] Deprecated polymorphic_allocator member function
The following member is declared in addition to those members specified in 20.4.3.3 [mem.poly.allocator.mem]:

```cpp
namespace std::pmr {
    template<class Tp = byte>
    class polymorphic_allocator {
    public:
        template <class T>
        void destroy(T* p);
    }
}
```

```cpp
template<class T>
void destroy(T* p);
```

Effects: As if by `p->~T()`.

7  Acknowledgements

Thanks to Michael Parks for the pandoc-based framework used to transform this document’s source from Markdown.

8  References

https://wg21.link/n4944

https://wg21.link/p0339r6

https://wg21.link/p2139r2

https://wg21.link/p2863r0