std::enable_shared_from_this

Defined in header <memory>

```
template< class T > class enable_shared_from_this;
```
(since C++11)

std::enable_shared_from_this allows an object t that is currently managed by a std::shared_ptr named pt to safely generate additional std::shared_ptr instances pt1, pt2, ... that all share ownership of t with pt.

Publicly inheriting from std::enable_shared_from_this<T> provides the type T with a member function shared_from_this. If an object t of type T is managed by a `std::shared_ptr<T>` named pt, then calling T::shared_from_this will return a new `std::shared_ptr<T>` that shares ownership of t with pt.

Member functions

(constructor)	constructs an enable_shared_from_this object (protected member function)
(destructor)	destroys an enable_shared_from_this object (protected member function)
operator=	returns a reference to `*this` (protected member function)
shared_from_this	returns a std::shared_ptr which shares ownership of `*this` (public member function)
weak_from_this (C++17)	returns a std::weak_ptr which shares ownership of `*this` (public member function)

Member objects

Member name	Definition
weak-this (exposition only)	std::weak_ptr object tracking the control block of the first shared owner of `*this`.

Notes

A common implementation for enable_shared_from_this is to hold a weak reference (such as std::weak_ptr) to `*this`. For the purpose of exposition, the weak reference is called *weak-this* and considered as a mutable std::weak_ptr member.

The constructors of std::shared_ptr detect the presence of an unambiguous and accessible (i.e. public inheritance is mandatory) enable_shared_from_this base and assign the newly created std::shared_ptr to *weak-this* if not already owned by a live std::shared_ptr. Constructing a std::shared_ptr for an object that is already managed by another std::shared_ptr will not consult *weak-this* and thus will lead to undefined behavior.

It is permitted to call shared_from_this only on a previously shared object, i.e. on an object managed by `std::shared_ptr<T>`. Otherwise, std::bad_weak_ptr is thrown (by the shared_ptr constructor from a default-constructed *weak-this*).

enable_shared_from_this provides the safe alternative to an expression like `std::shared_ptr<T>(this)`, which is likely to result in `this` being destructed more than once by multiple owners that are unaware of each other (see example below).

Example

Run this code

```
#include <iostream>
#include <memory>

class Good : public std::enable_shared_from_this<Good>
{
public:
    std::shared_ptr<Good> getptr()
    {
        return shared_from_this();
    }
};

class Best : public std::enable_shared_from_this<Best>
{
    struct Private{};

public:
    // Constructor is only usable by this class
    Best(Private) {}

    // Everyone else has to use this factory function
    // Hence all Best objects will be contained in shared_ptr
    static std::shared_ptr<Best> create()
    {
        return std::make_shared<Best>(Private());
    }

    std::shared_ptr<Best> getptr()
    {
        return shared_from_this();
    }
};


struct Bad
{
    std::shared_ptr<Bad> getptr()
    {
        return std::shared_ptr<Bad>(this);
    }
    ~Bad() { std::cout << "Bad::~Bad() called\n"; }
};

void testGood()
{
    // Good: the two shared_ptr's share the same object
    std::shared_ptr<Good> good0 = std::make_shared<Good>();
    std::shared_ptr<Good> good1 = good0->getptr();
    std::cout << "good1.use_count() = " << good1.use_count() << '\n';
}

void misuseGood()
{
    // Bad: shared_from_this is called without having std::shared_ptr owning the caller
    try
    {
        Good not_so_good;
        std::shared_ptr<Good> gp1 = not_so_good.getptr();
    }
    catch (std::bad_weak_ptr& e)
    {
        // undefined behavior (until C++17) and std::bad_weak_ptr thrown (since C++17)
        std::cout << e.what() << '\n';
    }
}

void testBest()
{
    // Best: Same but can't stack-allocate it:
    std::shared_ptr<Best> best0 = Best::create();
    std::shared_ptr<Best> best1 = best0->getptr();
    std::cout << "best1.use_count() = " << best1.use_count() << '\n';

    // Best stackBest; // <- Will not compile because Best::Best() is private.
}

void testBad()
{
    // Bad, each shared_ptr thinks it's the only owner of the object
    std::shared_ptr<Bad> bad0 = std::make_shared<Bad>();
    std::shared_ptr<Bad> bad1 = bad0->getptr();
    std::cout << "bad1.use_count() = " << bad1.use_count() << '\n';
} // UB: double-delete of Bad

int main()
{
    testGood();
    misuseGood();

    testBest();

    testBad();
}
```

Possible output:

```
good1.use_count() = 2
bad_weak_ptr
```

```
best1.use_count() = 2
bad1.use_count() = 1
Bad::~Bad() called
Bad::~Bad() called
*** glibc detected *** ./test: double free or corruption
```

Defect reports

The following behavior-changing defect reports were applied retroactively to previously published C++ standards.

DR	Applied to	Behavior as published	Co
LWG 2529 (https://cplusplus.github.io/LWG/issue2529)	C++11	specification for `enable_shared_from_this` was unclear and maybe unimplementable	cla

See also

`shared_ptr` (C++11)	smart pointer with shared object ownership semantics (class template)
`make_shared` **`make_shared_for_overwrite`** (C++20)	creates a shared pointer that manages a new object (function template)

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/memory/enable_shared_from_this&oldid=166001"