Why doesn’t my code compile when I change a shared_ptr(p) to an equivalent make_shared(p)?

devblogs.microsoft.com/oldnewthing/20240103-00

January 3, 2024

Raymond Chen

Today, we’re going to learn how to read C++ error messages.

I was providing code review feedback, and one of the changes I suggested was to change

```
auto widget = std::shared_ptr<Widget>(new Widget(this));
```

to

```
auto widget = std::make_shared<Widget>(this);
```

This change solves a few problems.

- It’s less typing and avoids repeating the name `Widget`.
- It follows the C++ core guideline R.11: Avoid calling `new` and `delete` explicitly.
- It allows the control block to be combined with the `Widget`, avoiding an extra allocation. (See Inside STL: The `shared_ptr` constructor vs `make_shared`.)

However, when the developer tried the alternative, a big cascade of errors was the result.

```
memory(874): error C2443: 'type cast': conversion from '_Ty' to 'IWidgetCallback *' exists,
but is inaccessible
    with
    [
        _Ty = WidgetContainer *
    ] (compiling source file widgetcontainer.cpp)
memory(973): note: see reference to function template instantiation
'std::_Ref_count_obj<_Ty>::_Ref_count_obj<WidgetContainer*>(_V0_t &&)' being compiled
    with
    [
        _Ty = Widget,
        _V0_t = WidgetContainer *
    ] (compiling source file widgetcontainer.cpp)
widgetcontainer.cpp(288): note: see reference to function template instantiation
'std::shared_ptr<_Ty> std::make_shared<Widget, WidgetContainer*>(_V0_t &&)' being compiled
    with
    [
        _Ty = Widget,
        _V0_t = WidgetContainer *
    ]
```

What does this all mean?

In general, C++ error messages put the most useful information at the start and at the end. All the stuff in the middle is to show you how the start and end got connected, but the important stuff usually isn't in the middle.

The Microsoft Visual C++ compiler puts the immediate problem at the start and works outward toward the code that triggered the problem.

The last line points us at line 288 of `widgetcontainer.cpp`, and that's the line that we wrote that the compiler is upset about:

```
    auto widget = std::make_shared<Widget>(this);
```

Okay, so let's look at the other end of the error output, the top. After doing the substitution of `_Ty`, you get

```
memory(874): error C2443: 'type cast': conversion from 'WidgetContainer *' to
'IWidgetCallback *' exists, but is inaccessible
```

The problem is that somebody has a `WidgetContainer *` and wants to convert it to a `IWidgetContainer *`. The conversion exists but is inaccessible. What does this mean?

In C++, there are three access levels: `public`, `protected`, and `private`.

Access	From same class	From same class or derived class	From anywhere else
`public`	Accessible	Accessible	Accessible
`protected`	Accessible	Accessible	Inaccessible
`private`	Accessible	Inaccessible	Inaccessible

We see from looking at the table that if If something is inaccessible, it means that it was declared as `private` and is being used outside the class, or it is `protected` and is being used outside the class or a derived class.

So let's see what the conversion's access specifier is.

```
class WidgetContainer : IWidgetCallback
{
    ⟦ ... ⟧
};
```

The default access specifier for a `class` is `private`, so `IWidgetCallback` is a private base class of `WidgetContainer`, and only the class itself can convert a pointer to itself into a pointer to the base class.

And that is consistent with the call site, which happens to be the `WidgetContainer` constructor.

```
WidgetContainer::WidgetContainer()
{
    ⟦ ... ⟧

    auto widget = std::make_shared<Widget>(this);

    ⟦ ... ⟧
}
```

When this code was written as

```
    auto widget = std::shared_ptr<Widget>(new Widget(this));
```

the conversion of `this` from a `WidgetContainer*` to a `IWidgetCallback*` happened inside the context of a `WidgetContainer`, so the conversion was allowed.

But changing to `make_shared` means that the parameter is passed into `make_shared` as a `WidgetContainer*`, and the conversion to a `IWidgetCallback*` doesn't happe until the `Widget` constructor is reached, which is not inside `WidgetContainer`.

So what can we do?

One approach is to force the conversion to happen in a context in which it is allowed.

```
auto widget = std::make_shared<Widget>(
    static_cast<IWidgetCallback*>(this));
```

This pre-converts the `WidgetContainer*` to a `IWidgetCallback*` so that `make_shared` doesn't have to do it.

But I suggested checking whether the private-ness of `WidgetContainer`'s use of `IWidgetCallback` as a base class was intentional. Maybe the `public` keyword was left off by mistake.

After some investigation, it seems that there was no need for `IWidgetCallback` to be a private base class, so we made it public, and that fixed it.

Bonus chatter: The clang compiler also puts the immediate problem at the start and works outward toward the code that triggered the problem.

```
// clang
In file included from widgetcontainer.cpp:5:
In file included from memory:66:
In file included from stl_tempbuf.h:61:
stl_construct.h:115:4: error: no matching function for call to 'construct_at'
  115 |           std::construct_at(__p, std::forward<_Args>(__args)...);
      |           ^~~~~~~~~~~~~~~~
alloc_traits.h:660:9: note: in instantiation of function template specialization
'std::_Construct<Widget, WidgetContainer *>' requested here
  660 |         { std::_Construct(__p, std::forward<_Args>(__args)...); }
      |                ^
shared_ptr_base.h:604:30: note: in instantiation of function template specialization
'std::allocator_traits<std::allocator<void>>::construct<Widget, WidgetContainer *>' requested
here
  604 |           allocator_traits<_Alloc>::construct(__a, _M_ptr(),
      |                                     ^
shared_ptr_base.h:972:6: note: in instantiation of function template specialization
'std::_Sp_counted_ptr_inplace<Widget, std::allocator<void>,
__gnu_cxx::_S_atomic>::_Sp_counted_ptr_inplace<WidgetContainer *>' requested here
  972 |             _Sp_cp_type(__a._M_a, std::forward<_Args>(__args)...);
      |             ^
shared_ptr_base.h:1712:14: note: in instantiation of function template specialization
'std::__shared_count<>::__shared_count<Widget, std::allocator<void>, WidgetContainer *>'
requested here
 1712 |         : _M_ptr(), _M_refcount(_M_ptr, __tag, std::forward<_Args>(__args)...)
      |                     ^
shared_ptr.h:464:4: note: in instantiation of function template specialization
'std::__shared_ptr<Widget>::__shared_ptr<std::allocator<void>, WidgetContainer *>' requested
here
  464 |         : __shared_ptr<_Tp>(__tag, std::forward<_Args>(__args)...)
      |           ^
shared_ptr.h:1009:14: note: in instantiation of function template specialization
'std::shared_ptr<Widget>::shared_ptr<std::allocator<void>, WidgetContainer *>' requested here
 1009 |       return shared_ptr<_Tp>(_Sp_alloc_shared_tag<_Alloc>{__a},
      |              ^
widgetcontainer.cpp:288:16: note: in instantiation of function template specialization
'std::make_shared<Widget, WidgetContainer *>' requested here
   15 |         auto widget = std::make_shared<Widget>(this);
      |                       ^
stl_construct.h:94:5: note: candidate template ignored: substitution failure [with _Tp =
Widget, _Args = <WidgetContainer *>]: cannot cast 'WidgetContainer' to its private base class
'IWidgetCallback'
   96 |     construct_at(_Tp* __location, _Args&&... __args)
      |     ^
   97 |     noexcept(noexcept(::new((void*)0) _Tp(std::declval<_Args>()...)))
   98 |     -> decltype(::new((void*)0) _Tp(std::declval<_Args>()...))
      |                                     ~~~
stl_construct.h:119:29: error: cannot cast 'WidgetContainer' to its private base class
'IWidgetCallback'
  119 |       ::new((void*)__p) _Tp(std::forward<_Args>(__args)...);
      |                         ^
```

```
widgetcontainer.cpp:30:25: note: implicitly declared private here
   10 | class WidgetContainer : IWidgetCallback
      |                         ^~~~~~~~~~~~~~~
```

The clang compiler is nice enough to explain why the conversion is inaccessible: “Implicitly declared private here.”

The gcc compiler goes in the opposite order, starting with the code that triggered the problem and working toward the code that encountered the immediate problem:

```
// gcc
In file included from stl_tempbuf.h:61,
                 from memory:66,
                 from widgetcontainer.cpp:5:
stl_construct.h: In instantiation of 'constexpr void std::_Construct(_Tp*, _Args&& ...) [with
_Tp = Widget; _Args = {WidgetContainer*}]':
alloc_traits.h:660:19:   required from 'static constexpr void
std::allocator_traits<std::allocator<void> >::construct(allocator_type&, _Up*, _Args&&& ...)
[with _Up = Widget; _Args = {WidgetContainer*}; allocator_type = std::allocator<void>]'
shared_ptr_base.h:604:39:   required from 'std::_Sp_counted_ptr_inplace<_Tp, _Alloc,
_Lp>::_Sp_counted_ptr_inplace(_Alloc, _Args&& ...) [with _Args = {WidgetContainer*}; _Tp =
Widget; _Alloc = std::allocator<void>; __gnu_cxx::_Lock_policy _Lp = __gnu_cxx::_S_atomic]'
shared_ptr_base.h:971:16:   required from 'std::__shared_count<_Lp>::__shared_count(_Tp*&,
std::_Sp_alloc_shared_tag<_Alloc>, _Args&& ...) [with _Tp = Widget; _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; __gnu_cxx::_Lock_policy _Lp =
__gnu_cxx::_S_atomic]'
shared_ptr_base.h:1712:14:   required from 'std::__shared_ptr<_Tp,
_Lp>::__shared_ptr(std::_Sp_alloc_shared_tag<_Tp>, _Args&& ...) [with _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; _Tp = Widget; __gnu_cxx::_Lock_policy _Lp =
__gnu_cxx::_S_atomic]'
shared_ptr.h:464:59:   required from
'std::shared_ptr<_Tp>::shared_ptr(std::_Sp_alloc_shared_tag<_Tp>, _Args&& ...) [with _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; _Tp = Widget]'
shared_ptr.h:1009:14:   required from 'std::shared_ptr<std::_NonArray<_Tp> >
std::make_shared(_Args&& ...) [with _Tp = Widget; _Args = {WidgetContainer*}; _NonArray<_Tp>
= Widget]'
<source>:15:35:   required from here
stl_construct.h:115:28: error: no matching function for call to 'construct_at(Widget*&,
WidgetContainer*)'
  115 |           std::construct_at(__p, std::forward<_Args>(__args)...);
      |           ~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
stl_construct.h:94:5: note: candidate: 'template<class _Tp, class ... _Args> constexpr
decltype (::new(void*(0)) _Tp) std::construct_at(_Tp*, _Args&& ...)'
   94 |     construct_at(_Tp* __location, _Args&&... __args)
      |     ^~~~~~~~~~~~
stl_construct.h:94:5: note:   template argument deduction/substitution failed:
stl_construct.h: In substitution of 'template<class _Tp, class ... _Args> constexpr decltype
(::new(void*(0)) _Tp) std::construct_at(_Tp*, _Args&& ...) [with _Tp = Widget; _Args =
{WidgetContainer*}]':
stl_construct.h:115:21:   required from 'constexpr void std::_Construct(_Tp*, _Args&& ...)
[with _Tp = Widget; _Args = {WidgetContainer*}]'
alloc_traits.h:660:19:   required from 'static constexpr void
std::allocator_traits<std::allocator<void> >::construct(allocator_type&, _Up*, _Args&& ...)
[with _Up = Widget; _Args = {WidgetContainer*}; allocator_type = std::allocator<void>]'
shared_ptr_base.h:604:39:   required from 'std::_Sp_counted_ptr_inplace<_Tp, _Alloc,
_Lp>::_Sp_counted_ptr_inplace(_Alloc, _Args&& ...) [with _Args = {WidgetContainer*}; _Tp =
Widget; _Alloc = std::allocator<void>; __gnu_cxx::_Lock_policy _Lp = __gnu_cxx::_S_atomic]'
shared_ptr_base.h:971:16:   required from 'std::__shared_count<_Lp>::__shared_count(_Tp*&,
std::_Sp_alloc_shared_tag<_Alloc>, _Args&& ...) [with _Tp = Widget; _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; __gnu_cxx::_Lock_policy _Lp =
__gnu_cxx::_S_atomic]'
```

```
shared_ptr_base.h:1712:14:   required from 'std::__shared_ptr<_Tp,
_Lp>::__shared_ptr(std::_Sp_alloc_shared_tag<_Tp>, _Args&& ...) [with _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; _Tp = Widget; __gnu_cxx::_Lock_policy _Lp =
__gnu_cxx::_S_atomic]'
shared_ptr.h:464:59:   required from
'std::shared_ptr<_Tp>::shared_ptr(std::_Sp_alloc_shared_tag<_Tp>, _Args&& ...) [with _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; _Tp = Widget]'
shared_ptr.h:1009:14:   required from 'std::shared_ptr<std::_NonArray<_Tp> >
std::make_shared(_Args&& ...) [with _Tp = Widget; _Args = {WidgetContainer*}; _NonArray<_Tp>
= Widget]'
widgetcontainer.cpp:288:35: required from here
stl_construct.h:96:17: error: 'IWidgetCallback' is an inaccessible base of 'WidgetContainer'
   96 |     -> decltype(::new((void*)0) _Tp(std::declval<_Args>()...))
      |                 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
stl_construct.h: In instantiation of 'constexpr void std::_Construct(_Tp*, _Args&& ...) [with
_Tp = Widget; _Args = {WidgetContainer*}]':
alloc_traits.h:660:19:   required from 'static constexpr void
std::allocator_traits<std::allocator<void> >::construct(allocator_type&, _Up*, _Args&& ...)
[with _Up = Widget; _Args = {WidgetContainer*}; allocator_type = std::allocator<void>]'
shared_ptr_base.h:604:39:   required from 'std::_Sp_counted_ptr_inplace<_Tp, _Alloc,
_Lp>::_Sp_counted_ptr_inplace(_Alloc, _Args&& ...) [with _Args = {WidgetContainer*}; _Tp =
Widget; _Alloc = std::allocator<void>; __gnu_cxx::_Lock_policy _Lp = __gnu_cxx::_S_atomic]'
shared_ptr_base.h:971:16:   required from 'std::__shared_count<_Lp>::__shared_count(_Tp*&,
std::_Sp_alloc_shared_tag<_Alloc>, _Args&& ...) [with _Tp = Widget; _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; __gnu_cxx::_Lock_policy _Lp =
__gnu_cxx::_S_atomic]'
shared_ptr_base.h:1712:14:   required from 'std::__shared_ptr<_Tp,
_Lp>::__shared_ptr(std::_Sp_alloc_shared_tag<_Tp>, _Args&& ...) [with _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; _Tp = Widget; __gnu_cxx::_Lock_policy _Lp =
__gnu_cxx::_S_atomic]'
shared_ptr.h:464:59:   required from
'std::shared_ptr<_Tp>::shared_ptr(std::_Sp_alloc_shared_tag<_Tp>, _Args&& ...) [with _Alloc =
std::allocator<void>; _Args = {WidgetContainer*}; _Tp = Widget]'
shared_ptr.h:1009:14:   required from 'std::shared_ptr<std::_NonArray<_Tp> >
std::make_shared(_Args&& ...) [with _Tp = Widget; _Args = {WidgetContainer*}; _NonArray<_Tp>
= Widget]'
widgetcontainer.cpp:288:35: required from here
stl_construct.h:119:7: error: 'IWidgetCallback' is an inaccessible base of 'WidgetContainer'
  119 |       ::new((void*)__p) _Tp(std::forward<_Args>(__args)...);
      |       ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

There it is: "`IWidgetCallback` is an inaccessible base of `WidgetContainer`."

(If this looks familiar, it's because it's basically the same thing we saw before when we learned that perfect forwarding can sometimes be too perfect.)