Subject: How to mark std::array<T, N> moveable if only T is moveable
Posted by busiek on Tue, 02 Jul 2024 15:54:00 GMT
View Forum Message <> Reply to Message

Hi all,
U++ does not have a container with std::array functionality. Is there any simple way to mark std::array<T, N> moveable if only T is moveable?
I didn't come with anything better than implementing my own version of std::array derived from Moveable<MyArray<T, N>>. By using macro NTL_MOVEABLE one can only mark moveable a specific type not a template. Even if I want to mark specific template type, macro does not work with template types having two or more parameters and I have to expand macro myself:
template<> inline void AssertMoveable<std::array<double, 2>>(std::array<double, 2>*) {}

Correct me if I miss something. Shouldn't the mechanism of marking type moveable be done through partial class instantiation? Or concepts with C++20?

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Lance on Wed, 03 Jul 2024 01:39:21 GMT
View Forum Message <> Reply to Message

Reminds me of some exploration I did with Moveable.

I vaguely recalled that when I tested a few years back, the way Moveable was designed were causing a compile time error with Windows/VSBuild. I would also welcome a redesign of Moveable using more recent language facilities.

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 18 Jul 2024 06:41:26 GMT
View Forum Message <> Reply to Message

Lance wrote on Wed, 03 July 2024 03:39Reminds me of some exploration I did with Moveable.

I vaguely recalled that when I tested a few years back, the way Moveable was designed were causing a compile time error with Windows/VSBuild. I would also welcome a redesign of Moveable using more recent language facilities.

Suggestions? (But it needs to be backward compatible).

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 18 Jul 2024 06:44:32 GMT
View Forum Message <> Reply to Message

busiek wrote on Tue, 02 July 2024 17:54Hi all,
U++ does not have a container with std::array functionality. Is there any simple way to mark

std::array<T, N> moveable if only T is moveable?
I didn't come with anything better than implementing my own version of std::array derived from Moveable<MyArray<T, N>>. By using macro NTL_MOVEABLE one can only mark moveable a specific type not a template. Even if I want to mark specific template type, macro does not work with template types having two or more parameters and I have to expand macro myself:
template<> inline void AssertMoveable<std::array<double, 2>>(std::array<double, 2>*) {}

Correct me if I miss something. Shouldn't the mechanism of marking type moveable be done through partial class instantiation? Or concepts with C++20?

Wait a moment there: Are you sure that std::array is Moveable? There is no guarantee...

Mirek

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by busiek on Thu, 18 Jul 2024 17:39:04 GMT
View Forum Message <> Reply to Message

I thought it was obvious if only T is moveable. In the reference for std::array it says it is an aggregate type for C-style array T[N]. What do you mean by guarantee?

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Fri, 19 Jul 2024 11:11:10 GMT
View Forum Message <> Reply to Message

Same semantics does not cover U++ moveable. Now I agree it is unlikely, but so I thought about std::string until I met implementation that is not U++ moveable...

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Lance on Fri, 19 Jul 2024 11:52:06 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 19 July 2024 07:11Same semantics does not cover U++ moveable. Now I agree it is unlikely, but so I thought about std::string until I met implementation that is not U++ moveable...

I looked into std::basic_string in this post, message #59178.

Quote:
Story of std::basic_string (GLIBCXX implementation)

It's surprising that a basic_string<ch> would cause trouble (core dump etc) when treated as raw

bytes. Digging into its implementation (in <bits/basic_string.h>), we have the data members ....

On the same post, I proposed to use class traits to handle Moveable. There are details I could not iron out. Also, it might not be compatible with very old c++ because I have an added goal to allow non-trivially relocatable class objects, which can be corrected pre- or post relocation, be housed in std::vector or upp::Vector.

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Lance on Fri, 19 Jul 2024 12:02:19 GMT
View Forum Message <> Reply to Message

last time I tested, AssertMoveable was preventing U++ be compiled with std=c++20. Eventually that might be a move we have to make.

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Didier on Fri, 19 Jul 2024 17:10:45 GMT
View Forum Message <> Reply to Message

I dont know about std::array but std::Vector is not upp::moveable
I recently ran into it in Someone else's code... And I had to replace it with upp::Vector.

I agree with Lance: keeping up with latest compiler standards may be a good move

---

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Mon, 22 Jul 2024 09:10:46 GMT
View Forum Message <> Reply to Message

Lance wrote on Fri, 19 July 2024 14:02last time I tested, AssertMoveable was preventing U++ be compiled with std=c++20. Eventually that might be a move we have to make.

I have just checked and it seems fine...

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Lance on Mon, 22 Jul 2024 23:07:51 GMT
View Forum Message <> Reply to Message

Sorry Mirek. I forgot to give more detail. It's been mentioned in this reply in the post I mentioned above. It happens when I compile theide (or anything use Core) with MSBT64x22. I haven't tried with more recent MSBuilder.

Lance wrote on Mon, 14 November 2022 08:52
Well it all starts with testing Upp code for C++20 compliance. let's try theide first. The ide compiles fine on both GCC and CLANG with -std=c++20 option, except some complaints on capturing this by default is deprecated in C++20, which are easy to fix or safe to ignore for now. But it's a total different story with MSC. with standard set to C++17, MSC rejects a bunch of stuff like

return somecondition? "a literal string" : AString;

These are also easy to fix if you don't mind your local version is slightly different from the main stream.

When standard is set to C++20 or c++latest, Upp::Moveable AssertMoveable0() is start to causing compilation failure, this one seems to be quite difficult to fix.

I was thinking it's just a mechanism to communicate to the compiler that it can treat object of this class as raw bytes, maybe we can do it differently with so much more facilities available in more recent c++ library.

So I start to do some experiment.

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by busiek on Tue, 23 Jul 2024 16:25:28 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 18 July 2024 08:41Lance wrote on Wed, 03 July 2024 03:39Reminds me of some exploration I did with Moveable.

I vaguely recalled that when I tested a few years back, the way Moveable was designed were causing a compile time error with Windows/VSBuild. I would also welcome a redesign of Moveable using more recent language facilities.

Suggestions? (But it needs to be backward compatible).

I was thinking about something like that:
```
#include <Core/Core.h>
#include <type_traits>

namespace Upp {
// First way of marking class moveable is to be an ancestor of NtlMoveableBase
template <class T> class NtlMoveableBase {};

// Second way is to specialize NtlMoveableClass
template <class T>
struct NtlMoveableClass
: std::integral_constant<bool,
```

```
    std::is_trivial_v<T>
 || std::is_base_of_v<Upp::NtlMoveableBase<T>, T>
 // below is not needed if we make them derived from NtlMoveableBase
 || std::is_base_of_v<Upp::Moveable_<T>, T>
 || std::is_base_of_v<Upp::Moveable<T>, T>
 || std::is_base_of_v<Upp::MoveableAndDeepCopyOption<T>, T>> {};

// For instance mark std::array<T, N> moveable if only T is moveable
template <class T, size_t N>
struct NtlMoveableClass<std::array<T, N>>
: std::integral_constant<bool, NtlMoveableClass<T>::value> {};

// Helper for checking whether class is moveable
template <class T>
inline constexpr bool IsNtlMoveable = NtlMoveableClass<T>::value;

// Optional concept for c++20
template <class T>
concept NtlMoveable = IsNtlMoveable<T>;

}

using namespace Upp;

// use of concept
template <NtlMoveable T>
struct MyVector
{
 T *ptr;
 // Or use static_assert
 ~MyVector() { static_assert(IsNtlMoveable<T>); }
};

CONSOLE_APP_MAIN
{
 // Checking if given type is moveable
 static_assert(IsNtlMoveable<int>);
 static_assert(IsNtlMoveable<const void *>);
 static_assert(IsNtlMoveable<Vector<int>>);
 static_assert(IsNtlMoveable<std::array<int, 5>>);
 static_assert(IsNtlMoveable<std::array<Vector<int>, 5>>);
 static_assert(!IsNtlMoveable<Thread>);
}
```

You could redefine NTL_MOVEABLE macro as a specialization of NtlMoveableClass.
But you need to change an assertion AssertMoveable((T*)0) to static_assert(IsNtlMoveable<T>).
Are those ideas usable?

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 25 Jul 2024 00:27:32 GMT
View Forum Message <> Reply to Message

busiek wrote on Tue, 23 July 2024 18:25mirek wrote on Thu, 18 July 2024 08:41Lance wrote on Wed, 03 July 2024 03:39Reminds me of some exploration I did with Moveable.

I vaguely recalled that when I tested a few years back, the way Moveable was designed were causing a compile time error with Windows/VSBuild. I would also welcome a redesign of Moveable using more recent language facilities.

Suggestions? (But it needs to be backward compatible).

I was thinking about something like that:

```
#include <Core/Core.h>
#include <type_traits>

namespace Upp {
// First way of marking class moveable is to be an ancestor of NtlMoveableBase
template <class T> class NtlMoveableBase {};

// Second way is to specialize NtlMoveableClass
template <class T>
struct NtlMoveableClass
: std::integral_constant<bool,
    std::is_trivial_v<T>
 || std::is_base_of_v<Upp::NtlMoveableBase<T>, T>
 // below is not needed if we make them derived from NtlMoveableBase
 || std::is_base_of_v<Upp::Moveable_<T>, T>
 || std::is_base_of_v<Upp::Moveable<T>, T>
 || std::is_base_of_v<Upp::MoveableAndDeepCopyOption<T>, T>> {};

// For instance mark std::array<T, N> moveable if only T is moveable
template <class T, size_t N>
struct NtlMoveableClass<std::array<T, N>>
: std::integral_constant<bool, NtlMoveableClass<T>::value> {};

// Helper for checking whether class is moveable
template <class T>
inline constexpr bool IsNtlMoveable = NtlMoveableClass<T>::value;

// Optional concept for c++20
template <class T>
concept NtlMoveable = IsNtlMoveable<T>;

}

using namespace Upp;
```

```
// use of concept
template <NtlMoveable T>
struct MyVector
{
 T *ptr;
 // Or use static_assert
 ~MyVector() { static_assert(IsNtlMoveable<T>); }
};

CONSOLE_APP_MAIN
{
 // Checking if given type is moveable
 static_assert(IsNtlMoveable<int>);
 static_assert(IsNtlMoveable<const void *>);
 static_assert(IsNtlMoveable<Vector<int>>);
 static_assert(IsNtlMoveable<std::array<int, 5>>);
 static_assert(IsNtlMoveable<std::array<Vector<int>, 5>>);
 static_assert(!IsNtlMoveable<Thread>);
}
```

You could redefine NTL_MOVEABLE macro as a specialization of NtlMoveableClass.
But you need to change an assertion AssertMoveable((T*)0) to static_assert(IsNtlMoveable<T>).
Are those ideas usable?

Yep. After tinkering and simplifying I think this should cover it all:

```
template <class T> struct Moveable {};

template <class T>
inline constexpr bool is_Moveable = std::is_trivially_copyable<T>::value ||
                                    std::is_base_of<moveable<T>, T>::value;
```

We will not need NTL_MOVEABLE macro anyway... Trivial types are covered and allowing non-trivial types to be explicitly marked Moveable is outright dangerous.

BTW, std::array passes is_Moveable out of box.

See any problem? (Apart for requiring C++17, but I guess we can go there for the next release)

Mirek

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Lance on Thu, 25 Jul 2024 01:16:02 GMT
View Forum Message <> Reply to Message

It works!

The capability to manually mark a class might still be an asset. Suppose we receive a third party class whose objects are trivially relocatable but not trivially copyable, the simplest way to make Upp::Vector accept it is to mark it. Wrongfully marked class will usually result in immediate runtime error, hence should not be a big concern.

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 25 Jul 2024 07:19:38 GMT
View Forum Message <> Reply to Message

Lance wrote on Thu, 25 July 2024 03:16
The capability to manually mark a class might still be an asset. Suppose we receive a third party class whose objects are trivially relocatable but not trivially copyable, the simplest way to make Upp::Vector accept it is to mark it. Wrongfully marked class will usually result in immediate runtime error, hence should not be a big concern.

I disagree. They might be relocatable in that version of library with given compiler only. Immediate runtime error might happen when you do testing, but developer might not be even aware he is supposed to test it.

Also from practical point of view, this never happened

What might be usefull though is the capability to manually mark a class to be relocated by regular move/copy constructor:

- I still think it is a good idea not to do move/copy as default option to force effectivness

- but this would allow things like Vector<std::string> with some performance penalty...

Another thing to consider: Tuple is moveable if all its components are moveable. Are we able to express that somehow? (If not, no big deal, I can make Tuple moveable and request that elements are).

Mirek

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 25 Jul 2024 07:32:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 25 July 2024 09:19
Another thing to consider: Tuple is moveable if all its components are moveable. Are we able to express that somehow? (If not, no big deal, I can make Tuple moveable and request that elements are).

Figured it out:

```
struct not_moveable {};

template <class A, class B>
class Two : std::conditional<is_moveable<A> && is_moveable<B>, moveable<Two<A, B>>,
not_moveable>::type {
 A a;
 B b;
};
```

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 25 Jul 2024 08:07:07 GMT
View Forum Message <> Reply to Message

```
// For instance mark std::array<T, N> moveable if only T is moveable
template <class T, size_t N>
struct NtlMoveableClass<std::array<T, N>>
: std::integral_constant<bool, NtlMoveableClass<T>::value> {};
```

BTW, easiest and really nice way how to add optional trait:

```
template <> inline constexpr bool is_moveable<std::string> = true;
```

(do not want this, but could be useful to know)

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by busiek on Thu, 25 Jul 2024 15:32:03 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 25 July 2024 09:32mirek wrote on Thu, 25 July 2024 09:19
Another thing to consider: Tuple is moveable if all its components are moveable. Are we able to express that somehow? (If not, no big deal, I can make Tuple moveable and request that elements are).

Figured it out:

```
struct not_moveable {};

template <class A, class B>
class Two : std::conditional<is_moveable<A> && is_moveable<B>, moveable<Two<A, B>>,
not_moveable>::type {
 A a;
 B b;
};
```

But what if one wants to use std::tuple instead of Upp::Tuple? How to mark it moveable if tuple components are moveable?
I sometimes prefer to use std::tuple because one can write:
auto [a, b] = some std::tuple
Alternatively, there is a way to implement Upp::Tuple that it works with structured binding?

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by Lance on Thu, 25 Jul 2024 22:00:16 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 25 July 2024 04:07

```
// For instance mark std::array<T, N> moveable if only T is moveable
template <class T, size_t N>
struct NtlMoveableClass<std::array<T, N>>
: std::integral_constant<bool, NtlMoveableClass<T>::value> {};
```

BTW, easiest and really nice way how to add optional trait:

```
template <> inline constexpr bool is_moveable<std::string> = true;
```

(do not want this, but could be useful to know)

Exactly, like it or not, a programmer can mark a class as moveable.

Another not very convincing example.

```
#include <iostream>
#include <type_traits>
#include <array>


template <class T> struct Moveable {};

template <class T>
inline constexpr bool is_moveable_v = std::is_trivially_copyable<T>::value ||
                                      std::is_base_of<Moveable<T>, T>::value;

class C: public Moveable<C>{
 C(const C& c){}
 C(C&& c){}
};

// uncomment to communicate the moveability of array<T,n> to the compiler
//template <class T, int n>
//inline constexpr bool is_moveable_v<std::array<T,n>> = is_moveable_v<T>;

int main()
{
 std::cout<<is_moveable_v<std::array<C,5>><<std::endl;
}
```

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Mon, 29 Jul 2024 06:47:50 GMT

View Forum Message <> Reply to Message

mirek wrote on Thu, 25 July 2024 09:32mirek wrote on Thu, 25 July 2024 09:19
Another thing to consider: Tuple is moveable if all its components are moveable. Are we able to express that somehow? (If not, no big deal, I can make Tuple moveable and request that elements are).

Figured it out:

```
struct not_moveable {};

template <class A, class B>
```

```
class Two : std::conditional<is_moveable<A> && is_moveable<B>, moveable<Two<A, B>>,
not_moveable>::type {
 A a;
 B b;
};
```

Nicer and simpler way:

```
template <class A, class B> inline constexpr bool is_moveable<Two<A, B>> = is_moveable<A>
&& is_moveable<B>;
```

I am about ready to try to introduce this into U++, but I am actually out of ideas how to name things....

Moveable / is_moveable is not perfect (future c++ might name this trivially_relocatable, but that is too long IMO), but u++ traditional, I guess I can live that.

However, new system should introduce a trait that says "I know I cannot memmove this type, but I still would like to store it into Vector anyway". E.g. to have Index<std::string> ... Really not sure what id should I use for that... Something like "is_vector_compatible_but_slower", or "std_moveable" or something?

(alternative would be to have "mem moveable" as option, but I want to force developer to be explicit, not to forget to mark moveable types)

Mirek

Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Fri, 23 Aug 2024 06:54:18 GMT
View Forum Message <> Reply to Message

https://www.ultimatepp.org/forums/index.php?t=msg&goto=6 0744&#msg_60744
