Moving to C++17 – personal experience

YAIR FRIEDMAN

Agenda

- C++17
- Personal experience
- Windows-Centric
- Not necessary representative code
- Try to (but not always) write modern C++ code
- Lot's of code from the field
- Long compiler messages
- Surprises
- Hopefully useful things
- Discussion

C++17

- C++17, also formerly known as C++1z, is the name of the most recent release of the C++ programming language, approved by ISO as of December 2017, replacing C++14.
- The name is derived from the tradition of naming language versions by the date of the specification's publication.
- Despite early plans for a major release, many of the proposed features were not mature enough and were dropped
- We will see Visual Studio 2017 which is mostly conforming to C++17

std::codecvt

Old Code

```
#include <string>
#include <codecvt>

std::string ws2s(const std::wstring& wstr)
{
   std::string str = std::wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t>().to_bytes(wstr);
   return str;
}
```

std::codecvt deprecation

Compiler message

```
1>error C4996:
'std::wstring_convert<std::codecvt_utf8<wchar_t,1114111,0>,wchar_t,std::allocator<wchar_t>,
std::allocator<char>>::to_bytes': warning STL4017: std::wbuffer_convert, std::wstring_convert,
and the <codecvt> header (containing std::codecvt_mode, std::codecvt_utf8, std::codecvt_utf16,
and std::codecvt_utf8_utf16) are deprecated in C++17. (The std::codecvt class template is NOT
deprecated.) The C++ Standard doesn't provide equivalent non-deprecated functionality; consider
using MultiByteToWideChar() and WideCharToMultiByte() from <Windows.h> instead. You can define
_SILENCE_CXX17_CODECVT_HEADER_DEPRECATION_WARNING or _SILENCE_ALL_CXX17_DEPRECATION_WARNINGS to
acknowledge that you have received this warning.

1> note: see declaration of
'std::wstring_convert<std::codecvt_utf8<wchar_t,1114111,0>,wchar_t,std::allocator<wchar_t>,std::
allocator<char>>::to_bytes'
```

std::codecvt deprecation

Old Code

```
#include <string>
#include <codecvt>

std::string ws2s(const std::wstring& wstr)
{
   std::string str = std::wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t>().to_bytes(wstr);
   return str;
}
```

std::codecvt deprecation

New Code

```
#define _SILENCE_CXX17_CODECVT_HEADER_DEPRECATION_WARNING
#include <string>
#include <codecvt>

std::string ws2s(const std::wstring& wstr)
{
   std::string str = std::wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t>().to_bytes(wstr);
   return str;
}
```

std::iterator

Old Code

```
#include <iterator>
#include <vector>
class TlvIterator : public std::iterator<std::forward_iterator_tag, uint16_t>
{
private:
    std::vector<uint8_t>::const_iterator _iterator;
public:




    TlvIterator(const std::vector<uint8_t>::const_iterator& it);
    TlvIterator(const std::vector<uint8_t>& buffer);
    ~TlvIterator();
    bool operator==(const TlvIterator& other) const;
    bool operator!=(const TlvIterator& other) const;
…
};
```

std::iterator modifications

Compiler message

```
c:\program files (x86)\microsoft visual studio\2017\enterprise\vc\tools\msvc\14.12.25827\include\xutility(620):
warning C4996: 'std::iterator<std::forward_iterator_tag,uint16_t,ptrdiff_t,_Ty *,_Ty &>::iterator_category':
warning STL4015: The std::iterator class template (used as a base class to provide typedefs) is deprecated in
C++17. (The <iterator> header is NOT deprecated.) The C++ Standard has never required user-defined iterators to
derive from std::iterator. To fix this warning, stop deriving from std::iterator and start providing publicly
accessible typedefs named iterator_category, value_type, difference_type, pointer, and reference. Note that
value_type is required to be non-const, even for constant iterators. You can define
_SILENCE_CXX17_ITERATOR_BASE_CLASS_DEPRECATION_WARNING or _SILENCE_ALL_CXX17_DEPRECATION_WARNINGS to acknowledge
that you have received this warning.
2>        with
2>        [
2>            _Ty=uint16_t
2>        ]
…
tlviterator.cpp(196): note: see reference to function template instantiation '_InIt
std::find<TlvIterator,int>(_InIt,_InIt,const _Ty &)' being compiled
2>        with
2>        [
2>            _InIt=TlvIterator,
2>            _Ty=int
2>        ]
…
```

std::iterator modifications

Old Code

```cpp
#include <iterator>
#include <vector>
class TlvIterator : public std::iterator<std::forward_iterator_tag, uint16_t>
{
private:
    std::vector<uint8_t>::const_iterator _iterator;
public:




    TlvIterator(const std::vector<uint8_t>::const_iterator& it);
    TlvIterator(const std::vector<uint8_t>& buffer);
    ~TlvIterator();
    bool operator==(const TlvIterator& other) const;
    bool operator!=(const TlvIterator& other) const;
…
};
```

std::iterator modifications

New Code

```
#include <iterator>
#include <vector>
class TlvIterator // Not inheriting from std::iterator
{
private:
    std::vector<uint8_t>::const_iterator _iterator;
public:
    // Iterator attributes:
    using iterator_category = std::forward_iterator_tag;
    using value_type = uint16_t;
    using difference_type = int32_t;
    TlvIterator(const std::vector<uint8_t>::const_iterator& it);
    TlvIterator(const std::vector<uint8_t>& buffer);
    ~TlvIterator();
    bool operator==(const TlvIterator& other) const;
    bool operator!=(const TlvIterator& other) const;
…
};
```

unused parameters

Old Code

```
HINSTANCE LibraryLoader::load(std::wstring fileName,
                              const std::vector<std::wstring>& allowedValues) const
{
…
#ifndef _DEBUG
…
  if (!Verifier::verifyFile(fileName, allowedValues))
  {
      return nullptr;
  }
…
#endif
…
  // return some good pointer
}
```

unused parameters

Compiler message

```
1> error C2220: warning treated as error - no 'object' file generated

1> warning C4100: 'allowedValues': unreferenced formal parameter
```

unused parameters

Old Code

```
HINSTANCE LibraryLoader::load(std::wstring fileName,
                              const std::vector<std::wstring>& allowedValues) const
{
…
#ifndef _DEBUG
…
  if (!Verifier::verifyFile(fileName, allowedValues))
  {
      return nullptr;
  }
…
#endif
…
  // return some good pointer
}
```

unused parameters

New Code

```
HINSTANCE LibraryLoader::load(std::wstring fileName,
                                        const std::vector<std::wstring>& allowedValues [[maybe_unused]]) const
{
…
#ifndef _DEBUG
…
  if (!Verifier::verifyFile(fileName, allowedValues))
  {
      return nullptr;
  }
…
#endif
…
  // return some good pointer
}
```

Casting Errors

Old Code and Compiler message

```
DWORD pid = 11;
HANDLE process = (HANDLE)pid, token = (HANDLE)123;

1>warning C4312: 'type cast': conversion from 'DWORD' to 'HANDLE' of greater size
```

```
ENGINE engine_handle = (ENGINE)0xdeadbeef;
1>warning C4312: 'type cast': conversion from 'unsigned int' to 'ENGINE' of greater size
```

Casting Errors

New Code

```
DWORD pid = 11;
auto hProcess = reinterpret_cast<HANDLE>(11);
auto hToken = reinterpret_cast<HANDLE>(123);




ENGINE engine_handle = reinterpret_cast<ENGINE>(uintptr_t{0xdeadbeef});
```

Random Numbers

Old Code and Compiler message

```
#include <stdlib.h>
#include <time.h>


srand((unsigned int)time(NULL));

ENGINE engine_handle = (ENGINE)rand();

1> error C2220: warning treated as error - no 'object' file generated
1> warning C4312: 'type cast': conversion from 'int' to 'ENGINE' of greater size
```

Random Numbers

New Code

```
#define NOMINMAX
#include <random>
#include <chrono>
#include <limits>

std::mt19937_64 engine(std::chrono::high_resolution_clock::now().time_since_epoch().count());
std::uniform_int_distribution<uintptr_t> dist(0, std::numeric_limits<uintptr_t>::max());
ENGINE engine_handle = reinterpret_cast<ENGINE>(dist(engine));
```

What is a byte?

Old Code and Compiler message

```
#include <Windows.h>



#include <cstddef>






byte group1244[] = {0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02, 0x01,
0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
```

```
1>error C2872: 'byte': ambiguous symbol
1>c:\program files (x86)\windows kits\10\include\10.0.16299.0\shared\rpcndr.h(191): note: could
be 'unsigned char byte'
1>c:\program files (x86)\microsoft visual
studio\2017\enterprise\vc\tools\msvc\14.12.25827\include\cstddef(22): note: or 'std::byte'
```

Is it a std::byte or an unsigned char byte?

Root cause

```
#include <Windows.h>
// #include <rpcndr.h>
typedef unsigned char byte;
#include <cstddef>
namespace std {
enum class byte : unsigned char {} ;
}
using namespace std;
…
byte group1244[] = {0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02, 0x01,
0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
1>error C2872: 'byte': ambiguous symbol
1>c:\program files (x86)\windows kits\10\include\10.0.16299.0\shared\rpcndr.h(191): note: could
be 'unsigned char byte'
1>c:\program files (x86)\microsoft visual
studio\2017\enterprise\vc\tools\msvc\14.12.25827\include\cstddef(22): note: or 'std::byte'
```

std::byte vs byte

Attempt #1: Remove unnecessary include files and using declarations

```
~~#include <Windows.h>~~
~~// #include <rpcndr.h>~~
~~typedef unsigned char byte;~~
#include <cstddef>
namespace std {
enum class byte : unsigned char {} ;
}
using namespace std;
…
byte group1244[] = {0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02, 0x01,
0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
```

std::byte vs byte

Attempt #1: Remove unnecessary include files and using declarations

```
#include <Windows.h>
// #include <rpcndr.h>
typedef unsigned char byte;
#include <cstddef>
namespace std {
enum class byte : unsigned char {} ;
}
~~using namespace std;~~
…
byte group1244[] = {0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02, 0x01,
0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
```

std::byte vs byte vs BYTE

Attempt #2 look on Usage and match

```
#include <Windows.h>
// #include <rpcndr.h>
typedef unsigned char byte;
#include <cstddef>
namespace std {
enum class byte : unsigned char {} ;
}
using namespace std;
…
BYTE group1244[] = {0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02, 0x01,
0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
…
func(group1244, …);
// defined as
func(const BYTE* vec, …);
```

std::byte vs byte vs BYTE vs uint8_t

Attempt #3 look on Usage and match, or change

```
#include <Windows.h>
// #include <rpcndr.h>
typedef unsigned char byte;
#include <cstddef>
namespace std {
enum class byte : unsigned char {} ;
}
using namespace std;
…
uint8_t group1244[] = {0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02, 0x01,
0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
…
func(group1244, …);
// defined as
func(const uint8_t* vec, …);
```

std::byte vs byte vs BYTE vs uint8_t

Future

std::byte doesn't have implicit conversion from (unsigned) int, so arrays of std::byte are not easy to create.

But we can have a helper function:

```
template<typename... Ts>
constexpr std::array<std::byte, sizeof...(Ts)>
make_bytes(Ts&&... args) noexcept {
    return { std::byte{std::forward<Ts>(args)}... };
}

auto group1244 = make_bytes(0x30, 0x82, 0x03, 0x3f, 0x30, 0x82, 0x02, 0xe6, 0xa0, 0x03, 0x02,
0x01, 0x02, 0x02, 0x02, 0x04, 0xdc, 0x30, 0x0a, 0x06, 0x08 …
…
func(group1244, …);
// possibly defined as
<size_t N>
func(const std::array<const std::byte, N>&vec, …);
```

Questions?