CTAD
Concept Typename
Auto Deduction

The Feature

Before C++17

```
vector<int> v = {1, 2, 3}; // Why do I need to say this is a vector of ints?
lock_guard<shared_mutex> lck{smtx}; // Doesn't the compiler know the smtx is a shared_mutex?
shared_lock<shared_mutex> lck2{smtx2}; // Same here
scoped_lock<shared_mutex, shared_lock<shared_mutex>> assign_lock{smtx, lck2}; // Yuck!
```

- DRY

C++17

```
vector v = {1, 2, 3}; // Deduces vector<int>, etc.
lock_guard lck{smtx};
shared_lock lck2{smtx2};
scoped_lock assign_lock{smtx, lck2};
```

The Issue

One problem is that common cases lead to “non-deducible” template contexts. To see what can go wrong, suppose you want to initialize a vector from two iterators:

```
// b and e are iterators
vector v2(b, e); // How does the compiler know it is supposed to deduce the iterator's value_type?
```

- Compiler may not know that you’re trying to create a vector from the value_type of the iterator, not the iterator itself.

The Fix

```
// Inside <vector>
template<typename Iter> // Explain how to deduce from iterator pair!
vector(Iter, Iter) -> vector<typename iterator_traits<Iter>::value_type>;
// Outside <vector>
vector v2(b, e); // Works now!
```

- Specify deduction guides for your class templates.

Where?

- Bimap
- Circular Buffer
- Compute
- Container
- Dynamic Bitset
- Graph?
- Intrusive
- MultiArray
- …

Case Study

```
+void cdat_constructor() {
+    {
+    std::vector v = { 1, 2, 3, 4, 5, 6 };
+    circular_buffer cb{v.begin(), v.end()};
+    auto it = cb.begin() + 1;
+
+    BOOST_CHECK(it[0] == 2);
+    BOOST_CHECK(it[-1] == 1);
+    BOOST_CHECK(it[2] == 4);
+    }
+}
+
```

```
test — richardp@Richards-MacBook-Pro-2 — ..r_buffer/test — -zsh — 82×24
...patience...
...patience...
...found 2779 targets...
...updating 8 targets...
darwin.compile.c++ ../../../bin.v2/libs/circular_buffer/test/base_test.test/darwin
-4.2.1/debug/cxxstd-1z-iso/base_test.o
[base_test.cpp:38:21: error: no viable constructor or deduction guide for deduction]
[ of template arguments of 'circular_buffer'                                       ]
    circular_buffer cb{v.begin(), v.end()};
                    ^
../../../boost/circular_buffer/base.hpp:1035:14: note: candidate template ignored:
 couldn't infer template argument 'T'
    explicit circular_buffer(capacity_type buffer_capacity, const allocator_type&
alloc = allocator_type())
             ^
../../../boost/circular_buffer/base.hpp:1054:5: note: candidate template ignored:
couldn't infer template argument 'T'
    circular_buffer(size_type n, param_value_type item, const allocator_type& allo
c = allocator_type())
    ^
../../../boost/circular_buffer/base.hpp:1146:5: note: candidate template ignored:
couldn't infer template argument 'T'
    circular_buffer(InputIterator first, InputIterator last, const allocator_type&
 alloc = allocator_type())
```

- Circular buffer

Fix?

```
    }
};

template<typename Iter> circular_buffer(Iter b, Iter e) -> circular_buffer<typename Iter::value_type>;

// Non-member functions
```

The Concern

- Without consistency general users may not adopt CDAT.
- Without Boost consistent adoption Boost may fall out of favor.

References

- Michael Spertis: https://github.com/CppCon/CppCon2017/tree/master/Posters/Best%20Practices%20for%20Constructor%20Template%20Argument%20Deduction

- Michael Spertis: https://youtu.be/Tl2to07dfqI

-