rof_egnar
reversed adapter
for(auto x:reversed(range))

Thomas Corbat / Prof. Peter Sommerlad
Rapperswil, 06.07.2018

Cevelop
Your C++ deserves it

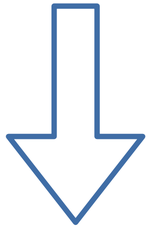
HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

1 The range-based `for` statement

`for (` *for-range-declaration* `:` *for-range-initializer* `)` *statement*

is equivalent to



```
{
    auto &&__range = for-range-initializer ;
    auto __begin = begin-expr ;
    auto __end = end-expr ;
    for ( ; __begin != __end; ++__begin ) {
        for-range-declaration = *__begin;
        statement
    }
}
```

lifetime extension

A naive range adapter that keeps just a reference

- **A simple adapter, what can go wrong**
- **just call rbegin for begin and rend for end**
- **OOPS, for temporaries we need the rvalue-reference overload**
- **but...**

```
template<typename Cont>
struct reverse{
   explicit constexpr
   reverse(Cont & c)
   :container(c){}

   explicit constexpr
   reverse(Cont && c) // allow binding to temporaries
   :container(c){}

   Cont& container;
   constexpr auto begin() { return std::rbegin(container);}
   constexpr auto begin() const { return std::rbegin(container);}
   constexpr auto end() { return std::rend(container);}
   constexpr auto end() const { return std::rend(container);}
};
```

- **does not compile**
- **OK, special case...**
- **CTAD to the rescue, match std::initializer_list by value**
  - Is that OK?

```
void testReverseInitializerList(){
   std::ostringstream out{};
   for(auto const &i:reverse({1,2,3,4,5,6})){
      out << i;
   }
   ASSERT_EQUAL("654321",out.str());
}
```

```
../src/Test.cpp:61:41: error: class template argument deduction failed:
  for(auto const &i:reverse({1,2,3,4,5,6})){
                                           ^
../src/Test.cpp:61:41: error: no matching function for call to
'reverse(<brace-enclosed initializer list>)'
```

```
template<typename T>
reverse(std::initializer_list<T>)→ reverse<std::initializer_list<T>>;
```

What can go wrong - using temporaries: Life-time extension is not transitive

- **when we keep a temporary internally by reference in the reverse object?**
- **How do we test that?**

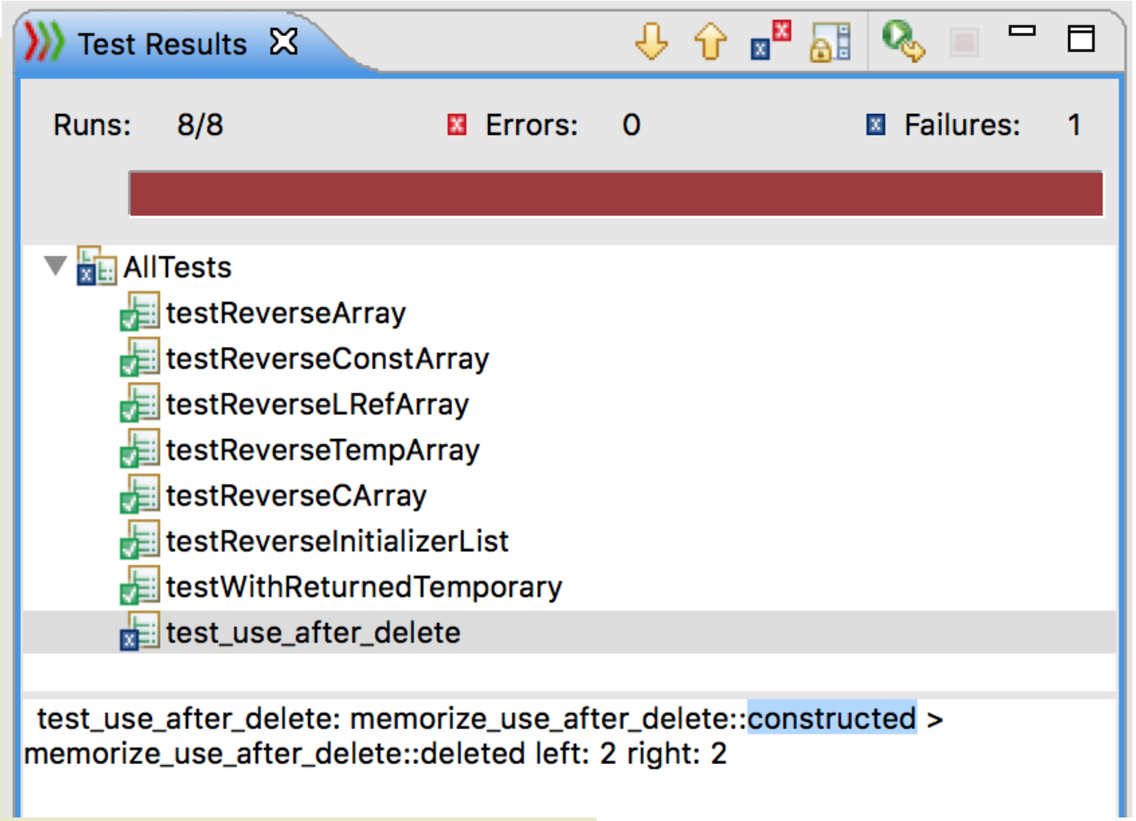
```
constexpr std::array<int,5> make_array(){
  return {1,2,3,4,5};
}
void testWithReturnedTemporary(){
  std::ostringstream out{};
  for(auto const &i:reverse(make_array())){
    out << i;
  }
  ASSERT_EQUAL("54321",out.str());
}
```

Testing for temporaries use after delete... needs ugly code


```
struct memorize_use_after_delete{
   static inline size_t deleted{};
   static inline size_t constructed{};
   memorize_use_after_delete(){
      ++constructed;
   }
   memorize_use_after_delete(memorize_use_after_delete const &){
      ++constructed;
   }
   memorize_use_after_delete(memorize_use_after_delete &&) noexcept {
      ++constructed;
   }
   ~memorize_use_after_delete(){
      ++deleted;
   }
};

std::vector<memorize_use_after_delete> factory(){
   return (std::vector<memorize_use_after_delete>(2));
}

void test_use_after_delete(){
   for(auto &i:reversed(factory())){
      ASSERT_GREATER(memorize_use_after_delete::constructed,memorize_use_after_delete::deleted);
   }
}
```



Another gotcha: std::reverse() exists... so we name our adapter "reversed"


```
#ifndef REVERSED_H_
#define REVERSED_H_
#include <type_traits>
#include <initializer_list>
#include <utility>
#include <iterator>

namespace adapter{

template<typename Cont>
struct reversed{
   explicit constexpr
   reversed(Cont & c)
   :container(c){}

   explicit constexpr
   reversed(std::remove_reference_t<Cont> &&c)
   :container(std::move(c)){}

   Cont container;
   constexpr auto begin() { return std::rbegin(container);}
   constexpr auto begin() const { return std::rbegin(container);}
   constexpr auto end() { return std::rend(container);}
   constexpr auto end() const { return std::rend(container);}
};
template<typename Cont>
reversed(Cont &) -> reversed<Cont &>;
template<typename Cont>
reversed(Cont  &&) -> reversed<Cont>;
template<typename T>
reversed(std::initializer_list<T>)-> reversed<std::initializer_list<T>>;
}
#endif /* REVERSED_H_ */
```

- **Deduction guides help**
- **Please help me find my bugs**
- **AFAIK Ranges reverse adapter does not work nicely with temporaries**

move temporary container into reversed

Keep container by reference if not a temporary

special treatment of initializer_list by value

Bonus Slide: Referring stuff obtained from temporaries is dangerous

- **Just an idea (may be worth a ISO C++ paper?)**
  - provide deleted overloads for begin(), end() etc for rvalue references.
  - might break already wrong code
  - members returning elements by reference should return by value for temporaries


```
void testTemporaryArrayAccess(){
   ASSERT_EQUAL(2,(std::array{1,2,3}).at(1));
   int &i = std::array{2,3}[0];
   i=1; // UB
}
void testBeginTemporaryShouldNotCompile(){
   auto it = std::array{1,2,3}.begin();
   ASSERT_EQUAL(1,*it);
}
```

```
constexpr iterator
begin() & noexcept
{ return iterator(data()); }
constexpr const_iterator
begin() const & noexcept
{ return const_iterator(data()); }
constexpr iterator
begin() && noexcept  = delete;
constexpr iterator
begin() const && noexcept  = delete;
```

```
constexpr reference
operator[](size_type __n)  & noexcept
{ return _AT_Type::_S_ref(_M_elems, __n); }
constexpr const_reference
operator[](size_type __n) const & noexcept
{ return _AT_Type::_S_ref(_M_elems, __n); }
constexpr value_type
operator[](size_type __n) && noexcept
{ return std::move(_M_elems[__n]); }
```

- https://github.com/PeterSommerlad/ReverseAdapter

Cevelop

Your C++ deserves it

Download IDE at:
www.cevelop.com