Customization Point Objects / Niebloids

PV264 Advanced Programming in C++

Nikola Beneš Jan Mrázek Vladimír Štill

Faculty of Informatics, Masaryk University

Autumn 2020

Customization Points

- allow programmers to customize certain functions (in their own namespace)

swap

- generic code is supposed to call `swap` like this:

```
using std::swap;
swap(a, b);
```

- this is also how the standard library calls `swap` (see https://en.cppreference.com/w/cpp/named_req/Swappable)

Customization Points

begin / end

- the semantics of
  `for (type var: container) { do_something(var) }`
  is actually something like this:

```
{
    using std::begin;
    using std::end;
    auto&& __l = container;
    auto __i = begin(__l);
    auto __e = end(__l);
    for (; __i != __e; ++__i) {
        type var = *__i;
        do_something(var);
    }
}
```

Customization Points

Issues with customization points

- need to type more
- easy to use incorrectly (`std::swap(a, b)`)
- do not work well with **concepts**
  - we would like to specify the constraints in one place and let them be checked regardless of customization

Customization Points

Solution?

- wrap it in a function

```
namespace my_std {
template<typename T> requires /* swappable constraints */
void my_swap(T& a, T& b) {
    using std::swap;
    swap(a, b);
}
} // namespace my_std
```

- is this OK?

Customization Points

Solution?

- wrap it in a function

```
namespace my_std {
template<typename T> requires /* swappable constraints */
void my_swap(T& a, T& b) {
    using std::swap;
    swap(a, b);
}
} // namespace my_std
```

- is this OK? not really

```
/* somewhere in code */
using namespace my_std;
my_swap(x, y);  // what does this call?
```

- we would like to **switch off ADL**

Ranges Library vs. ADL

```
// std:
template< /* ... */ >
bool all_of( InputIt first, InputIt last, UnaryPredicate p );

// ranges:
template< /* ... */ > constexpr
bool all_of( I first, S last, Pred pred, Proj proj = {} );

/* somewhere in code */
    using namespace std::ranges;
    bool result = all_of(iter1, iter2, pred);
```

ADL breaks ranges

- `std` algorithms are usually more specialized than `ranges` algorithms
- we would like to **switch off ADL**

Switching off ADL

Niebloids

- a trick invented by Eric Niebler
- note: ADL only works for *functions*

Switching off ADL

Niebloids

- a trick invented by Eric Niebler
- note: ADL only works for *functions*, not for **function objects**!
- idea: instead of a function, create a (global) function object

```
namespace my_std {
  namespace _detail {
    struct _my_swap {
        template<typename T> requires /* constraints */
        void operator()(T& a, T& b) const {
            using std::swap;
            swap(a, b);
        }
    };
  } // namespace _detail
// since C++17 we can have inline variables, yay!
inline constexpr _detail::_my_swap my_swap{};
} // namespace my_std
```

Niebloids

Properties

- are *invisible* to ADL (cannot be found through ADL)
- inhibit ADL (if one is found, no ADL is performed)
- the template parameters of the function call operator cannot be explicitly specified
  - but some objects are themselves templated, such as `views::elements`

Niebloids

Properties

- are *invisible* to ADL (cannot be found through ADL)
- inhibit ADL (if one is found, no ADL is performed)
- the template parameters of the function call operator cannot be explicitly specified
  - but some objects are themselves templated, such as `views::elements`

Niebloids in the Standard Library (Ranges)

- true CPOs (customization point objects)
  - `swap`, `begin`, `end`, `size`, `data`, `empty`, ...
- other niebloids
  - `cbegin`, `cend`, ...
  - all the range algorithms
  - range adaptors etc.