



C++20 Ranges und Constrained Algorithms

Felix Racz, Tim Wende

29. Mai 2022



Inhalt

1. Vorwissen

- Iteratoren
- Concepts

2. Motivation

- Schlechte Fehlermeldungen
- Unübersichtlicher Code
- (Compiler-) Geschwindigkeitsoptimierung

3. Anwendung

- Constrained Algorithms
- Projection
- Range Concepts
- Views

4. Aufgaben



Ranges

A *range* is something that can be iterated over.



Constrained Algorithm

A constrained algorithm is an algorithm that takes ranges.



Ranges in anderen Programmiersprachen

■ Java:

```
IntStream.range(0, 10)  
          .filter(x -> w%2 == 1)  
          .map(x -> x*x)
```



Ranges in anderen Programmiersprachen

■ Java:

```
IntStream.range(0, 10)
          .filter(x -> x%2 == 1)
          .map(x -> x*x)
```

■ Python:

```
map(lambda x: x*x,
    filter(lambda x: x%2 == 1,
        range(10)
    )
)
```



Vorwissen



Iteratoren

Der Begriff *Iterator* stammt aus dem Bereich der Softwareentwicklung und bezeichnet einen Zeiger, mit dem die Elemente einer Menge durchlaufen werden können (z. B. eine Liste, ein Vector etc.).



Iteratoren - Beispiel (Code)

```
std::vector<int> v{ 3, 1, 4, 1, 5, ... };

// std::vector<int>::iterator
for (auto it = v.begin(); it != v.end(); ++it)
    std::cout << *it << ", ";

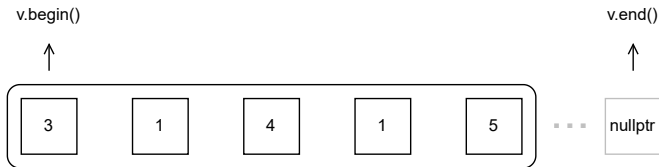
std::cout << std::endl;

// 3, 1, 4, 5, 1, ...,
```



Iteratoren - Beispiel (Graphisch)

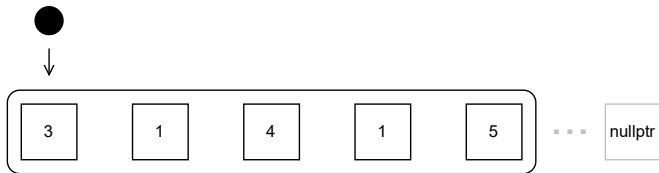
```
auto it = v.begin();
```





Iteratoren - Beispiel (Graphisch)

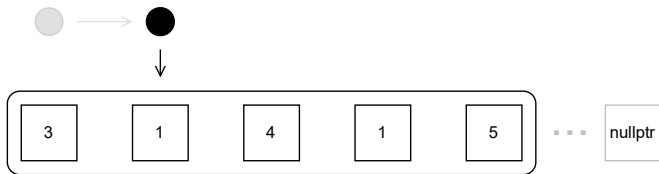
```
std::cout << *it << ", ";
```





Iteratoren - Beispiel (Graphisch)

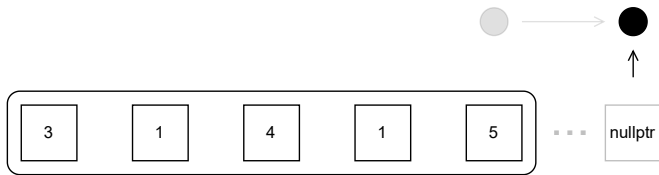
```
++it;
```





Iteratoren - Beispiel (Graphisch)

```
it == v.end();
```





Concepts

Concepts allow us to control the instantiation of templates by testing syntactic conditions.

It is a compile-time predicate which is true if the given type(s) meet the requirements.



Concepts - Beispiel

```
template <typename T>  
    requires string_convertible<T>  
std::string convert(const T& t) { return t.to_string(); }
```



Motivation



Schlechte Fehlermeldungen

```
std::list<int> list{ 3, 1, 4, 1, 5, ... };  
std::sort(list.begin(), list.end());
```



Zu lange Fehlermeldungen

In file included from error_messages.cpp:2:

In file included from /usr/bin/./lib/gcc/x86_64-linux-gnu/11/./././././
/usr/bin/./lib/gcc/x86_64-linux-gnu/11/././././././include/c++/11/bits

error: invalid operands to binary expression ('std::_List_iterator<int>
std::_lg(__last - __first) * 2,
~~~~~ ^ ~~~~~

/usr/bin/./lib/gcc/x86\_64-linux-gnu/11/././././././include/c++/11/bits  
std::\_sort(\_\_first, \_\_last, \_\_gnu\_cxx::\_\_ops::\_\_iter\_less\_iter  
^

error\_messages.cpp:8:10: note: in instantiation of function template sp  
std::sort(list.begin(), list.end());  
^

/usr/bin/./lib/gcc/x86\_64-linux-gnu/11/././././././include/c++/11/bits  
operator-(const move\_iterator<\_IteratorL>& \_\_x,  
^



## Zu lange Fehlermeldungen

In file included from error\_messages.cpp:2:

In file included from /usr/bin/./lib/gcc/x86\_64-linux-gnu/11/./././././  
/usr/bin/./lib/gcc/x86\_64-linux-gnu/11/././././././include/c++/11/bits

**error: invalid operands to binary expression** ('std::\_List\_iterator<int>  
std::\_lg(\_\_last - \_\_first) \* 2,  
~~~~~ ^ ~~~~~

/usr/bin/./lib/gcc/x86_64-linux-gnu/11/././././././include/c++/11/bits
std::_sort(__first, __last, __gnu_cxx::__ops::__iter_less_iter
^

error_messages.cpp:8:10: note: in instantiation of function template sp
std::sort(list.begin(), list.end());
^

/usr/bin/./lib/gcc/x86_64-linux-gnu/11/././././././include/c++/11/bits
operator-(const move_iterator<_IteratorL>& __x,
^



Zu lange Fehlermeldungen

In file included from error_messages.cpp:2:

In file included from /usr/bin/./lib/gcc/x86_64-linux-gnu/11/./././././
/usr/bin/./lib/gcc/x86_64-linux-gnu/11/././././././include/c++/11/bits

```
    if (__last - __first > int(_S_threshold))
```

```
        ~~~~~ ^ ~~~~~
```

/usr/bin/./lib/gcc/x86_64-linux-gnu/11/././././././include/c++/11/bits

```
    std::__final_insertion_sort(__first, __last, __comp);
```

```
    ^
```

/usr/bin/./lib/gcc/x86_64-linux-gnu/11/././././././include/c++/11/bits

```
    std::__sort(__first, __last, __gnu_cxx::__ops::__iter_less_iter
```

```
    ^
```

error_messages.cpp:8:10: note: in instantiation of function template sp

```
    std::sort(list.begin(), list.end());
```

```
    ^
```



Zu lange Fehlermeldungen

```
/usr/bin/../../lib/gcc/x86_64-linux-gnu/11/../../include/c++/11/bits  
operator-(const reverse_iterator<_IteratorL>& __x,  
^
```

```
/usr/bin/../../lib/gcc/x86_64-linux-gnu/11/../../include/c++/11/bits  
operator-(const move_iterator<_IteratorL>& __x,  
^
```

2 errors generated.



Bessere Fehlermeldungen

```
std::list<int> list{ 3, 1, 4, 1, 5, ... };  
std::ranges::sort(list.begin(), list.end());
```



Bessere Fehlermeldungen

```
error_messages.cpp:8:5:
error: no matching function for call
to object of type 'const std::ranges::__sort_fn'
    std::ranges::sort(list.begin(), list.end());
    ~~~~~
/usr/bin/../../lib/gcc/x86_64-linux-gnu/11/../../../../include/c++/11/bits
    operator()(_Iter __first, _Sent __last,
    ^
/usr/bin/../../lib/gcc/x86_64-linux-gnu/11/../../../../include/c++/11/bits
note: because 'std::_List_iterator<int>' does not satisfy
'random_access_iterator'
    template<random_access_iterator _Iter, sentinel_for<_Iter> _Sent,
    ^
[...]
```



Bessere Fehlermeldungen

error_messages.cpp:8:5:

error: no matching function for call

to object of type 'const std::ranges::__sort_fn'

std::ranges::sort(list.begin(), list.end());

^~~~~~

/usr/bin/../../lib/gcc/x86_64-linux-gnu/11/../../include/c++/11/bits

operator()(_Iter __first, _Sent __last,

^

/usr/bin/../../lib/gcc/x86_64-linux-gnu/11/../../include/c++/11/bits

note: because 'std::List_iterator<int>' does not satisfy

'random_access_iterator'

template<random_access_iterator _Iter, sentinel_for<_Iter> _Sent,

^

[...]



Unübersichtlicher Code

```
std::vector<int> v{ 3, 1, 4, 1, 5, ... };  
  
std::find(v.begin(), v.end(), 5);  
std::sort(v.begin(), v.end());
```



Unübersichtlicher Code

```
std::vector<int> v{ 3, 1, 4, 1, 5, ... };  
  
std::ranges::find(v, 5);  
std::ranges::sort(v);
```



Unübersichtlicher Code - Implementierung

```
1. std::ranges::sort(v);
```



Unübersichtlicher Code - Implementierung

1. `std::ranges::sort(v);`
2. `struct sort_fn {...}`



Unübersichtlicher Code - Implementierung

```
1. std::ranges::sort(v);  
2. struct sort_fn {...}  
3. std::ranges::sort(  
    std::ranges::begin(v),  
    std::ranges::end(v)  
);
```



Unübersichtlicher Code - Implementierung

1. `std::ranges::sort(v);`
2. `struct sort_fn {...}`
3. `std::ranges::sort(
 std::ranges::begin(v),
 std::ranges::end(v)
);`
4. `std::ranges::sort(v.begin(), v.end());`



Unübersichtlicher Code - Implementierung

1. `std::ranges::sort(v);`
2. `struct sort_fn {...}`
3. `std::ranges::sort(
 std::ranges::begin(v),
 std::ranges::end(v)
);`
4. `std::ranges::sort(v.begin(), v.end());`
5. `std::sort(v.begin(), v.end());`



(Compiler-) Geschwindigkeitsoptimierung

```
std::vector<int> v{ 3, 1, 4, 1, 5, ... };  
  
std::find(  
    v.begin(), // iterator -> v[0]  
    v.end(),   // iterator -> nullptr  
    5          // ist immer im vector enthalten  
);
```




(Compiler-) Geschwindigkeitsoptimierung - Implementierung

```
std::vector<int>::iterator find(  
    std::vector<int>::iterator first,  
    std::vector<int>::iterator last,  
    const int& val  
) {  
    for (; first != last; ++first)  
        if (*first == val) break;  
  
    return first;  
}
```



A *sentinel* is some type that is `equality_comparable_with` its corresponding iterator, which denotes the end of the range.

Für diese Beispiel benötigen wir den `unreachable_sentinel`.



(Compiler-) Geschwindigkeitsoptimierung

```
std::vector<int> v{ 3, 1, 4, 1, 5, ... };

std::ranges::find(
    v.begin(), // iterator -> v[0]
    std::ranges::unreachable_sentinel, // == it -> false
    5 // ist immer im vector enthalten
);
```



(Compiler-) Geschwindigkeitsoptimierung - Implementierung

```
std::vector<int>::iterator find(  
    std::vector<int>::iterator          first,  
    std::ranges::unreachable_sentinel_t last,  
    const int&                          val  
) {  
    // first == last -> false  
    for (; first != last; ++first)  
        if (*first == val) break;  
  
    return first;  
}
```



(Compiler-) Geschwindigkeitsoptimierung - Implementierung

```
std::vector<int>::iterator find(  
    std::vector<int>::iterator          first,  
    std::ranges::unreachable_sentinel_t last,  
    const int&                          val  
) {  
    // first == last -> false  
    for (; true; ++first)  
        if (*first == val) break;  
  
    return first;  
}
```



(Compiler-) Geschwindigkeitsoptimierung

Mit dem bekannten Iterator `.end()` wird jedes Element des Vectors 2 mal „angefasst“:

- `first != last`
- `*first == val`

Durch Sentinals verringern wir diese Zugriffe um die Hälfte, haben jedoch Probleme, wenn `val` nicht im Vector enthalten ist.

- `*first == val`



Anwendung



Constrained Algorithms <algorithm>

- `std::ranges::find()`
→ `std::find()`



Constrained Algorithms <algorithm>

- `std::ranges::find()`
→ `std::find()`
- `std::ranges::sort()`
→ `std::sort()`



Constrained Algorithms <algorithm>

- `std::ranges::find()`
→ `std::find()`
- `std::ranges::sort()`
→ `std::sort()`
- `std::ranges::equal()`
→ `std::equal()`



Constrained Algorithms <algorithm>

- `std::ranges::find()`
→ `std::find()`
- `std::ranges::sort()`
→ `std::sort()`
- `std::ranges::equal()`
→ `std::equal()`
- `std::ranges::move()`
→ `std::move()`



Constrained Algorithms <algorithm>

- `std::ranges::find()`
→ `std::find()`
- `std::ranges::sort()`
→ `std::sort()`
- `std::ranges::equal()`
→ `std::equal()`
- `std::ranges::move()`
→ `std::move()`
- `std::ranges::minmax_element()`
→ `std::minmax_element()`



Constrained Algorithms <numeric>

C++23



Projection

A *projection* is a unary callable which may be passed to most algorithms.

Projections modify the view of the data that the algorithms sees.



Projection - Beispiel

```
struct Pokemon {  
    int id;  
    std::string name;  
    ...  
}  
  
struct PokedexEintrag {  
    int pokemon_id;  
    std::string beschreibung;  
}  
  
std::vector<Pokemon> pokemon;  
std::vector<PokedexEintrag> pokedex;
```



Projection - Beispiel

Aufgabe:

Finden Sie heraus, ob jedes Pokémon einen PokédexEintrag hat



Projection - Beispiel

Erster Ansatz:

1. Sortiere alle Pokémon (nach `id`)



Projection - Beispiel

Erster Ansatz:

1. **Sortiere alle Pokémon (nach `id`)**
2. **Sortiere alle PokédexEinträge (nach `pokemon_id`)**



Projection - Beispiel

Erster Ansatz:

1. **Sortiere alle Pokémon (nach `id`)**
2. **Sortiere alle PokédexEinträge (nach `pokemon_id`)**
3. **Gehe von oben alle Einträge durch. Sobald die eine Liste die andere „überholt“, gib `false` zurück.**



Projection - Beispiel

Erster Ansatz:

1. **Sortiere alle Pokémon (nach `id`)**
2. **Sortiere alle PokédexEinträge (nach `pokemon_id`)**
3. **Gehe von oben alle Einträge durch. Sobald die eine Liste die andere „überholt“, gib `false` zurück.**
4. **Wenn wir am Ende der Listen sind, gib `true` zurück**



Projection - Beispiel

```
std::sort(pokemon.begin(), pokemon.end(),  
    [] (const Pokemon& p1, const Pokemon& p2) {  
        return p1.id <= p2.id;  
    });
```

```
std::sort(pokedex.begin(), pokedex.end(),  
    [] (const PokedexEintrag& e1,  
        const PokedexEintrag& e2) {  
        return e1.pokemon_id <= e2.pokemon_id;  
    });
```



Projection - Beispiel

```
std::equal(
    pokemon.begin(), pokemon.end(),
    pokedex.begin(), pokedex.end(),
    [] (const Pokemon& p, const PokedexEintrag& e) {
        return p.id == e.pokemon_id;
    });
```



Projection - Beispiel

Zweiter Ansatz:

- **Erweitere Iteratoren durch Ranges**

- Ersetze `.begin()` bzw. `.end()`



Projection - Beispiel

```
std::ranges::sort(pokemon,  
    [] (const Pokemon& p1, const Pokemon& p2) {  
        return p1.id <= p2.id;  
    });
```

```
std::ranges::sort(pokedex,  
    [] (const PokedexEintrag& e1,  
        const PokedexEintrag& e2) {  
        return e1.pokemon_id <= e2.pokemon_id;  
    });
```




Projection - Beispiel

```
std::ranges::equal(pokemon, pokedex,  
    [] (const Pokemon& p, const PokedexEintrag& e) {  
        return p.id == e.pokemon_id;  
    });
```



Projection - Beispiel

Dritter Ansatz:

■ Erweitere Ranges durch Projection

- Reduziere jedes Pokémon auf einen `int`
- Reduziere analog jeden PokédexEintrag auf einen `int`
- Überlasse dem Compiler den Umgang mit diesen primitiven Datentypen



Projection - Beispiel

```
std::ranges::sort(pokemon, std::ranges::less{},  
    [] (const Pokemon& p) { return p.id; });
```

```
std::ranges::sort(pokedex, std::ranges::less{},  
    [] (const PokedexEintrag& e) { return e.pokemon_id; });
```

```
std::ranges::equal(pokemon, pokedex,  
    std::ranges::equal_to{},  
    [] (const Pokemon& p) { return p.id; }  
    [] (const PokedexEintrag& e) { return e.pokemon_id; }  
);
```



Projection - Beispiel

Finaler Ansatz:

- Ersetze „getter“-Lambdas durch die direkten Attribute



Projection - Beispiel

```
std::ranges::sort(pokemon, std::ranges::less{},  
    &Pokemon::id);
```

```
std::ranges::sort(pokedex, std::ranges::less{},  
    &PokedexEintrag::pokemon_id);
```

```
std::ranges::equal(pokemon, pokedex,  
    std::ranges::equal_to{},  
    &Pokemon::id, &PokedexEintrag::pokemon_id);
```



Projection - Beispiel

✧ **Finaler Ansatz:** ✧



Projection - Beispiel

```
std::ranges::sort(pokemon, {},  
    &Pokemon::id);
```

```
std::ranges::sort(pokedex, {},  
    &PokedexEintrag::pokemon_id);
```

```
std::ranges::equal(pokemon, pokedex, {},  
    &Pokemon::id, &PokedexEintrag::pokemon_id);
```



Projection - Beispiel

Zusammengefasst:

```
// stl  
std::sort(pokemon.begin(), pokemon.end(),  
    [] (const Pokemon& p1, const Pokemon& p2) {  
        return p1.id <= p2.id;  
    });
```

```
// ranges  
std::ranges::sort(pokemon, {}, &Pokemon::id);
```




Range Concepts <ranges>

■ `std::ranges::input_range`
→ `std::cin`



Range Concepts <ranges>

- `std::ranges::input_range`
 - `std::cin`
- `std::ranges::bidirectional_range`
 - `std::list`



Range Concepts <ranges>

- `std::ranges::input_range`
 - `std::cin`
- `std::ranges::bidirectional_range`
 - `std::list`
- `std::ranges::random_access_range`
 - `std::unordered_map`



Range Concepts <ranges>

- `std::ranges::input_range`
 - `std::cin`
- `std::ranges::bidirectional_range`
 - `std::list`
- `std::ranges::random_access_range`
 - `std::unordered_map`
- `std::ranges::sized_range`
 - `std::vector`



Views

- Views leben in `std::ranges::views`



Views

- Views leben in `std::ranges::views`
- Es gibt den alias `std::views`



Views

- Views leben in `std::ranges::views`
- Es gibt den alias `std::views`
- Was kennzeichnet Views aus?



Views

- **Views leben in** `std::ranges::views`
- **Es gibt den alias** `std::views`
- **Was kennzeichnet Views aus?**
 - lazy evaluation



Views

- **Views leben in** `std::ranges::views`
- **Es gibt den alias** `std::views`
- **Was kennzeichnet Views aus?**
 - lazy evaluation
 - besitzen Inhalt nicht, referenzieren nur



Views

- **Views leben in** `std::ranges::views`
- **Es gibt den alias** `std::views`
- **Was kennzeichnet Views aus?**
 - lazy evaluation
 - besitzen Inhalt nicht, referenzieren nur
 - constant time copy and move



Views

- **Views leben in** `std::ranges::views`
- **Es gibt den alias** `std::views`
- **Was kennzeichnet Views aus?**
 - lazy evaluation
 - besitzen Inhalt nicht, referenzieren nur
 - constant time copy and move
 - einfach und effizient kombinierbar



Views - Beispiel

Aufgabe:

Berechnen Sie für alle Pokémon des Typs Feuer den BMI

$$\text{bmi} = \frac{\text{gewicht}}{\text{groesse}^2}$$



Klassisch

```
std::vector<Pokemon> get_pokemon() {...};  
auto pokemon = get_pokemon();  
std::vector<int> bmis{};  
  
for (const Pokemon& p : vec) {  
    if (p.primaer_typ == "Feuer" || p.sekundaer_typ = "Feuer") {  
        int bmi = p.gewicht() / (p.groesse * p.groesse);  
        bmis.push_back(bmi);  
    }  
}
```



Alte STL

```
std::vector<Pokemon> get_pokemon() {...};  
auto pokemon = get_pokemon();  
std::vector<double> bmis;  
auto hat_typ_feuer = [](const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
auto bmi = [](const Pokemon &p) {  
    return (p.gewicht / (p.groesse * p.groesse));  
};
```



Alte STL

```
std::copy_if(
    pokemon.begin(), pokemon.end(),
    std::back_inserter(feuer_pokemon),
    hat_typ_feuer);
std::transform(
    feuer_pokemon.begin(), feuer_pokemon.end(),
    std::back_inserter(bmi),
    bmi);
```



Mit Ranges

```
std::vector<Pokemon> get_pokemon() {...};  
  
auto pokemon = get_pokemon();  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
  
auto bmi = [] (const Pokemon &p) {  
    return (p.gewicht / (p.groesse * p.groesse));  
};  
  
auto view = transform(filter(vec, hat_typ_feuer), bmi);
```




Mit Ranges und Pipes

```
std::vector<Pokemon> get_pokemon() {...};  
  
auto pokemon = get_pokemon();  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
  
auto bmi = [] (const Pokemon &p) {  
    return (p.gewicht / (p.groesse * p.groesse));  
};  
  
auto view = vec | filter(hat_typ_feuer) | transform(bmi);
```



Mit Ranges und Pipes

```
std::vector<Pokemon> get_pokemon() {...};  
auto pokemon = get_pokemon();  
auto view = vec  
| filter([](const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
    || p.sekundaer_typ == "Feuer";  
})  
| transform([](const Pokemon &p) {  
    return (p.gewicht / (p.groesse * p.groesse));  
});
```



Dangling Iterator

```
std::vector<Pokemon> get_pokemon() {...};
```

```
auto pokemon = get_pokemon();  
auto it = min_element(  
    pokemon, {},  
    &Pokemon::groesse  
);
```

```
std::cout << *it << std::endl;
```



Dangling Iterator

```
std::vector<Pokemon> get_pokemon() {...};  
  
auto it = min_element(  
    get_pokemon(), {},  
    &Pokemon::groesse  
);  
std::cout << *it << std::endl;
```



Dangling Iterator

```
std::vector<Pokemon> get_pokemon() {...};

auto it = min_element(
    get_pokemon(), {},
    &Pokemon::groesse
);
std::cout << *it << std::endl;
//error: no match for 'operator*'
//(operand type is 'std::ranges::dangling')
```



Dangling View

```
std::vector<Pokemon> get_pokemon() {...};  
  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
  
auto pokemon = get_pokemon();  
auto view = views::filter(pokemon, hat_typ_feuer);  
for(const Pokemon& p : view) {  
    std::cout << p << std::endl;  
}
```



Dangling View

```
std::vector<Pokemon> get_pokemon() {...};  
  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
  
auto view = views::filter(get_pokemon(), hat_typ_feuer);  
for(const Pokemon& p : view)  
    std::cout << p << std::endl;
```



Dangling View

```
std::vector<Pokemon> get_pokemon() {...};  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
  
auto view = views::filter(get_pokemon(), hat_typ_feuer);  
for(const Pokemon& p : view)  
    std::cout << p << std::endl;  
  
//note: candidate: 'template<class _Range, class _Pred>  
//requires (viewable_range<_Range>)
```




Dangling View

```
std::vector<Pokemon> get_pokemon() {...};  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
auto get_feuer_pokemon(const std::vector<Pokemon>& p) {  
    return p | views::filter(hat_typ_feuer);  
}  
  
auto view = get_feuer_pokemon(get_pokemon())  
for(const Pokemon& p : view)  
    std::cout << p << std::endl;
```



Dangling View

```
std::vector<Pokemon> get_pokemon() {...};  
auto hat_typ_feuer = [] (const Pokemon &p) {  
    return p.primaer_typ == "Feuer"  
        || p.sekundaer_typ == "Feuer";  
};  
auto get_feuer_pokemon(const std::vector<Pokemon>& p) {  
    return p | views::filter(hat_typ_feuer);  
}  
  
auto view = get_feuer_pokemon(get_pokemon())  
for(const Pokemon& p : view)  
    std::cout << p << std::endl;  
  
//segfault
```



Dangling View

```
std::string s{" Das Ist Ein Langer String. "};  
auto s_view(const std::string& s) {  
    [...]  
    return std::string_view{s};  
}  
std::string_view sv = s_view(s);  
auto view = sv  
    | filter([](char c) { return c!= ' '; });  
for (char c : view) {  
    std::cout << c;  
} // "DasIstEinLangerString"
```



Dangling View

```
std::string s{" Das Ist Ein Langer String. "};  
auto s_view(const std::string& s) {  
    [...]  
    return std::string_view{s};  
}  
auto view = s_view(s)  
    | filter([](char c) { return c!= ' '; });  
for (char c : view) {  
    std::cout << c;  
} // "DasIstEinLangerString"
```



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?

Alles was eine `std::ranges::viewable_range` ist!



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?

Alles was eine `std::ranges::viewable_range` ist!

- **Ivalue reference (alles mit einem Namen)**



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?

Alles was eine `std::ranges::viewable_range` ist!

- **Ivalue reference (alles mit einem Namen)**
- `std::borrowed_range`



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?

Alles was eine `std::ranges::viewable_range` ist!

- **Ivalue reference (alles mit einem Namen)**
- `std::borrowed_range`
 - Alle views



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?

Alles was eine `std::ranges::viewable_range` ist!

- **Ivalue reference (alles mit einem Namen)**
- `std::borrowed_range`
 - Alle views
 - Typen mit `std::enable_borrowed_range<T> true`



`std::ranges::viewable_range`

Was kann man einem Range adaptor alles übergeben?

Alles was eine `std::ranges::viewable_range` ist!

- **Ivalue reference (alles mit einem Namen)**
- `std::borrowed_range`
 - Alle views
 - Typen mit `std::enable_borrowed_range<T> true`
 - `std::span`, `std::string_view`



Performance

```
std::vector<long long> eager_transform(  
    const std::vector<long long> &vec) {  
    std::vector<long long> result;  
    for (long long i : vec) {  
        if (i % 2 == 0){  
            result.push_back(i * i);  
        }  
    }  
    return result;  
}
```



Performance

```
auto lazy_transform(  
    const std::vector<long long> &vec) {  
    auto is_even = [](long long i){  
        return i % 2 == 0;  
    };  
    auto square = [](long long i){  
        return i * i;  
    };  
    return vec | filter(is_even) | transform(square);  
}
```



Performance

```
long long do_something(auto &vec, int n) {  
    long long sum = 0;  
    for (auto i : vec){  
        sum += i; n--;  
        if (n < 0)  
            break;  
    }  
    return sum;  
}
```



Performance

```
int main() {  
    std::vector<int> v = get_vektor(100000);  
    auto v1 = eager_transform(v); // 3018μs  
    auto v2 = lazy_transform(v);  // 0μs  
    do_something(v1, 100000);     // 425μs  
    do_something(v2, 100000);     // 8293μs  
    do_something(v1, 10000);      // 114μs  
    do_something(v2, 10000);      // 1352μs  
}
```



Eigene View

```
template <std::ranges::input_range R>
    requires std::ranges::viewable_range<R>
    class view_convert : public std::ranges::view_base {
    private:
        R r;

        struct iterator_type : std::ranges::iterator_t<R>
```




Eigene View - Iterator I

```
struct iterator_type : std::ranges::iterator_t<R> {  
    using base = std::ranges::iterator_t<R>;  
    iterator_type() = default;  
    iterator_type(base const &b) : base{b} {}  
  
    iterator_type &operator++() {  
        ++static_cast<base &>(*this);  
        return (*this);  
    }  
}
```



Eigene View - Iterator II

```
int operator*() const {  
    auto original = *static_cast<base>(*this);  
    return original*10;  
}  
  
int operator[](size_t n) const  
requires std::ranges::random_access_range<R> {  
    auto original = (static_cast<base>(*this))[n];  
    return original*10;  
}  
  
};
```



Eigene View I

```
public:
    view_convert(R &&r) : r{std::forward<R>(r)} {}

    iterator_type begin() const {
        return std::begin(r);
    }

    iterator_type end() const {
        return std::end(r);
    }
```



Eigene View II

```
int operator[](size_t n) const
requires std::ranges::random_access_range<R>{
    return begin()[n];
}
};
```



Eigene View - Adaptor I

```
template <typename R>
view_convert(R &&) -> view_convert<R>;

struct convert_view_fn
{
    template <typename R>
    auto operator()(R &&r) const
    {
        return view_convert{std::forward<R>(r)};
    }
}
```



Eigene View - Adaptor II

```
template <typename R>
friend auto operator|(R &&r, convert_view_fn const &)
{
    return view_convert{std::forward<R>(r)};
}

};

namespace view
{
    convert_view_fn convert;
}
```



Eigene View - Beispiel I

```
std::vector<int> vec{1, 3, 5, 7, 9};  
auto view = vec | view::convert;  
std::cout << view[2] << std::endl; //50  
for (auto i : view) {  
    std::cout << i << ", ";  
}  
std::cout << std::endl; // 10, 30, 50, 70, 90,
```



Aufgaben



Aufgaben

https://github.com/tmwnd/adv_cpp_example.git



pokemon.hpp

```
struct Pokemon;      // id, name, primaer_typ, ...
struct Entwicklung;  // pokemon_id, level

std::vector<Pokemon> get_pokemon();
std::map<int, Entwicklung> get_entwicklung();
```



`pokemon_ranges_q.cpp`

Aufgabe:

Überführen Sie die gegebenen STL Algorithmen in Ranges



pokemon_ranges_q.cpp

Aufgabe:

Überführen Sie die gegebenen STL Algorithmen in Ranges

- a) **Kopieren Sie alle Starter Pokémon (inkl. Entwicklungen) in einen neuen Vector**



pokemon_ranges_q.cpp

Aufgabe:

Überführen Sie die gegebenen STL Algorithmen in Ranges

- a) **Kopieren Sie alle Starter Pokémon (inkl. Entwicklungen) in einen neuen Vector**
- b) **Sortieren Sie diese nun nach Typ**



pokemon_ranges_q.cpp

Aufgabe:

Überführen Sie die gegebenen STL Algorithmen in Ranges

- a) **Kopieren Sie alle Starter Pokémon (inkl. Entwicklungen) in einen neuen Vector**
- b) **Sortieren Sie diese nun nach Typ**
- c) **Finden Sie in dem Vector nun das „beste“ Pokémon**



pokemon_ranges_q.cpp

Aufgabe:

Überführen Sie die gegebenen STL Algorithmen in Ranges

- a) **Kopieren Sie alle Starter Pokémon (inkl. Entwicklungen) in einen neuen Vector**
- b) **Sortieren Sie diese nun nach Typ**
- c) **Finden Sie in dem Vector nun das „beste“ Pokémon**
- d) **Zuletzt geben Sie das Pokémon mit der kleinsten und größten ID aus**



`pokemon_ranges_q.cpp`

Zusatzaufgaben:



`pokemon_ranges_q.cpp`

Zusatzaufgaben:

✨) **Kann man irgendwo projections benutzen?**



`pokemon_ranges_q.cpp`

Zusatzaufgaben:

- ✦) Kann man irgendwo projections benutzen?
- ✦) Kann man überall projection benutzen?



`pokemon_ranges_q.cpp`

Zusatzaufgaben:

- ✧) Kann man irgendwo projections benutzen?
- ✧) Kann man überall projection benutzen?
- ✧) Wie sieht es mit der Performance aus?



pokemon_ranges_q.cpp

```
auto pokemon = get_pokemon();
```



pokemon_ranges_q.cpp

```
auto filter_starter = [](const Pokemon& p) {  
    return p.id <= 9;  
};  
auto sort_id = [](const Pokemon& p1, const Pokemon& p2) {  
    return p1.id < p2.id;  
};
```



pokemon_ranges_q.cpp

```
auto type_value = [](const Pokemon& p) {  
    return p.primaer_typ * 100 + p.sekundaer_typ;  
};  
auto sort_type = [&type_value](const Pokemon& p1,  
    const Pokemon& p2) {  
    return type_value(p1) < type_value(p2);  
};  
auto find_best = [](const Pokemon& p) {  
    return p.id == 4;  
};
```



`pokemon_ranges_q.cpp`

Code!



`pokemon_views_q.cpp`

Aufgabe:

Errechnen Sie das durchschnittliche Entwicklungslevel aller Feuerpokemon



`pokemon_views_q.cpp`

Aufgabe:

Errechnen Sie das durchschnittliche Entwicklungslevel aller Feuerpokemon

- a) **Erstellen Sie zuerst eine View die alle Pokémon mit Primärtyp Feuer enthält**



`pokemon_views_q.cpp`

Aufgabe:

Errechnen Sie das durchschnittliche Entwicklungslevel aller Feuerpokemon

- a) **Erstellen Sie zuerst eine View die alle Pokémon mit Primärtyp Feuer enthält**
- b) **Transformieren Sie die View sodass Sie zu jedem Pokémon das Entwicklungslevel erhalten**



`pokemon_views_q.cpp`

Aufgabe:

Errechnen Sie das durchschnittliche Entwicklungslevel aller Feuerpokemon

- a) **Erstellen Sie zuerst eine View die alle Pokémon mit Primärtyp Feuer enthält**
- b) **Transformieren Sie die View sodass Sie zu jedem Pokémon das Entwicklungslevel erhalten**
- c) **Filtern sie nun alle Pokémon die sich nicht entwickeln**



`pokemon_views_q.cpp`

Aufgabe:

Errechnen Sie das durchschnittliche Entwicklungslevel aller Feuerpokemon

- a) **Erstellen Sie zuerst eine View die alle Pokémon mit Primärtyp Feuer enthält**
- b) **Transformieren Sie die View sodass Sie zu jedem Pokémon das Entwicklungslevel erhalten**
- c) **Filtern sie nun alle Pokémon die sich nicht entwickeln**
- d) **Addieren Sie nun alle Entwicklungslevel und dividieren durch die Anzahl der Pokémon**



pokemon_views_q.cpp

```
auto pokemon = get_pokemon();  
auto entwicklungen = get_entwicklung();
```



`pokemon_views_q.cpp`

Code!



`pokemon_evolve_q.cpp`

Aufgabe:

Erstellen Sie nun eine View, welche alle gegebenen Pokémon eine Entwicklung „nach vorne“ bewegt



`pokemon_evolve_q.cpp`

Aufgabe:

Erstellen Sie nun eine View, welche alle gegebenen Pokémon eine Entwicklung „nach vorne“ bewegt

a) **Erstellen Sie ein `data_member` Struct in der View**



pokemon_evolve_q.cpp

Aufgabe:

Erstellen Sie nun eine View, welche alle gegebenen Pokémon eine Entwicklung „nach vorne“ bewegt

- a) **Erstellen Sie ein data_member Struct in der View**
- b) **Erstellen Sie die jeweiligen Operatoren für:**
 - die View
 - die fn



pokemon_evolve_q.cpp

Aufgabe:

Erstellen Sie nun eine View, welche alle gegebenen Pokémon eine Entwicklung „nach vorne“ bewegt

- a) **Erstellen Sie ein data_member Struct in der View**
- b) **Erstellen Sie die jeweiligen Operatoren für:**
 - die View
 - die fn
- c) **Füllen Sie die Methoden begin() und end()**



pokemon_evolve_q.cpp

Aufgabe:

Erstellen Sie nun eine View, welche alle gegebenen Pokémon eine Entwicklung „nach vorne“ bewegt

- a) Erstellen Sie ein `data_member` Struct in der View
- b) Erstellen Sie die jeweiligen Operatoren für:
 - die View
 - die fn
- c) Füllen Sie die Methoden `begin()` und `end()`
- d) Nutzen Sie die View mithilfe von Pipes



pokemon_evolve_q.cpp

```
auto pokemon = get_pokemon();  
auto entwicklungen = get_entwicklung();
```



Tipps

Schauen wir uns die gegebenen Daten an:

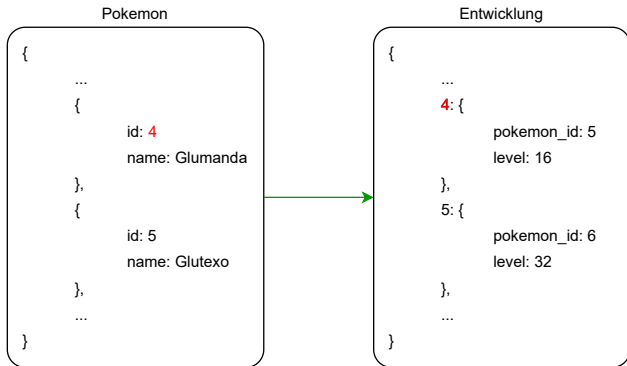
Pokemon

```
{
    ...
    {
        id: 4
        name: Glumanda
    },
    {
        id: 5
        name: Glutexo
    },
    ...
}
```

Entwicklung

```
{
    ...
    4: {
        pokemon_id: 5
        level: 16
    },
    5: {
        pokemon_id: 6
        level: 32
    },
    ...
}
```

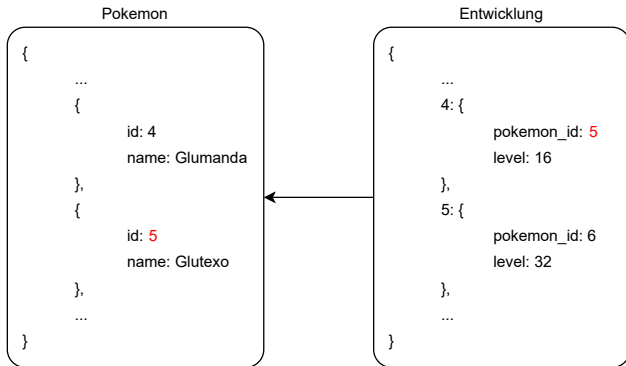
Zuerst vergleichen wir die `n`-te `Pokemon.id` mit `Entwicklung.von`:





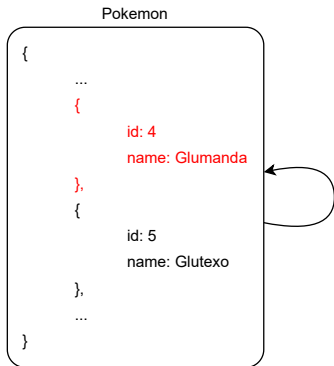
Tipps

Die erhaltene `Entwicklung.zu` finden wir in `Pokemon.id`:



Tipps

Nun überschreiben wir das n-te Pokémon mit seiner Entwicklung:





Tipps

Und erhalten:

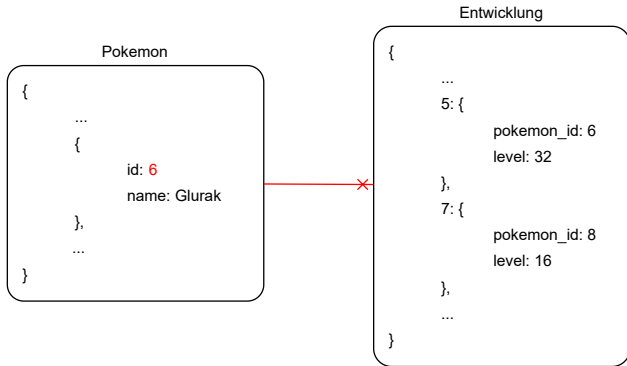
Pokemon

```
{  
    ...  
    {  
        id: 5  
        name: Glutexo  
    },  
    {  
        id: 5  
        name: Glutexo  
    },  
    ...  
}
```



Tipps

Sollten wir das n-te Pokémon nicht finden:





Tipps

entfernen wir dieses aus unserer View:

Pokemon

```
{  
    ...  
    {  
        id: 6  
        name: Glurak  
    },  
    ...  
}
```



`pokemon_evolve_q.cpp`

Code!



Quellen I

<https://en.cppreference.com/>

<https://de.wikipedia.org/wiki/Iterator>

<https://hannes.hauswedell.net/post/2018/04/11/view1/>

<https://mariusbancila.ro/blog/2020/06/06/a-custom-cpp20-range-view/>

https://hannes.hauswedell.net/post/2019/11/30/range_intro/

<https://github.com/ericniebler/range-v3>

<https://www.heise.de/developer/artikel/C-20-Die-Ranges-Bibliothek-4661566.html>

<https://www.stackoverflow.com/>

<https://mariusbancila.ro/blog/2019/01/20/cpp-code-samples-before-and-after-ranges/>

<https://www.internalpointers.com/post/writing-custom-iterators-modern-cpp>



Quellen II

<https://paddel.xyz/>
<https://github.com/pblan/matse-spicker-db>

cppcon:

<https://www.youtube.com/watch?v=SYLgG7Q5Zws>
https://www.youtube.com/watch?v=d_E-VLyUnzc
<https://www.youtube.com/watch?v=vJ290qlAbbw>



Felix Racz, Tim Wende –

RWTH Aachen University
Templergraben 55
52056 Aachen

https://www.github.com/tmwnd/adv_cpp_seminar