C++20 Ranges und Constrained Algorithms

Felix Racz, Tim Wende

23. Mai 2022

Überblick

Implementierung anderer Programmiersprachen

Vorwissen

Iteratoren

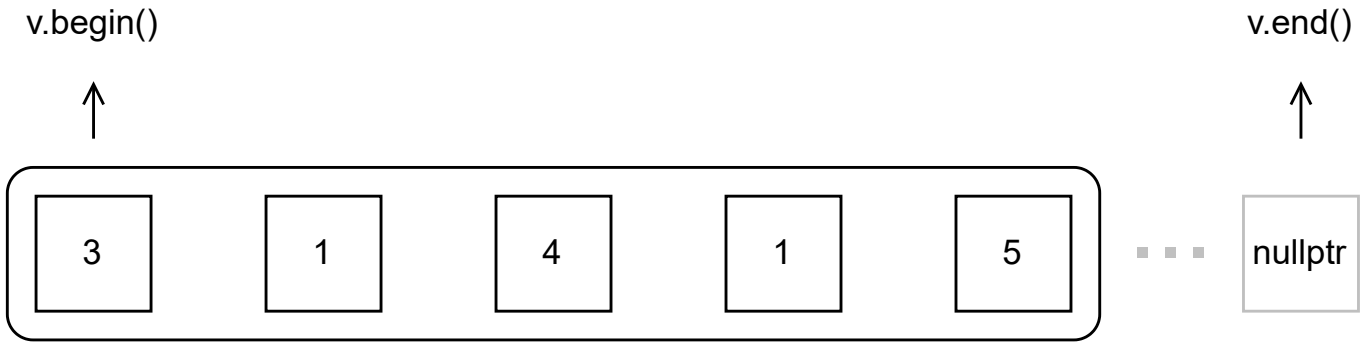
Der Begriff *Iterator* stammt aus dem Bereich der Softwareentwicklung und bezeichnet einen Zeiger, mit dem die Elemente einer Menge durchlaufen werden können (z. B. eine Liste, ein Vector etc.).

Beispiel

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

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

std::cout << std::endl;

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

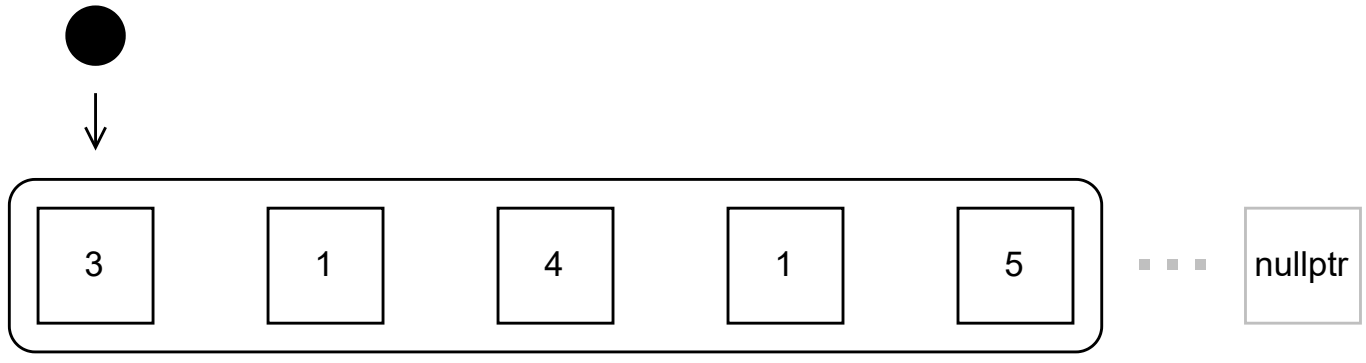
Iteratoren - Beispiel (Code)

```
std::vector<int>::iterator it = v.begin();
```



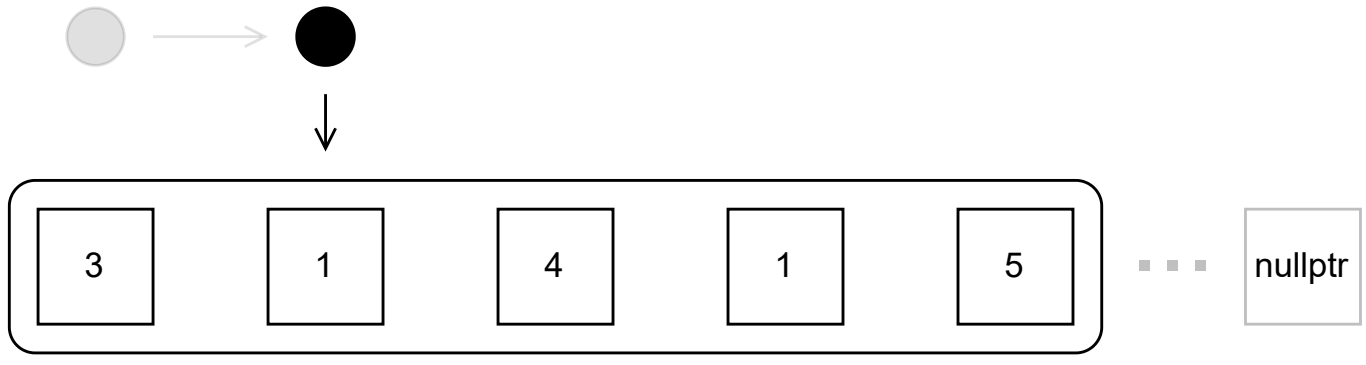
Iteratoren - Beispiel (Graphisch)

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



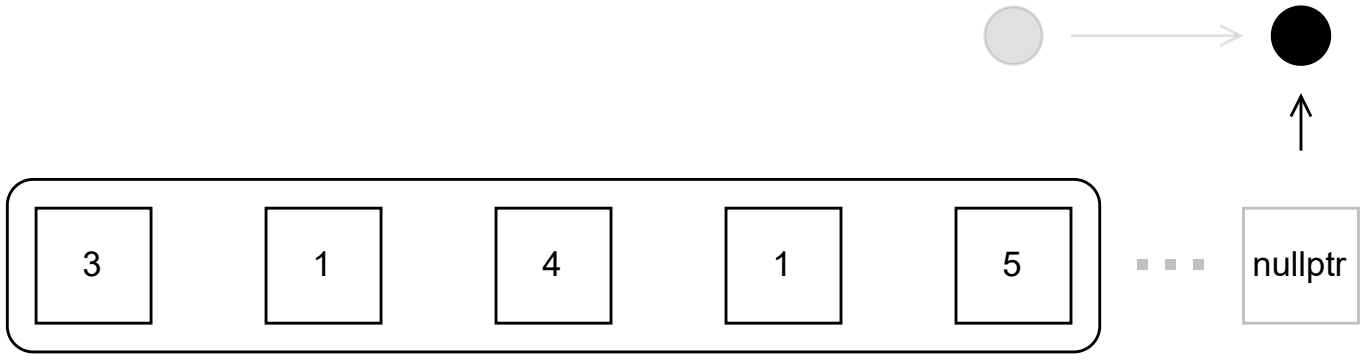
Iteratoren - Beispiel (Graphisch)

```
++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.

„SFINAE on steroids“

Concepts - Beispiel

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

Einleitung

Zu lange Fehlermeldungen

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.begin(), v.end());
```

(Compiler-) Geschwindigkeitsoptimierung

```cpp
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

```
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;
}
```

(Compiler-) Geschwindigkeitsoptimierung

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

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

(Compiler-) Geschwindigkeitsoptimierung

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_sentinal`.

(Compiler-) Geschwindigkeitsoptimierung

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

    return first;
}
```

(Compiler-) Geschwindigkeitsoptimierung

```
std::vector<int>::iterator find(
    std::vector<int>::iterator      first,
    ranges::unreachable_sentinal_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.

Anwendung

Range Types

Projection

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

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

Projection - Beispiel

```cpp
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:
Finde heraus, ob jedes Pokémon einen PokedexEintrag hat.

Projection - Beispiel

Erster Ansatz:

1. **Sortiere alle Pokemon (nach `id`)**
2. **Sortiere alle PokedexEinträge (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 == p.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 == p.pokemon_id;
    });
```

Projection - Beispiel

Dritter Ansatz:

- **Erweitere Ranges durch Projection**
- **Reduziere jedes Pokemon auf einen `int`**
- **Reduziere analog jeden PokedexEintrag 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);
```

Views

Pipes

Aufgaben

Beispiel 1

Quellen

https://www.youtube.com/watch?v=SYLgG7Q5Zws

Felix Racz, Tim Wende –

RWTH Aachen University
Templergraben 55
52056 Aachen

github.com/tmwnd/adv_cpp_seminar