C++

Spring 2021

compscicenter.ru

Basharin Egor

t.me/egorbasharin

Lecture XI

C++20 Ranges

Section 0

Motivation

Filtering & Transforming

STL vs Ranges

Filtering & Transforming

STL

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

int main() {

    std::vector vs{1,2,3,4,5,6,7,8,9,10};

    std::vector<int> even;
    std::copy_if(
        vs.begin(), vs.end(),
        std::back_inserter(even),
        [](int n){ return n % 2 == 0; });

    std::vector<int> result;
    std::transform(
        even.begin(), even.end(),
        std::back_inserter(result),
        [](int n) { return 2 * n; });

    for (auto v : result) {
        std::cout << v << " ";
    }
}
```

Filtering & Transforming

Ranges

```cpp
#include <iostream>
#include <ranges>
#include <vector>

using namespace std::views;

int main() {

    std::vector vs{1,2,3,4,5,6,7,8,9,10};

    auto range = vs
         | filter([](int n){ return n % 2 == 0; })
         | transform([](int n){ return n * 2; });

    for (auto v : range) {
        std::cout << v << " ";
    }
}
```

Ranges

- Lazy-evaluation
- Compatible with std-containters
- Composition

Section 1

Range

Range As Concept

```
template< class T >
concept range = requires(T& t) {
  ranges::begin(t);
  ranges::end  (t);
};
```

cppreference

Range

- based on iterators

> *They are implemented in terms iterators but they don't show in this interface.*

Section 2

Range Algorithms

`std::ranges::sort` vs `std::sort`

- std::list
- std::vector

Range Algorithms

- `<algorithms>` (c++20)
- `<numeric>` (c++23)

Sentinels

- `std::find` vs `std::ranges::find`

Sentinels

```
std::vector vs{1,2,4,5,6,7,8};
std::ranges::find(
    vs.begin(), std::unreachable_sentinel, 5);
```

Projections

- unary functions / functional objects
- modifies view of the data

Projections

```
#include <vector>
#include <algorithm>

struct Entry {
    int id;
};

int main() {
    std::vector<Entry> entries;

    std::sort(
        entries.begin(), entries.end(),
        [](const auto& lhs, const auto& rhs)
        {
            return lhs.id < rhs.id;
        });
}
```

Projections

```
std::sort(
    entries.begin(), entries.end(),
    [](const auto& lhs, const auto& rhs)
    { return lhs.id < rhs.id; });

std::ranges::sort(
    entries,
    [](const auto& lhs, const auto& rhs)
    { return lhs.id < rhs.id; });
```

Projections

```
std::sort(
    entries.begin(), entries.end(),
    [](const auto& lhs, const auto& rhs)
    { return lhs.id < rhs.id; });

std::ranges::sort(
    entries,
    [](const auto& lhs, const auto& rhs)
    { return lhs.id < rhs.id; });

std::ranges::sort(
    entries, std::less{},
    [](const auto& entry) { return entry.id; });
```

Projections

```
    std::ranges::sort(entries, std::less{}, &Entry::id);
```

Section 3

Range adaptors

Range adaptors

Example I: cout squares

Range adaptors

Example I: cout squares

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <ranges>

int main() {
    std::vector vs{1,2,3,4,5,6,7};

    std::ranges::transform(
        vs,
        std::ostream_iterator<int>{std::cout, " "},
        [](int i) { return i*i; });

    std::cout << std::endl;

    auto T = std::views::transform(vs, [](int i){ return i * i; });
    std::ranges::copy(T, std::ostream_iterator<int>{std::cout, " "});
}
```

Range adaptors

Example II: cout square of even numbers

Range adaptors

Example II: cout square of even numbers

```
std::vector vs{1,2,3,4,5,6,7};
auto rm = std::ranges::remove_if(
    vs, [](int i) { return i % 2 != 0; }); // modification
std::ranges::transform(
    vs.begin(), rm.begin(),
    std::ostream_iterator<int>{std::cout, " "},
    [](int i) { return i*i; });
```

Range adaptors

Example II: cout square of even numbers

```
std::vector vs{1,2,3,4,5,6,7};

auto T = std::views::transform(
    std::views::filter(vs, [](int i) { return i % 2 == 0; }),
    [](int i){ return i * i; });

std::ranges::copy(T, std::ostream_iterator<int>{std::cout, " "});
```

Range adaptors

Pipe operator

```
namespace view = std::views;
auto is_even = [](int i) { return i % 2 == 0; };
auto square = [](int i){ return i * i; };

auto T = vs | view::filter(is_even) | view::transform(square);

std::ranges::copy(T, std::ostream_iterator<int>{std::cout, " "});
```