* Ranges Dersin:

```
using namespace std;

cout << typeid(ranges::sort).name() << '\n';
```

std::sort bir function template

Fakat ranges::sort bir constexpr objeyi

```
ranges::sort()
}
▲ 1 of 2 ▼ constexpr std::conditional<...>::type operator()<_Rng, _Pr, _Pj>(_Rng &&_Range, _Pr _Pred = {}, _Pj _Proj = {}) con
```

fonksiyon çağrı operatörü

* Langtred Algorithms:

- Tüm STL algoritmaları range algoritmalarına henüz dönüşmedi!
- STL algoritmaları şöyle multithread çalışamaz X
- Bir range'in begin ve end iteratörü aynı olmak zorunda değil!

```
template <typename Iter, typename Sentinel>
void algo(Iter beg, Sentinel end)
```

→ Aynı tür olursa common range olur.

```
struct NullSent {
    bool operator==(auto x)const
    {
        return *x == '\0';
    }
};


int main()
{
    using namespace std;

    char str[] = "mustafa hekimoglu";

    ranges::for_each(str, NullSent{}, [](char c) {cout << "(" << c << ") "; });

}
```

son karakter \0 olduğu için

→ Kendi sentinelimizi yazdık! O sentinel ile range'in sonu olarak kullandık!

```
template <auto ENDVAL>
struct EndSent {

    bool operator==(auto pos)const
    {
        return *pos == ENDVAL;
    }
};


int main()
{
    using namespace std;

    vector ivec{ 1, 5, 7, 9, 2, 3, 6, 79, 90 };

    ranges::sort(ivec.begin(), EndSent<3>{});

    for (auto val : ivec) {
        cout << val << " ";
    }
}
```

```
1 2 5 7 9 3 6 79 90
C:\Users\necat\source\re
Press any key to close t
```

→ 3'e kadar olan range'i sort ettik!

```
using namespace std;

vector<int> ivec(100);

mt19937 eng{ random_device{}() };
uniform_int_distribution dist{ 0, 100 };

ranges::generate(ivec, [&] {return dist(eng); });

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

```
using namespace std;

vector<int> ivec(100);

mt19937 eng{ random_device{}() };
ranges::iota(ivec, 0);
ranges::shuffle(ivec, eng);
print(ivec);

int val;

cout << "aranacak degeri girin: ";
cin >> val;

auto iter = ranges::find(ivec.begin(), unreachable_sentinel, val);

cout << iter - ivec.begin() << "indeksli eleman olarak bulundu\n";

}
```

karşılaştırma yapılmasını sağlıyor!

→ begin != end artık sormuyor!

→ eğer end olmasaydı, sadece begin = end sorgusunu yapacaktı + *iter = val sorgusu

→ şimdi sadece (*iter = val) sorgusu var!

* Projection:

```
template <typename InIter, typename T, typename Projection>
InIter Find(InIter beg, InIter end, const T& val, Projection pr)
{
    while (beg != end) {
        if (pr(*beg) == val)
            return beg;

        ++beg;
    }

    return end;
}
```

→ baru std::invoke ile çağırıyoruz!

Artık iteratörün değil, o iteratörü bir callable'a geçirdiğimiz sonraki değerini kullanıyor oluyor!

```
struct Point {
    Point() = default;
    Point(int x, int y) : mx{x}, my{y} {}
    friend std::ostream& operator<<(std::ostream& os, const Point& p)
    {
        return os << '[' << p.mx << ", " << p.my << ']';
    }
    int mx{};
    int my{};
};


Point create_random_point()
{
    Irand rand{ 0, 99 };

    return Point{ rand(), rand() };
}

int main()
{
    using namespace std;

    vector<Point> pvec(20);

    ranges::generate(pvec, create_random_point);

    ranges::sort(pvec, {}, &Point::my);

    ranges::copy(pvec, ostream_iterator<Point>(cout, "\n"));


}
```

No issues found

less

projection olarak, Point'in my si verildi!

=> Artık y'ye göre sırlıyor.

```
int main()
{
    using namespace std;

    vector<string> svec;
    rfill(svec, 10000, [] {return rname() + ' ' + rfname(); });
    //ranges::copy(svec, ostream_iterator<string>{cout, "\n"});

    size_t length;
    cout << "enter length : ";
    cin >> length;


    if (auto iter = ranges::find(svec, length, [](const std::string& s) {return s.size(); }); iter != svec.end()) {
        std::cout << "bulundu: " << *iter << "\n";
    }
    else {
        std::cout << "bulunamadi\n";
    }
}
```

```
template<std::input_iterator Iter, std::sentinel_for<Iter> SenType, typename Init = std::iter_value_t<It
typename Op = std::plus<>, typename Proj = std::identity>
Init Accumulate(Iter beg, SenType end, Init init = Init{}, Op op = {}, Proj proj = {})
{
    while (beg != end)
    {
        init = std::invoke(op, std::move(init), std::invoke(proj, *beg));
        ++beg;
    }
    return init;
}
//
```

```
template<typename Iter, typename SenType, typename Init, typename Op = std::plus<>>
Init Accumulate2(Iter beg, SenType end, Init init, Op op = {})
{
    while (beg != end)
    {
        init = op(std::move(init), *beg);
        //init = std::invoke(op, std::move(init), *beg);
        ++beg;
    }
    return init;
}
```

fonksiyon çağrı operatörü olmalı

std::invoke'un bir takım avantajları daha var.

bir sonraki dersde anlatacak!

*View: - View da bir range
- Constant time da copy/move garantisi var!

```
//std::views::take
//std::views::filter
//std::views::drop
//std::views::take_while
//std::views::drop_while
```

```
int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 100, Irand{ 0, 999 });
    print(ivec);

    for (auto x : views::filter(ivec, [](int x) {return x % 5 == 0;}) {
        cout << x << " ";
    }

}
```

→ Sadece 5'e bölünenler!

*Bir range'in view olabilmesi için, range::view conceptini desteklemeli! Fakat her view bir rangedir!

```
using namespace std;

vector<int> ivec;
rfill(ivec, 10, Irand{ 0, 999 });
print(ivec);

for (auto val : views::reverse(ivec)) {
    cout << val << " ";
}
```

→ range'i tersine çevirdi!

```
for (auto val : views::reverse(views::take(ivec, 5))) {
    cout << val << " ";
}
```

→ sadece ilk 5 öğeyi tersine çevirdi!

* Pipeline yapılabilir!

```
using namespace std;

vector<int> ivec;
rfill(&: ivec, n: 20, frand: Irand{ 0, 999 });
print(ivec);

//views::filter(views::reverse(views::take(ivec, 10)), [](int x) {return x % 2 == 0; })

ivec | views::take(_Length: 10) | views::reverse | views::filter(_Pred: [](int x) {return x % 2 == 0; })
```

aynı!