2023 11 03

std::for_each

```
template <typename Iter, typename F>
F ForEach(Iter beg, Iter end, F f)
{
    while (beg != end)
    {
        f(*beg++);
    }
    return f;
}

//////////

int main()
{
     using namespace std;
     vector<string> svec;

     rfill(svec, 1000, []{return rname() + ' ' + rfname(); });

     for_each(svec.begin(), svec.end(), [](const auto& s)
     {
         cout << s << ' '; // her eleman için dönecek
     });
}
```

```
class Functor
{
    public:
        void operator ()(const std::string& s)
        {
            if (s.size() > 12)
                ++m_count;
            std::cout << s << '\n';
        }

        int get_count()const
        {
            return m_count;
        }
    private:
        std::size_t m_count{};
};

int main()
{
    using namespace std;

    vector<string> svec;
    rfill(svec, 100, []{return rname() + ' ' + rfname(); });

    auto f = for_each(svec.begin(), svec.end(), Functor{});
    std::cout << f.get_count() << '\n';
}
```

any_of, all_of and none_of

```
    avoid raw loops

    vector<Fighter> fighter_vec;

    bool flag = false;

    for(size_t i{}; i < fighter_vec.size(); ++i)
    {
        if (is_wounded(fighter_vec[i]))
        {
            flag = true;
            break;
        }
    }

    böyle bir kod uygun değil

    auto is_wounded=
    if (any_of(x.begin(), x.end(), is_wounded)
```

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

    vector<string> svec;
    rfill(svec, 100, rname);

    if (any_of(svec.begin(), svec.end(), [](const string& s)
    {
        return s.contains('x'))
    }
    {
        // eğer bu vector'lerin içinde x geçiyorsa
    }

    /*
        any_of : herhangi biri
        all_of : hepsi
        none_of : hiçbiri

    */

    vector<int> ivec;
    // vector boş
    // true
    bool b = all_of(ivec.begin(), ivec.end(), [](int x) {return x % 2 == 0;});
    // true
    b = none_of(ivec.begin(), ivec.end(), [](int x) {return x % 2 == 0;});
    // false
    b = any_of(ivec.begin(), ivec.end(), [](int x) {return x % 2 == 0;});
}
```

```
// replace
int main()
{
    using namespace std;
    vector<int> ivec;

    rfill(ivec, 100, Irand {0, 9})
    // değeri 5 olanları 9999 yapacak
    replace(ivec.begin(), ivec.end(), 5, 9999);
}
```

Bazı Algoritmaların Sonunda:

reverse_copy ,replace_copy ,remove_copy, reverse_copy_if ,replace_copy_if ,remove_copy_if

copy bulunur. Yapılan değişikliği başka yere kopyalar.

std::reverse_copy

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

    vector<string> svec;
    rfill(svec, 20, rname);

    list<string> slist;
    // svec tersi slis'te kopyalandı
    reverse_copy(svec.begin(), svec.end(), back_inserter(slist))


    vector<int> ivec{ 1, 2, 3, 1, 2, 4, 1, 5, 1, 1, 9};
    vector<int> desvec(10);

    // 1 dışındakileri kopyalar
    auto iter = remove_copy(ivec.begin(), ivec.end(), desvec.begin(), 1);

    std::cout << "toplam" << distance(destvec.begin, iter) << "eleman kopyalandı";
}
```

std::remove_copy_if

```
int main()
{
    using namespace std;
    vector<Date> dvec;
    rfill(dvec, 100, Date::random);
    vector<Date> dest_vec;
    // kasım ayı hariç kalanları dest_vec kopyalar
    remove_copy_if(dvec.begin(), dvec.end(), back_inserter(dest_vec),
    [](const Date& d)
    {
    return d.month() == 11;
    });
}
```

std::replace_copy and std::replace_copy_if

```
// replace_copy and replace_copy_if
int main()
{
    using namespace std;
    vector<int> ivec;

    rfill(ivec, 100, Irand {0, 9})

    vector<int> dest_vec;
    replace_copy(ivec.begin(), ivec.end(), back_inserter(dest_vec), 5, 3333);

    vector<int> dest_vec_if;

    replace_copy_if(ivec.begin(), ivec.end(), back_inserter(dest_vec_if),
                []()int x) {return x % 2 == 0;}, -1);
}
```

Containers

Bir işi 2 farklı bir biçimde yapabiliriz. Biri container sınıfın member fonksiyonunu çağrarak diğeri ise algoritmaları kullanarak. Hangisin tercih etmeliyiz.

Eğer container sınıfı üye fonksiyonları işimi görüyorsa onları kullanmamız yeterli.

```
    container'ların ortak member fonksiyonları:
    .size()
    .empty()
    .clear()
    .erase() // 2 overloading
```

Container tipleri:

Sequence containers

- std::vector
- std::deque
- std::list
- std::forward_list
- std::array

(array hariç) özel fonksiyonlar:

- front
- back
- resize

Associative Containers

- std::set
- std::multiset
- std::map
- std::multimap

Unordered Associative Containers

- std::unordered_set
- std::unordered_multiset
- std::unordered_map
- std::unordered_multimap

std::vector

```
template<typename T>
class Vector
{
    public:
        void push_back(T&&)
        {
            // bir nesnesin move ctor çağırılır
            std::move(r)
        }; // R reference
        void push_back(const T&)
        {
            // copy ctor çağrılır
        }; // L reference

        template<typename U>
        void foo(U&&) // Universel reference
}
```

Container Emplace Fonksiyonları

```
template<typename T>
class Vector
{
    public:
        void push_back(T&&)
        {
            // bir nesnesin move ctor çağırılır
            std::move(r)
        }; // R reference
        void push_back(const T&)
        {
            // copy ctor çağrılır
        }; // L reference
        template<typename ...Args>
        void emplace_back(Args&& ...args)
        {
            // perfect forwading
            T(std::forward<U>(args)...)
        }
}
```