

* Find / Find-if / find_if-not:

Find: linear search $\rightarrow O(n)$

```
template <typename InIter, typename T> InIter Find(InIter beg, InIter end, const T& tval)
{
    while (beg != end) {
        if (*beg == tval)
            return beg;
        ++beg;
    }
    return end;
}
```

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

    vector<string> svec;
    rfill(svec, 20, "name"); //nutility
    print(svec); //nutility
    string name;
    cout << "aranacak ismi girin: ";
    cin >> name;

    vector<string>::iterator iter = find(svec.begin(), svec.end(), name);
    buru yorum not: iter = auto
}
```

Foxit direkt edilmiş, burdaki her scope leakage'a sebebiyet verir.

→ Görüm 1: scope eklenir
→ Görüm 2: if with initializer.

unutma bu sentinel değer
son eleman
svec.end() - 1'de

```
if (iter != svec.end()) {
    std::cout << *iter << "\n";
    cout << iter - svec.begin() << " indeksli eleman\n";
} ]→ vector itersten → random access iterator.
else {
    std::cout << "bulunamadi\n";
}
```

→ svec. back() kullanılır.
svec. front()

→ aynı da svec.begin()
ile son elemanı erişir.

→ If with initializer kılavuzu:

```
if (auto iter = find(svec.begin(), svec.end(), name); iter != svec.end()) {
    cout << *iter << "\n";
    cout << iter - svec.begin() << " indeksli eleman\n";
}
else {
    ++iter;
    cout << "bulunamadi\n";
}
```

→ Find_if(): Sonu if biten stl fonksiyonu, birden fazla predicate fonksiyon isteyebilir.

↳ sonucda true'da kalırız..

```
I

template <typename InIter, typename UnPred> <T> Provide sample template arguments for Intellisense ✓
InIter FindIf(InIter beg, InIter end, UnPred f)
{
    while (beg != end) {
        if (f(*beg)) {
            return beg;
        }
        ++beg;
    }

    return end;
}
```

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

    vector<int> ivec;
    rfill(ivec, 100, Irand{ 5'000'000, 6'000'000 });
    print(ivec);

    if (auto iter = find_if(begin(ivec), end(ivec), isprime); iter != end(ivec)) {
        std::cout << "ilk asal sayı : " << *iter << "\n";
    }
    else {
        std::cout << "bulunamadı\n";
    }
}
```

notifity.h da var.

* Iteratör Adaptörleri:

- Basılı bir sınıf (kullanıcı, O sınıftının farklı bir interface olusurdu)

* Modelîmza: CTAD,

```
vector ivec{ 2, 5, 6, 9, 3, 34, 76, 19 }; //CTAD
```

→ class template argument deduction

→ <int> adımı olduğu kendi endi.

→ Reverse Iterator: iterator.begin ve end'in basına "r" harfi eklenince, rbegin / rend → reverse iterator dir.

```
vector ivec{ 2, 5, 6, 9, 3, 34, 76, 19 }; //CTAD
reverse_iterator<vector<int>::iterator> iter;
```

→ vector<int>::reverse_iterator ile aynı
↓

rbegin'in return değerini bu.

* Fikir: reverse iterator, + ile, bir önceki iteratorın konumunu tutuyor.

* Bir vektördək ən 2 rakamını silme:

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

    vector<int> ivec{ 1, 2, 4, 5, 2, 6, 7, 2, 9};

    auto riter = find(ivec.rbegin(), ivec.rend(), 2);

    ivec.erase(riter.base() - 1);
    print(ivec);
}
```

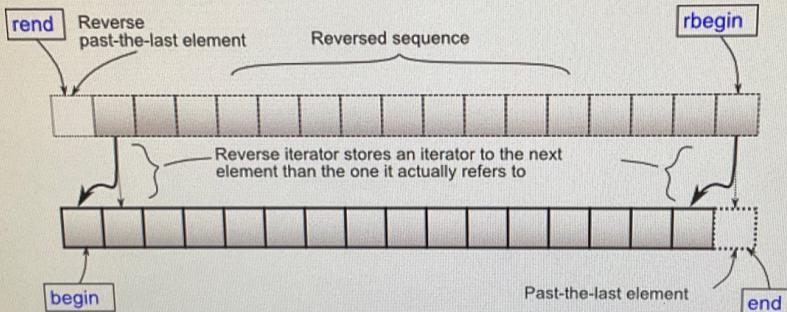
reverse iterator
rte senden
base bələdçi
başdır.

riter.base() → base 9-ü verir.
→ riter.base() - 1 ilə 2-ü sildi.

base() returns the underlying base iterator.

The base iterator refers to the element that is next to the element the reverse_iterator is currently pointing to. That is `std::reverse_iterator(it).base() == std::next(it)`.

You can learn more about `reverse_iterator` [here](#).



Share Improve this answer Follow

11 14 21 2013

* Insert Iterators:

• Hərəkətma:

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

    vector ivec{ 2, 5, 7, 1, 9, 3 };
    vector <int> destvec;

    copy(ivec.begin(), ivec.end(), destvec.begin());
}
```

• Bu kod çalışır X Təmiz Dəvət

• Cənək: kopyalama nümunənin destination
bos!!

```
using namespace std;
```

```
vector ivec{ 2, 5, 7, 1, 9, 3 };
vector <int> destvec(6);
```

6 element var.

→ ÜstHək: kod bu şəhərde çalışır.

→ Fakat bu tenimsiz davranış söyle可以说:

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

    vector<int> ivec{ 1, 2, 3, 4, 5 };
    vector<int> destvec;

    copy(ivec.begin(), ivec.end(), back_inserter(destvec)); !  
    print(destvec);
}
```

back_insert_iterator	con.push_back(value)
back_inserter	
front_insert_iterator	con.push_back(value) → tenim front_inserter
insert_iterator	con.insert(iter, value)
inserter	

* Fakat: her container'ın push_front ve push_back'i
yapıyor. Onlar tern inserter ver.

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

    vector<int> ivec{ 34, 7, 7, 12, 9, 9, 18, 1, 123, 56 };
    set<int> myset;

    copy(ivec.begin(), ivec.end(), inserter(myset, myset.begin()));

    print(myset);

}
```

* Constek:

```
using namespace std;

vector<int> ivec{ 10, 20, 30, 40, 50 };
const vector<int>::iterator iter = ivec.begin();
```

*iter = 75; burada eriştiğiniz nesne const değil. iter nesne const → Top level const gibi

→ low level const gibi salt okuma amaçlı kullanıldığında, const迭代器 nested type'i kullanılır.

```
list<int> mylist{ 10, 20, 30, 40, 50 };

for (auto iter = mylist.cbegin(); iter != mylist.cend(); ++iter) {
    cout << *iter << "\n";
}
```

```
list<int> mylist{ 10, 20, 30, 40, 50 };

for (list<int>::const_iterator iter = mylist.cbegin(); iter != mylist.cend(); ++iter) {
    cout << *iter << "\n";
}
```

```

456 list<int>::iterator
457 mylist.begin(), mylist.end()

458 list<int>::const_iterator
459 mylist.cbegin(), mylist.cend()

460 list<int>::reverse_iterator
461 mylist.rbegin(), mylist.rend()

462 list<int>::const_reverse_iterator
463 mylist.crbegin(), mylist.crend()

```

* Iterator kullanımları: → iteratörler manipüle eden bir tür template dir.

- advance
- distance
- next
- prev
- iter_swap

- Advance:

```

using namespace std;

vector<string> svec{ "eray", "deniz", "fatih", "selda" , "ece" };
auto iter = svec.begin(); → iter su an eray

cout << *iter << "\n"; I

iter += 2; → fatih dylemi ki bir 2'inci terin tütüğü yerin 2'sini tütmesini istiyoruz.

```

→ Advance bu random.access iterator olmazsa
 "+= " operatörü çalışmaz (sendox hata) → list

```

using container = std::list<std::string>; notasyonu kullanma.
                                              I ortak list iteratorde da increment ettiğimiz.

int main()
{
    using namespace std;

    container con{ "eray", "deniz", "fatih", "selda" , "ece" };
    auto iter = con.begin();

    std::cout << *iter << "\n"; → eray
    advance(iter, 2);
    std::cout << *iter << "\n"; → fatih

}

```

→ advance'in colisyonunu saglayan mekanizma : tag dispatch

```
template <typename Iter>
void Advance_Impl(Iter& iter, int n, std::random_access_iterator_tag)
{
    std::cout << "random access iterator\n";
    iter += n;
}

template <typename Iter> void Advance_Impl(Iter& iter, int n, std::bidirectional_iterator_tag)
{
    std::cout << "bidirectional iterator\n";
    while (n--)
        ++iter;
}

template <typename Iter>
void Advance(Iter& iter, int n)
{
    Advance_Impl(iter, n, typename Iter::iterator_category{});
}
```

parametre tag yorumlu
context and tag
resolutionunda
kullaniliyor.

→ tag dispatch'lik compile time de
tari n'e oldugu anlasilir.

*→ Aynı seferde advance, end'den baslayip, negatif değerlerde abs kullanabilir.

fakat negatif degise
colisabilecegicin, bidirectional olması gerekiyor!

- Distance:

```
int main()
{
    //std::distance(iterator1, iterator2)
}
```

- Iter1 kab lkr iterator2'ye
ugraşır.
Bunu standart.