

[Sign up](#)[Code](#)[Pull requests](#) 0[Projects](#) 0[Actions](#)[Security](#)[Pulse](#)

Branch: master ▾

[Find file](#)[Copy path](#)[Cplusplus\\_Ders\\_Notlari](#) / [any\\_of\\_none\\_of\\_all\\_of.md](#)

necatiergin Update any\_of\_none\_of\_all\_of.md ff9526d Feb 9, 2020

1 contributor

253 lines (205 sloc) | 6.28 KB

[Raw](#)[Blame](#)[History](#)

Bu yazımızın konusu `C++11` ile standart kütüphaneye basit ama faydalı `3` algoritma.

`any_of` algoritmasıyla bir aralık (`range`) içindeki değerlerden herhangi birinin bir koşulu sağlayıp sağlamadığını sınavabiliyoruz:

```
template<class InIter, class UnPred>
bool any_of(InIter first, InIter last, UnPred f);
```

Şablondan üretilecek işlevin ilk iki parametresi ilgili aralığa ilişkin adımlayıcılar (`iterator`) . İşlevin son parametresi tek parametrelili bir sınavıcı (`predicate`) . Eğer `[first, last)` aralığındaki herhangi bir öğe için sınavıcımız olan `f` , `"true"` değer üretirse işlevimiz `"true"` değer döndürecek. Eğer aralıktaki hiçbir değer için `f` , `"true"` değer üretmez ise işlevimiz `"false"` değerini döndürecek. Kısaca bu işlev ile `[first, last)` aralığında en az bir değer verilen bir koşulu sağlayıp sağlamadığını sınavabiliyoruz:

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

using namespace std;

bool isprime(int);

int main()
{
    vector<int> ivec{ 138651, 43523, 462231, 2340097, 20414123 };

    if (any_of(ivec.begin(), ivec.end(), &isprime)) {
        cout << "en az bir asal sayı var\n";
    }
    else {
        cout << "sayılardan hicbiri asal degil\n";
    }
    //
}
```

Yukarıdaki kodda `ivec` `vector` 'ünde bulunan tamsayılar içinde en az bir asal sayı bulunup bulunmadığını test ediyoruz. Bu sınamayı pekala `find_if` algoritmasıyla da gerçekleştirebilirdik, değil mi?

```
template<class InIter, class UnPred>
InIter find_if(InIter first, InIter last, UnPred f)
{
    while (first != last) {
        if (f(*first))
            return first;
        ++first;
    }
    return first;
}
```

`find_if` algoritması bir aralık içinde bir koşulu sağlayan ilk öğeyi arıyor ve koşulu sağlayan ilk öğenin konumunu döndürüyor. Eğer öğelerden hiçbiri ilgili koşulu sağlamıyor ise algoritmanın geri dönüş değeri kendisine geçilen `last` konumu.

```
template<class InIter, class UnPred>
bool any_of(InIter first, InIter last, UnPred f)
{
    while (first != last) {
        if (f(*first))
            return true;
        ++first;
    }

    return false;
}
```

`any_of` algoritmasını şu şekilde de kodlayabilirdik:

```
template<class InIter, class UnPred>
bool any_of(InIter first, InIter last, UnPred f)
{
    return find_if(first, last, f) != last;
}
```

`all_of` algoritması ile bir aralıktaki tüm değerlerin bir koşulu sağlayıp sağlamadığını oynayabiliyoruz:

```
template<class InIter, class UnPred>
bool all_of(InIter first, InIter last, UnPred f);
```

Aşağıdaki kod parçasına bakalım:

```

#include <list>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
    int n;
    list<int> ls {25, 32, 43, 19, 6, 8, 20, 11, 9, 27, 5, 7};
    cout << "pozitif bir tamsayi girin : ";
        cin >> n;
    if (all_of(ls.begin(), ls.end(), [n](int x){return x % n == 0; }))
        cout << "listedeki tum degerler " << n << " ile tam bolunuyor\n";
    //
}

```

`ls` isimli listedeki tüm sayıların `n` tamsayısına tam olarak bölünüp bölünmediğini sınamak için `all_of` algoritmasının kullanıldığını görüyorsunuz. Bu örnekte sınavıcı parametresine bir `lambda` ifadesi geçiliyor. `lambda` ifadesinde yerel değişken olan `n` yakalanıyor (capture ediliyor).

`all_of` işlev şablonu şu şekilde kodlanabilir:

```

template<class InIter, class UnPred>
bool all_of(InIter first, InIter last, UnPred f)
{
    while (first != last) {
        if (!f(*first))
            return false;
        ++first;
    }
    return true;
}

```

`all_of` algoritmasının başka bir gerçekleştirimi şöyle olabilirdi:

```

template<class InIter, class UnPred>
bool all_of(InIter first, InIter last, UnPred f)
{
    return find_if_not(first, last, f) == last;
}

```

Bu arada `find_if_not` algoritmasının da C++11 ile geldiğini hatırlatalım. `find_if_not` algoritması ile bir aralıkta bulunan değerlerden bir koşulu sağlamayan ilk öğenin konumunu bulabiliyoruz:

```

template<class InIter, class UnPred>
InIter find_if_not(InIter first, InIter last, UnPred f)
{
    while (first != last) {
        if (!f(*first))
            return first;
        ++first;
    }
    return first;
}

```

Şimdi de `none_of` algoritmasını inceleyelim. `none_of` algoritması bir aralıktaki değerlerden hiçbirinin verilen bir

koşulu gerçekleştiriyor ise "true" değer döndürüyor:

```
template<class InIter, class UnPred>
bool none_of(InIter first, InIter last, UnPred f);
```

Aşağıdaki main işlevini inceleyelim:

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

using namespace std;

int main()
{
    vector<string> svec{ "selim", "can", "berrin", "murat", "kayhan", "nedim" };
    //
    if (none_of(svec.begin(), svec.end(), [](const string &s){return s.length() < 6; })) {
        //
    }
    //
}
```

Yukarıdaki kodda svec içinde tutulan isimlerden hiçbirinin uzunluğu 6 'dan fazla değilse programın akışı if deyiminin içine girecek. none\_of algoritması aşağıdaki gibi kodlanabilir:

```
template<class InIter, class UnPred>
bool none_of(InIter first, InIter last, UnPred f)
{
    while (first != last) {
        if (f(*first)) {
            return false;
        }
        ++first;
    }
    return true;
}
```

none\_of algoritmasının any\_of algoritmasının lojik değili olduğu açık, değil mi?

```
template<class InIter, class UnPred>
bool none_of(InIter first, InIter last, UnPred f)
{
    return find_if_not(first, last, f) == last;
}
```

Şimdi de aşağıdaki programı derleyerek çalıştırın:

```

#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    vector<int> ivec{ 4, 6, 2, 7, 3, 8, 1, 7, 3 };
    int ival;

    cout << "bir sayi giriniz : ";
    cin >> ival;
    auto f = [ival](int x) {return x < ival; };
    cout.setf(ios_base::boolalpha);
    cout << any_of(ivec.begin(), ivec.end(), f) << "\n";
    cout << all_of(ivec.begin(), ivec.end(), f) << "\n";
    cout << none_of(ivec.begin(), ivec.end(), f) << "\n";
}

```

Peki, verilen aralık boş ise işlevlerimiz hangi değerleri üretecek. Aşağıdaki kod bunu gösteriyor:

```

#include <iostream>
#include <vector>
#include <algorithm>

int main()
{
    using namespace std;

    vector<int> ivec; //ivec boş
    auto pred{ [](int x) {return x == 5; } };
    cout << boolalpha;
    cout << any_of(ivec.begin(), ivec.end(), pred) << "\n";
    cout << none_of(ivec.begin(), ivec.end(), pred) << "\n";
    cout << all_of(ivec.begin(), ivec.end(), pred) << "\n";
}

```