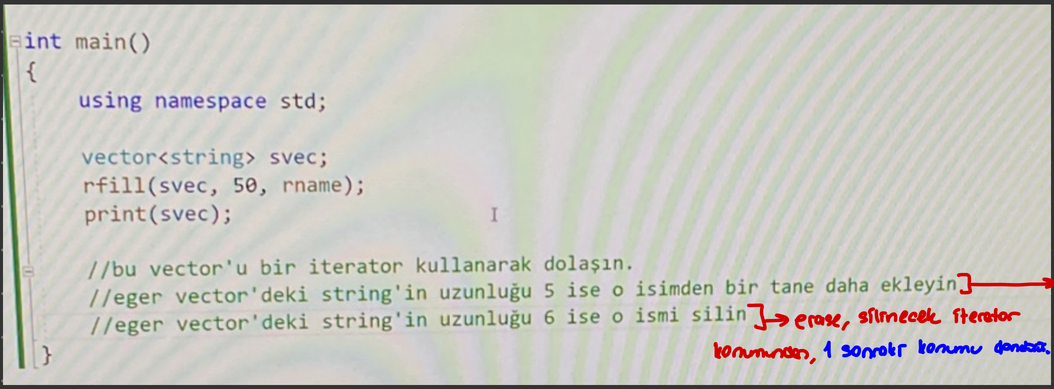
Mülakat Sorusu:



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

    vector<string> svec;
    rfill(svec, 50, rname);
    print(svec);

    //bu vector'u bir iterator kullanarak dolaşın.
    //eger vector'deki string'in uzunluğu 5 ise o isimden bir tane daha ekleyin
    //eger vector'deki string'in uzunluğu 6 ise o ismi silin
}
```

$\rightarrow$ insert, ekleme yapılacak konumu döndürür.

$\rightarrow$ erase, silinecek iterator konumundan, 1 sonraki konumu döndürür.

* iterator and reference invalidation:

→ standartlar bunu açıklıyor !!

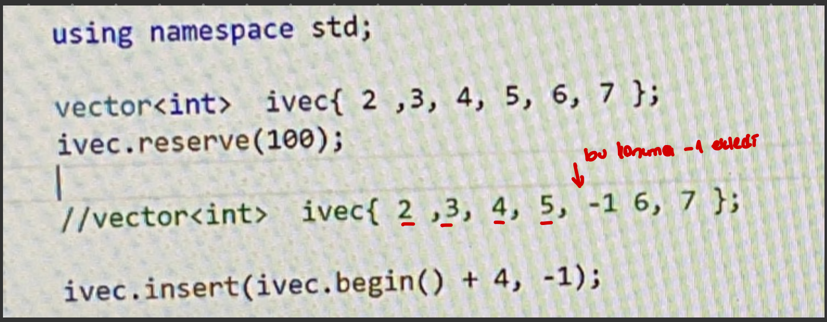
→ containerdan container'a fark gösteriyor.

→ Container'ın tuttuğu nesneler, pointer, iterator, reference olabilir. Fakat, bazı ekleme silme işlemi yapan fonksiyonlar, konum tutan bu pointer / iterator / ref'ı invalidate edebilir. ⟹ invalidate edilmiş kullanmak / onun gösterdiği yere erişmek = undefined behavior.

* Örneğin realloc her şeyi invalid eder.

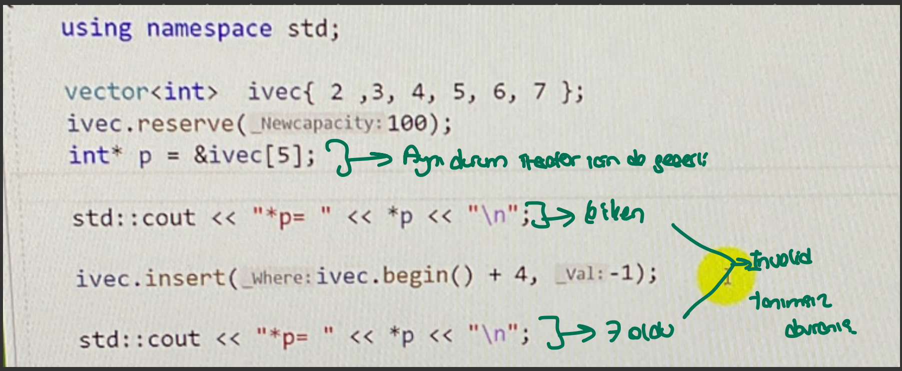
→ Ancak, realloc olmadan insertion yaparsak, insert ettiğimiz konuma kadar valid ✓
↓
deletion

insert ettiğimiz konumdan sonrası invalid ✗



```cpp
using namespace std;

vector<int>  ivec{ 2 ,3, 4, 5, 6, 7 };
ivec.reserve(100);

//vector<int>  ivec{ 2 ,3, 4, 5, -1 6, 7 };

ivec.insert(ivec.begin() + 4, -1);
```

bu konuma -1 eklendi



```cpp
using namespace std;

vector<int>  ivec{ 2 ,3, 4, 5, 6, 7 };
ivec.reserve(_Newcapacity: 100);
int* p = &ivec[5];

std::cout << "*p= " << *p << "\n";

ivec.insert(_Where: ivec.begin() + 4, _Val: -1);

std::cout << "*p= " << *p << "\n";
```

→ Aynı durum iterator için de geçerli

→ 6 iken

→ 7 oldu

→ invalid

tanımsız davranış

* Molokat Sorusunun Çözümü:

→ erase için: erase, silme işleminden sonra, silinenden bir sonraki konumun iterator'ünü döndürür.

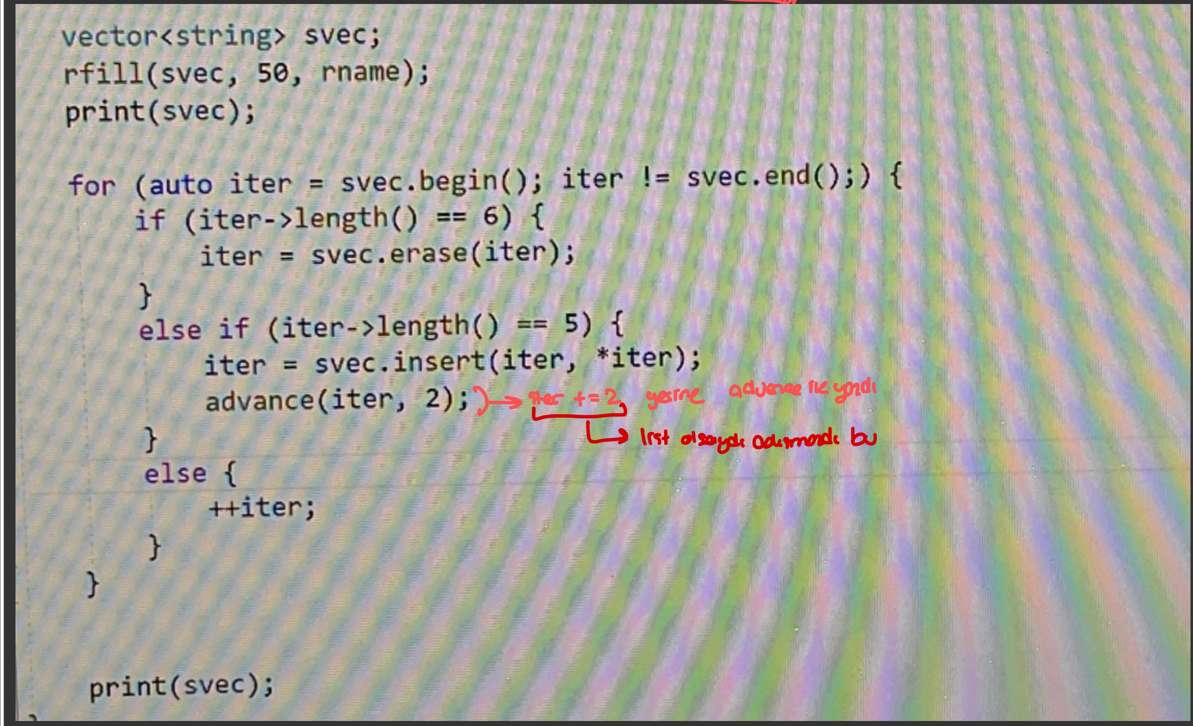
→ insert için: eklenen konuma iterator döndürür.

→ **erase de invalidationdan kaçınmak için:**

eğer silme yaptıysak, return edecek iterator değer yine aynı iterator değerine atamalıyız.

→ **insertte invalidationdan kaçınmak için:**

eğer insert yapıyorsak, return değer iteratörü kontenyarı ilerletebilmek için $\boxed{\text{iterator } +2}$ olmalı



```
vector<string> svec;
rfill(svec, 50, rname);
print(svec);

for (auto iter = svec.begin(); iter != svec.end();) {
    if (iter->length() == 6) {
        iter = svec.erase(iter);
    }
    else if (iter->length() == 5) {
        iter = svec.insert(iter, *iter);
        advance(iter, 2);
    }
    else {
        ++iter;
    }
}

print(svec);
```

→ iter += 2 yerine advance ile yazdı

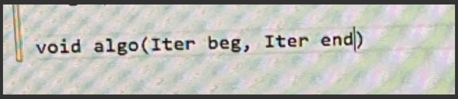
→ list olsaydı olamazdı bu

* Silme Algoritmaları:

→ Range parametreli bir algoritma, söz konusu rangeden öğe silemez.

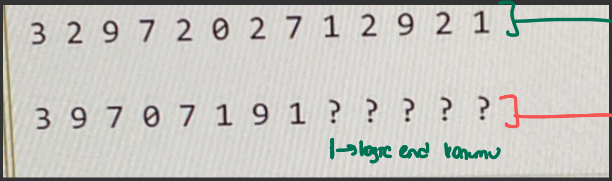
↳ silme için bir container'ın .erase() / vektörün resize'ı ile yapılır.

→ Silme algoritmaları logic silme yapıyor. / logic erasure



```
void algo(Iter beg, Iter end)
```

→ **Remove Algoritması:**



```
3 2 9 7 2 0 2 7 1 2 9 2 1
3 9 7 0 7 1 9 1 ? ? ? ? ?
```

→ sadece 2'lerin remove edildiği bir algoritma

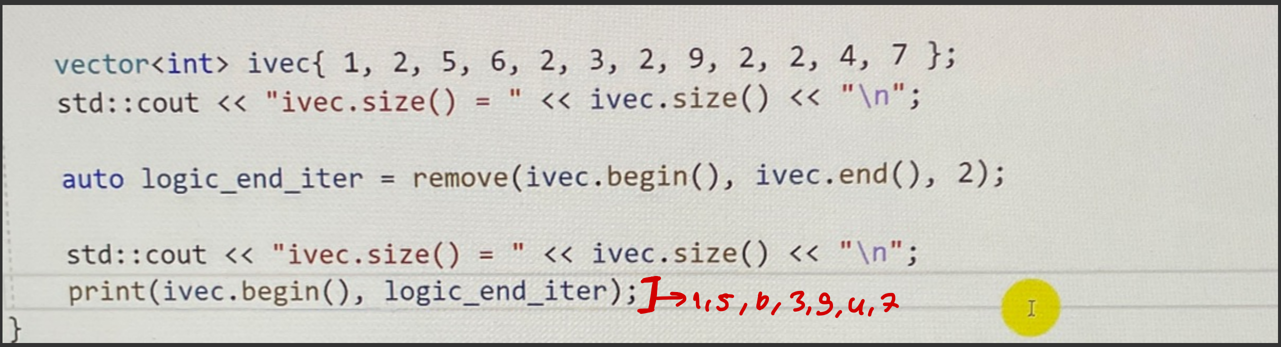
→ Fiziksel olarak silmek yerine, silinmiş gibi öğeleri yeniden düzenliyor. ve logic end konumu döndürür.

↳ logic end konumu

↳ Container'ın size'ı değişmedi. Fiziksel olarak da silmek için Range logic end

↓

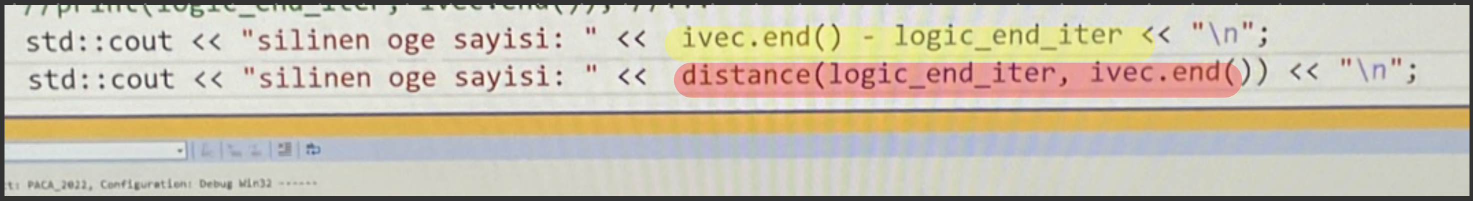
end'de sil.



```
vector<int> ivec{ 1, 2, 5, 6, 2, 3, 2, 9, 2, 2, 4, 7 };
std::cout << "ivec.size() = " << ivec.size() << "\n";

auto logic_end_iter = remove(ivec.begin(), ivec.end(), 2);

std::cout << "ivec.size() = " << ivec.size() << "\n";
print(ivec.begin(), logic_end_iter);
}
```

$\rightarrow$ 1,5,6,3,9,4,7

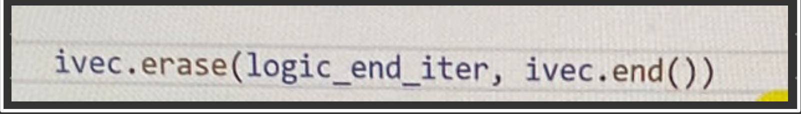
* ivec_end() - logic_end_iter() = silinen öge sayısı.



```
//print(logic_end_iter, ivec.end()); //...
std::cout << "silinen oge sayisi: " << ivec.end() - logic_end_iter << "\n";
std::cout << "silinen oge sayisi: " << distance(logic_end_iter, ivec.end()) << "\n";
```

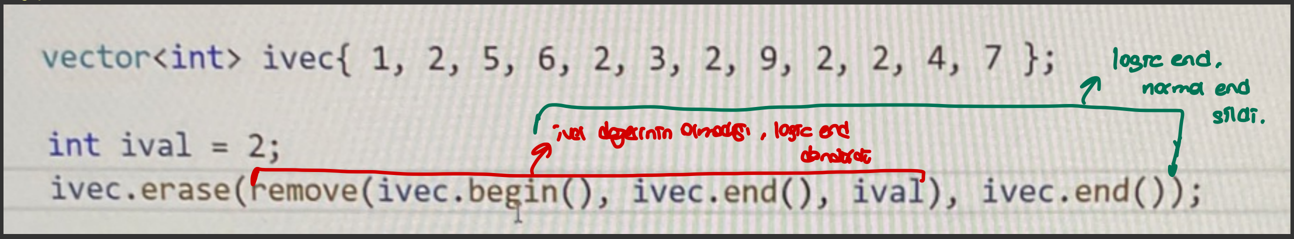
```
t: PACA_2022, Configuration: Debug Win32 ------
```

* Erase fonksiyonu ile gerçekten silebiliriz. → size'ı azaltıyor. → Fiziksel silme işlemi.



```
ivec.erase(logic_end_iter, ivec.end())
```

* Remove - Erase Idiom: → Tek seferde, containerdan öge silme

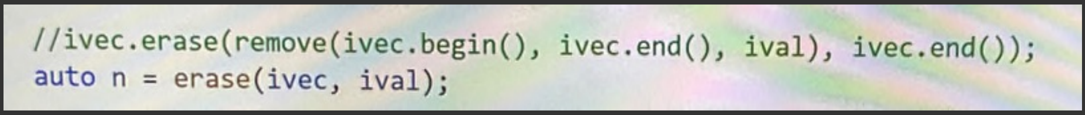


```
vector<int> ivec{ 1, 2, 5, 6, 2, 3, 2, 9, 2, 2, 4, 7 };

int ival = 2;
ivec.erase(remove(ivec.begin(), ivec.end(), ival), ivec.end());
```

ival değerinin olmadığı, logic end döndürdü

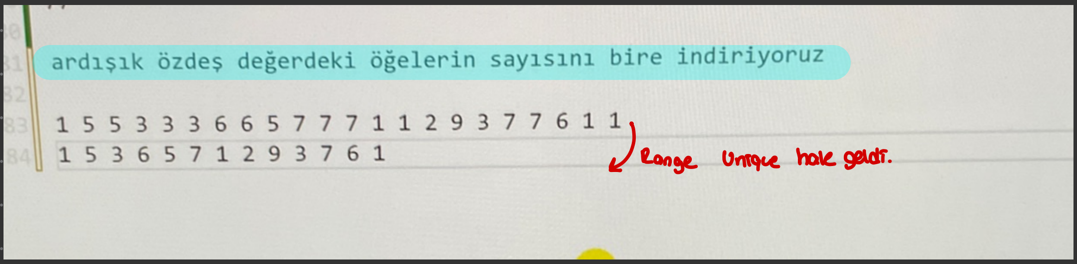
logic end, normal end sildi.

• Cpp 20de, global fonksiyon olarak erase geliyor.



```
//ivec.erase(remove(ivec.begin(), ivec.end(), ival), ivec.end());
auto n = erase(ivec, ival);
```

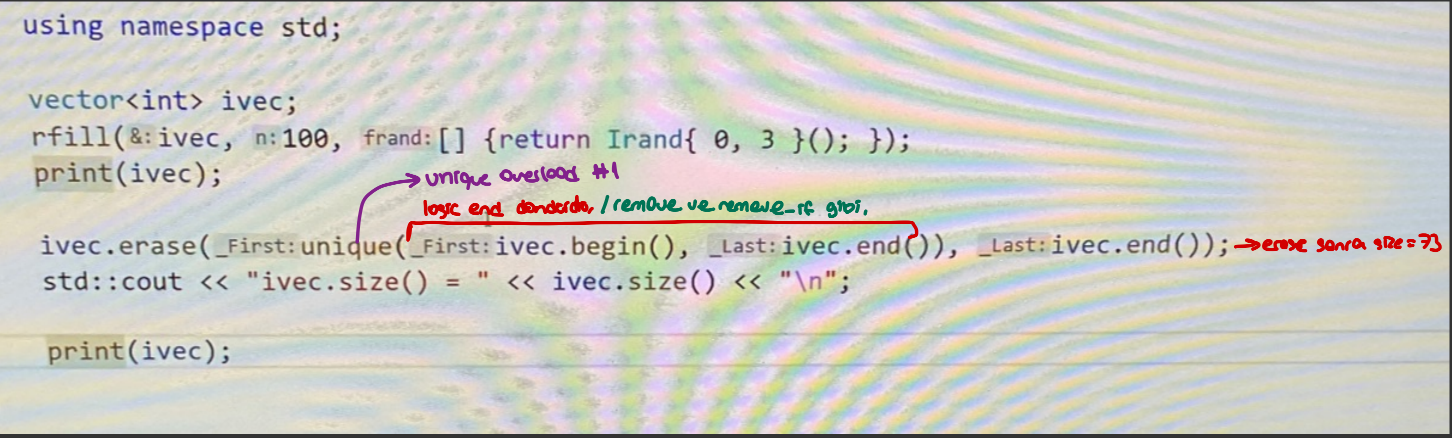
* Unique Algoritması



```
ardışık özdeş değerdeki öğelerin sayısını bire indiriyoruz

1 5 5 3 3 3 6 6 5 7 7 7 1 1 2 9 3 7 7 6 1 1
1 5 3 6 5 7 1 2 9 3 7 6 1
```

range unique hale geldi.

→ eğer, her öğeden sadece 1 tane olsun isteniyorsa, önce sıralamalıyız.



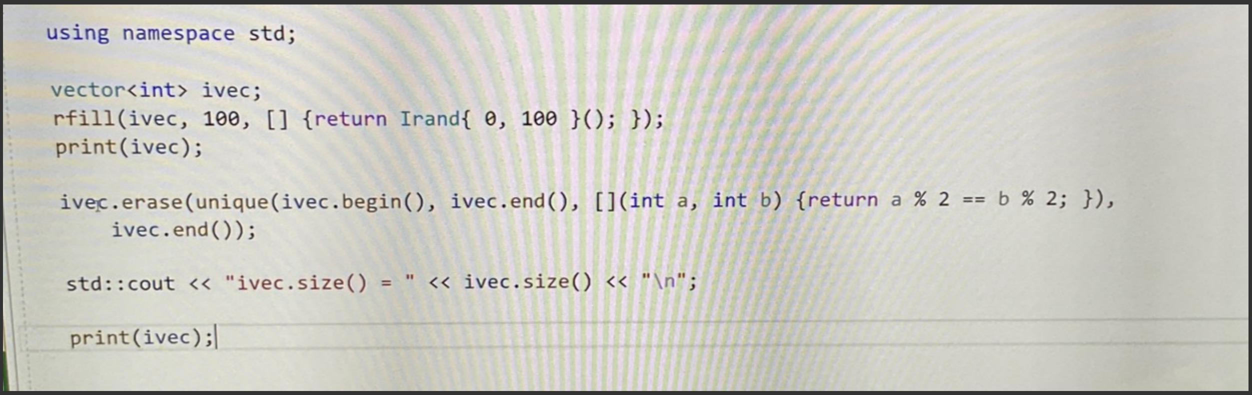
```cpp
using namespace std;

vector<int> ivec;
rfill(&:ivec, n:100, frand:[] {return Irand{ 0, 3 }(); });
print(ivec);

ivec.erase(_First:unique(_First:ivec.begin(), _Last:ivec.end()), _Last:ivec.end());
std::cout << "ivec.size() = " << ivec.size() << "\n";

print(ivec);
```

unique overload #1

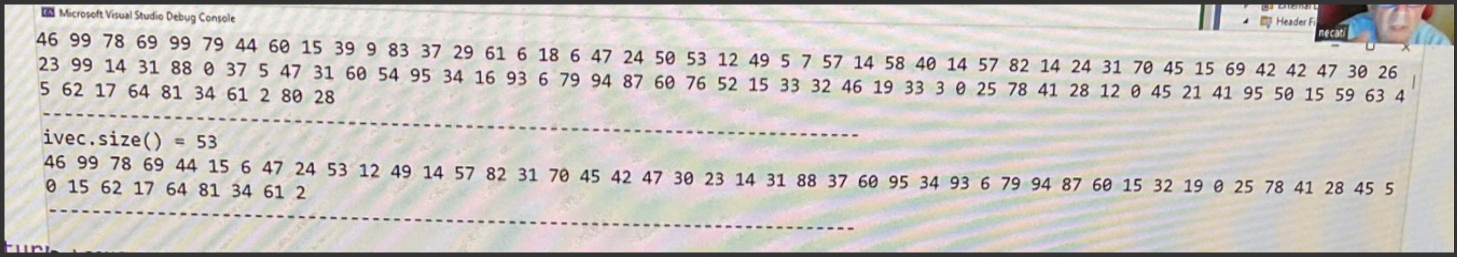
logic end döndürdü, / remove ve remove_if gibi.

→ erase sonra size=73

* ikinci overload için, verdiği örnek: Ardışık aynı değil, ikisi ardışık sayı tek ise / çift ise sil.



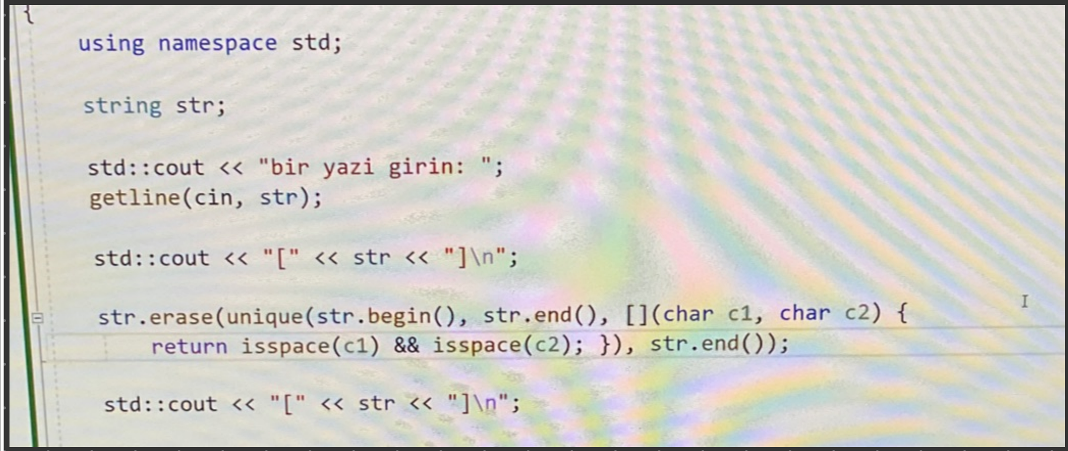
```cpp
using namespace std;

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

ivec.erase(unique(ivec.begin(), ivec.end(), [](int a, int b) {return a % 2 == b % 2; }),
    ivec.end());

std::cout << "ivec.size() = " << ivec.size() << "\n";

print(ivec);
```



Microsoft Visual Studio Debug Console

```
46 99 78 69 99 79 44 60 15 39 9 83 37 29 61 6 18 6 47 24 50 53 12 49 5 7 57 14 58 40 14 57 82 14 24 31 70 45 15 69 42 42 47 30 26
23 99 14 31 88 0 37 5 47 31 60 54 95 34 16 93 6 79 94 87 60 76 52 15 33 32 46 19 33 3 0 25 78 41 28 12 0 45 21 41 95 50 15 59 63 4
5 62 17 64 81 34 61 2 80 28
--------------------------------------------------------------------------
ivec.size() = 53
46 99 78 69 44 15 6 47 24 53 12 49 14 57 82 31 70 45 42 47 30 23 14 31 88 37 60 95 34 93 6 79 94 87 60 15 32 19 0 25 78 41 28 45 5
0 15 62 17 64 81 34 61 2
--------------------------------------------------------------------------
```

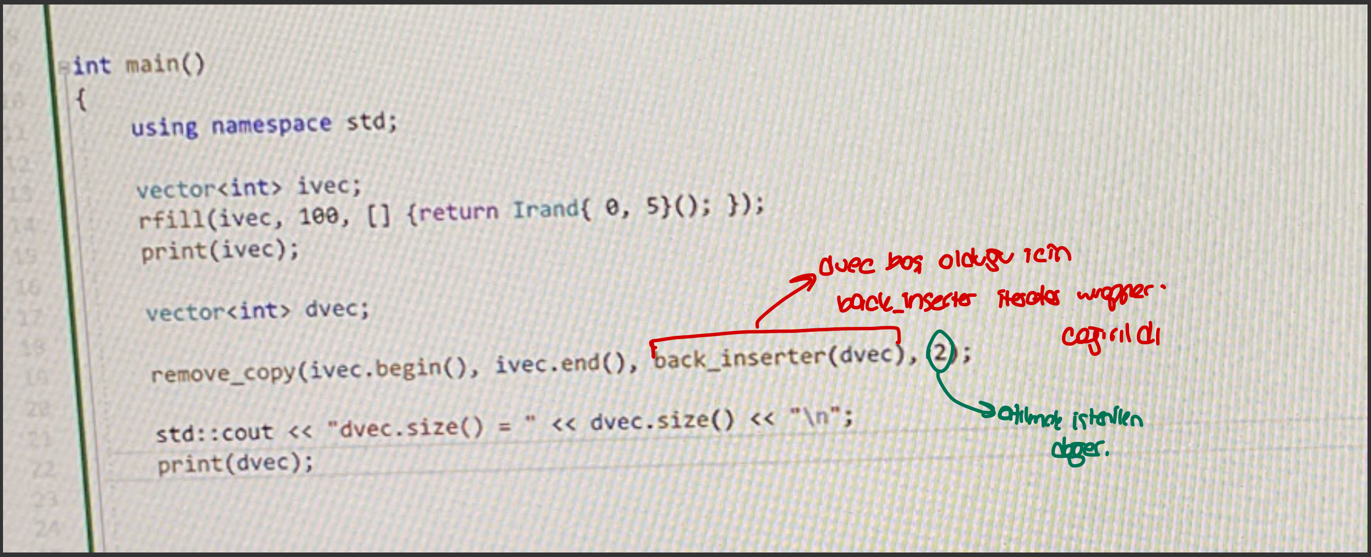
* Bir stringden boşluk silme örneği:



```cpp
{
    using namespace std;

    string str;

    std::cout << "bir yazi girin: ";
    getline(cin, str);

    std::cout << "[" << str << "]\n";

    str.erase(unique(str.begin(), str.end(), [](char c1, char c2) {
        return isspace(c1) && isspace(c2); }), str.end());

    std::cout << "[" << str << "]\n";
```

*** Remove Copy:** → Bir değer dışı öge hariç hepsini kopyala.

* Sonunda copy olan algoritmalar, bu işlemleri in-place değil, başka bir range için yapar.

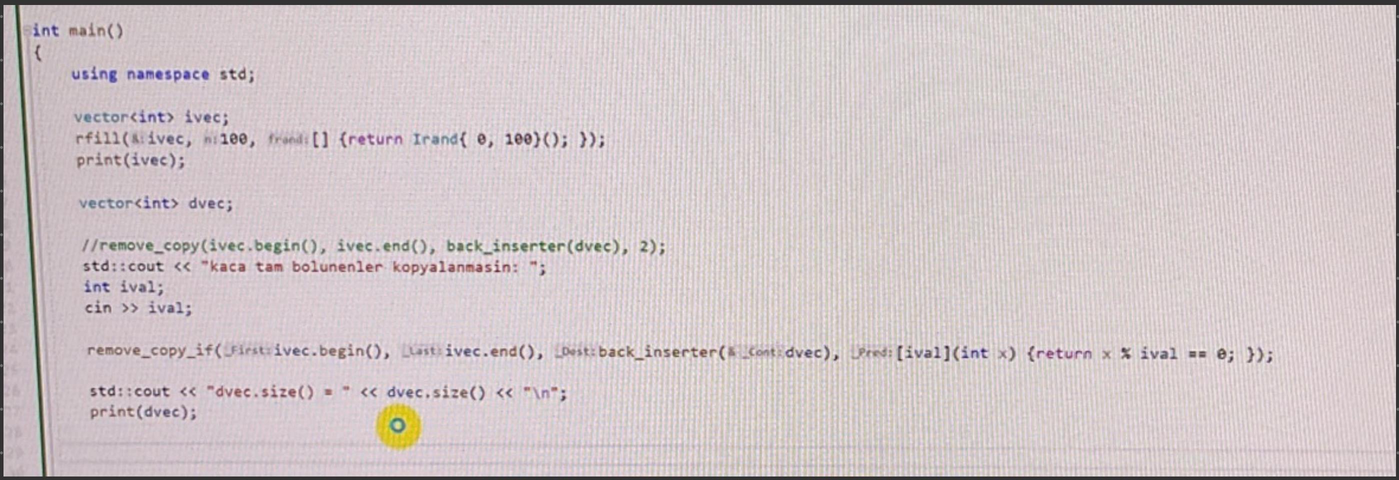


```cpp
int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 100, [] {return Irand{ 0, 5}(); });
    print(ivec);

    vector<int> dvec;

    remove_copy(ivec.begin(), ivec.end(), back_inserter(dvec), 2);

    std::cout << "dvec.size() = " << dvec.size() << "\n";
    print(dvec);
```

dvec boş olduğu için back_inserter iterator wrapper çağrıldı

atılmak istenilen değer.

*** Remove_Copy_If:**

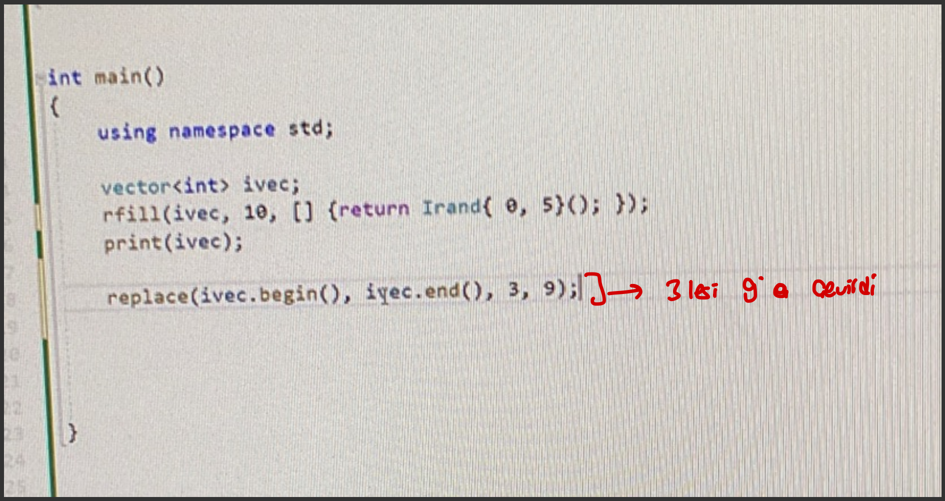


```cpp
int main()
{
    using namespace std;

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

    vector<int> dvec;

    //remove_copy(ivec.begin(), ivec.end(), back_inserter(dvec), 2);
    std::cout << "kaca tam bolunenler kopyalanmasin: ";
    int ival;
    cin >> ival;

    remove_copy_if(ivec.begin(), ivec.end(), back_inserter(dvec), [ival](int x) {return x % ival == 0; });

    std::cout << "dvec.size() = " << dvec.size() << "\n";
    print(dvec);
```

*** Reverse / Reverse Copy:**

→ range'i tersine çevirir.

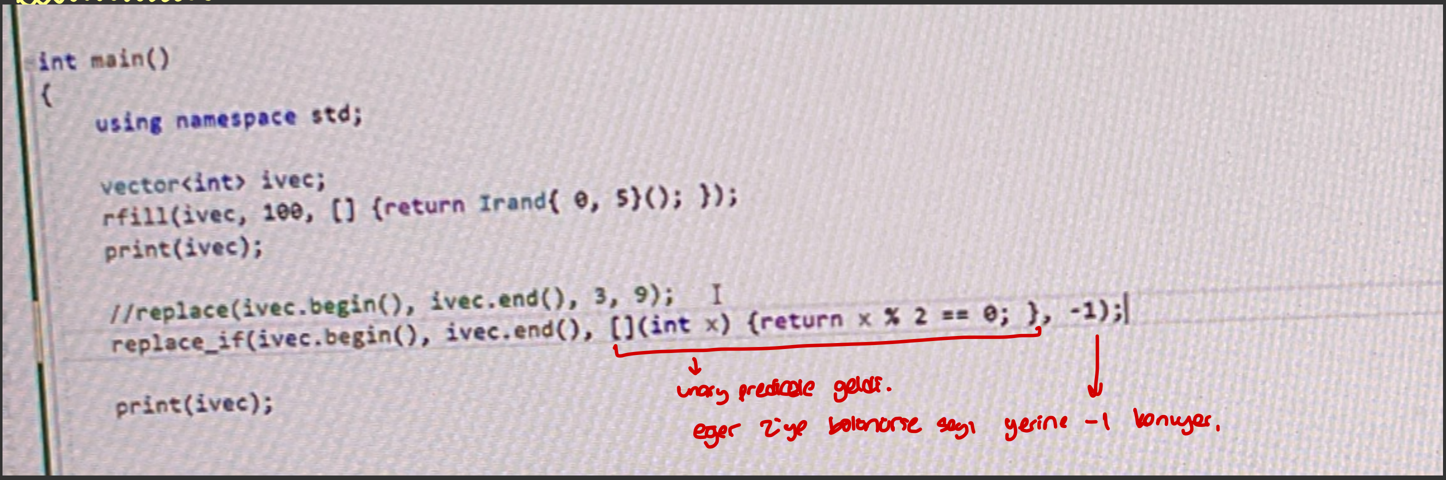
*** Replace:**



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

    vector<int> ivec;
    rfill(ivec, 10, [] {return Irand{ 0, 5}(); });
    print(ivec);

    replace(ivec.begin(), ivec.end(), 3, 9);



}
```

→ 3'leri 9'a çevirdi

* Replace_if():

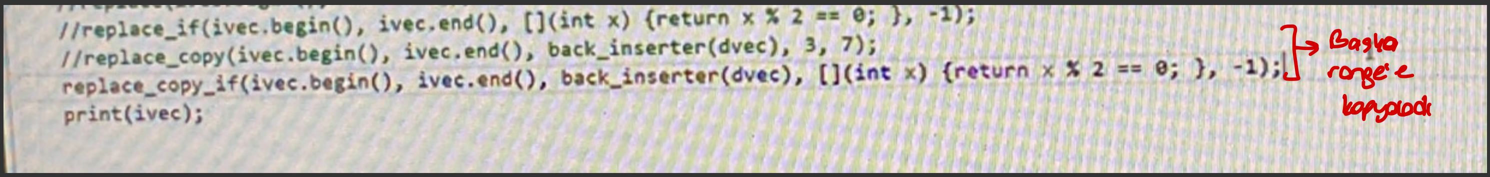


```cpp
int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 100, [] {return Irand{ 0, 5}(); });
    print(ivec);

    //replace(ivec.begin(), ivec.end(), 3, 9);
    replace_if(ivec.begin(), ivec.end(), [](int x) {return x % 2 == 0; }, -1);

    print(ivec);
```

unary predicate geldi.

eger 2'ye bolunurse sayı yerine -1 koyuyor.

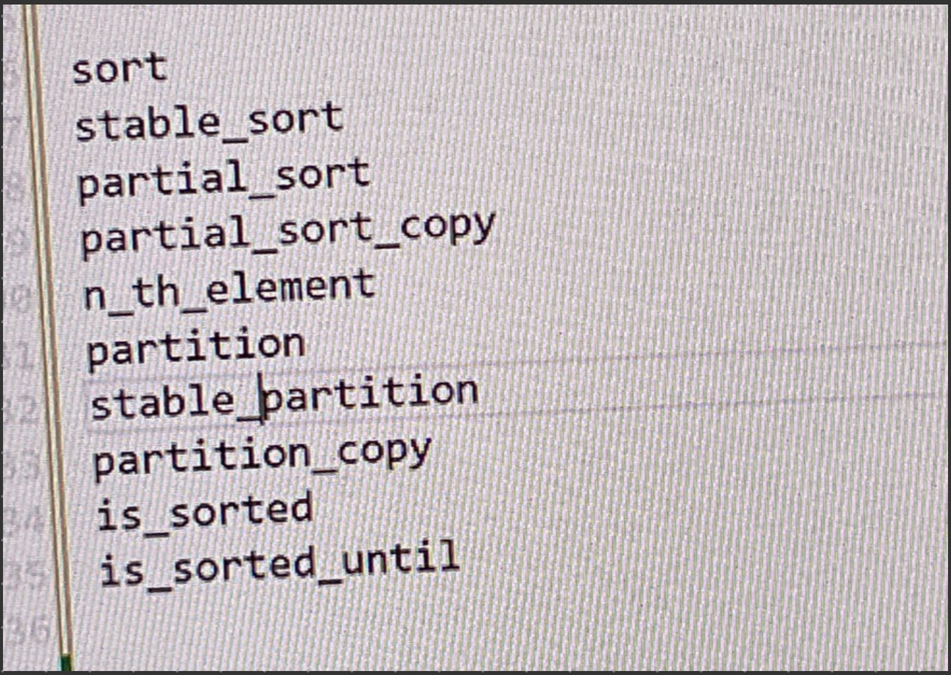
* Replace Copy / Replace Copy If:



```cpp
//replace_if(ivec.begin(), ivec.end(), [](int x) {return x % 2 == 0; }, -1);
//replace_copy(ivec.begin(), ivec.end(), back_inserter(dvec), 3, 7);
replace_copy_if(ivec.begin(), ivec.end(), back_inserter(dvec), [](int x) {return x % 2 == 0; }, -1);
print(ivec);
```

→ Başka range'e kopyaladı

* Sıralamaya ilişkin Algoritmalar:

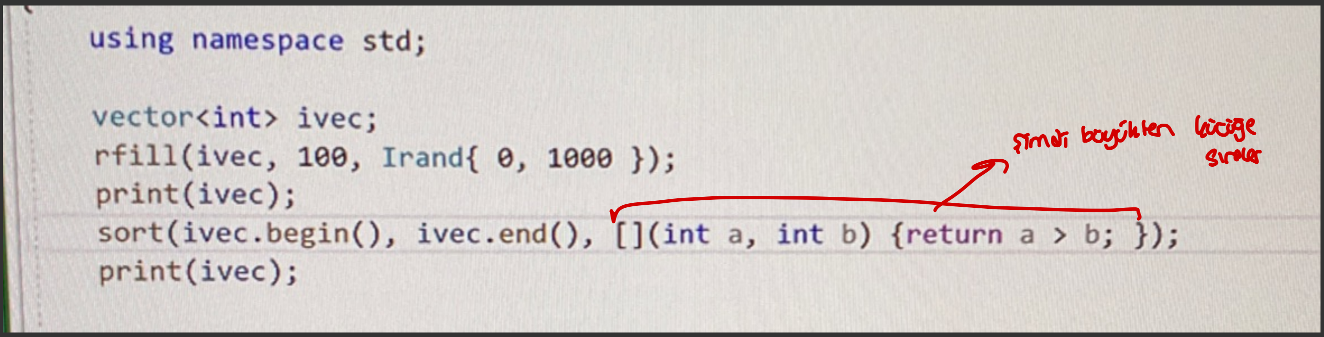


```
sort
stable_sort
partial_sort
partial_sort_copy
n_th_element
partition
stable_partition
partition_copy
is_sorted
is_sorted_until
```

* Sort:

→ N log (n)

→ random access iterator alır.



```cpp
using namespace std;

vector<int> ivec;
rfill(ivec, 100, Irand{ 0, 1000 });
print(ivec);
sort(ivec.begin(), ivec.end(), [](int a, int b) {return a > b; });
print(ivec);
```

şimdi büyükten küçüğe sıralar