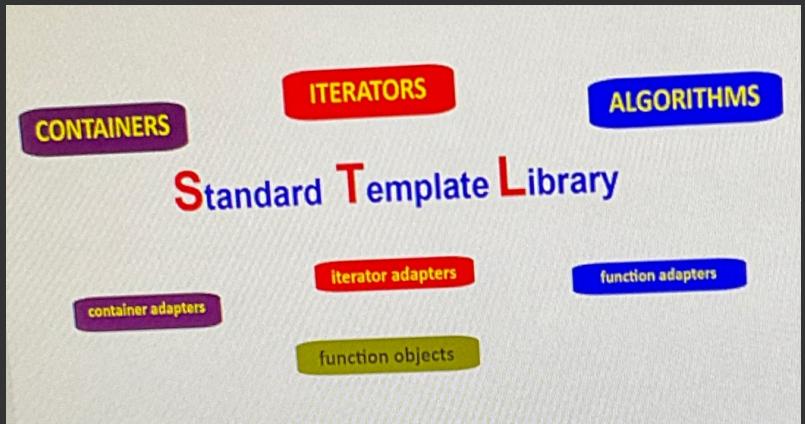


## \* STL:

- Standard Template Library
  - **Containers**: Tipde direkt veya yepitlenme implement eden sınıflar
  - **Iterators**: Kullanılan nesnelerin sınıfları, veriyapısında kullan zeminin konumunu taze pointerler de iterator.
  - **algorithms**: Farklışın template'sidir. Iteratorler kullanıd, containerlar içinde işlem yapmanının sağları.



sequence containers	associative containers
std::vector	std::set
std::deque	std::multiset
std::list	std::map
std::forward_list	std::multimap
std::array	
std::string	

C++11      C++11

unordered associative containers
std::unordered_set
std::unordered_multiset
std::unordered_map
std::unordered_multimap

C++11      C++11      C++11      C++11

- \* Range:**
- Bir aralık, bir veriyapısında  $n$  tane veriden oluşan birim
  - Bu n'ın içi konum bilgisini gerektir. Bir pointer 1 konum 1'den artıra artıra konum 2'ye gelme

```

void Print(const int* ps, const int* pe)
{
    while (ps != pe) {
        printf("%d ", *ps++);
    }
    printf("\n");
}

int main()
{
    int a[10] = { 1, 4, 7, 8, 9, 2, 7, 6, 3, 10 };
    Print(a, a + 10);
}

```

range ==> aralık

[konum1 konum2)

```

#include <iostream>
#include <vector>
#include <list>

template <typename Iter>
void Print(Iter ps, Iter pe)
{
    while (ps != pe) {
        std::cout << *ps++ << " ";
    }
    printf("\n");
}

int main()
{
    std::list<int> ilist{ 1, 4, 6, 8, 9, 12, 89, 91 };

    Print(ilist.begin(), ilist.end());
}

```

doubly linked list  
d. link list type farksızdır.

- Bu template ile farklı classların elementlerini print edebilidik, Ortak bir arayüz oluşturduk.
  - BIZ buada iterator durum, sınıf type farksızı gondedik.
  - `first.begin()` / `vector.begin()` ... gibi dairesel değer olan sınıf Operator Overloading mekanizmamıyla pointerlarda yepitilişimiz türlerin yapınanımıza elenmiş sağlıyor ✓ (gelen sayıları ornek)
- Böyle sınıflara iterator deniyor!!!

```

#include <iostream>
#include <vector>
#include <list>

template <typename Iter> void Print(Iter ps, Iter pe)
{
    while (ps != pe) {
        std::cout << *ps++ << " ";
    }
    printf("\n");
}

int main()
{
    std::vector<int> ivec{ 1, 4, 6, 8, 9, 12, 89, 91 };

    Print(ivec.begin(), ivec.end());
}

```

vector type farksızdır.

### \* Iteratorların Alt tipisi: Su şekilde:

```

63 template<typename T> class list {
64     class iterator {
65         public:
66             class iterator {
67                 public:
68                     T& operator*();
69                     operator++();
70                     operator!=();
71             };
72             iterator begin();
73             iterator end();
74         };
75     };
76 };
77 };
78 };
79 };
80 };
81 }

```

List classının class template'i  
public interface'ı  
Tasarruf edilen Iterator  
needed class'ı var.  
saglıklı olumsuzluk  
operator overloadleri  
yapılıyorum.  
Iterator return type'i  
begin ve end farklıdır,

```

int main()
{
    std::list<int> ilist{ 1, 4, 6, 8, 9, 12, 89, 91 };

    Print(ilist.begin(), ilist.end());
}

Bu şekilde yapılan  
çalıştırıldığında Template Argmet.  
Değerin yapısını. Specialization su şekilde

```

```

void Print(std::list<int>::iterator ps, std::list<int>::iterator pe)
{
    while (ps != pe) {
        std::cout << *ps++ << " ";
    }
    printf("\n");
}

```

Algorithması

## \* Copy:

```

9
10 template <typename InIter, typename OutIter>
11 OutIter Copy(InIter beg, InIter end, OutIter destbeg);
12
13 bağlıt range [ ] bağlıt blokları
14

```

- Konveksiyon dursak: kaynak hedef bitti.
- Return dursak: Bir sen nesne yordaysa, ondan sonraki konum, 4'üncü yordigi range'in end. konum.

```

template <typename InIter, typename OutIter> <T> Provide sample template arg
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    while (beg != end) {
        *destbeg++ = *beg++;
    }
    return destbeg;
}

```

- Bu kod ile var vector, bir liste gibi olabilir. Sadece bu sıfırtan adresindeki vector birbiriyle eklebilir olursa.

```

int main()
{
    using namespace std;
    vector<int> ivec{ 1, 4, 7, 2, 3}; double'a int atanır.
    list<double> dlist(6);
    copy(ivec.begin(), ivec.end(), dlist.begin());
    print(dlist);
}

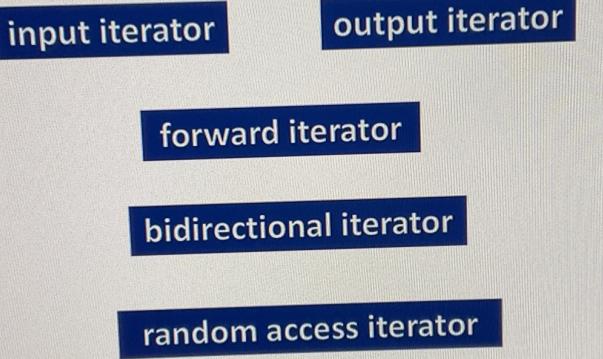
```

## \* Iterator Category:

- Iteratorin hangi işleme sözleşmesi varsa belirtirken dekt ( ++, --, c. = gibi işlevler)

→ Kategori kriterinden biri olmalıdır.

→ Bu sınıfı implement ettiğimiz bir sınıfın içinde etmeliyim.



## iterator kategorileri

## operasyonlar

output iterator	<pre>copy constructible ++it it++ *it  it-&gt; (sol taraf değeri)</pre>	ostream_iterator ostreambuf_iterator
input iterator	<pre>copy constructible ++it it++ *it  it-&gt; (sağ taraf değeri) it1 == it2      it1 != it2</pre>	istream_iterator istreambuf_iterator
forward iterator	<pre>copy constructible default constructible ++it it++ *it  it-&gt; (sağ taraf değeri) (sol taraf değeri) it1 == it2      it1 != it2</pre>	forward_list unordered_set unordered_multiset unordered_map unordered_multimap
bidirectional iterator	<pre>copy constructible default constructible ++it it++      --it it-- *it  it-&gt; (sağ taraf değeri) (sol taraf değeri) it1 == it2      it1 != it2</pre>	list set multiset map multimap
random access iterator	<pre>copy constructible default constructible ++it it++      --it it-- *it  it-&gt; (sağ taraf değeri) (sol taraf değeri) it1 == it2      it1 != it2 it + n  n + it  it - n it + n  it - n it1 - it2 it[n] it1 &lt; it2  it1 &lt;= it2  it1 &gt; it2  it1 &gt;= it2</pre>	vector deque array string C arrays

maçelik  
Aynı  
pointer gibi

→ iterator class, Iterator Category needed type'ı vector, Type Id ile ögrenilebilir.

```
using namespace std;
int main()
{
    cout << typeid(vector<int>::iterator::iterator_category).name() << "\n";
    cout << typeid(list<int>::iterator::iterator_category).name() << "\n";
    cout << typeid(forward_list<int>::iterator::iterator_category).name() << "\n";
    cout << typeid(istream_iterator<int>::iterator_category).name() << "\n";
    cout << typeid(ostream_iterator<int>::iterator_category).name() << "\n";
}
```

→ Sort, random access iteratörler, list iteratörleri daha uygun değil!

\* Bu algoritmelerin sonu if ne bulursa, return değer boolean

↓  
prependice Functions

## \* copy\_if Algorithm:

```
using namespace std;

template <typename InIter, typename OutIter, typename Pred> <T> Provide sample
OutIter CopyIf(InIter beg, InIter end, OutIter destbeg, Pred f)
{
}
```

f callable dir.

```
using namespace std;
```

```
template <typename InIter, typename OutIter, typename Pred> <T> Provide sa  
OutIter CopyIf(InIter beg, InIter end, OutIter destbeg, Pred f)  
{  
    while (beg != end) {  
        if (f(*beg)) {  
            *destbeg++ = *beg;  
        }  
        ++beg;  
    }  
    return destbeg;  
}
```

*- her elementi f callable function'a verir.  
- f predicate function → kendi segmenrsa true  
segmenrsa false*

*f() { val 1,2 == 0 }*

*:*

```
int main()  
  
vector<int> ivec;  
rfill(ivec, 100, Irand{ 0, 1000 });  
print(ivec);  
  
list<int> ilist(100);  
  
auto iter = copy_if(ivec.begin(), ivec.end(), ilist.begin(), &is even);  
  
print(ilist.begin(), iter);
```

*f callable'na  
farkiyen  
adres  
göndür.*

*ya da böyle bir fonksiyon  
yerine, böyle bir fonksiyona sınıf class oluşturur.*

```
class DivPred {  
public:  
    DivPred(int val) : m_x{val}{}  
    bool operator()(int x) const  
    {  
        return x % m_x == 0;  
    }  
private:  
    int m_x;  
};
```

*\* Not: bu classı, örneğin compiler'a yollandıktan sonra  
gerek bir name oluşturularak  
bu ne lambda  
expression'dır.*

*bu bir overload edilmiş  
predicate function yaratır.*

```
int main()  
{  
    vector<int> ivec;  
    rfill(ivec, 100, Irand{ 0, 1000 });  
    print(ivec);  
  
    list<int> ilist(100);  
  
    auto iter = copy_if(ivec.begin(), ivec.end(), ilist.begin(), DivPred{});
```

*Bu, function object (functor class)  
dir.*