**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**)...)**

**}**

**}**