**2023 11 17**

**set.insert() -- hint insert**

**int** main**()**

**{**

**using** **namespace** std**;**

set**<**string**>** myset**{** "ayca"**,** "berna"**,** "derya"**,** "meliha"**,** "saliha"**};**

***// eğer doğru konum verirsek performans verir.***

myset**.**insert**(**myset**.**begin**(),** "aliye"**);** ***// hint insert***

vector**<**string**>** svec **{**"eda"**,** "naz"**,** "tan"**};**

copy**(**svec**.**begin**(),** svec**.**end**(),** inserter**(**myset**,** myset**.**begin**());**

**}**

**inserter**

***// inserter***

**int** main**()**

**{**

**using** **namespace** std**;**

vector**<**string**>** svec **{**"eda"**,** "naz"**,** "tan"**};**

set**<**string**>** myset**{** "ayca"**,** "berna"**,** "derya"**,** "meliha"**,** "saliha"**};**

copy**(**svec**.**begin**(),** svec**.**end**(),** inserter**(**myset**,** myset**.**begin**());**

**}**

**set.emplace\_hint()**

***// emplace hint***

**int** main**()**

**{**

***// perfect forwarding kullanarak set'in sağladığı bellek alanında nesneyi oluşturur***

myset**.**emplace\_hint**(**myset**.**begin**(),** 10**,** 'A'**);**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

set**<**string**>** myset**;**

rfill**(**myset**,** 10**,** rname**);**

cout **<<** "eski ve yeni ismi giriniz:"**;**

string old\_name**,** new\_name**;**

cin **>>** old\_name **>>** new\_name**;**

***// eski cpp için find ile bulup silip daha sonra yeni elemanı eklicez***

**if** **(auto** iter **=** myset**.**find**(**old\_name**);** iter **!=** myset**.**end**())**

**{**

myset**.**erase**(**iter**);**

myset**.**insert**(**new\_name**);**

std**::**cout **<<** "anahtar degistirildi\n"**;**

**}**

**else**

**{**

cout **<<** "bulanamadi\n"**;**

**}**

**}**

**set.extract()**

**int** main**()**

**{**

**using** **namespace** std**;**

set**<**string**>** myset**;**

rfill**(**myset**,** 10**,** rname**);**

cout **<<** "eski ve yeni ismi giriniz:"**;**

string old\_name**,** new\_name**;**

cin **>>** old\_name **>>** new\_name**;**

**auto** iter **=** myset**.**find**(**old\_name**);**

**auto** handle **=** myset**.**extract**(**iter**);**

cout **<<** "size = " **<<** myset**.**size**()** **<<** "\n"**;** ***// size = 9***

handle**.**value**()** **=** new\_name**;**

myset**.**insert**(**std**::**move**(**handle**));**

cout **<<** "size = " **<<** myset**.**size**()** **<<** "\n"**;** ***// size = 10***

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

multiset**<int>** myset**;**

rfill**(**myset**,** 20**,** Irand**{**0**,** 5**});**

print**(**myset**);**

**auto** **[**iter\_first**,** iter\_last**]** **=** myset**.**equal\_range**(**3**);**

**}**

**Set Lambda Karşılaştırma Kriteri**

**int** main**()**

**{**

**using** **namespace** std**;**

set**<int,** **decltype([](int** x**,** **int** y**)**

**{**

**return** x **%** 100 **<** y **%** 100**;**

**})>** myset**;**

**}**

**std::map**

**int** main**()**

**{**

**using** **namespace** std**;**

***// key value***

map**<int,** string**,** less**{}>** mx**;**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

mx**.**insert**(**pair**{** rand**(),** rname**()});**

**}**

**for** **(const** **auto&**p **:** mx**)**

**{**

cout **<<** p**.**first **<<** " " **<<** p**.**second **<<** "\n"**;** ***// key'e göre sıralar***

**}**

**auto** iter **=** mx**.**begin**();**

iter**->**first **=** 323**;** ***// syntax hatası çünkü key const***

iter**->**second **=** "emre"**;** ***// hata yok***

**}**

**std::map'a öğe ekleme**

**int** main**()**

**{**

**using** **namespace** std**;**

map**<**string**,** **int>** mx**;**

pair**<**string**,** **int>** p1 **{**"tayfun"**,** 767**};**

mx**.**insert**(**p1**);**

mx**.**insert**(**pair**{** "alican"**,** 871**});**

mx**.**insert**({**"emrecan"**,** 823**});**

mx**.**insert**(**make\_pair**(**"deniz"**,** 761**));**

mx**.**emplace**(**"damla"**,** 732**);**

**for** **(auto** iter **=** mx**.**begin**();** iter **!=** mx**.**end**();** **++**iter**)**

**{**

std**::**cout **<<** iter**->**first **<<** " " **<<** iter**->**second **<<** "\n"**;**

**}**

***// for-ranged***

**for** **(const** **auto&** p **:** mx**)**

**{**

std**::**cout **<<** p**.**first **<<** " " **<<** p**.**second **<<** "\n"**;**

**}**

***// structured binding C++17***

**for** **(const** **auto&** **[**name**,** grade**]** **:** mx**)**

**{**

std**::**cout **<<** name **<<** " " **<<** grade**;**

**}**

**}**

**[] operator fonksiyonu**

**int** main**()**

**{**

**using** **namespace** std**;**

map**<int,** string**>** mymap**;**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

mymap**.**emplace**(**Irand**{**0 **,** 70**}(),** rname**());**

**}**

print**(**mymap**,** "\n"**);**

**int** key**;**

cin **>>** key**;**

***// ref semantiği ile key'e erişebiliriyoruz***

mymap**[**key**]** **=** "sadullah"**;**

***// eğer key yoksa o keyi eklemiş oluyoruz***

mymap**.**at**(**key**)** **=** "sadullah"**;** ***// anahtar yoksa exception throw eder***

**}**

***// vectordeki isimlerden kaç tane olduğunu map'a yazıyoruz***

**int** main**()**

**{**

**using** **namespace** std**;**

vector**<**sting**>** svec**;**

rfill**(**svec**,** 10000**,** rname**);**

map**<**string**,** **int>** cmap**;**

**for** **(const** **auto&** name **:** svec**)**

**{**

**++**cmap**[**name**]**

**}**

vector**<**pair**<**string**,** **int>>** myvec**{**cmap**.**begin**(),** cmap**.**end**()};**

sort**(**myvec**.**begin**(),,** myvec**.**end**(),** **[](const** **auto&** p1**,** **const** **auto&** p2**)**

**{**

**return** p1**.**second **>** p2**.**second**;** ***// sıralama yaptık sayıya göre***

**});**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

map**<**string**,** **int>::**value\_type ***// pair<string, int>***

map**<**string**,** **int>::**key\_type ***// string***

map**<**string**,** **int>::**mapped\_type ***// int***

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

map**<**string**,** **int>** mymap**;**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

mymap**.**emplace**(**rname**(),** Irand **{** 0**,** 70**}());**

**}**

string old\_key**,** new\_key**;**

cout **<<** "eski ve yeni anahtarları girin: "**;**

cin **>>** old\_key **>>** new\_key**;**

**auto** handle **=** mymap**.**extract**(**old\_key**);**

**if** **(**handle**)**

**{**

handle**.**key**()** **=** new\_key**;**

mymap**.**insert**(**move**(**handle**));**

cout **<<** "anahtar degistirildi\n"**;**

**}**

**}**

**Hash Table**

* Hash'ta anahtar ile arama constant time complexity
* Sette anahtar ile arama linear time complexity

#include <functional>

**int** main**()**

**{**

**using** **namespace** std**;**

hash**<int>** hasher**;**

**size\_t** val **=** hasher**(**87423**);**

cout **<<** val **<<** "\n"**;** ***// hash değeri***

**}**

#include <functional>

**int** main**()**

**{**

**using** **namespace** std**;**

hash**<**string**>** hasher**;**

cout **<<** hasher**(**"deniz"**);** **<<** "\n"**;** ***// hash değeri***

cout **<<** hasher**(**"denis"**);** **<<** "\n"**;** ***// hash değeri***

cout **<<** hasher**(**"denizcan"**);** **<<** "\n"**;** ***// hash değeri***

**}**

**struct** DateHasher

**{**

std**::size\_t** **operator()(const** Date**&** dt**)const**

**{**

std**::**hash**<int>** hasher**;**

**return** hasher**(**dt**.**month\_day**())** **+** hasher**(**dt**.**month**())** **+** hasher**(**dt**.**year**());**

**}**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

Date mydate**{** 17**,** 11**,** 2023**};**

cout **<<** DateHasher**{}(**mydate**)** **<<** '\n'**;**

**++**mydate**;**

cout **<<** DateHasher**{}(**mydate**)** **<<** '\n'**;**

**}**

**std::unordered\_set**

**int** main**()**

**{**

**using** **namespace** std**;**

unordered\_set**<int,** hash**<int>,** equal\_to**<int>>** ***// unordered\_set<int>***

**}**

***// Eğer kendi veri tipimizi kullancaksak***

**class** DateHasher

**{**

std**::size\_t** **operator()(const** Date**&)const;**

**};**

**struct** DateEqual

**{**

**bool** **operator()(const** Date**&,** **const** Date**&)const;**

**};**

**int** main**()**

**{**

**using** **namespace** std**;**

unordered\_set**<**Date**,** DateHasher**>** myset**;**

**}**

**lambda kullanarak unordered\_set oluşturma**

***// lambda kullanarak unordered\_set oluşturma***

**int** main**()**

**{**

**using** **namespace** std**;**

**auto** fhash **=** **[](const** Date**&)**

**{**

**return** 12u**;**

**};**

**auto** eq **=** **[](const** Date**&,** **const** Date**&)**

**{**

**return** **true;**

**};**

unordered\_set**<**Date**,** **decltype(**fhash**),** **decltype(**eq**)>;**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

unordered\_set**<int>** myset**(**400**);** ***// bucket sayısını belirleriz***

cout **<<** myset**.**bucket\_count**()** **<<** "\n"**;**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

unordered\_set**<**string**>** myset**;**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

myset**.**insert**(**rname**());**

**}**

print**(**myset**);**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

string name**;**

cout **<<** "isim girin: "**;**

cin **>>** name**;**

**if** **(**myset**.**contains**(**name**))**

**{**

**}**

**}**

**}**