

* Set Template Parameters:

```
1129 =====
1130
1131 template<typename Key, typename Comp = std::less<Key>, typename A = std::allocator<Key>>
1132 class Set;
1133
1134 Für greater
Küllermet için → #include <functional>
```

```
1
template <typename T>
using gset = std::set<T, std::greater<T>>;
2int main() Bu set ile küllerim, nog
{
    using namespace std;

    gset<int> myset{ 3, 6, 9, 1, 2, 8 };
    gset<string> myset;
```

* Kendi binary predicate'umuzu nasıl ekleriz?

1.

```
bool abs_comp(int a, int b)
{
    return std::abs(a) < std::abs(b);
}

// -4 -5 5 7 9 12 -12 Eğer multiset
olsa bu cout'da olurdu.

int main()
{
    using namespace std;
    set<int, bool (*)> myset(&abs_comp);

    myset.insert(12);
    myset.insert(-5);
    myset.insert(7);
    myset.insert(5);
    myset.insert(9);
    myset.insert(-4);
    myset.insert(-12);

    for (auto i : myset)
        cout << i << ' ';
}
```

→ Cevap: -4, -5, 7, 9, 12

2.

```
bool abs_comp(int a, int b)
{
    return std::abs(a) < std::abs(b);
}

// -4 -5 5 7 9 12 -12

int main()
{
    using namespace std;
    set<int, bool (*)> myset(&abs_comp);
    // set<int, decltype(&abs_comp)> myset(&abs_comp);
```

→ with decltype

3.

```

3. struct Comp {
4.     bool operator()(int a, int b) const
5.     {
6.         return std::abs(a) < std::abs(b);
7.     }
8. };
9.
10. // -4 -5 5 7 9 12 -12
11.
12. int main()
13. {
14.     using namespace std;
15.
16.     set<int, Comp> myset;
17.
18.     myset.insert(12);
19.     myset.insert(-5);
20.     myset.insert(7);
21.     myset.insert(5);
22.     myset.insert(9);
23.     myset.insert(-4);
24.     myset.insert(-12);
25.
26.     for (auto i : myset)
27.         cout << i << ' ';
28. }

```

With
a Functor
class

4.

```

4. struct Comp {
5.     bool operator()(int a, int b) const
6.     {
7.         return std::abs(a) < std::abs(b);
8.     }
9.
10. // -4 -5 5 7 9 12 -12
11.
12. int main()
13. {
14.     using namespace std;
15.
16.     auto fcomp = [] (int a, int b)
17.     {
18.         return std::abs(a) < std::abs(b);
19.     };
20.
21.     // set<int, decltype(fcomp)> myset(fcomp);
22.     set<int, decltype(fcomp)> myset;
23.
24.     myset.insert(12);
25.     myset.insert(-5);
26.     myset.insert(7);
27.     myset.insert(5);
28.     myset.insert(9);
29.     myset.insert(-4);
30.     myset.insert(-12);
31.
32.     for (auto i : myset)
33.         cout << i << ' ';
34. }

```

with
Lambda
expression

Cpp 11 de
ortak
değişiklik
parametresi
gönderebilir
de olur

* Set Member Functions:

- `insert`: Arguman olarak key olan fonksiyon.

```

using namespace std;

set<string> myset;

myset.insert("nedim");
myset.insert("ayse");
myset.insert("fuat");
myset.insert("nedim");

print(myset);

```

Set oldugu için 2. nedim eklendi.

* `Set'in return type'i: pair<iterator, bool>`. Gurktu ekrana nümeri her zaman
gördürmeyecektir. Yani `pair<iterator, false>` Ekleme yapıldı `pair<iterator, true>`

* Eğer bir anahtarın değerini değiştirmek istiyorsak: - var olan anahtarı silip onu yeniden insert.

- Fakat var olan anahtarın konumunu sorgu yapıp ekrana XX Anahtar: değiştirdim !!

* Hint `Insert`: `·insert overload`u`

```

using namespace std;

set<string> myset;
rfill(myset, 10, rname);
print(myset);

//hint insert
myset.insert(myset.begin(), "yilmaz");
print(myset);

```

EEER O KONUMA

EVLİNGİÇİSE , Yılmaz değeri ekli.
Yoksa Yılmaz değeri , ölmeli geleneklere
gödük. Bz yol gösterdi.

* Emplace_Hint:

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

    set<Date> myset;
    rfill(myset, 10, Date::random);
    print(myset, "\n");

    myset.emplace_hint(myset.begin(), 12, 5, 1987);

    print(myset, "\n");
```

→ insert overload ile aynı

* Erase:

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

    set<string> myset;
    rfill(myset, 10, rtown);
    print(myset);

    myset.erase(myset.begin());           → header arguman
    print(myset);
    myset.erase(prev(myset.end()));       → range
    print(myset);
    myset.erase(next(myset.begin()), prev(myset.end())); → range
    print(myset);
```

* Benzer ek operatör, key value ile erase de
yapılır.

* Erase_if:

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

    multiset<string> myset;
    rfill(myset, 200, rtown);

    auto n = erase_if(myset, [](const string& s) {return s.contains('a'); });
    print(myset);

    cout << n << " anahtar silindi\n";
}
```

* Count: → set için var mı yok mu

multiset için looks like var

```
using namespace std;

multiset<string> myset;
rfill(myset, 20, rtown);

string town;
cout << "sehri girin: ";
cin >> town;

if (myset.count(town)) {
    //var
} else {
    //yok
}
```

→ alternatif countnes.

* Find(): Eger verilen key'in iterator konumunu
bulursa.

```
multiset<string> myset;
rfill(myset, 20, rtown);

string town;
cout << "sehri girin: ";
cin >> town;

if (auto iter = myset.find(town); iter != myset.end()) {
```

Fakat multiset oluyagi,
var olan ilk iterator konum

Anahtar Değerini Degistirme

- * extract fonksiyonu ile set'in node'un tipinden karsi, *Custom types'ın en direkt kullanılışıdır!

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

    set<string> myset;
    rfill(myset, 10, rtown);
    print(myset);

    auto iter = next(myset.begin());
    cout << *iter << "\n";           → return değer node type

    auto handle = myset.extract(iter);   ←
    handle.value() = "barcelona";       → iter overload ver
    myset.insert(move(handle));         → ya稀缺 olan
                                         → ya ol olmayan.

    print(myset);
}
```

* Lower Bound:

- * Range de sınırlamış değerlerin sırasını bozmadan elemanları ekleme işlemi için kullanılır.
- * Bir anahtarın varlığı bir konum.
- * Anahtardan büyük ya da eşitlik egerin konumu.

* Upper Bound:

- * Sırası bozmadan elemanları ekleme işlemi için kullanılır.
- * Anahtardan büyük olan ilk değerin konumu.

* Equal Range:

[lower bound, upper bound] → elementin key
→ distance : key aralığı.

```
{ using namespace std;

multiset<int> myset;

rfill(myset, 20, Irand{ 0, 20 });
print(myset);

int ival;

cout << "anahtar degerini girin: ";
cin >> ival;

auto iter_lower = myset.lower_bound(ival);
auto iter_upper = myset.upper_bound(ival);

cout << "equal range distance : " << distance(iter_lower, iter_upper) << "\n";
print(iter_lower, iter_upper);
```

```

int main()
{
    using namespace std;

    vector<string> svec;

    //dikkat, mulakat sorusu

    for (int i = 0; i < 20; ++i) {
        auto name = rname();
        cout << name << "\n";
        auto iter = lower_bound(svec.begin(), svec.end(), name);
        svec.insert(iter, name);
        print(svec);
        (void)getchar();
    }
}

```

→ verilere
girdi eklemeye
yaptık.

```

int main()
{
    using namespace std;

    vector<string> svec;
    const auto fcomp = [] (const std::string& s1, const std::string& s2) {
        return s1.length() < s2.length() || s1.length() == s2.length() && s1 < s2;
    };

    //dikkat, mulakat sorusu

    for (int i = 0; i < 20; ++i) {
        auto name = rname();
        cout << name << "\n";
        auto iter = lower_bound(svec.begin(), svec.end(), name, fcomp);
        svec.insert(iter, name);
        print(svec);
        (void)getchar();
    }
}

```

I

Birim binary predicate'ınıza
gore göre.

#NOT:

```

int main()
{
    using namespace std;

    //set<int>::key_type
    set<int>::value_type
}

```

I

set icin key type ile value type
ayndır.

* Map:

```
}
```

```
map<int, string>
```

```
template<typename Key, typename Val, typename Comp = less<Key>, typename A = allocator<pair<Key, Value>>
```

```
class Map {
```

```
}
```

I

→ Amaç, anıktan look up yapın, bkr obje erişimi.

→ Set ile interface'i neredeyse aynı

```
int main()
```

```
{
```

```
    using namespace std;
```

```

    map<int, string> imap;
```

```

    for (int i = 0; i < 10; ++i) {
        imap.insert(pair<int, string>{rand(), rname()}); → Filler pair
        imap.insert({rand(), rname()}); → pair'in constructorine gonderilen argüman ] → ile type checked olur.
        imap.insert(make_pair(i, rname())); → make pair'in returno
    }
```

aynı şekildeemplace kullanılır.

```
for (int i = 0; i < 10; ++i) {
    imap.insert(pair<int, string>{rand(), rname()});
    imap.insert({rand(), rname()});
    imap.insert(make_pair(i, rname()));
    imap.emplace(rand(), rname()); [
```

* Uzune range based for loop ile traverse edilebilir.

```
for (const auto& p : imap) {
    cout << p.first << " " << p.second << "\n";
}
```

```
for (const auto&[id, name] : imap) {
    cout << id << " " << name << "\n";
}
```

→ Structured Binding

* Greater Orneği:

```
using namespace std;
```

```

map<string, int, greater<>> imap;
```

```

for (int i = 0; i < 10; ++i) {
    imap.insert({_val:pair<string, int>{rname(), rand() % 100}});
    imap.insert({_val:{rname(), rand() % 100}});
    imap.insert({_val:make_pair(_val1:rname(), &_val2:i)});
    imap.emplace(_vals:rname(), _vals:rand() % 100);
}
```

```

for (const auto&[ const std::string & name, const int & id] : imap) {
    cout << name << " " << id << "\n";
}
```

first

second

• Başlıklar koyucu similar

• Bu sefer stringler, key oldu. Orden sıralanacak.

+ Map içinde değer arama:

```
using namespace std;

map<string, int> imap;

for (int i = 0; i < 10; ++i) {
    imap.insert(pair<string, int>{rname(), rand() % 100});
    imap.insert({rname(), rand() % 100});
    imap.insert(make_pair(rname(), i));
    imap.emplace(rname(), rand() % 100);
}

for (const auto&[name, id] : imap) {
    cout << name << " " << id << "\n";
}

string name;
std::cout << "aranacak ismi girin: ";
cin >> name;

if (auto iter = imap.find(name); iter != imap.end()) {
    cout << "bulundu degeri " << iter->second << "\n";
}
```

Find o argument
olarak key verir.