2023.10.27

Algorithm

```
// copy
// Not : algoritmalar exception throw etmez
template <typename InIter, typename OutIter>
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    /*
        -beg arttır arttır ama hiçbir zaman end  eşit olmazsa tanımsız davranış
        -destbeg'dan sonra container'da üye yoksa tanımsız davranış
        -eğer bir yazma işlemi yapıyorsak fonksiyonun geri dönüş değeri en son
        yazdığımız konumdan sonraki yerdir.

    */

    // beg en az input_iterator içeriğine sahip olmalı çünkü InIter
    // destbeg en az output_iterator içeriğine sahip olmalı çünkü OutIter

    while (beg != end)
    {
        *destbeg++ = *beg++;
    }
    return destbeg;

    // farklı container üzerinde kopyalama işlemleri yapabiliriz.
}

int main()
{
    using namespace std;

    vector<int> ivec{ 2, 5, 7, 9, 1, 3};
    list<int> ilist;

    // tanımsız davranış çünkü list'e boş
    Copy(ivec.begin(), ivec.end(), ilist.begin());

    list<int> ilist1(10);
    auto iter = Copy(ivec.begin(), ivec.end(), ilist1.begin());

    ilist.begin(), iter // algoritma'nın yazdığı öğeler var
}
```

Algoritmaların parametre değişkenleri iterator'dür: Böylece bir algoritma örneğin şunları yapabilir:

- Bir iterator konumundaki nesneye atama yapabilir.
- Aynı range'deki iki nesneyi takas edebilir.
- Range'in ilişkin olduğu olduğu kaba ekleme ve silme yapamaz

```
int main()
{
    const char* const p[] = { "emre" , "bilge", "damla", "tamer", "gokhan" };
    vector<string> svec(6);

    copy(begin(p), end(p), begin(svec));
}
```

Predicate: bool döndüren callable'lar denir. bkz bool isEven(int);

Copy If

```
template<typename Inter, typenameOutIter, typename Upred>
OutIter CopyIf(Inter beg, InIter end, OutIter destbeg, Upred f)
{
    while (beg != end)
    {
        if (f(*beg))
        {
            *destbeg++ = *beg;
        }
        ++beg;
    }

    return destbeg;
}
bool isEven(int x)
{
    return x % 2 == 0;
}

int main()
{
    using namaspace std;

    vector<int> ivec{2, 5, 7, 10, 4, 3, 6, 8, 1, 9};
    list<int> ilist(10);

    CopyIf(ivec.begin(), ivec.end(), ilist.begin(), iseven);

    int n;
    cin >> n;

    CopyIf(ivec.begin(), ivec.end(), ilist.begin(),
        [n](int x) {return x % n == 0; } ); // lambda ifadesi
        /*
            lambda ifadelerinde derleyici bir class  yazar ve
            geçeci nesne oluşturur  ( closure object)
        */

}
```

Count If

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

    vector<string> svec;
    rfill(svec, 100'000, rname);

    cout << count(svec.begin(), svec.end(), "tamer") << "\n";

    char c = 'e';
    count_if(begin(svec), end(svec), [c](const string& s)
    {
        return s.find(c) != std::npos;
    });

    sort(svec.begin(), svec.end()); // küçükten büyüğe vectörü sıralar
}
```

Iter Const'luk

```
int main()
{
    using namaspace std;
    vector<string> svec{  "ali", "can", "ece", "tan" };

    const vector<string>::iterator iter = svec.begin();

    ++iter; // syntax hatası
    *iter = "murathan"; // legal
    iter.operator*() = "murathan" // böyle bir fonksiyon var const olan


    vector<string>::const_iterator iter = svec.begin();
    ++iter; // legal
    *iter = "murathan" // syntax hatası

    auto iter = svec.begin(); // const değil
    iter = svec.cbegin(); // const

    auto iter = end(svec); // const değil
    iter = cend(svec); // const
}
```

Reverse Iterator

```
int main()
{
    vector<string> svec{  "ali", "can", "ece", "tan" };
    vector<string>::reverse_iterator iter = svec.begin();

    cout << *iter << "\n"; // tan
    cout << *iter++ << "\n" // ece

    auto it = iter.base(); // normal iter döndürür
}
```

Find

```
InIter Find(InIter beg, InIter end, const Key& val)
{
    while (beg != end)
    {
        if (*beg == val)
                return beg;
        ++beg;
    }

    return beg;
}

int main()
{
    using namespace std;

    vector<int> ivec{ 4, 6, 7, 8, 1, 3 , 98};

    int ival = 1;

    auto iter = Find(ivec.begin(), ivec.end(), ival);

    if (iter != ivec.end())
    {
        cout << "bulundu" << *iter << "\n";
        cout << "indeks = " << iter - ivec.begin() << "\n";
    }

    auto riter = Find(ivec.rbegin(), ivec.rend(), ival);

    cout << *iter << "\n" // 1 yazar
    cout << *iter.base() << "\n" // 3 yazar
}
```