2023 11 13

std::minmax_element

```cpp
// minmax_element
int main()
{
    using namespace std;

    vector<string> svec;
    rfill(svec, 20, rname);
    print(svec);

    pair<vector<string>::iterator, vector<string>::iterator> ret
        = minmax_element(svec.begin(), svec.end()); // std::pair döndür

    auto ip = minmax_element(svec.begin(), svec.end());

    cout << "min = " << *ip.first << '\n';
    cout << "max = " << *ip.second << '\n';
    // Structured binding
    auto [iter_min, iter_max] = minmax_element(svec.begin(), svec.end());
}
```

std::partition_point

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

    vector ivec{ 2, 4, 6, 8, 2, 20, 4, 3, 7, 9, 101};

    auto iter = partition_point(ivec.begin(), ivec.end(), [](int x )
    {
        return x % 2 == 0; // tekin başladığı yeri döndürür 3'ü
    });

    cout << *iter << '\n';
}
```

std:: is_sorted

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

    vector ivec { 2, 4, 6, 8, 20};

    auto b = is_sorted(ivec.begin(), ivec.end()); // b = true
}
```

std::is_sorted_until

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

    vector ivec { 22, 14, 6, 18, 1, -1};
    // büyükten küçüğe sıralamanın bozulduğu konumu verir. *iter = 18
    auto iter = is_sorted_until(ivec.begin(), ivec.end(), greater{});
}
```

heap veri yapısı

```
// heap veri yapısı
int main()
{
    using namespace std;
    vector<int> ivec;

    rfill(ivec, 20, IRand{ 0, 1000});

    make_heap(ivec.begin(), ivec.end()); // max heap
    make_heap(ivec.begin(), ivec.end(), greater{});  // min heap

    // heap'in en başındaki öğeyi en sona getirir çıkarabilmek için
    pop_heap(ivec.begin(), ivec.end());
    // heap olma özelliğini korurs
    ivec.pop_back(); // sondaki öğeyi çıkarır

    // heap'e veri ekleme
    ivec.push_back(9021);
    push_heap(ivec.begin(), ivec.end());

    while (!ivec.empty())
    {
        pop_heap(ivec.begin(), ivec.end());
        ivec.pop_back();
    }
}
```

std::queue

```
#include <queue>
int main()
{
    /*
        -kuyruğa farklı sırada elemanları ekledik
        -kuyruktan eleman çıkarıyoruz ama çıkan elemanlar en büyük değer

    */
    using namespace std;

    priority_queue<int> x;

    Irand myrand{ 0, 19887};

    for (int i = 0; i < 20; ++i)
    {
        int ival = myrand();
        x.push(ival);
        cout << ival << " kuyruga eklendi\n ";
    }

    while (!x.empty())
    {
        cout << x.top();
        x.pop();
    }
}
```

std::list

- Ortalardan ekleme ve silme işlemleride vector'e göre daha avantajlı
- Swap işlemlerinde ve swap işlemlerir yapan algoritmalarda vector'e göre daha avantajlı
- Çok büyük verilerde list vector'e göre daha avantajlı

uyarı: profiling yapmadan list mi vector mu seçilmeli kararı verilmemeli

bidirectional_iterator sahip bu yüzden:

- iki iteratorü büyüklük küçüklük karşılaştırması yapmak
- iki iteratorün farkını almak
- += -= gibi işlemler yapmak
- [] parantez işlemleri yapmak

bunlar syntax hatasıdır.

std::reverse, std::remove ve std::remove_if

```
int main()
{
    using namespace std;
    // eğer vector olsaydı algoritma kütüphanesini kullancaktık.
    list<string> mylist;

    rfill(mylist, 10, rname);
    print(mylist);

    mylist.reverse(); // listeye özel


    list<int> ilist;
    rfill(ilist, 100, Irand{0, 5});
    int ival = 3;

    auto n = ilist.remove(ival);

    list<int> ilist1;
    rfill(ilist1, 100, Irand{0, 5});

    auto n = ilist1.remove_if([ival](int x) {return x % ival == 0;});

}
```

std:: merge

```
int main()
{
    using namespace std;
    // karşılaştırma kriteri bilinmeli
    list<string> x;
    list<string> y;

    auto fgen = [] {return rname() + ' ' + rfname();};

    rfill(x, 20, fgen);
    rfill(y, 20, fgen);

    x.sort();
    y.sort();

    x.merge(y);

    cout << x.size() << "\n"; // 40;
    cout << y.size() << "\n"; // 0;
}
```

std::splice (bir listeden çıkartıp başka listeye ekler)

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

    list<string> f{ "esin", "ayse", "fatma", "gizem", "lale", "hulya"};
    list<string> m{ "fuat", "ceyhun", "tarik", "cetin", "kaan"};

    f.splice(f.begin(), m); // en başa m listesini ekledik
}
```

std::lower_bound, std:: upper_bound ve std::equal_range

```
/*

    4  6  6  7  7  7  9  12  23  34  45  90

    lower bound: verilen bir değer için öyle bir konum ki o değeri sırayı
    bozmadan ekleyebileceğimiz ilk konum
    (verilen değer 7 ise ekliyebileceğim indeks 3)

    upper bound: verilen bir değer için öyle bir konum ki o değeri sırayı
    bozmadan ekleyebileceğimiz son konum
    (verilen değer 7 ise ekliyebileceğim indeks 6)

    equal range: lower_bound ile upper_bound arasındaki range
    yani 7 değeri için 3 6 arası

*/
```

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

    vector<int> ivec;
    // karşılaştırma kriteri bilinmeli
    rfill(ivec, 20, Irand{ 0, 5});
    sort(ivec.begin(), ivec.end()), greater{});

    int key;
    cin >> key;

    auto iterlower = lower_bound(ivec.begin(), ivec.end(), key, greater{});
    cout << "ilk ekleme yapilabilecek yerin indeksi : " << distance(ivec.begin(),
iterlower) << '\n';
    auto iterupper = upper_bound(ivec.begin(), ivec.end(), key, greater{});
    cout << "son ekleme yapilabilecek yerin indeksi : " << distance(ivec.begin(),
iterupper) << '\n';

    auto [iterlower, iterupper] = equal_range(ivec.begin(), ivec.end(), key);

    cout << key << " degerinden " << distance(iterlower, iterupper) << " tane
var\n";

}
```

Mülakata sorusu: bir vektöre eleman eklicez ama sıralama bozulmıcak

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

    vector<string> svec;

    auto fgen = [] {return rname() + ' ' + rfname(); };

    for (int i = 0; i < 20; ++i)
    {
        auto name = fgen();
        std::cout << name  << "ekleniyor\n";
        auto iter = lower_bound(svec.begin(), svec.end(), name);
        svec.insert(iter, std::move(name));
    }
}
```

sorted range algorithms

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

    vector<string> x;
    vector<string> y;

    rfill(x, 30, rtown);
    rfill(y 40, rtown);

    sort(x.begin(), x.end());
    sort(y.begin(), y.end());

    // x ve y kesişimi
    list<string> slist_inter;
    set_intersection(x.begin(), x.end(), y.begin(), y.end(),
back_inserter(slist_inter));
    // x ve y birleşimi
    list<string> slist_union;
    set_union(x.begin(), x.end(), y.begin(), y.end(), back_inserter(slist_union));
    // x fark y
    list<string> slist_diff;
    set_union(x.begin(), x.end(), y.begin(), y.end(), back_inserter(slist_diff));

}
```