2023 11 15

forwad list (tekli bağlı liste) C++11

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

    // size fonksiyonu yok
    forwad_list<int> mylist{ 2, 3, 1, 2, 5, 12, 123, 11};

    copy(mylist.begin(), mylist.end(), ostream_iterator<int>{cout, " "});
    std::cout << "\n";

    // push_back fonksiyonu yok
    mylist.push_front(123);
    mylist.push_front(999);

    copy(mylist.begin(), mylist.end(), ostream_iterator<int>{cout, " "});
    std::cout << "\n";
    // pop_back fonksiyonu yok
    mylist.pop_front();


    auto iter = next(mylist.begin(), 3);
    cout << *iter << "\n";

    // iter konumundan bir sonraki konuma ekler
    mylist.insert_after(iter, 7777);

    // en başa ekler, before_begin() en baştaki öğeden önceki adresi verir
    mylist.insert_after(mylist.before_begin(), { -7, -8, 1, 3});
    // ilk 3 öğeyi siler
    mylist.erase_after(mylist.before_begin(), next(mylist.begin(), 3));
}
```

Container Adapters

- stack: (LIFO: Last in first out) son girenin ilk çıktğı bir veri yapısı
- queue: ilk girenin ilk çıktığı bir veri yapısı
- priority_queue: önceliği en yüksek olan ilk çıkar

std::stack

```
template <typename T, typename C = std::deque<T>>
class Stack
{
    public:
        void push(const T& val)
        {
            c.push_back(val);
        }
        T& top()
        {
            return c.back();
        }
        void pop()
        {
            c.pop_back();
        }
        bool empty()const
        {
            return c.empty();
        }
        auto size()const
        {
            return c.size();
        }
};
```

```
#include <stack>
// default template argumanı deque
int main()
{
    stack<int, deque<int>> mystack;  // stack<int>
    stack<int, vector<int>> mystack1;
}
```

```
// stack class'ını kalıtım yoluyla kullanabiliriz
class NecStack : public std:stack<int>
{

}
```

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

    stack<int> mystack;

    mystack.push(2);
    mystack.push(3);
    mystack.push(5);
    mystack.push(7);
    mystack.push(11);

    cout << "size = " << mystack.size() << "\n";
    cout << "mystack.top() = " << mystack.top() << "\n";

    while (!mystack.empty())
    {
        cotu << mystact.top() << "\n";
        mystack.pop(); // 11 7 5 3 2 diye çıkar
    }

    //syntax hatası
    stack<int> s{3, 13, 23, 12, 4}; // init list ctor ypk

    deque dx{ 3, 5, 7, 7};
    stack<int mystack { dx }; // geçerli
}
```

std:: queue

```cpp
// template argumanı deque
#include <queue>
int main()
{
    using namespace std;

    queue<string> names;

    names.push("melike");
    names.push("emre");
    names.push("tamer");
    names.push("furkan");
    names.push("selim");
    names.push("yasar    ");

    std::cout << "kuyrukta " << names.size() << " kisi var\n";
    std::cout << "kuyruk basi " << names.front() << "\n";
    std::cout << "kuyruk sonu " << names.back() << "\n";

    while (!names.empty())
    {
        cout << names.front() << "\n";
        names.pop(); // ilk giren ilk çıkar (melike , emre, tamer ...)
    }
}
```

std::priority_queue

```
// priority_queue
template<typename T, typename C = std::vector<T>, typename Comp =
std::less<typename C::value_type>>
class PrioritQueue{};
// template argumanı vector ve less (karşılaştırma kriteri)
int main()
{
    priority_queue<string> x;

    for (int i = 0; i < 10; ++i)
    {
        auto name = rname();
        x.push(name);
        cout << name << " eklendi\n";
    }

    // en büyükten küçüğe doğru çıkar
    while (!x.empty())
    {
        std::cout << x.top() << " kuyrak cikiyor\n";
        x.pop();
    }
}
```

```
// alias template
template <typename T>
using minpq = std::priority_queue<T, std::vector<T>, std:.greater<T>>;
```

Associative Container

- set
- multiset
- map
- multimap

std::set

```
// bidirectional_iterator
#include <set>
int main()
{
    set<int, less<int>, allocator<int>> myset; // set<int>

    set<int> s {3, 1, 4, 2, 5, 7};

    for (auto iter = s.begin(); iter != s.end(); ++iter)
        cout << *iter << '\n';
    for(int ival : s)
        cout << ival << '\n';
}
```

```cpp
int main()
{
    set<string> myset;
    for (int i = 0; i < 1000; ++i)
    {
        myset.insert(rname());
    }
    // size 1000'dan az olabilir çünkü bir değerden sadace 1 tane olur
    cout << "myset.size() = " << myset.size() << "\n";

    multiset<string> mymultiset;
    for (int i = 0; i < 1000; ++i)
    {
        mymultiset.insert(rname());
    }
    // size 1000 olacak. Multisette bir değerden birçok tane olabilir
    cout << "mymultiset.size() = " << mymultiset.size() << "\n";
}
```

Set Karşılaştırma Kriteri

```cpp
class scomp
{
    public:
        bool operator()(const std::string& s1, const std::string& s2)const
        {
            return s1.size() < s2.size() || (s1.size() == s2.size() && s1 < s2);
        }
};

int main()
{
    using namespace std;

    set<string, scomp> myset;

    for (int i = 0; i < 100; ++i)
    {
        myset.insert(rname());
    }

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

    // artk scomp'a göre karşılaştırması olacak
    for (const auto& s : myset)
    {
        std::cout << s << " ";
    }
}
```

```cpp
bool mycomp(const std::string& s1, const std::string& s2)
{
    return s1.size() < s2.size() || (s1.size() == s2.size() && s1 < s2);
}
int main()
{
    using namespace std;

    set<string, decltype(&mycomp)> myset(mycomp);

    for (int i = 0; i < 100; ++i)
    {
        myset.insert(rname());
    }

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

    // artk scomp'a göre karşılaştırması olacak
    for (const auto& s : myset)
    {
        std::cout << s << " ";
```

```cpp
int main()
{
    using namespace std;
    auto fcomp = [](const std::string& s1, const std::string& s2)
    {
        return s1.size() < s2.size() || (s1.size() == s2.size() && s1 < s2);
    };
    set<string, decltype(fcomp)> myset(fcomp);

    for (int i = 0; i < 100; ++i)
    {
        myset.insert(rname());
    }

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

    // artk scomp'a göre karşılaştırması olacak
    for (const auto& s : myset)
    {
        std::cout << s << " ";
    }
}
```

Strict Weak Ordering

Karşılaştırma fonksiyonu yazarken:

- a < b true ise b > a false olmalı (antisymmetric)
- a < a false olmalı (küçük eşittir olmaz) (irreflexive)
- a operator b true ve b operator c true ise a operator c true olmalı (transitive)

- !(a < b) && !(b < a) true ise ve !(b < c) && !(c < b) true ise
- !(a < c) && !(c < a) true olmalı (transitivity of equivalence)

set.insert()

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

    set<string> myset;
    vector<string> svec { "derya", "ceyhun", "nalan", "tekin" };

    myset.insert("ayse")
    myset.insert({"ali", "zeki", "nuri", "derya"});
    myset.insert(svec.begin(), svec.end());

    cout << "eklenecek isim girin : " ;
    string name{};

    cin >> name;

    //pair<set<string>::iterator, bool> p = myset.insert(name);
    auto p = myset.insert(name);
    // sette varsa bool false döner iterator var olan yeri döner

    if (p.second)
    {
        std::cout << "ekleme yapildi... \n";
        cout << *p.first << "\n";
    }
    else
    {
        std::cout << name << "sette var\n";
        cout << *p.first << "\n";

    }

}
```

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

    set<string> myset;

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

    cout << "aranacak isim: "
    string name{};
    cin >> name;
    // logaritmik karmaşıklıkta
    if (auto iter = myset.find(name); iter != myset.end())
    {
        std::cout << "bulundu..." << *iter << "\n";
    }
    else
    {
        std::cout << "bulunamadı..." << *iter << "\n";
    }
    // iterator kullanmayacaksak böyle kullanabiliriz
    if (myset.count(name))
    {
        std::cout << "bulundu...\n";
    }

    if (myset.contains(name)) // cpp 20 de geldi
    {

    }
}
```