* Deque Kullanım Senaryoları:

→ Constant time'da hem sona hem başa erişim + indexle erişim

→ Bellek kullanımı. vector → tek bir bellek bölgesinde

Deque → chunk halinde tutulur. (Eğer vektör için yer yoksa, chunk chunk daha mantıklı)

→ Bool vektör tutmak istiyorsa, Fakat STL'deki bool vektörü bir portal specialization. Bit seviyesinde tutuluyor. Eğer amacımız boolları tutmaksa → deque<bool> tercih edilebilir.

→ Deque iterator invalidation kuralları, vektörünkinden farklı. Avantaj olabilir.

→ Random Access iterator'e sahip. Random Access iterator gerekiyorsa algoritmalara deque rahatça geçilebilir.

* Deque İterator Invalidation:

↪ Baştan / sondan silme yapıyorsak, sadece silinen öğeye ilişkin iterator / referanslar invalid olur

↪ Baştan / sondan ekleme yapıyorsak, referanslar valid diğer iteratorler invalid.

↪ Ortadan ekleme yapıyorsak, referanslar ve iteratorler invalid (hepsi)

- deque : An insertion in the middle of the deque invalidates all the iterators and references to elements of the deque. An insertion at either end of the deque invalidates all the iterators to the deque, but has no effect on the validity of references to elements of the deque. [26.3.8.4/1]

- deque : An erase operation that erases the last element of a deque invalidates only the past-the-end iterator and all iterators and references to the erased elements. An erase operation that erases the first element of a deque but not the last element invalidates only iterators and references to the erased elements. An erase operation that erases neither the first element nor the last element of a deque invalidates the past-the-end iterator and all iterators and references to all the elements of the deque . [Note: pop_front and pop_back are erase operations. —end note] [26.3.8.4/4]

* Deque Dezavantajları:

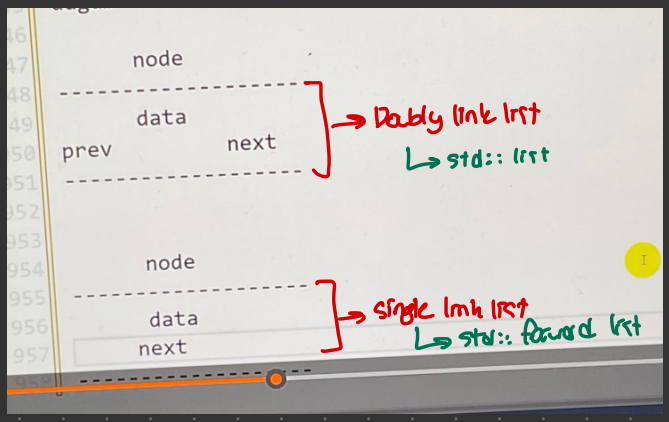
↪ Contiguous Değil !! , C API'lerine argüman olarak gönderilemez. (Vektör'e 1 pointer 1 size fonksiyon api'ye gönderebiliriz.)

↪ Deque'te hem chunklar için, hem de chunkların başlangıç adreslerinin pointerları için yer tutulur.

```
using namespace std;

deque<Date> dc;

for (int i = 0; i < 5'000; ++i) {
    auto mydate = Date::random();
    //...
    if (mydate.month_day() % 2 == 0) {
        dc.push_front(mydate);
    }
    else {
        dc.push_back(mydate);
    }
}

//
for (size_t idx = 0; idx < dc.size(); ++idx) {
    cout << dc[idx] << "\n";
}
```

* List: → Konumunu bildiğimiz bir yere ekleme yapma / silme → Constant time

→ STL List: Doubly link list.



→ Bellek kullanımı açısından dezavantaj var X.

Çünkü her veri (eklenen eleman için) bir node oluşturmak gerekir. → data
→ prev pointer = int için 12 byte
→ next pointer alloc.

→ Ayrıca cache friendly değil !! Çünkü contiguous değil bir.

→ İndex ile erişimde linear time erişim.

* Vector'e benzer. swap işleminde direkt datalar değil, node'ların pointerları swap edilir (swap işlemi, sort, reverse, unique işlemlerinde merge, splice)

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

    vector ivec{ 1, 4, 5, 7, 9, 12 };

    list<int> mylist{ ivec.begin(), ivec.end() };
}
```

vector içeri, list'e eklidi.

→ Constructor grubu vector / deque ile aynı.

→ unique, aynı değerdeki ardışık olanları teke indiriyor.

```
{
    using namespace std;

    list<string> slist;
    rfill(slist, 200, rname);

    slist.unique([](const string& s1, const string& s2) {return s1.length() == s2.length(); });

    print(slist);
}
```

→ uzunluğa göre unique

→ Önce sort + sonra unique = Her özeden 1 adet

* remove_if:

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

    list<string> slist;

    rfill(slist, 1000, rname);

    auto n = slist.remove_if([](const string& s) {return s.length() > 3; });


}
```

* Splice:

- Node rangi listeye ait olduğunu bilmez. Bir bir node'u başka bir list'in konumuna ekler.

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

    list<string> male_list{ "ahmet", "selami", "hakki", "recep", "binali", "bilal", "suleyman" };
    list<string> female_list{ "lale", "nurdan", "suheyla", "aleyna", "tijen", "nuriye", "huriye" };


    male_list.splice(next(male_list.begin()), female_list);
    print(male_list)

}
```

selamin konumu

→ splice'in overloadlerine bak.

```
ahmet lale nurdan suheyla aleyna tijen nuriye huriye selami hakki recep binali bilal suleyman
---------------------------------------------------------------------------

D:\CONCURRENCY\PACA_2022\Debug\PACA_2022.exe (process 33380) exited with code 0.
Press any key to close this window . . .
```

female list eklendi

* Forward list: → singly linked list (slist)

Modern c++ ile geldi.

- Iterator konumuna insert / delete yapılamaz. Bir sonraki node için yapılabilir. → insert after
- Sınıfın hem begin hem de before begin var. → erase after

↓

başa birden fazla öğe eklemek için.

- size fonksiyonu yok. Size elde etmek için distance kullan

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

    forward_list<string> mylist{ "eray", "musa", "samet" };
    print(mylist);

    cout << distance(mylist.begin(), mylist.end()) << "\n";

}
```

- push_front ile başa eklenir.

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

    forward_list<string> mylist{ "eray", "musa", "samet", "emre", "cem"};
    print(mylist);

    mylist.insert_after(mylist.begin(), 5, "recep");


}
```

→ eraydan sonra 3 adet recep ekledi

* Associative Containers:

→ key: aranacak / eklenecek değer.

→ Veri yapısındaki aramayı O(log N)'e çekmek (örneğin 1024 öğe varsa 10 sorgulda)

→ Arka planda red black binary search tree var.

- Bu containerlar: set → sette container'ın değeri unique → ekleme/silme/arama key değerine göre
  multiset → multisette aynı değerse sahip birden fazla container,
  map → bunlarda her containera karşılık bir değer var → pair first = key, pair second = value
  multimap → ekleme çıkarma silme yine containera göre

```
==========

equality  (eşitlik)

equivalance  (eş değerlik)

a  ==  b → equality

!(a < b) &&  !(b < a); → equivalence
```

→ set ve multisette 2. template parametresi bizim comparator'ımız. Yani less, greater.

→ Default olarak less, ama greate girilirse büyükten küçüğe dizilir.

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

    set<int> myset;

    Irand myrand{ 0, 100 };

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

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

set olduğu için 100 değer olma garantisi yok. Çünkü aynı değeri 2 kez insert olmaz.

```cpp
struct Comp {

    bool operator()(const std::string& r1, const std::string& r2)const
    {
        return r1.size() < r2.size();
    }
};


int main()
{
    using namespace std;

    set<string, Comp> myset;

    rfill(myset, 5, rname);
    print(myset);


}
```

Kendi functor classımızı yazdık.

string size'ı küçükten büyüğe sıraladı.

```
efe nuri selin handan cebrail

D:\CONCURRENCY\PACA_2022\Debug\PACA_2022.exe (process 27348) exited with co
Press any key to close this window . . .
```

→ Fakat lambda expression ogrendigimiz interesydik

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

    auto fcomp = [](const std::string& r1, const std::string& r2)
    {
        return r1.size() < r2.size();
    };

    set<string, decltype(fcomp)> myset(fcomp);

    rfill(myset, 5, rname);
    print(myset);

}
```

decltype operatörü

bu ifade evaluate.

2020'den önce olusan bu closer type'ın default constructor'ı olmadığı için böyle yazıldı

* Eğer bir comparator ifadesini yazıyorsak: → Strict weak ordering'e dikkat edilmeli

1. anti-simetrik (antisymmetric).
x < y doğru ise y < x yanlış olmalı

op(x,y) doğru ise op(y,x) yanlış olmalı

2. geçişken (transitive).
x < y doğru ve y < z doğru ise bu durumda x < z doğru olmalı
op(x,y) doğru ve op(y,z) doğru ise op(x,z) doğru olmalı

3. irreflexive
x < x her zaman yanlış olmalı
op(x,x) her zaman yanlış olmalı

4. eşitliğin geçişkenliği (transitivity of equivalence)
!(a<b) && !(b<a) doğru ve !(b<c) && !(c<b) doğru ise bu durumda !(a<c) && !(c<a) doğru olmalı