

* String Sınıfı:

- hem vektor, hem string abstrak structure olarak dynamic arraydır.

→ Dynamic Array:

- Tutaran üye sayısı → size
 - Dizin tutabilecek üye sayısı → capacity
- Size = capacity olduğunda, yeni bir insertionda, realloc edilir. (maliyetli)
Capacity artışı compiler'a bağlı ama reserved bir time ile yapılır.

* realloc takes time

↳ realloc invalidates pointers and refs. → yeni adrese atıldığında değişim pointer riskli.

- Üyeler contiguous → aynı ortama dizi → index'i bilmen'e erişim constant time.
→ sondan okunur

- String aslında, tür esr. Bir class template (generic)

```
using string = std::basic_string<char, std::char_traits<char>, std::allocator<char>>;
```

```
int main()  
{  
}
```

↓
tür

↓
trait

↓
resme nasıl
yapılır

- String'in her durumu kapsayan member function'ları var. → Container'lar var.

↳ Gevretli algoritmalar, B.M.: string reverse

```
sequential containers  
=====
```

```
std::vector  
std::deque  
std::list  
std::forward_list  
std::array  
std::string
```

```
associative containers  
=====
```

```
std::set  
std::multiset  
std::map  
std::multimap
```

```
unordered associative containers  
=====
```

```
std::unordered_set  
std::unordered_multiset  
std::unordered_map  
std::unordered_multimap
```

- Bol bol function ve operator overloading var. + toplama

↳ sondan okunma gibi

- Cstring → sonunda null karakter olan bir yığın adresi "mer + " gibi
- Cstring parametreleri fonksiyonlar `const char *` argüman alır. Fakat sonunda NULL karakter olmama gerekir. → Bütün programcıların

```
int main()
{
    char str[100];

    str[0] = 'O';
    str[1] = 'K';

    str
```

* str burada null terminated byte stream. Değil!
(Cstring)

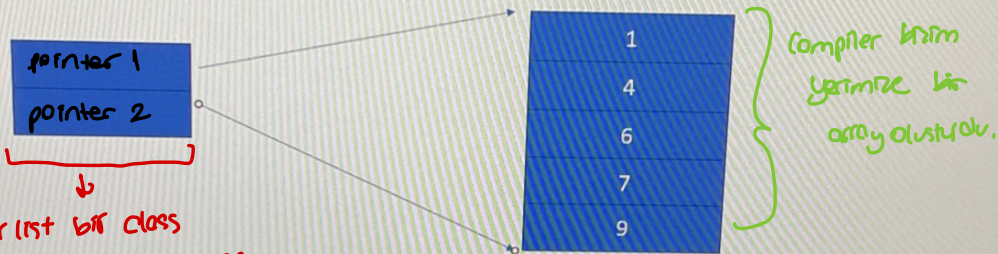
→ Çünkü son karakter null değil, ve cstring ile ilgili için tehlikeli durum.

std::string Constructorları (Parametreleri):

```
=====
string          const std::string &
string          std::string && → tasama semantiği kullanılır.
cstring parametre const char * → null char belirtir
data            const char *, string::size_type n → tasama hatası önümüzü durdurur,
range           [konum1 konum2)
fill            string::size_type, char
char            char
substring       const string&str, string::size_type idx
substring       const string&str, string::size_type idx, string::size_type n
initializer_list std::initializer_list
```

• initializer_list Nedir?

std::initializer_list<int> x{ 1, 4, 6, 7, 9 }; → int değil / farklı değil / da değil.



↓
Initializer list bir class
Bu class iki adet pointer içerir.
Array başlangıç adresi ve bitiş adresini tutar.

- Sol + okuma amaçlı bir liste oluştur.


```

1 #include <string>
2 #include <iostream>
3 #include <initializer_list>
4
5 int main()
6 {
7     std::initializer_list<int> x{ 1, 2, 3, 4, 5 };
8
9     std::cout << "sizeof(x) = " << sizeof(x) << "\n";
10    std::cout << "sizeof(int*) = " << sizeof(int*) << "\n";
11
12 }

```

Init. list'in size'i 2 adet pointer kodudur.

$\rightarrow 4 \times 2 = 8$

```

1 #include <string>
2 #include <iostream>
3 #include <initializer_list>
4
5 int main()
6 {
7     std::initializer_list<int> mylist{1, 2, 3, 4, 5};
8
9     std::cout << "mylist.size() = " << mylist.size() << "\n";
10 }

```

size kavramı sadece elementlerin sayısını döndürür.

- **Begin ve End** fonksiyonları da, başlangıç ve bitiş adreslerini tutan pointer'i döndürür.

```

1 #include <string>
2 #include <iostream>
3 #include <initializer_list>
4
5 int main()
6 {
7     std::initializer_list<int> mylist{1, 2, 3, 4, 5, 9, 2};
8
9     auto p1 = mylist.begin();
10    auto p2 = mylist.end();
11
12 }

```

const int *

- **Range based for loop** kullanılır.

```

1 #include <string>
2 #include <iostream>
3 #include <initializer_list>
4
5 using namespace std;
6
7 int main()
8 {
9     std::initializer_list<int> mylist{1, 2, 3, 4, 5, 9, 2};
10
11    for (auto i : mylist) {
12        std::cout << i << "\n";
13    }
14 }

```

range based for loop

• Cpp 17'de **type deduction** kullanılarak oluşturma eklendi

```
1 #include <string>
2 #include <iostream>
3 #include <initializer_list>
4
5 using namespace std;
6
7
8
9
10
11 int main()
12 {
13     auto x = { 1, 3, 5, 6, 7 };
14     //std::initializer_list<int> x = { 1, 3, 5, 6, 7 };
15 }
```

"=" token'i kullanılmak ZORUNDA

öğelerin hepsi aynı türden olmalı.

* Fakat Bir class'ın initializer listli constructor'ı varsa, Darenin kullanılgı geçer !

- Vektor ve string bunlara dahildir.

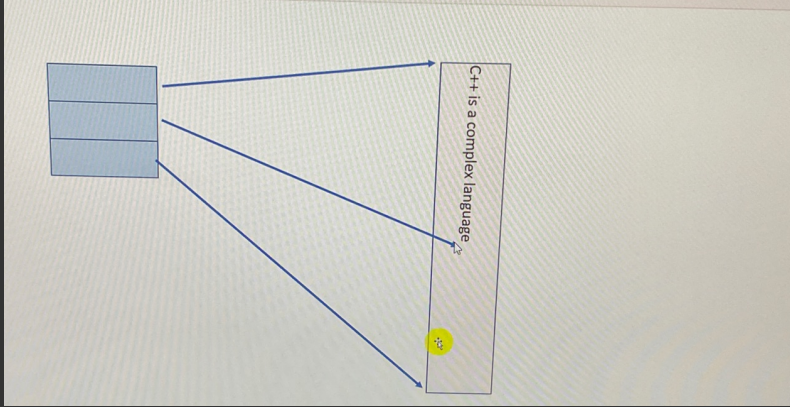
```
7 class MyClass {
8 public:
9     MyClass(std::initializer_list<int> x)
10     {
11         std::cout << "Myclass(init_list)\n";
12     }
13
14     MyClass(int)
15     {
16         std::cout << "Myclass(int)\n";
17     }
18 };
19
20
21 int main()
22 {
23     MyClass m1 = { 1, 2, 3, 4, 5 }; → Init list
24     MyClass m2{ 10 }; → * Bu da init list !! Bir öğe bile olsa geçer.
25     MyClass m3(10); → int
26 }
```

```
int main()
{
    string s1(55, 'A'); → 55 tane A yazdır, Fill constructor
    std::cout << "s1.size() = " << s1.size() << "\n"; → 55
    std::cout << "[" << s1 << "]" \n"; → [ AAA.....A ]
    string s2{ 55, 'A' }; → Init list ile çağır. 55'in ASCII değeri ve A ile üretilir.
    std::cout << "s1.size() = " << s1.size() << "\n"; → 2
    std::cout << "[" << s1 << "]" \n"; → [AA]
```

- Small String Optimization:

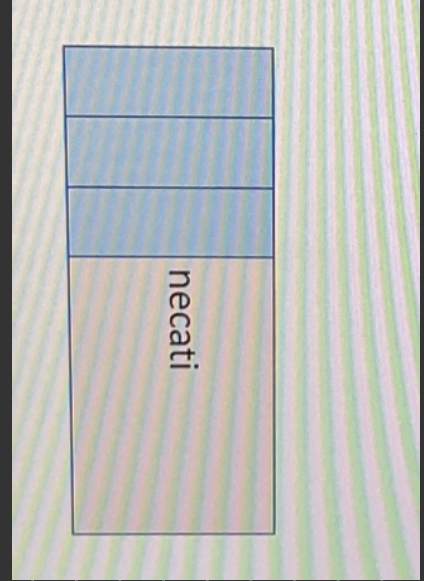
- String sınıfı genelde 3 pointerdan oluşur. → Dinamik bellek alanının başlangıcı
- String'in sonu (ekleme yapılabilecek yer)
- Kapasite olarak tutulan, son bellek bilgisi

```
int main()
{
    string str{"C++ is a complex language"}
}
```



modern implementasyonlar

class içinde, bellek sınıfı
içerisinde bir buffer var.



- Küçük bir yazı, string nesnesi içindeki bufforda tutulur.
- Yazı büyük olduğunda, new ile yeni bellek alanı tahsis ediliyor.
- sizeof(std::string) = 24.
→ 3 pointer + buffer.