* String Sınıfı:

• hem vector, hem string data structure olarak dynamic array'dir.

→ Dynamic Array:

• Tutulan öğe sayısı → size
• Dizinin tutabileceği öğe sayısı → capacity

→ Size = capacity olduğunda, yeni bir insertionda, realloc edilir. (maliyetli)
→ Capacity artışı compiler'a bağlı ama reserve bir func ile ayarlanır.

* realloc takes time
* realloc invalidates pointers and refs → yeni adrese atandığında dangling pointer riski

• Öğeler contiguous → arka arkaya dizili → index'i bilinen'e erişim constant time
→ sondan ekleme

• String aslında tür eş ismi. Bir class template (generic)

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

int main()
{
}
```

↓ tür — ↓ trait — ↓ bellek nasıl yapılır

• String'in her durumu kapsayan member function'ı gör. → Container'lar var.
↳ Çeşitli algoritmalar, örn/: 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 ekleme gibi

• C string → sonunda null karakter olan bir yazının adresi. "mert" gibi

• C string parametreler, fonksiyonlar const char * argüman alır. Fakat sonunda NULL karakter olmasına güvenir. → Bizim sorumluluğumuzda

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

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

    str
```

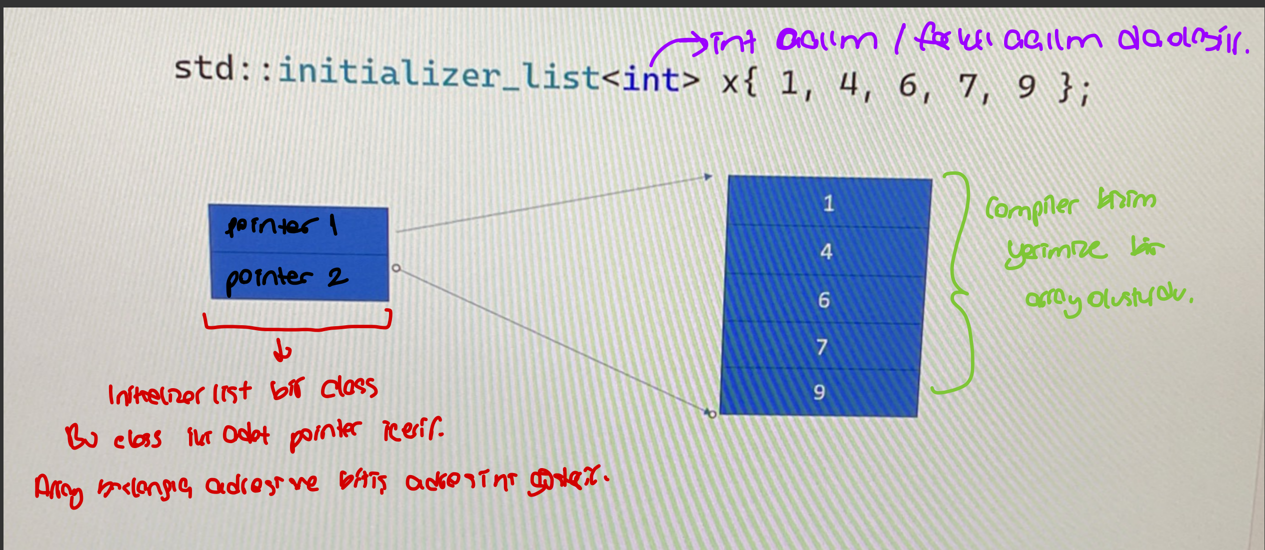
* str burada null terminated byte stream Değil! (C string)

→ Çünkü son karakter null değil, ve cstring isteyen fonc için tanımsız davranış

std::string Constructor'ları (Parametreleri):

string	const std::string &
string	std::string && → taşıma semantiği kullanır.
cstring parametre	const char * → null char karakteri
data	const char *, string::size_type n → taşma hatası tanımsız davranış.
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?



• Salt okuma amaçlı bir liste oluşturur.

```cpp
#include <string>
#include <iostream>
#include <initializer_list>

int main()
{
    std::initializer_list<int> x{ 1, 2, 3, 4, 5 };

    std::cout << "sizeof(x) = " << sizeof(x) << "\n";
    std::cout << "sizeof(int*) = " << sizeof(int*) << "\n";

}
```

init. list'in size'ı 2 adet pointer kadardır.

4x2 = 8

```cpp
#include <string>
#include <iostream>
#include <initializer_list>

int main()
{
    std::initializer_list<int> mylist{1, 2, 3, 4, 5};

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

size komutu initializer elemanların sayısını döndürür.

- begin ve end fonksiyonları da, başlangıç ve bitiş adreslerini tutan pointerı döndürür.

```cpp
#include <string>
#include <iostream>
#include <initializer_list>

int main()
{
    std::initializer_list<int> mylist{1, 2, 3, 4, 5, 9, 2};

    auto p1 = mylist.begin();
    auto p2 = mylist.end();


}
```

const int *

- Range based for loop'la kullanılabilir.

```cpp
#include <string>
#include <iostream>
#include <initializer_list>

using namespace std;

int main()
{
    std::initializer_list<int> mylist{1, 2, 3, 4, 5, 9, 2};

    for (auto i : mylist) {
        std::cout << i << "\n";
    }
```

range based for loop

• Cpp 17'de type deduction kullanarak oluşturma eklendi

```cpp
#include <string>
#include <iostream>
#include <initializer_list>

using namespace std;

int main()
{
    auto x = { 1, 3, 5, 6, 7 };
    //std::initializer_list<int> x = { 1, 3, 5, 6, 7 };
}
```

"=" token'i kullanılmak ZORUNDA

öğelerin hepsi aynı türden olmalı.

* Eğer bir class'ın initializer list'li constructor'ı varsa parantezin kıymığı kaçar!

- Vektör ve string bunlara dahildir.

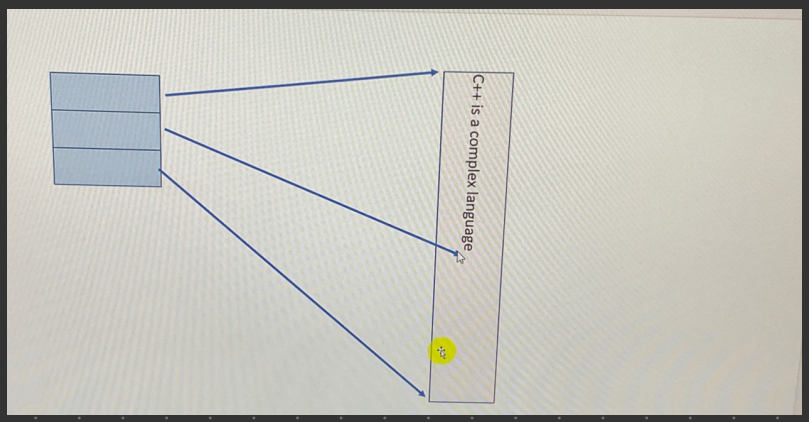
```cpp
class Myclass {
public:
    Myclass(std::initializer_list<int> x)
    {
        std::cout << "Myclass(init_list)\n";
    }

    Myclass(int)
    {
        std::cout << "Myclass(int)\n";
    }
};

int main()
{
    Myclass m1 = { 1, 2, 3, 4, 5 }; → init list
    Myclass m2{ 10}; → * Bu da init list!! Bir öge bile olsa geçerli.
    Myclass m3(10); → int
}
```

```cpp
int main()
{
    string s1(55, 'A'); → 55 tane A yazar, Fill constructor

    std::cout << "s1.size() = " << s1.size() << "\n"; → 55
    std::cout << "[" << s1 << "]\n"; → [AAA..... A]
                                        55 tane

    string s2{ 55, 'A' }; → init list ile çağırır. 55'in ASCII değeri ve A'ne yaratılır.

    std::cout << "s1.size() = " << s1.size() << "\n"; → 2
    std::cout << "[" << s1 << "]\n"; [7A]
```

- Small String Optimization:

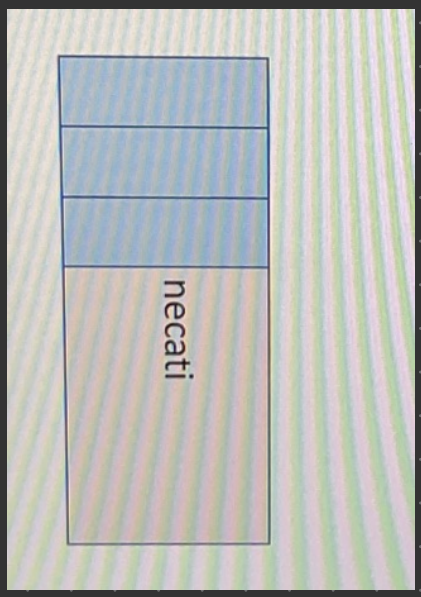
- String sınıfı genelde 3 pointerdan oluşur. → Dinamik bellek alanının başlangıcı
→ String'in sonu (ekleme yapılacak yer)
→ Kapasite olarak tutulan , son bellek bilgisi

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



modern implementasyonlar

class içinde küçük stringi tutabilen bir buffer var.



- Küçük bir yazı string nesnesi içindeki buffer'da tutulur.
- Yazı boyutu artınca, new ile yeni bellek alanı tahsis edilir.
- sizeof(str:string) = 24
↳ 3 pointer + buffer.