

* Son boldi: → endianness bilen had

```
int main()
{
    unsigned int n = 1;

    if (*(char *) &n)
    {
        std::cout << "LITTLE ENDIAN" << '\n';
    }
    else
    {
        std::cout << "BIG ENDIAN" << '\n';
    }
}
```

Char tarzı yapılan cast yöntemleri

Aynı tür sağa taranın işaretli ve işaretçisi arasında
Struct'in ilk elementin tarzı CAST SÖBÜLTÜRÜZ.

AŞAĞI TANIMSLI DAVRANISI

```
3 struct Data {
4     int a, b, c;
5 };
6
7 int main()
8 {
9     Data mydata{ 12, 56, 90 };
10
11     int* p = (int*)&mydata;
12
13 }
```

Struct'in en başında osla padding olmaz!

* Cooked İndeksine Deverim:

```
1 Date operator""_dt(const char*p, size_t size)
2 {
3     std::cout << p << '\n';
4     std::cout << "size = " << size << '\n';
5
6     return {};
7 }
8
9 int main()
10 {
11     auto Date x = "12-05-1987"_dt;
12 }
```

String'in adresi

size=10

```
1 #include <iostream>
2 #include "date.h"
3 #include "nutility.h"
4
5 constexpr int operator""_i(char c)
6 {
7     return static_cast<int>(c);
8 }
9
10 int main()
11 {
12     constexpr auto ival = 'A'_i;
13 }
```

artık tarz int

-i amasya c'de tarz int

c'de tarz int

çap'de tarz char

```

#include <iostream>
#include "date.h"
#include "nutility.h"
#include <string>

std::string operator""_s2(const char* ptr, std::size_t) {
    return std::string{ptr} + std::string{ptr};
}

int main()
{
    using namespace std;

    cout << "alican"s << "\n";
    cout << "alican"_s2 << "\n";
}

```

*cokeci
en çok
üçük
gelişti
size
array*

* Uncaught Functions: → size_t parametresi yok.

```

int operator""_b(const char* p)
{
    int result = 0; → gerekli size_t versi null terminated byte string, while'in içindeki ifade  
null character'a gelene kadar işlenir!
    while (*p) {
        if (*p != '0' && *p != '1') → eğer dolayısıyla throw except!
            throw(std::runtime_error{ "invalid binary character!\n" });
        result = result * 2 + (*p - '0');
        ++p;
    }
    return result;
}

```

*trıple return yapmadık!
Direkt null terminated
byte string gönderiliyor.*

* Matematik Kodu: → Hex sayının decimal değer.

```

int main()
{
    char str[] = "AC4201";

    int result = 0;

    for (int i = 0; str[i] != '\0'; ++i) {
        if (isdigit(str[i])) {
            result = result * 16 + str[i] - '0';
        } sayılar için
        else if (isxdigit(str[i])) {
            result = result * 16 + toupper(str[i]) - 'A' + 10;
        }
    }
}

```

A - F için

* String View Sınıfı:

• C++ 17'de eklendi.

→ Model kütüphanesi genelde, stringview kullanılarak olur.

```
void func(const std::string& str)
{
}

int main()
{
    func("ali buraya gelsin hemen....")
}
```

String View ile çalışmaya

Burada func parametresi bir string nesnesine referans olur.

→ Ümeyen, çok önemli bir string'in substringini oluşturmak isterse, substring size(), kader ver kopgebilir

```
class StringView {
public:
    std::size_t length() const;
private:
    const char* ps;
    std::size_t len;
};
```

→ Böyle bir yapı oluşturduktan sonra bir kullanımcı, gerekiz kopgebilirken onu görür. (Sıkıcıdır amaçlı.)

→ Buradaki sıkıntısı, biz 0 argument olan string nesnesinin sonhar değiliz! Nesne değişik edildiğinde fizikselde olsalar da, stringview'ın onun hedefinin kılıp kalmadığını bilmemizdir.

```
void func(std::string_view x)
```

x.

→ String sınıfının interface'ine sahiptir. Sadece yorumları yok!

→ Dergileyice göre değişebilir. Anıksızaof.c t

```
#include <string>
#include <string_view>

int main()
{
    using namespace std;

    char sa[] = "Hasan Kaynar";
    string_view sv = sa;

    cout << sv << "\n";
    cout << "uzunluk = " << sv.length() << "\n";
    cout << sv.back() << "\n";
}
```

12
↳ r harfi

```
using namespace std;
char sa[] = "Hasan Kaynar";

string_view sv = sa;

cout << sv << "\n";
cout << "uzunluk = " << sv.length() << "\n";
cout << sv.back() << "\n";

sa[0] = 'Y';
sa[2] = 'M';

cout << sv << "\n";
```

Bu kodda Hasan yazına ısrar edildi. Fakat, eğer character yazımı string olسا ve string reallocation'a gitse → dangling pointer!

```
f(); → notify.h'da file açılır.
using namespace std;

string str{ "eray goksu" };
string_view sv{ str };

str += " bu akşam C++ dersine katılıyor";
cout << str << '\n';

cout << "sv = " << sv << "\n";
```

reallocation, dangling pointer olabilir.

```
1 eray goksu bu akşam C++ dersine katılıyor
2 sv = ENÜΦ goksu I
```

*Stringview Constructors:

```
char str[] = "Bjarne Stroustrup";
std::string s{ "cpp is the best programming language" };
std::array<char, 6> ar{ 'n', 'e', 'c', 'a', 't', 'i' };
```

→ Stringview constructor'ına gelebilecek argumentolar

```
3 std::string_view sv1;
4
5
6 std::cout << "(sv1.length : " << sv1.length() << ") |" << sv1 << "|\\n";
7
8 std::string_view sv2 = "Necati Ergin";
9
10 std::cout << "(sv2.length : " << sv2.length() << ") |" << sv2 << "|\\n";
11
12 std::string_view sv3{ str, 6 };
13 std::cout << "(sv3.length : " << sv3.length() << ") |" << sv3 << "|\\n";
14
15
16 std::string_view sv4{ s };
17 std::cout << "(sv4.length : " << sv4.length() << ") |" << sv4 << "|\\n";
18
19 std::string_view sv5{ s.data() + 11, 4 }; yeni adres
20 std::cout << "(sv5.length : " << sv5.length() << ") |" << sv5 << "|\\n";
21
22 std::string_view sv6{ ar.data(), size(ar) };
23 std::cout << "(sv6.length : " << sv6.length() << ") |" << sv6 << "|\\n";
24 std::string_view sv7{ str, str + 3 };
25 std::cout << "(sv7.length : " << sv7.length() << ") |" << sv7 << "|\\n";
```

```
C:\nec>a
(sv1.length : 0) ||
(sv2.length : 12) |Necati Ergin|
(sv3.length : 12) |Bjarne|
(sv4.length : 36) |cpp is the best programming language|
(sv5.length : 4) |best|
(sv6.length : 6) |necati|
(sv7.length : 3) |Bja|
```

Output

*Empty Stringview:

```
1 int main()
2 {
3     using namespace std;
4
5     f();
6
7     string_view sv;
8
9     cout << sv.length() << '\\n';
10    cout << boolalpha << sv.empty() << '\\n';
11    cout << boolalpha << (sv.data() == nullptr) << '\\n';
12 }
```

empty true döndürür
data ise nullptr

+ mavi et srasu:

```
#include <iostream>
#include <string_view>

int main()
{
    char s[] { 'n', 'e', 'c', 'o' };

    std::string_view sv{ s, 4 };

    std::cout << sv << '\n'; //no problem

    std::cout << sv.data() << '\n'; //ub
```

null character
yaz

string view nessi, null character olmadan da standard cikis okunma string bicer satir 10 degr.

Ancak data, adres dondurulce, null character'da kader ki string yaziilmak zornde null char olmadigi icin satir 13 => undefined behavior!

```
int main()
{
    char s[] { 'n', 'e', 'c', 'o', 'm', '\0' };
    std::string_view sv{ s, 4 }; I Uchar ile const-ot-edirgi
    std::cout << sv << '\n'; //no problem sinde yaz

    std::cout << sv.data() << '\n'; //ub data adres yodgumuz
    //std::cout << s << '\n'; //ub ren "herem" yaz!
}
```

* Returning A Stringview: • Erisili !
 • Dangling Pointer Olusuma, Tidimeli Golu Yolcul !

```
auto foo()
{
    char str[] = "sofiya oznacar"; I foo funksiyundan
    serra moyet bilcel
    ///
    std::string_view sv{ str };

    return sv; I return str ile dyn!
}

int main()
{
    auto sv = foo(); I moyet bimsis bir char[]'in odasini
                    tutuyu!
}
```

Eger static oluydu dangling pointer olmazdi !

```

int main()
{
    using namespace std;

    string_view sv{ "mustafa" }; → legal

    //string name = "ali"; → legal
    string name = sv; → constructor explicit exchange
                           sentels return
}

```

```

int main()
{
    using namespace std;

    string_view sv{ "mustafa" };

    //string name = "ali";
    string name{ sv }; → legal almost won
                           direct initialization
                           optimal
}

```

```

#include <iostream>
#include <string_view>

std::string foo()
{
    return std::string_view{};
}

```

Ayni şekilde string return eden
 farklı gün return değer stringviews
 olmaz X
 → une syntax hatalı

*Remove Prefix / Remove Suffix:

```

int main()
{
    std::string_view sw{ "The good the bad and the ugly" };

    std::cout << sw.length() << "\n";
    std::cout << "sw = " << sw << "\n";

    sw.remove_prefix(8);

    std::cout << sw.length() << "\n";
    std::cout << "sw = " << sw << "\n";
}

```

C:\nec>a
 29
 SW = The good the bad and the ugly
 21
 SW = the bad and the ugly
 C:\nec>

```

int main()
{
    std::string_view sw{ "The good the bad and the ugly" };

    std::cout << sw.length() << "\n";
    std::cout << "sw = " << sw << "\n";

    sw.remove_suffix(5);

    std::cout << sw.length() << "\n";
    std::cout << "sw = " << sw << "\n";
}

```

C:\nec>a
 nd 29
 nn SW = The good the bad and the ugly
 24
 ou SW = The good the bad and the
 C:\nec>

* Starts with / Ends with:

```
#include <iostream>
#include <string_view>

using namespace std;

int main()
{
    string_view str{ "life is very short" };

    std::cout << boolalpha << str.starts_with("life") << "\n";
    std::cout << boolalpha << str.starts_with("time") << "\n";
    std::cout << boolalpha << str.ends_with("short") << "\n";
    std::cout << boolalpha << str.ends_with("long") << "\n";
}
```

→ Stringde olduğu gibi

* Mete Onay:

```
#include <iostream>
#include <string_view>

using namespace std;

std::string_view get_reverse(std::string str)
{
    reverse(begin(str), end(str));
    return str;
}

int main()
{
    auto s = get_reverse("necati ergin");
    std::cout << s << "\n";
}
```

→ Geçerli string nesnesi, orgünen de olunmuş ve stringview yine return edilmiştir.

→ DANGEROUS POINTER USE!

```
int main()
{
    std::string str{ "necati ergin" };
    std::string_view sw{ str };
    std::cout << "[" << sw << "] \n";
    str.front() = 'X';
    str.back() = 'X';
    std::cout << "[" << sw << "] \n";

    str.append(500, 'A'); → Bu adıda reallocation olacak ve dangling hole
    std::cout << "[" << sw << "] \n";                                olacak
}                                         (Stringview etsi pointerdir) tutar
```

```
std::string getstr(int ival)
{
    return std::to_string(ival);
}

int main()
{
    int ival;
    std::cout << "bir tamsayı girin: ";
    std::cin >> ival;
    //auto &s1 = getstr(ival); //gecersiz → sol taraf refereksi, sağ taraf static oluyor!
    const auto& s2 = getstr(ival); //life extension → const & value ref
    auto&& s3 = getstr(ival); //life extension → universal reference → life time extend
    const char* p1 = getstr(ival).c_str(); //ub
    auto p2 = getstr(ival).c_str(); //ub
    std::string_view s = getstr(ival); //ub
    std::cout << "(" << s << ") \n";
}
```

cm
char

sol taraf refereksi, sağ taraf static oluyor!

const & value ref → life time extend

universal reference → life time extend

eder scope begins

* Back to Lambda Expressions:

- Hatırlatmalar:
 - Lambda expressionlerde, delegeler ifadesi kesisik bir sınıf kodu oluşturur. Ve o türden bir gelenek nesne ya da metotlar.
 - Closure type: Delegenin oluşturduğu sınıfın türü
 - Closure object: Delegenin oluşturduğu gelenek nesne
 - Stateless lambdalar, otomatik olarak function pointer'a dönüştür.
(hic bir şey capture etmem)

```
int main()
{
    int(*p)(int) = [](int x) {return x + 10; };
}
```

```
int main()
{
    auto f = +[](int x) {return x + 10; };
}
```

- Template Argument obdevetmeli de kullanılabılır.

→ Immediately Invoked Function Expression:

- Bir kodda bağlı durum, değer üretme için

```
int main()
{
    int a = 45;
    //
    int x = a > 10 ? 30 : a;
}
```

alternatif

```
int main()
{
    int a = 45;
    //
    int x = ([](int y) {
        // ...
        return y + 5;
    })(a);
}
```

→ Ün/:

```
void func()
{
    static int _ = [] {
        std::cout << "merhaba arkadaşlar\n";
        return 0;
    }();
}

int main()
{
    func();
    func();
    func();
}
```

→ Fonksiyonun static yerel değişkenleri, fonksiyon çağrıldığında her zaman aynı! → ilke çağrıda ilk defa yerel ve loca

→ static yerel değişkenler thread-safe