→ Referans semantiğine devam etmeden önce Scope Leakage'dan bahsetti

* Scope Leakage:

→ Bir ismin ilgili kullanıldığı kod alanının dışında bilinir olması / kullanılabilir olması durumu.

→ Leakage'i önlemek için C'de olmayan, C++'da gelen, → If / switch scope içinde değişken tanımlama
→ C++ 17'de gelen if with initializer

1. Değişken isimlerini istenmeyen şekilde kullanmış olabiliriz hata ile.

2. Bir class'ı, pointer'ı, smart pointer'ı kullanılması gereken zamandan daha uzun süre (bir container, server, DB ya da heap) tanıtılıyor olabilir.

```
int func();

int main()
{
    int x;
    x = func();

    if (x != 0) {
        std::cout << "true path x = " << x << "\n";
    }
    else {
        std::cout << "false path x = " << x << "\n";
    }
}
```

→ x scope dışında

```
int main()
{
    if (int x = func()) {
        std::cout << "true path x = " << x << "\n";
    }
    else {
        std::cout << "false path x = " << x << "\n";
    }
}
```

→ Buna C++ izin vermiyor.
→ Burada logic değişkendirme var.
→ 0 değilse
← 0 ise

→ **If with initializer:** C++ 17'de geldi.

```
//if with initializer

int foo();

int main()
{
    if (int x = foo(); x > 10) {
        ///
        //use x
    }
    else {
        //use
    }
}
```

variable tanımlama kısmı
Logic kısmı

* Ödevler bu videodan sonra verilmeye başlanmış

→ Referanslara Devam:

* Const L Value References:

- int &r = 10
  - → hatalı çünkü lvalue expression r value ile init ediliyor.

→ Fakat: const int &r = 10 ; hatalı değil.

- double dval = 3.4
  int &r = dval
  - → syntax hatası çünkü referans olan ismi, bir değişkene başlatılan aynı türden olması gerekir.

→ Fakat: double dval = 3.4 ;
const int &r = dval ; hatalı değil.

→ Cünkü const referans aslında geçici bir nesne oluşturuyor ve ona ilk değer atamasını yapıyor,

Önemli!!

→ const int &r = 10 örneğinde
  - int tmp = 10
    const int &r = tmp

→ double dval = 3.4
const int &r = dval örneğinde
  - int tmp = dval (Lvalue ter dönüşümü olur / narrowing çünkü)
    const int &r = tmp

* Bunun bize katkısı, const kullanarak bir lvalue expression'ı r value ile çağrı yapabiliriz, init edebiliriz.
pr value

* R value Reference: → Modern C++ ile eklendi

→ Move semantics:
- R value'ya ihtiyaç duyulma nedenlerinden
- kopyalamaya alternatif bir yapı
- Hayatı bitecek bir nesnenin kaynaklarını, hayata gelecek bir nesneye aktaran araç seti.

→ Generic Programlama → Perfect Forwarding:
- R value'ya ihtiyaç duyulma nedeni

- int &&r = 10
  - → bu da yalnızca r value expression ile init edilebilir.

* Type Deduction:

→ Tamamen Derleme Zamanına ilişkin bir araç.
→ C'de yok, C++'da eklendi.

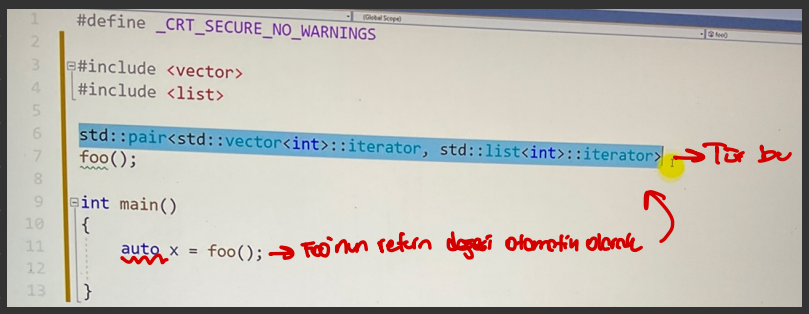
→ Farklı araçlarla yapılabilir. En meşhuru auto type deduction, diğer araçlar ise declytype ve declytype (auto)

1. Auto Type Deduction:

auto x = expr; ← Burada x'in türü değil, auto'ya karşılık gelen türü deduct ediyoruz!!! Dolayısıyla x int

auto → bu bir place holder.

Örn:



→ Ayrıca generic programlamada bolca kullanılır.

Özelik: - Kod yazmayı kolaylaştırır
- Tür bildirimini değiştirdiğimizde, tek re-compile ile, takip etmeden değiştir.

→ Arada Bonsella: Integral Promotion

- Integer dan küçük olan type'lar (char, short ...) aritmetik işleme sokulursa, integer mertebesine yükseltir.
- Bu özellik hem C hem C++'da var

```
int main()
{
    char c1 = '10'
    char c2 = '20'
    auto c1 + c2
```

→ türü int olur

→ auto deduction yapılırken: const int x = 10
auto y = x
↳ y'nin türü düz int
const değil!!

const int x = 10
const auto y = x
↳ simdi türü const auto

→ Const Düşer!!

→ Ayrıca referanslık da düşer.

→ `int a[3] = {1,2,3}` → Array to int* decay!! , B'nin türü int*
`auto b = a`

`auto p = "mert"` , → const char*

* auto & c = expression:

↳ referans belirteçlerini auto

`int a[3]: {1,2,3}` → array decay olmaz, referans olmayan array, nasıl decay olur, hem C hem C++
`auto & x = a` → x'in türü int[3] / Aynı şekilde const'luğunu da korumaz

→ `auto & x = "mert"`
↳ auto const char[5] , x auto olmadan const char(&x)[5] = "mert"

* Decltype:

→ decltype(expr) şeklinde kullanılır. Türün kullanıldığı her yerde. Fakat isimde farklı, expressionda farklı

```
int x = 10;
decltype(x) ival = 5 ; ival'in türü int olur.
```

* Decltype const'luğu düşürmez

→ Örn: `int a[5] = {};`
`decltype x` → x'in türü int array 5 elemanlı

→ Fakat unutma: const ve ref'ler init edilmeli!!!

→ `decltype((expr)) x` → çift parantez = expr & türünden olur x!