

→ Referans sarrantırgıne devam etmeden önce Scope Leavage dan bahsetti

* Scope Leavage:

→ Bir ismin ailen kullandığı kod alanının dışında bilimsir olması / kullanılabilir olması durumu.

→ Leavage: öğrenmek için C'de olmayan, C++'da gelen, → if/switch: scope tanımla değışken tanımlama
→ C++ 17'de gelen if with initializer

1. Değişken isimlerini islemeyen şekilde kullanmış olabiliriz hatta ile.

2. Bir class', pointer', smart pointer' kullanılması gereken zamanlarda daha uzun süre scope tanıtıyor duruyor.

```
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 dışındadır
```

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

→ Bina C++ rem veriyor.
→ Burada logic değeri değişime var.
← 0' değere
← 0'sa

→ 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 verilmeğe başlanmıştır

→ Referanslara Dönem:

* Const L Value References:

• `int &r = 10`

→ hatalı const lvalue expression
r value ile init. ediliyor.

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

• `double dual = 3.4`

`int &r = dual`

→ syntax hatası const
referans olan `int`, bir değere başlangıç
aynı taraftan olması gerekir.

Fakat: `double dual = 3.4` ;

`const int &r = dual` ; hatalı değil.

→ Genelde const reference altında global bir nesne oluşturur ve ona ilk değer ataması yapılıyor.

Önemli!!

→ `const int &r = 10` içinde

↳ `int &n = 10`

`const int &r = &n`

→ `double dual = 3.4`

`const int &r = dual` içinde

↳ `int &n = dual` (ve ter dönüştürme olur / narrowing cast)

`const int &r = &n`

* Bunun birer katkısı, const kullanarak bir lvalue expression'i r value ile değeri yapabiliriz, Init edebiliriz

pr value

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

→ Move semantics:

• R value'ya ihtiyaç duyulma nedenlerinden

• Kopyalamaya alternatif bir yapı

• Hayati bir şekilde bir nesnenin kaynaklarını, hayati olarak bir nesneye aktaran araç seti.

→ Generic Programlarda → Perfect Forwarding: • R value'ya ihtiyaç duyulma nedeni

• `int &&r = 10`

→ bu da yalnızca rvalue
expression ile init edilebilir.

* Type Deduction:

→ Tamamen Derleme Zamanında İşleyen bir araç.

→ C'de yok, C++'da eklendi.

→ Farklı araçlarla yapılabilir. En meşhur **auto type deduction**, diğer araçlar ise **decltype** ve **decltype(auto)**

1. Auto Type Deduction:

auto $x = \text{expr};$ → Burada x 'in türü değil, **auto**ya karşılık gelen türü dedekt ediyoruz!!! Doğru yoldan x int bu bir place holder.

Örn:

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3 #include <vector>
4 #include <list>
5
6 std::pair<std::vector<int>::iterator, std::list<int>::iterator>
7 foo();
8
9 int main()
10 {
11     auto x = foo();
12 }
13
```

→ Tür bu

→ Foo'nun return değeri otomatik olarak

→ Ayrıca genelde programlamada sıkca kullanılır.

Özellik: - Kod yazmayı kolaylaştırır

- Tür bildirimi değiştirilmezse, tek re-compile ile, takip etmeden değiştirir.

→ Arada Bonzetti: Integral Promotion

• Integer dan küçük abstrak type'ları (char, short ...) aritmetik işleme sokarsa, integer mertebesine yükseltir.

• Bu özellik hem C hem C++'da var

• `int main()`

{

char c1 = '10';

char c2 = '20';

auto c1 + c2

→ tür int olur

→ auto deduction yaparken: `const int x = 10`

`auto y = x`

→ y'nin türü `int` const değil!!

`const int x = 10`

`const auto y = x`

→ `y` türü `const auto`

→ **Const Değer !!**

→ Aynı referanslık da diğer.

→ `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 deklarasyonu `auto`

`int a[3] = {1, 2, 3}` } → Array decay olmaz / Aynı şekilde const'ünü de
`auto &x = a` } x'in türü `int[3]` kaybetmez

→ `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, expression'da farklı.

`int x = 10;`

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

* `decltype` const'ü deşirmez

→ DRY: `int a[5] = {3};`

`decltype x` → x'in türü `int array 5 eleml.`

→ Fakat unutmak: `const` ve `ref`'ler `int` edilmeli!!!