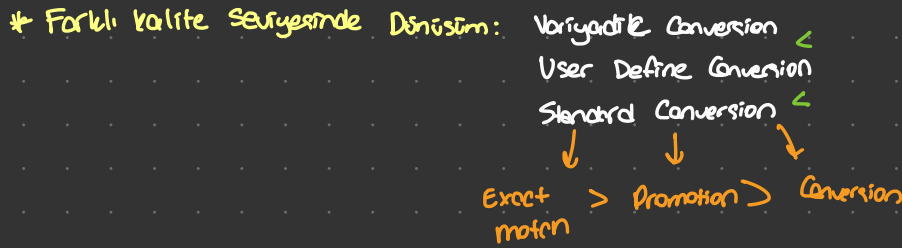


→ Function overload Resolution ile ilgili:

- İki durumda resolve edemez. 1) NO-match
2) Ambiguity



* Type Cast Operators:

• implicit: örtük - gizli → talimat vermeden compiler yapar.

• explicit: fiilen typecast yapılması

* C dilinde 1 adet typecast var. fiilen c++'in aynı typecast operatörü var. : → static cast
→ const cast
→ reinterpret cast
→ dynamic cast

kullanılabilir ama kullanılmamalı.

1. Static Cast: Birim kontrollerinde olan, otomatik uygulan. + c++'deki C++'de ifade static cast edilecek e'den en uygun bir kalite.

2. Const Cast: Const nesneden, low level const pointerın + volatile ifadeleri değiştirilen.
→ program oku kriterleri değiştiriyor.

3. Reinterpret: Nesnenin bitlerini, adresini farklı seviyede kullanma işi.

⇒ static_cast < int > (dual)
↓ ↓
const. adl. hafıza (değişkenler)
 tür

en iyi örnek, biz .bin (formatör) dosyası char* dene oluyoruz. Bunu C++'de reinterpret_cast < char* > yap.

* Enums:

static const

→ Arithmetiklerden enum'ın otomatik dönüşüm yok. Ama enum'ın arithmetiklere otomatik dönüşüm var.

* Modern C++'da: Scoped enums eklendi.

* enum Color : unsigned char { red, green, blue }

↓
underlying type'i
belirtebilirsiniz.

* enum class Color { red, blue, green } → yine underlying type belirtilebilir.

→ Artık bu enum'ların kendi scope'ları var.

```
5 enum class Color { red, blue, green };
7 int main()
8 {
9     auto c = Color::red;
10 }
```

→ bu şekilde belirtilmese red'i bulamaz.

```
3
4
5 enum class Color { red, blue, green };
6 enum class TrafficLight { red, yellow, green };
7 enum class TextBoxColor { red, yellow, green };
8 enum class WindowBaseColor { red, yellow, green };
9
10 int main()
11 {
12     Color c = Color::red;
13     TrafficLight tl = TrafficLight::red;
14 }
15
16
```

→ Scope'ları farklı olduğu için aynı isimlendirmeye yapabiliriz.

→ Bu enum'ların sayı değerleri olarak kullanmak için

⇒ static const <int> (c)

```
enum class Color {red, blue, green, brown, magenta, black};
```

```
void func()
```

```
{  
    using enum Color; → using kullanarak bu scope'ı visible hale getirebiliriz.
```

```
    auto c1 = red;
```

```
    auto c2 = red;  
}
```

```
int main()
```

```
{  
  
}
```

* C kuralı:

- **name lookup:** Derleyicinin bir ismin, hangi varlığa ait olduğunu onlara gösterir.

- **context control:** name lookup'ten sonra (isim bulunduktan sonra), name lookup biter, sonra bağlamı kontrol edilir.

* Scope Resolution Operator:

```
1 #include <cstdio>
```

```
2  
3 int foo();
```

```
4  
5 int main()
```

```
6 {  
7     int foo = 6;
```

```
8  
9     ::foo();  
10 }
```

→ artık foo'yu global namespace'de arar. Eğer aynı adlı fonksiyon scope'da farklı foo objesi için hata verir.

