* Attributes Devam:

[[deprecated]]:

```cpp
struct [[deprecated("use struct ErgNec")]] Nec {
    int a, b, c;
};

int main()
{
    Nec mynec;
    ///
}
```

→ Compiler bunun deprecated olduğu uyarısını verecek!

→ parantez içine yazılan string literalini bize uyarı olarak verir.

[[likely]] ve [[unlikely]]:

- Compiler dependent

→ for branch prediction ←

```cpp
enum class Color {white, blue, black, red, magenta};

void f_white();
void f_blue();
void f_black();
void f_red();
void f_magenta();

void foo(Color c)
{
    //...
    switch (c) {
    case Color::white: f_white(); break;
    case Color::blue: f_blue(); break;
    [[likely]] case Color::black: f_black(); break;
    case Color::red: f_red(); break;
    case Color::magenta: f_magenta(); break;
    }
}
```

→ bu path'e girme ihtimali daha muhtemel!

```cpp
constexpr double pow(double x, long long n) noexcept
{
    if (n > 0) [[likely]]
        return x * pow(x, n - 1);
    else [[unlikely]]
        return 1;
}

constexpr long long fact(long long n) noexcept
{
    if (n > 1) [[likely]]
        return n * fact(n - 1);
    else [[unlikely]]
        return 1;
}

constexpr double cosin(double x) noexcept
{
    constexpr long long precision{ 16LL };
    double y{};
    for (auto n{ 0LL }; n < precision; n += 2LL) [[likely]]
        y += pow(x, n) / (n & 2LL ? -fact(n) : fact(n));
        return y;
}
```

[[maybe_unused]]: - kullanılmayan parametrel değişkenlere compiler uyarı veriyor!

```cpp
[[maybe_unused]] static int foo(int x)
{
    ////
    return x * x;
}
```

→ foo fonksiyonu tanımlandıktan sonra, kullanılmasaydı compiler uyarı verirdi.

→ herhangi bir nedenden ötürü kullanmayacaksak, maybe_unused attribute'u ile uyarıyı ortadan kaldırırız.

→ Değişkenler için de kullanılır!

```cpp
[[maybe_unused]] void f([[maybe_unused]] bool thing1,  [[maybe_unused]] bool thing2)
{
    [[maybe_unused]] bool b = thing1 && thing2;
    assert(b); // in release mode, assert is compiled out, and b is unused
    // no warning because it is declared [[maybe_unused]]
} // parameters thing1 and thing2 are not used, no warning

int main() {}
```

[[noreturn]]:

```cpp
[[noreturn]] void Exit();
```

→ Fonksiyon return etmeden exit ediyoruz!

→ Bu fonksiyon return edemeyecek!

→ return edemeden exception throw etmek de buna dahil!

```cpp
[[noreturn]] void bar(int i)
{
    if (i < 0) {
        throw std::runtime_error{ "error!!!!" };
    }
}
```

→ Bu kod undefined behavior. Çünkü fonksiyonun sonuna ulaşabiliriz. (Reachable)

*Return değerinin void olması, kesinlikle aynı anlama gelmez! [[noreturn]] o fonksiyonun tamamlanmayacağı anlamına gelir!

[[fallthrough]]: - switch statementlarında!

```cpp
void f1();
void f2();
void f3();

void func(int x)
{
    switch (x) {
        case 1: f1(); [[fallthrough]];
        case 2: f2(); break;
        case 3: f3(); break;
        //
    }
}
```

→ bilerek ve isteyerek break yapmadığımızı söyledik!

*Hatırlatma: → type identity ile template'de tür çıkarımını engellemek!

```cpp
template <typename T>
struct TypeIdentity {
    using type = T;
};

template <typename T>
using TypeIdentity_t = TypeIdentity<T>::type;
          alias template!

template <typename T>
void foo(T, TypeIdentity_t<T>);

int main()
{
    foo(1.4, 5);
}
```

<T> Provide sample template arguments for IntelliS

→ bunun için int çıkarımı yapılmamış oldu!

*iota:

```cpp
using namespace std;

vector<int> ivec(10);

iota(ivec.begin(), ivec.end(), 0);

for (auto i : ivec)
    cout << i << " ";
```

⇒ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- verilen sayıdan başlayarak, range'in başından sonuna kadar +1 eklenerek yazar.

"Range +1'lenmeyi destekleyen türden olmalı!

*Std:: ranges: - Cpp 20

- Eğer STL ile çözdüğümüz bir senaryo (sort, for_each...) varsa, Std değil ranges kullan!

mümkünse yenisini kullanalım.
yenisini kullanmak daha pratik
yenisini kullanmak daha güvenli
yenisini kullanmak daha verimli (efficient)
yenisini kullanmak ilave avantajlar (esneklik) kazandırıyor

- Diğer algoritmalarla çakışmaması için, namespace alias kullanılır!

```cpp
namespace rng = std::ranges;
```

⇒ namespace alias!

a) yeni algoritmalar (conscept) constarined → ranges için bir concept
b) bildirimlerin bir kısmı farklı yapıda

c) iterator göndermek yerine, direkt o collection'ı gönderiyoruz (?)

```
template <std::ranges::range T>
void foo(T &&c)
```

Normal STL:

```
vector<Date> dvec;
rfill(dvec, 20, Date::random);

sort(dvec.begin(), dvec.end());
```

With Ranges:

```
vector<Date> dvec;
rfill(dvec, 20, Date::random);

ranges::sort(dvec);
```

→ range yerine tüm container'ı gönderiyoruz

Son 30 dakika → range terimin iteratorlerin subsumption'ı

⋮