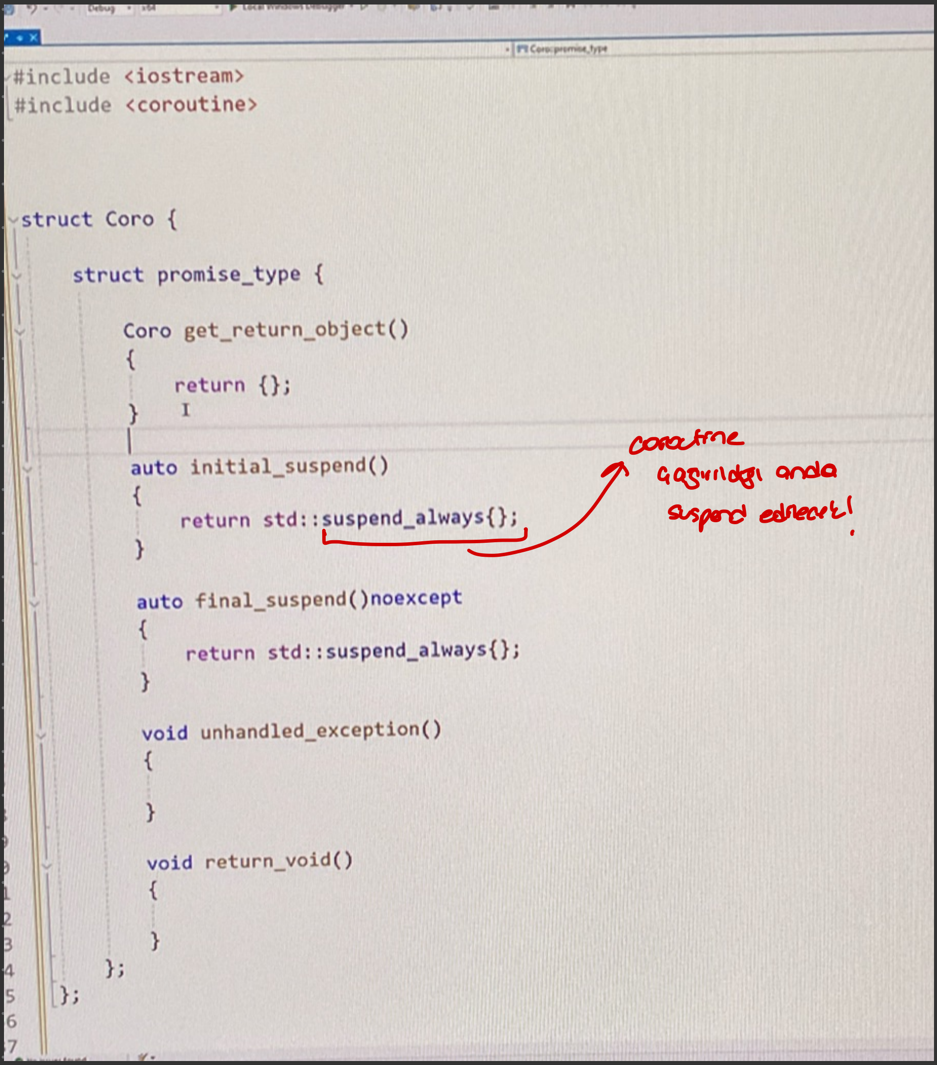
* Awaiter / Awaitable Example:

1.



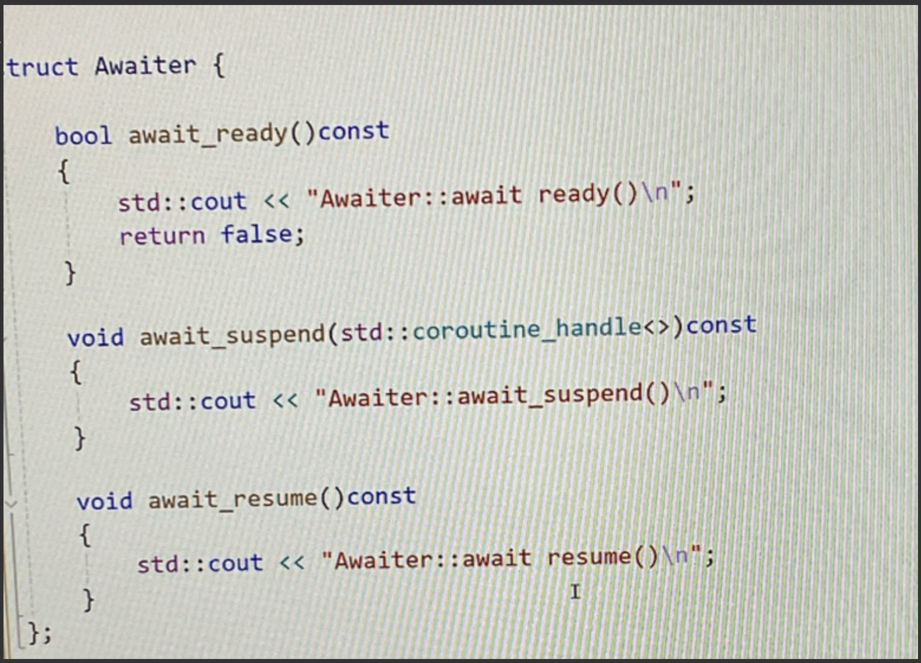
```cpp
#include <iostream>
#include <coroutine>


struct Coro {

    struct promise_type {

        Coro get_return_object()
        {
            return {};
        }

        auto initial_suspend()
        {
            return std::suspend_always{};
        }

        auto final_suspend()noexcept
        {
            return std::suspend_always{};
        }

        void unhandled_exception()
        {

        }

        void return_void()
        {

        }
    };
};
```

coroutine çagırıldığı anda suspend edilecek!

→ Coroutine interface definition

→ coroutine interface return type'ımız bu olacak

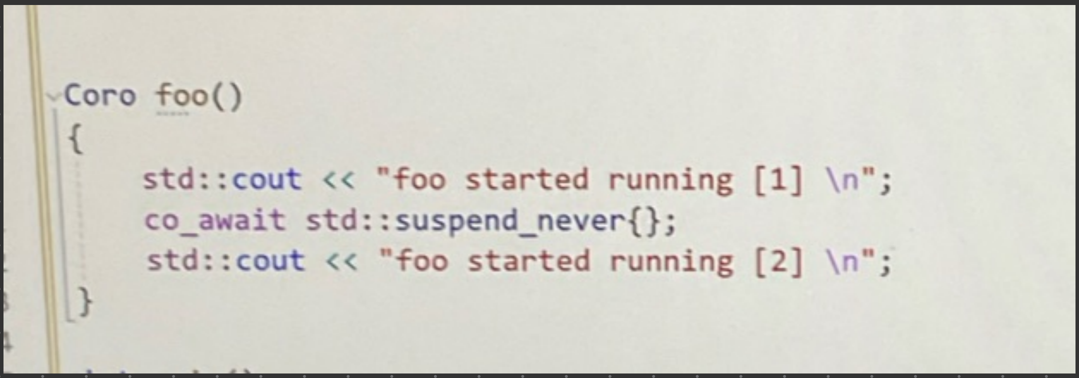
2.



```cpp
truct Awaiter {

    bool await_ready()const
    {
        std::cout << "Awaiter::await ready()\n";
        return false;
    }

    void await_suspend(std::coroutine_handle<>)const
    {
        std::cout << "Awaiter::await_suspend()\n";
    }

    void await_resume()const
    {
        std::cout << "Awaiter::await resume()\n";
    }
};
```

→ awaiter struct'ımız

3.



```
Coro foo()
{
    std::cout << "foo started running [1] \n";
    co_await std::suspend_never{};
    std::cout << "foo started running [2] \n";
}
```

→ coroutine'imiz!

→ fakat bu kodu runladığımızda, hiç bir çıktı alamayız.!

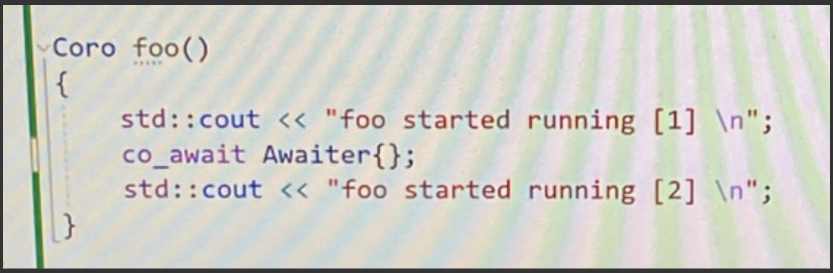
→ Çünkü initial_suspend fonksiyonumuz

↳ suspend_always

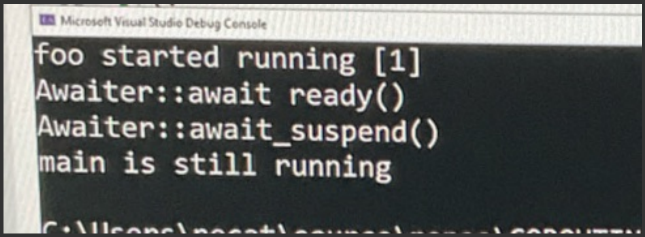
Derleyici co_await operatörünü bir kaç yerde obna çağırır!

$\Rightarrow$ co_await $\underbrace{\text{prom.initial\_suspend()}}_{\text{biz co\_await'i suspend\_never ile çağırdık!}}$

3.1



```
Coro foo()
{
    std::cout << "foo started running [1] \n";
    co_await Awaiter{};
    std::cout << "foo started running [2] \n";
}
```



```
foo started running [1]
Awaiter::await ready()
Awaiter::await_suspend()
main is still running

C:\Users\necat\source\repos\COROUTIN
```

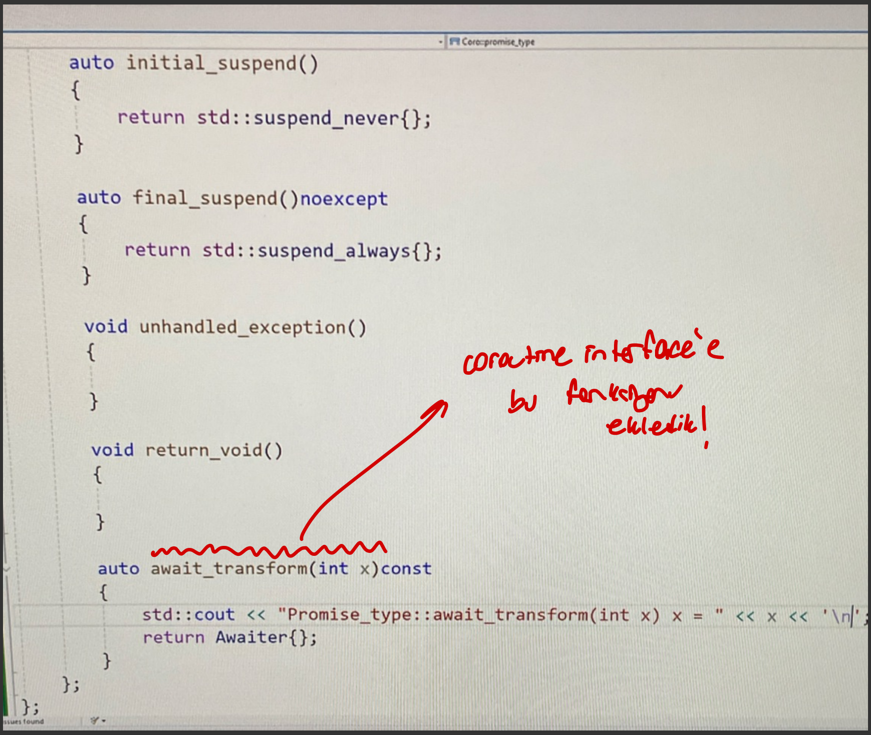
→ awaiter'ın await ready'sini çağırdık

↳ false return ettiği için await_suspend çağılacak!

3.1. Ai

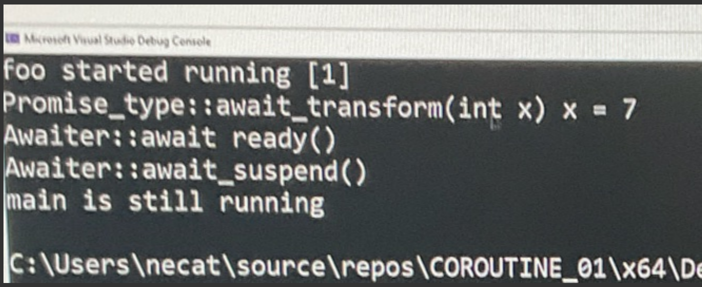
↳ coawait'in operand olarak aldığı ifadenin awaitable'a dönüşmesi

await transform function



```
auto initial_suspend()
{
    return std::suspend_never{};
}

auto final_suspend()noexcept
{
    return std::suspend_always{};
}

void unhandled_exception()
{

}

void return_void()
{

}

auto await_transform(int x)const
{
    std::cout << "Promise_type::await_transform(int x) x = " << x << '\n';
    return Awaiter{};
}
};
};
```

coroutine interface'e bu fonksiyonu ekledik!



```
foo started running [1]
Promise_type::await_transform(int x) x = 7
Awaiter::await ready()
Awaiter::await_suspend()
main is still running

C:\Users\necat\source\repos\COROUTINE_01\x64\De
```

3.1.A.1:

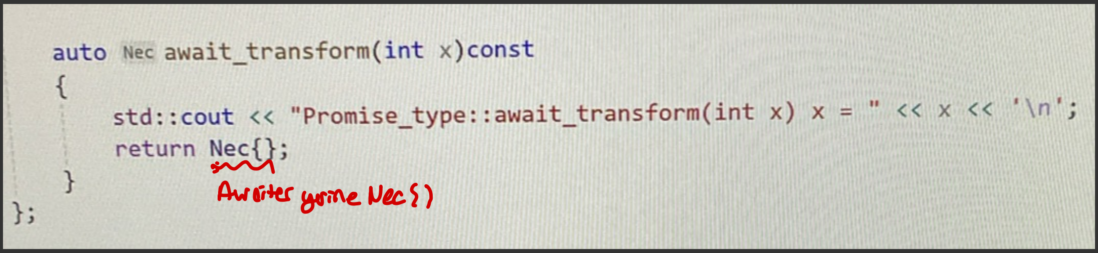
↳ await transform fonksiyonu awaiter yerine farklı bir sınıftan geçici nesne dönerse,

↓

return edilen sınıfın,

co_await operator'ü olmalı!

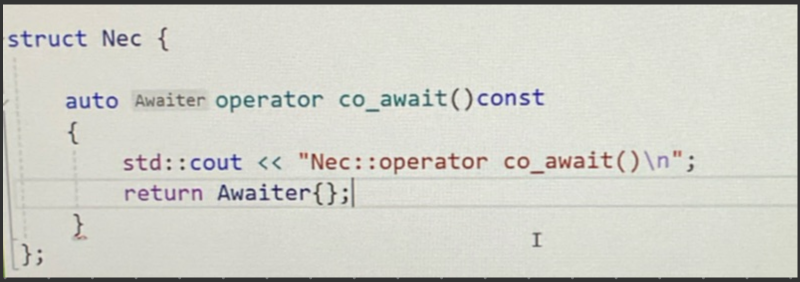
*



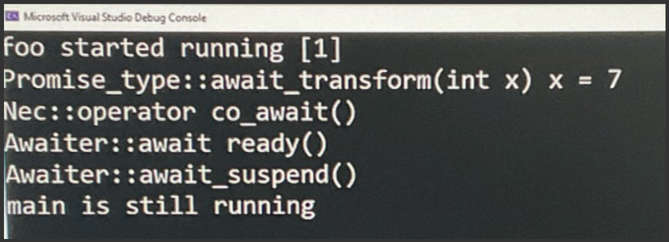
```cpp
auto Nec await_transform(int x)const
{
    std::cout << "Promise_type::await_transform(int x) x = " << x << '\n';
    return Nec{};
}
};
```

Awaiter yerine Nec{}

*



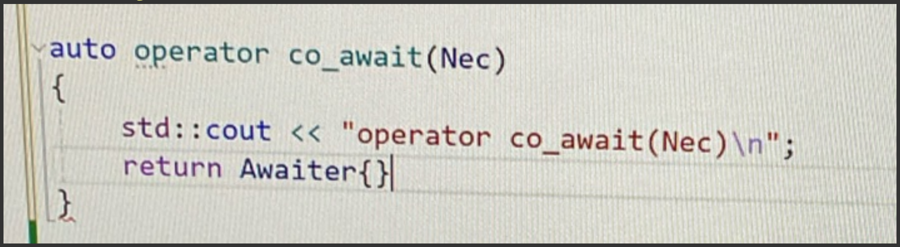
```cpp
struct Nec {

    auto Awaiter operator co_await()const
    {
        std::cout << "Nec::operator co_await()\n";
        return Awaiter{};
    }
};
```



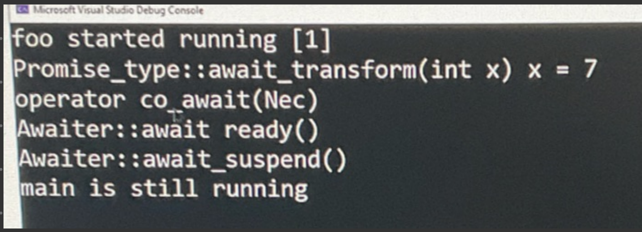
```
foo started running [1]
Promise_type::await_transform(int x) x = 7
Nec::operator co_await()
Awaiter::await ready()
Awaiter::await_suspend()
main is still running
```

3.1.A.2:

↳ global co-await operatörü:



```cpp
auto operator co_await(Nec)
{
    std::cout << "operator co_await(Nec)\n";
    return Awaiter{}
}
```



```
foo started running [1]
Promise_type::await_transform(int x) x = 7
operator co_await(Nec)
Awaiter::await ready()
Awaiter::await_suspend()
main is still running
```