* Gecen Dersler :

main.cpp*

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

class Coro {
public:
    struct promise_type;
    using handle_type = std::coroutine_handle<promise_type>;

    struct promise_type {

        Coro get_return_object()
        {
            return Coro{ handle_type::from_promise(*this) };
        }

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

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

        void unhandled_exception()
        {

        }

        void return_void()
        {

        }

    };

    Coro(handle_type h) : h_(h) {}
private:
    handle_type h_;

};

Coro cfunc()
{
    co_await std::suspend_never{};
    std::cout << "necati ergin\n";
}
```

```cpp
int main()
{
    using namespace std;
    boolalpha(cout);

    Coro::handle_type handle1;

    cout << (handle1 == nullptr) << '\n';
```

default init edilebilir!
nullptr == sorgusu yapilabilir!

```cpp
int main()
{
    using namespace std;
    boolalpha(cout);

    auto f = cfunc();
    auto handle = f.h_;

    cout << (handle != nullptr) << '\n';
    if (handle) {
        cout << "dolu durumda\n";
    }
```

* nullable type olduğu için, nullptr atması yapılabilir!

operator bool macet!

*Address / From Address Functions:

```cpp
int main()
{
    using namespace std;

    auto f = cfunc();

    auto handle = f.h_;

    void* vptr = handle.address();

    //

    auto handle2 = decltype(handle)::from_address(vptr);

    cout << (handle == handle2) << '\n';
}
```

İkisi aynı!

true statement

*Coroutine Handle'ın kopyalanması:

Coroutine handle bir generic typetır. promise type'ı template argümanı olarak alır. Fakat sınıfın <void> specialization'ı void pointer gibi davranır! ⟶ Yani coroutine handle nameless, coroutine handle<void> türüne kopyalayabiliriz!

```cpp
int main()
{
    using namespace std;

    auto f = cfunc();

    coroutine_handle<> vhandle = f.h_;
```

Fakat void çağrılmının promise fonksiyonu yok!

```cpp
coroutine_handle<> vhandle = f.h_;

vhandle.
```

- address
- destroy
- done
- from_address
- operator bool
- operator()
- operator=
- resume
- _Ptr