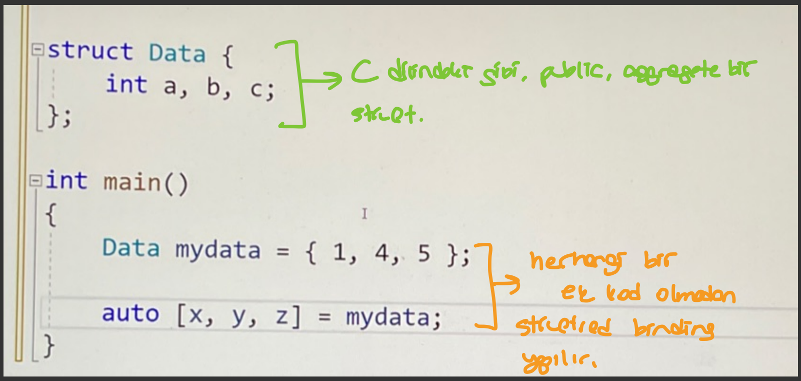
Tekrar:

- Structured Binding :

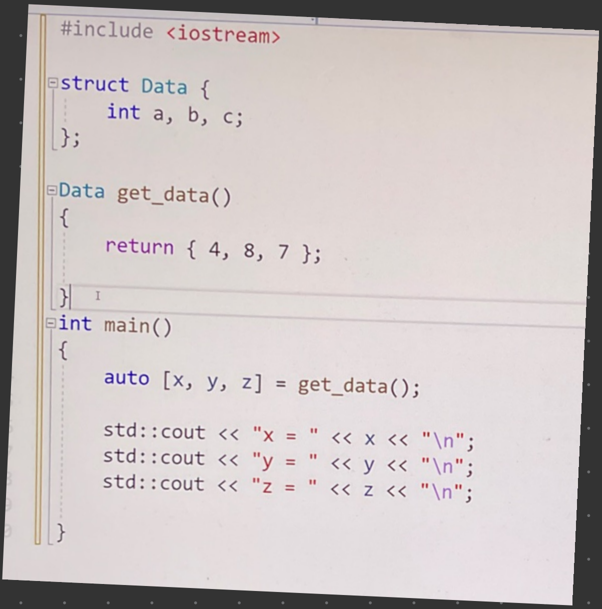


```cpp
struct Data {
    int a, b, c;
};

int main()
{
    Data mydata = { 1, 4, 5 };

    auto [x, y, z] = mydata;
}
```

C dilindeki gibi, public, aggregate bir struct.

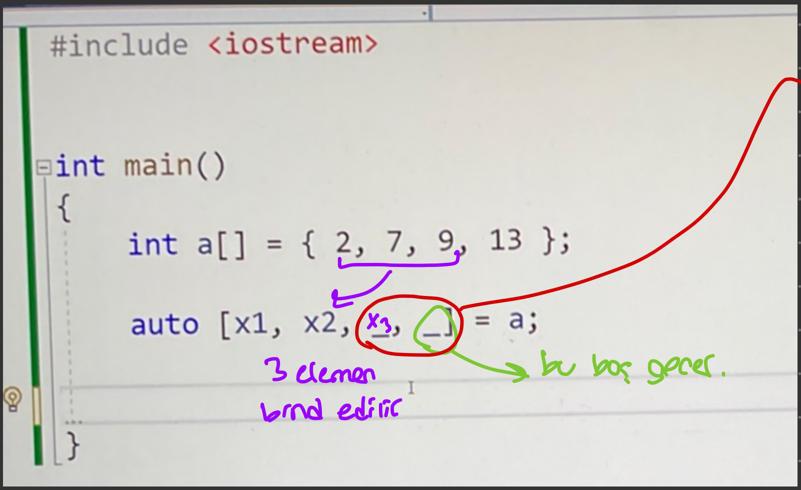
herhangi bir ek kod olmadan structured binding yapılır.

Benzeri şekilde



```cpp
#include <iostream>

struct Data {
    int a, b, c;
};

Data get_data()
{
    return { 4, 8, 7 };

}
int main()
{
    auto [x, y, z] = get_data();

    std::cout << "x = " << x << "\n";
    std::cout << "y = " << y << "\n";
    std::cout << "z = " << z << "\n";
}
```

→ Structured binding, arrayler için de geçerli.

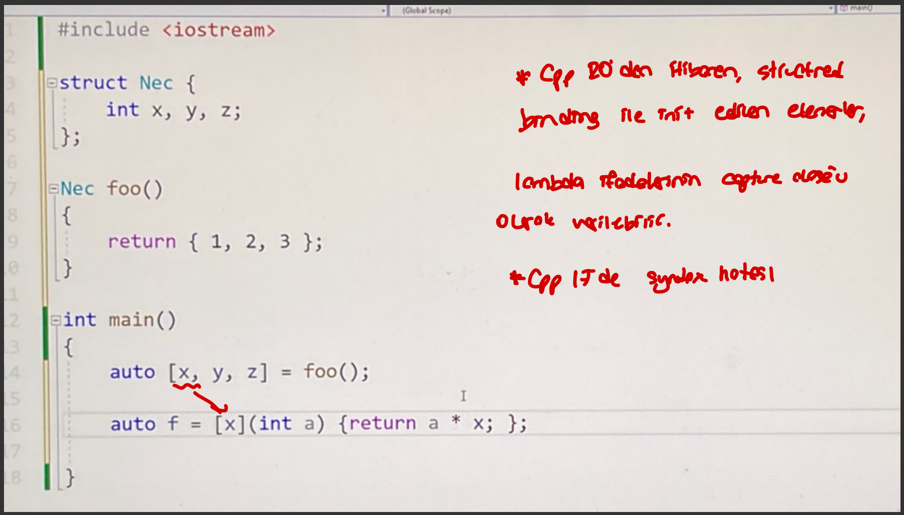


```cpp
#include <iostream>


int main()
{
    int a[] = { 2, 7, 9, 13 };

    auto [x1, x2, x3, _] = a;

}
```

3 eleman bind edilir

bu boş geçer.

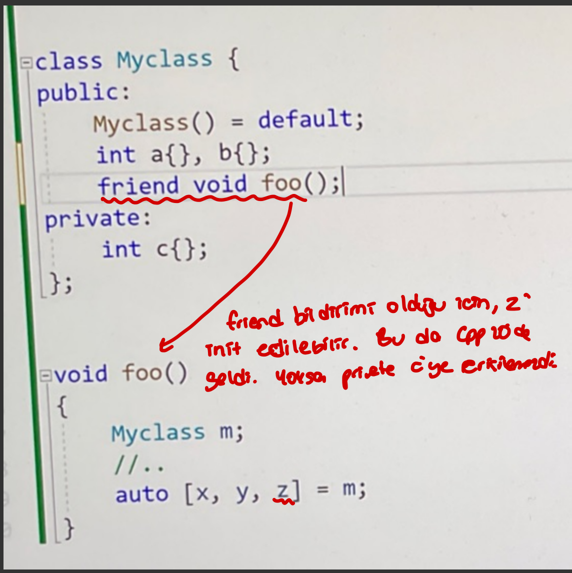
Ancak aynı scope'ta ikisini de underscore olarak yazmak syntax hatası !!



```cpp
#include <iostream>

struct Nec {
    int x, y, z;
};

Nec foo()
{
    return { 1, 2, 3 };
}

int main()
{
    auto [x, y, z] = foo();

    auto f = [x](int a) {return a * x; };
}
```

* Cpp 20'den itibaren, structured binding ile init edilen elemanlar, lambda ifadelerinin capture closeu olarak verilebilir.

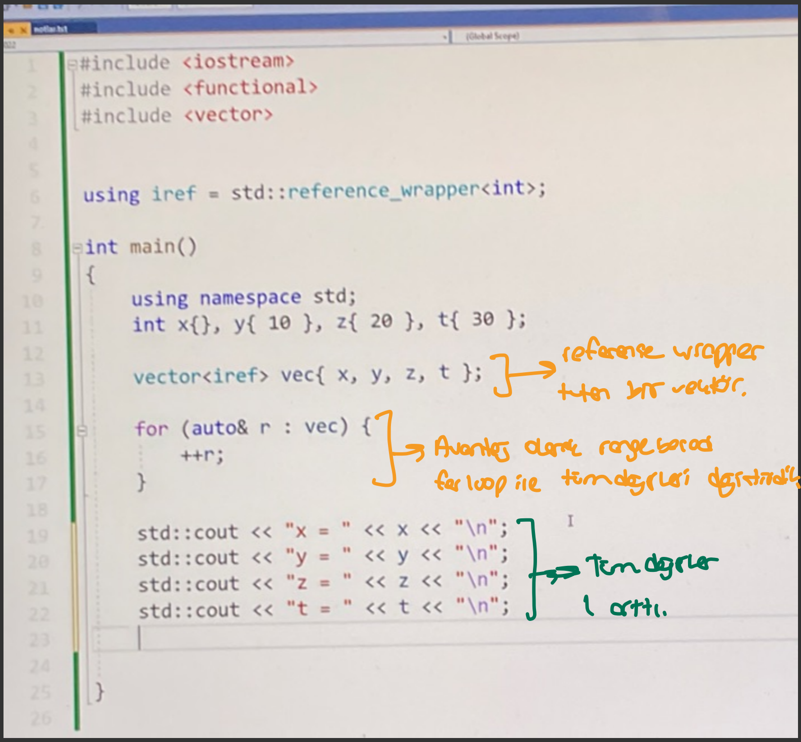
* Cpp 17'de syntax hatası



```cpp
class Myclass {
public:
    Myclass() = default;
    int a{}, b{};
    friend void foo();
private:
    int c{};
};

void foo()
{
    Myclass m;
    //..
    auto [x, y, z] = m;
}
```

friend bildirimi olduğu için, z' init edilebilir. Bu da Cpp 20'de geldi. Yoksa private c'ye erişilemezdi.

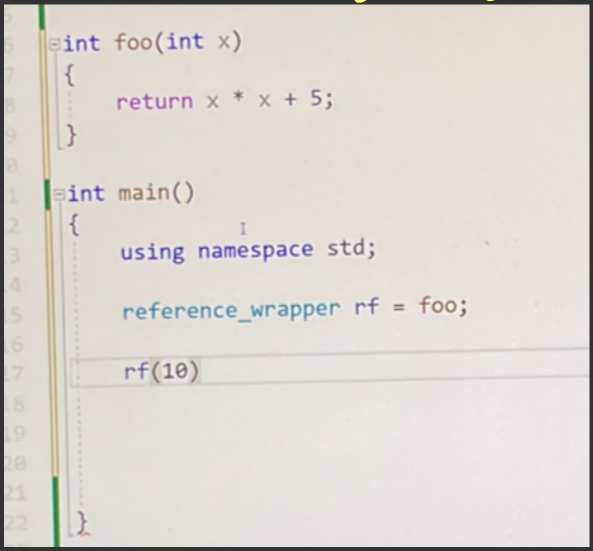
→ Referance wrapper tutan containerlar:



```cpp
#include <iostream>
#include <functional>
#include <vector>


using iref = std::reference_wrapper<int>;

int main()
{
    using namespace std;
    int x{}, y{ 10 }, z{ 20 }, t{ 30 };

    vector<iref> vec{ x, y, z, t };

    for (auto& r : vec) {
        ++r;
    }

    std::cout << "x = " << x << "\n";
    std::cout << "y = " << y << "\n";
    std::cout << "z = " << z << "\n";
    std::cout << "t = " << t << "\n";


}
```

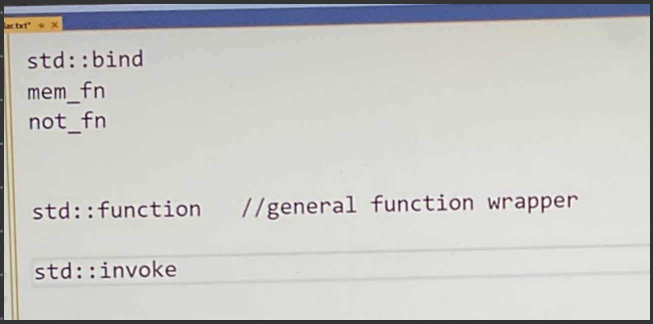
→ referense wrapper tutan bir vektör.
→ Avantaj olarak range based for loop ile tüm değerleri değiştirdik.
→ Tüm değerler 1 arttı.

→ Referance wrapperlar'a fonksiyonlar da bağlanabilir.



```cpp
int foo(int x)
{
    return x * x + 5;
}

int main()
{
    using namespace std;

    reference_wrapper rf = foo;

    rf(10)


}
```

#Function Adaptors:

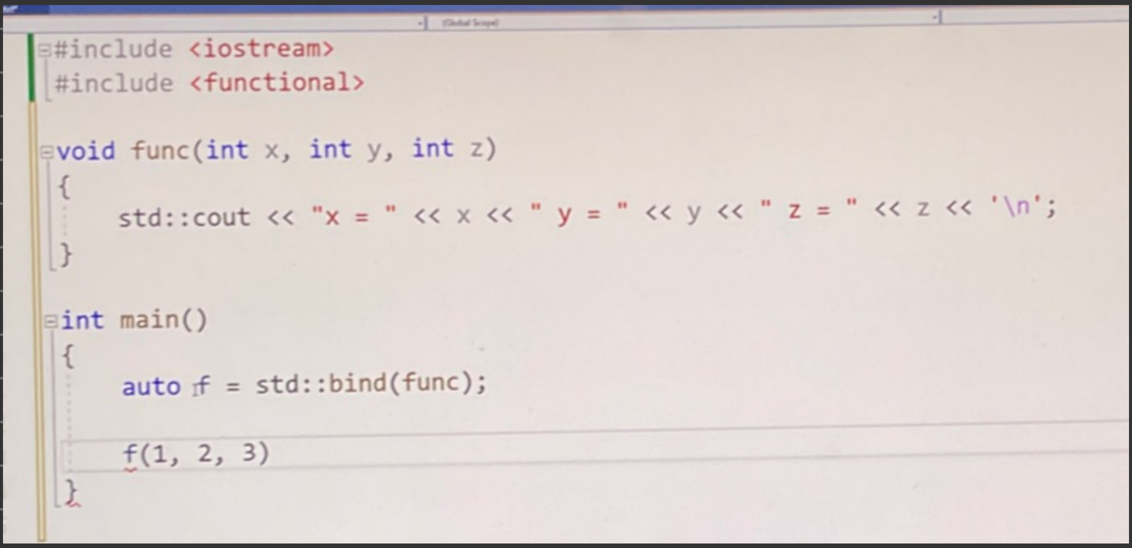


```cpp
std::bind
mem_fn
not_fn


std::function    //general function wrapper

std::invoke
```

→ Fonksiyon adaptörü nedir?

- Bizden bir callable alıp, bize callable döndüren bir wrapper.
- Örneğin 3 argümanlı bir callable'ı 2 elemanla çağırmak hale getirebiliriz. std::Bind gerekiyor.

* Std:: Bind:



```
#include <iostream>
#include <functional>

void func(int x, int y, int z)
{
    std::cout << "x = " << x << " y = " << y << " z = " << z << '\n';
}

int main()
{
    auto f = std::bind(func);

    f(1, 2, 3)
}
```