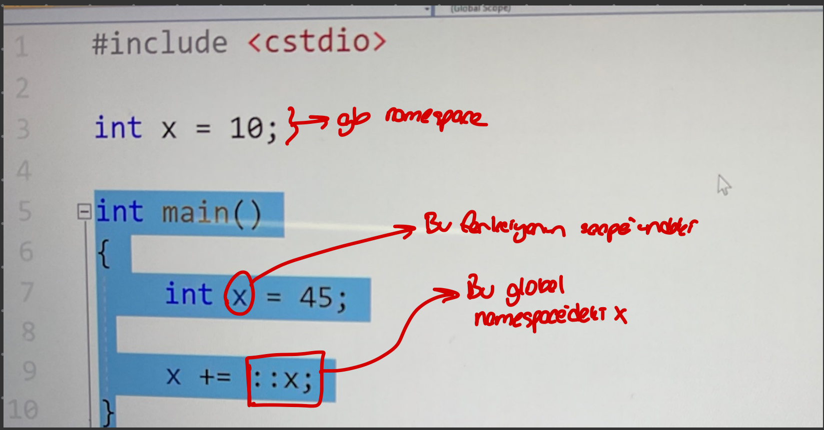
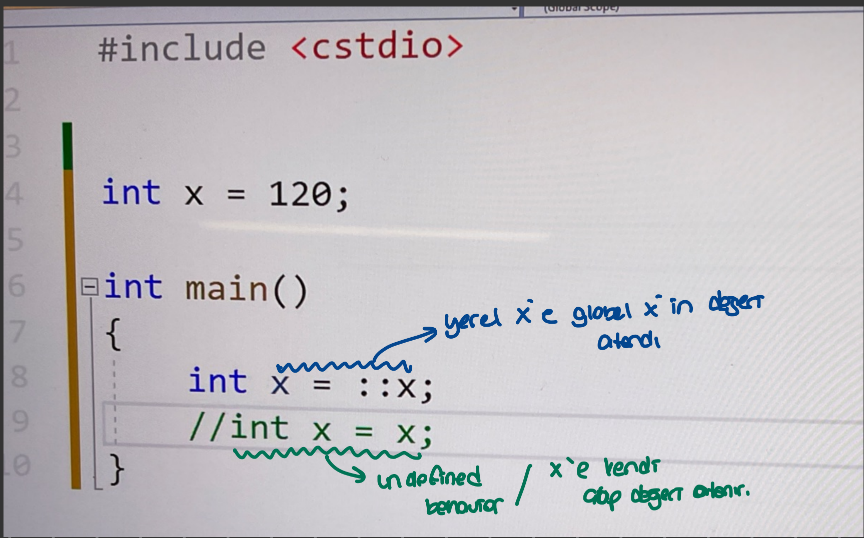
→ Tekrar:

• Scope Resolution Operator ::x → ismin tanıtıldığı namespace'te aratmak için





* Classlar:

→ Yazılımsal olarak bir problem / çözüm domainindeki bir veriyi temsil eder

• class Myclass
{
}

Myclass → class tag

→ Class Definition / Complete Type'tır.

int main()
{
myClass m
}

myClass → class tag: doğrudan tek başına bu türü temsil edebilir / return type, pointer ... (Typedef gibi)

• class Myclass → Class Declaration / Incomplete Type

- class → class anahtar sözcüğü
- Myclass → class'ın adı

→ Ana blok içindeki tanımlara member denir. (Bir şey hariç). Hiç member'ı olmayan classlar Empty Class'tır.

↳ Data member
Member function
Type member / nested type / member type
↳ class içindeki diğer classlar / typedef'ler

• Data member'lar static / non static olabilir. Non static data memberlar, sınıfın genel elemanlarıdır. Sınıfın içinde tanımlıdır. (Yer kaplar) Static data member'lar ise class dışında yer kaplar.

• C'deki functionlar C++'da free function olarak geçer. Class içinde member function → java method
global function
stand-alone function

• Data members gibi member functionlar da static / non-static tanımlanabilir.

• Non-static member functionlar default olarak, ilgili sınıfın adresiyle çağırılır.

• Access Control: Sınıfların elemanlarına erişimi kontrol eder → public: kısıtlama yok
private: sınıfın client'larının erişimine kapalı
protected: protected elemanlar private gibi clientlara kapalı ancak inherit eden diğer classlara açık.

↓
name look-up
↓
context control
↓
access control

Bu keywordlere access specifier adı verilir.

* Eğer farklı bir access specifier belirtilmezse, default olarak private member olur. (Struct ise default olarak public)

• Public ve private bölümler Scope Değildir!!. Yalnızca erişim kontrolü için

```
1  #include <iostream>
2
3  //access control
4
5  struct Myclass {
6  public:
7      int x;
8  private:
9      void x();
10 public:
11
12 };
13
```

Compiler aynı ismin verilmesine çalışır.

• C++ Member Function vs C Struct Definition:

```
class Myclass {
public:
    void foo();
    void func(int);
private:
    int a, b, c;
};


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

void foo(Myclass_* p);
void func(Myclass_* p, int);
```

bir adres göndermemiz gerek.

Structta fonksiyon tanımlamadığımız için, global fonksiyonlara parametre olarak ilgili struct'ın pointer'ı eklenir.

Member fonksiyonlar bunu default olarak yapar.

→ Çağırırken:

```
int main()
{
    Myclass_ cm;
    Myclass m;

    foo(&cm);
    m.foo();

    func(&cm, 10);
    m.func(10);
}
```

struct

class

struct adresi

• Function Overloading ve Access Control:

Derlerken sırayla: name lookup → context control → Access Control

```
class Myclass {
public:
    void func(int);
private:
    void func(double);
};


int main()
{
    Myclass m;

    m.func(1.3);
}
```

Burada ilk func'ı buldu, sonra overload olduğunu anladı sonra priv. olduğu için Access control'e takıldı ve A.C. hatası verdi

* Member Function Tanıtımı:

→ Global fonksiyonlar gibi, class'ın tanıtıldığı header'da bildirilip, ilgili source'da tanımlanabilir.

* Bu header multiple inclusion'a karşı guard etmeli.

```
#ifndef MYCLASS_H
#define MYCLASS_H

//

#endif
```

→ #pragma once preprocessor. Neden kullanmıyormuş

* int m_x , m_y , m_z veya
  x_ , y_ , z_ idiomları ile data member tanıtılabilir.

* Örnek:

```
class A {

public:
    void foo();
private:
    int x;
};

int x = 10;

void A::foo()
{
    //int x = 20;

    int ival = x;
}
```

→ Scopelara bakış

→ x en başta 20.
→ Eğer aynı scope'ta yoksa, class'takı
→ Class'ta yoksa global olacak

→ Buna name lookup deniyor.

Block scope masks > Class scope masks > Global scope bunu unutma!!!!

*Örnek 2:

```
4   class A {
5
6   public:
7       void foo(A);
8   private:
9       int m_x;
10  };
11
12  A ga;
13
14  void A::foo(A pa)
15  {
16      A ax;
17      pa.m_x;
18      m_x = 5;
19      ax.m_x = 10;
20      ga.m_x;
21
22  }
23
```

→ Bu kısmı bir daha izle