

→ Tekrar:

- Scope Resolution Operator `::x` → ismin tanımlığı namespace'te oluşturmak için

```
1 #include <stdio>
2
3 int x = 10; } → global namespace
4
5 int main()
6 {
7     int x = 45;
8
9     x += ::x;
10 }
```

Bu fonksiyonun scope'ındadır

Bu global namespace'deki x

```
1 #include <stdio>
2
3
4 int x = 120;
5
6 int main()
7 {
8     int x = ::x;
9     //int x = x;
10 }
```

yerel x'e global x'in değeri atandı

undefined behavior / x'e kendisi değeri atlanır.

* Classlar: → Yozulimsal olarak bir problem / çözüm alanında bir veriyi temsil eder

```
• class MyClass
{
}
} → class Definition / complete type tr.
```

```
int main()
{
    MyClass m
}
} → class Tag
```

değişkenlere benzer
bu türü temsil eder / return type
(Typeof gibi)

```
• class MyClass } → class Declaration / incomplete type
```

↓ class
anlatıcı + adı
sözcüğü

→ Ana blok içindeki tanımlara member denir. (Bir şey varsa). Hic member'i 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 elementleridir. Sınıfın içinde tanımlıdır. (yer kaplar) Static data member'lar ise class dışında, yer kaplar.

• C'deki functionlar C'de 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 adresine çağırılır.

• Access Control: Sınıfın elementlerine erişimi kontrol eder → public: herkese açık

↓
name look-up
↓
context control
↓
access control

private: sınıfın client'lerinin erişimine kapalı

protected: protected elementler private gibi client'da kapalı ancak
inherit eden diğer classlara açık.

↓
Bu keywordlere
access specifier adı verilir.

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

• Public ve private bölgeler 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
varlığına control.

• C++ Member Function vs C Struct Definition:

```
4 class MyClass {
5 public:
6     void foo();
7     void func(int);
8 private:
9     int a, b, c;
10 };
11
12
13 struct MyClass_ {
14     int a, b, c;
15 };
16
17 void foo(MyClass_* p);
18 void func(MyClass_* p, int);
19
```

bir adres gondermemiz gerek.

Structta fonksiyon tanımladığımız için, global fonksiyonlara parametre olarak diğer struct'ın pointer'ı eklenir.

Member fonksiyonlar bunu default olarak yapar.

→ Çözümleri: `int main()`

```
{
    MyClass_ cm; } → struct
    MyClass m; } → class

    foo(&cm); } → struct adresi
    m.foo();

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

• Function Overloading ve Access Control:

Derlerken sırayla: name lookup → Context control → Access Control

```
4 class MyClass {
5 public:
6     void func(int);
7 private:
8     void func(double);
9 };
10
11
12 int main()
13 {
14     MyClass m;
15
16     m.func(1.3);
17 }
18
```

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

* Member Function Tanıtımı:

→ Global Fonksiyonlar gibi, class'ın tanıldığı 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. Nere kullanılmış mı?

* int m_x, m_y, m_z veya
x_, y_, z_

İkinci ile data member tanımlanabilir.

* Örnek:

```
4 class A {
5
6 public:
7     void foo();
8 private:
9     int x;
10 };
11
12 int x = 10;
13
14 void A::foo()
15 {
16     //int x = 20;
17
18     int ival = x;
19 }
20
```

→ Scope'lara
dikkat.

← 20.
X en başta
→ Eğer aynı scope'ta yoksa, class-ıdır
→ Class'ta yoksa global'dir

Block scope marks → Class scope marks → Global scope bunu unutma!!!!

Buna markalama
denir.

```
void A::foo()
{
    A::x
}
```

bu şekilde
markalama
etebilirsiniz!

*Örnek 2:

```
4 class A {
5
6 public:
7     void foo(A);
8 private:
9     int m_x;
10 };
11
12 A ga; → global değişken
13
14 void A::foo(A pa) → üye fonksiyonlar private memberlara erişebilir!
15 {
16     A ax;
17     pa.m_x;
18     m_x = 5; → NON static global member olduğu için, ilgilili her neresinde m_x'i!
19     ax.m_x = 10;
20     ga.m_x;
21 }
22 → global değişken olması, sınıfın üye fonksiyonu erişim kısıtlarını değiştirmez!
23     → Hangi nesne ile çalışıldığından değişir!
```

→ Aynı class'tan oluşturulan nesneler, birbirlerinin private verileri elementlerine erişebilir!