

Fonksiyona recursive yaprı:

```
void func()
{
    int func=0;
    ::func(0); //global'de orar bulur.
}
```

empty class: Sınıfın hiçbir elemanı olmamalıdır.

class members: a) data member b) member function
c) type member : tamamı sınıfın içinde yapılan türlerle denir.
→ static data member
→ non-static data member

class MyClass {

int x; //x değişkeni sınıf nesnesinin içinde bulunur.

static int s; //sınıf başına tek, sınıf nesnesinin içinde depolıdır.

}

name lookup

Context control : (function, variable, ...)

access control : sınıfın her elemanına erişim kontrolü

a) public : erişimde kısıtlama yok

b) private : kısıtlı, sadece sınıfın kendi kolları erişebilir.
client'larla kapalı.

c) protected : tipki private gibi clientlara kapalı.
kalıtım (inheritance) ile ulaşılabilir.

Sayıyla
toplantı

Kalıtımından bahsetmeden $\text{private} = \text{protected}$

```
class MyClass {  
    int mx; // default private tr. struct'ta tam tersi  
}
```

üye fonksiyon içinde nitelenmemiş ismin aranması : unqualified namelookup
önce kullanıldığı blok içinde (kullanıldığı yere kadar)
(bulunamaz ise) kapsayan bloklarda
(bulunamaz ise) class definition içinde
(bulunamaz ise) global scope'da aranır

```
int x=10;  
int main()  
{  
    int x=45;  
    {  
        cout << "::x" << ::x << '\n'  
    }  
}
```

$x=10$, unary prefix operator
 x 'i globalde tanımlı
sayıyor.

void func() \Rightarrow global scope

```
{  
    int func = 0;  
  
    func(); // önce blokta aranır. int func bulur. ?? neden globale bakmío ???  
}
```

namelookup bitti. ismi buldu. biter.

```
// recursive çağrı yapmak isteseydim
void func() // global isim alanı ✘
{
    int func = 0;

    ::func(); // doğru, global isim alanında arar, ve bulur
}
```

// alttakileri h. file'a koymada sorun var mı ? (iyi olup olması değil)

- ✗ int x = 10; // yanlış, ODR ihlali global değişken ✓
- ✗ static int y = 10; // yanlışlık yok, static olarak tanımlanan değişkenler internal linkage'da, her değişken ayrı bir değer olur çakışmaz, fakat tercih edilmez
- ✗ namespace { // unnamed namespace, static int y = 10; yerine kullanılır
int t = 40;
}
- ✗ inline int z = 10; // hata yok, inline variable, linker bir tane görür ✓

// alttakileri h. file koymada sorun var mı ? (iyi olup olması değil)

- ✗ const int x = 10; // hata yok, global const nesneler internal linkage'da, herbir değişken ayrı bir değer olur.
Bunu bilmiyordum.

```
ali.h
const int x = 10;
void foo();
```

```
ali.cpp
#include "ali.h"
#include <iostream>

void foo()
{
    std::cout << "&x = " << &x << "\n";
}
```

```
main.cpp
#include "ali.h"
#include <iostream>

int main()
{
    foo();
    std::cout << "&x = " << &x << "\n"; // foo ile burada yazılan
adres farklı // ise bunlar ayrı değişkenler , iç bağlantıda, demekki
bunlar ayrı değişkenler
}
```

eğer yukarıdaki x değişkeni
constexpr int x = 10; //olsa herhangi farklılık olmaz, x'ler ayrı
değişkenler
inline int x = 10; // bu durumda x'ler aynı değişkendir.

Const, constexpr durumda x'ler ayrı değişkenler
inline olunca aynı değişkenler olur.

Sınıflar (classes) :

C++ multiparadigm programlama dilidir, sadece OOP değil

sınıfları kullandık diye nesne yönelimli program yaptık anlamına gelmiyor
Paradigmadaki yaklaşım önemli

Complete type, incomplete type

class declaration (forward declaration) :

```
class Myclass; // incomplete type
```

class definition:

```
class Myclass{
```

```
    } // complete type
```

```
class Myclass{
```

```
    int mx; // data member
```

```
    void func(int); // member function
```

```
    class Nested{ // type member
```

```
};
```

```
    typedef int Word; // type member
```

```
};
```

```
class Myclass{
```

```
    int x; // her oluşturululan Myclass  
nesnesinin int x elemanı var
```

```
    // x değişkeni Myclass nesnesinin  
içinde bulunur !!!
```

```
    static int s; // sınıf nesnesinin içinde  
değil, sınıf başına tek
```

```
};
```

```
int main()
```

```
{
```

```
    int* ptr;
```

```
    ptr // lvalue
```

```
    int x = 10;
```

```
&x // ifade türü int* türü, value kategorisi pr value expression'dır
```

```
}
```



pointer başka pointer değişken başkadır.

pointer genel olarak adres anlamında kullanılan bir terim
bir adres ifadesi lvalue veya rvalue expression olabilir !!

class scope: bildirimi sınıf tanımı içinde yapılan (bir istisna hariç) isimler sınıfın elemanıdır ve bu isimler tüm isimlerde olduğu gibi bir kapsamı vardır.

namespace scope

class scope

block scope

function scope

function prototype scope

bir ismin bir sınıfın tanımı içinde aranması için şu koşullar gereklidir:

- 1) isim nokta operatörünün (member selection) dot operatorunun sağ operandı olarak kullanılmış ise
- 2) isim ok operatörünün (member selection) arrow operatorunun sağ operandı olarak kullanılmış ise
- 3) isim scope resolution operatörünün sağ operandı olarak kullanılmış ise (sol operandı bir sınıf ismi ise)

x.y

ptr -> a

Myclass::a

Gizli parametre :

```
class Myclass{  
public:  
    void func(); // member function  
};  
  
int main()  
{  
    Myclass m;  
  
    m.func(); // derleyici m nesnenin adresini bu fonksiyonun bildirimde  
    gösterilmeyen gizli parametre değişkenine argüman olarak geçiyor.  
    // c'de karşılığı func(&m), üye fonksiyon yerine global fonk. olacaktı.  
}
```

class (sınıf) ve obje(nesne) farklı

```
class MyClass{  
public:  
    void foo();  
    void func(int);  
private:  
    int a, b, c;  
};  
  
int main()  
{  
    MyClass m;  
    MyClass* pm = &m;  
  
    m.foo();  
    pm->foo();  
    (*pm).foo();  
}
```

```
int x = 120;  
  
int main()  
{  
    int x = x; // tanımsız davranış, yerel x'e  
    // çöp değeriyle ilk değer verdiğimizden  
    int x = ::x; // doğru, yerel x'e global x ile ilk  
    // değer vermek  
}
```

```
class MyClass{  
public:  
    void func(int);  
private:  
    void func(double); // overload  
};  
  
int main()  
{  
    MyClass m;  
  
    m.func(1.3); // sentax hatası  
    önce namelookup yapıldı function overload olduğu anlaşıldı,  
    private'daki func'a gidildi.  
    sonra diğer context kontrol, en son access control yapıldı.  
    ✖ Erişilmiş olan fonksiyonun access kontrolü yapıldı, private olduğundan  
    sentax hatası oldu  
}
```

bir üye fonksiyonun cpp dosyasında tanımlanmasıyla sınıfın içinde tanımlanması aynı şey değil !!!

multiple inclusion guard

```
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass {
public:
    void set(int a, int b); // fonksiyonların bildiriminde a ve b yazmana gerek yok
    // default argüman tanımda olursa bildirimde olmamalı,
    void setin (int a, int b = 0);

private:
    int m_x, m_y;
    int a_, b_, c_;
};

#endif
```

otomotiv => misra standardı

C++ core_guideliness

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>

global bir fonksiyon içerisinde nitelenmeden kullanılan isimlerin aranmasıyla üye fonksiyonların nitelenmeden kullanılan isimlerin aranması farklıdır !!!

unqualified namelookup :

üye fonksiyon içinde nitelenmemiş ismin aranması :
önce kullanıldığı blok içinde (kullanıldığı yere kadar)
(bulunamaz ise) kapsayan bloklarda
(bulunamaz ise) class definition içinde
(bulunamaz ise) global scope'da aranır

(member function)

C'de

```
struct Fighter{  
    ///  
    ///  
}  
  
// bir savaşçı diğer savaşçuya saldıracak  
void attack_enemy(Fighter *p, Fighter *);  
  
attack(&myfighter, &yourfighter); // kim kime saldırıyor ?? ne bileyim ben  
// koddan doğrudan anlaşılmaz
```

C++'da

```
myfighter.attack(yourfighter); // daha belirgin
```

```

class A
{
public:
    void foo();
private:
    // int x; // 2) x sonra burada
};

int x = 10; // 3) x sonra burada aranır

void A::foo()
{
    // int x = 20; // 1) x önce burada

    int ival = x;
}

```

Maskelemeyi aşmanın yolları:

```

void A::foo()
{
    A::x; // isim doğrudan class
scope'da aranır !! blokta veya global
isim alanında aranmaz // fakat m_x
olarak tanımlarsak maskeleme olmaz
zaten

::x; // global namespace'de aranır
}

```

 Aynı class'tan oluşturulan nesneler birbirlerinin private fonksiyon ve private verilerine erişebilirler !!! bunun sentaks hatası olduğunu düşünen çok kişi varmış

```

class A
{
public:
    void foo(A);
private:
    int m_x;
};

```

A ga;

```

void A::foo(A pa)
{
    A ax;
    pa.m_x;
    m_x = 5;
    ax.m_x = 10;
    ga.m_x;
}

```