

* Tekrar:

Nitelermenis isim, Block Scope → Class Scope → Namespace scope

- Rustta resmeler default olarak non-mutable (const)

- C'de de bir struct varsa →

```
1 struct Data {  
2     int a, b, c;  
3 };  
4  
5 void func(struct Data*); → Bu bir mutator  
6 void foo(const struct Data*); → Bu bir accessor  
7  
8 I
```

→ C'te da bunlar el
deki referans ile
yok olur.

- C++'da iki tür member function var: Non-const member function
: Const member function

```
1 class MyClass {  
2  
3     public:  
4         void func(int); → gidiş parametresi MyClass*  
5         void foo() const; → Fonksiyon bildiriminin  
6     private:  
7         int m_x; → sağda const anahtar sözcüğü  
8     }; → eklenebilir.  
9  
10    → Her bildirim hem  
11    → hem class hem de aynı.
```

foo(const MyClass*)

* Const member Function Kuraları:

- 1- Const member functionlar, o nesnenin içi elemanını değiştiremez.

```
1 class MyClass {  
2  
3     public:  
4         void func();  
5         void foo() const;  
6     private:  
7         int m_x;  
8     };  
9  
10    void MyClass::foo() const  
11    {  
12        auto val = m_x; → salt okuma  
13        m_x = 10; → amaçlı  
14    } → erişim. Bu fonksiyon  
15    → m_x = otomatik  
16    → Syntax hatası  
17  
18    void MyClass::func()  
19    {  
20        m_x = 10;  
21    }
```

* Fovet Bu Durum *
Syntax İzin

2- Const olmayan bir sınıf nesnesiyle, sınıfın const tipi fonksiyonları çağırılabilir.

p^* , const p^* 'a abisidir ama const p^* , p^* 'a abisidir.

Ancak, const bir sınıf nesnesiyle, sadece const member function çağırılabilir. Non-const member function çağrımıza gerek yok.

```
class Myclass {  
public:  
    void foo();  
    void foo()const; };  
  
void Myclass::foo()  
{  
    std::cout << "Myclass::foo()\n"; }  
  
void Myclass::foo()const  
{  
    std::cout << "Myclass::foo()const\n"; }  
  
int main()  
{  
    const Myclass cm;  
    Myclass m;  
    cm.foo();  
    m.foo(); };
```

const overriding var, eğer override tanımlanıysa, non-const olarak çağırılır.

const nesne, non-const çağırılır.

3- Const member functionlar, non-const member function çağırılmaz.

* Syntax vs. Semantic Const Member F:

```
#include <iostream>  
#include <vector>  
#include <map>  
  
class Fighter {  
public:  
    void print()const; }  
private:  
    mutable int m_debug_call_counter{};  
};  
void Fighter::print()const  
{  
    ++m_debug_call_counter; }
```

print, bu nesnenin özellğini, doğası gereğinden bir fonksiyon olsuz, Bu nesne ile semantic olarak const Nesnenin tildeğini etkilemez.

Bunun önüne geçmek için mutable eklenmiştir.

Yani bu data member'in class'in onlemeleri, objesine etkisi yok.

Fakat const member func, non-const data member'i değiştirmez, Fikirde hatalı

* Member function'in const olup olmamasının next elementiyle doğrudan ilgisi yok.
Class'ın 0 fonksiyonun çağrısızlığı, problem domaindeki anlamına yonelik bir değişim yapır, yapmaması onemlidir!!!
İşleymeye yonelik

* Const member function, 0 nesnenin stateini değiştirmemeli. State değiştirmeğen fonksiyonlar → const olarak tanımlanır.
Aynı şekilde, state değiştirmeğen data memberler, → mutable anahtar sözcüğü ile tanımlanır, yoksa, syntax error.

* This Keyword:

- Sadece sınıfın non-static type fonksiyonlarının tanımında kullanılır.
- This keyword'ü bir pointerdir. Hangi nesne için çağrıldıysa, o nesnenin adresini tutar. Diğer dillerde, class'ın kendisini temsil eder. *this

```
void func();  
void foo();  
  
private:  
    int mx, my;  
};  
  
void Myclass::foo()  
{  
    mx = 10; // Member selection  
    this->mx = 10; // Operator between  
    Myclass::mx = 10; // addr.  
}  
  
Bu iki de aynı anlarda
```

→ Kullanım Alanları: 1. Global Bir Fonksiyon, Class tarafından pointer ya da referans istedir. / Global fonksiyon argumanında

```
void gf1(Myclass*);  
void gf2(Myclass &);  
  
void Myclass::foo()  
{  
    gf1(this); // myclass içinde this  
    gf2(* this); // ile değiştir.  
}
```

2. Return Değer Dörek, çağrılan nesnenin adresini / kendisini döndürür. (Chaining)

```
7 public:
8     Myclass& func();
9     Myclass& foo();
10    Myclass& bar();
11 };
12
13 Myclass& Myclass::func()
14 {
15     std::cout << "Myclass::func cagrildi\n";
16     //code
17     return *this;
18 }
19
20 int main()
21 {
22     Myclass m;
23
24     m.func().foo().bar();
25 }
```

her
bars
myclass obiectini. Böylediile obiectler
cagirdi.