20 Nisan 2022

Üye fonksiyonlar assembly katmanında bir farklılık taşımaz.
Dil katmanında bir farklılık taşır.

Üye fonksiyon : tanımı cpp dosyasında

üye fonksiyonun sınıfın içerisinde tanımlanması ayrı bir sentax özelliği =>> sınıfın inline fonksiyonları

yanlış : cpp'de üye fonksiyonun implemente edildiği yerde class'ın private bölümüne erişim var. Ama bu fonksiyon hangi nesne için çağrıldıysa onun private elemanlarına erişebiliyorum
doğru: aynı türden her nesnenin private bölümüne erişebiliriz. !!!!

```
void func(Data &);  // mutator , nesneyi değiştiren
void(const Data&)  // accessor, nesneyi değiştirmeyecek
```

const üye fonksiyon sınıfın veri elemanlarına sadece okumak için erişebilir

1. sınıfın const üye fonksiyonları (hangi nesne için çağırılmışlarsa)
o nesnenin veri elemanlarını değiştiremezler

2. const Myclass ===> sadece const üye fonksiyon
non-const Myclass ===> const üye fonksiyon
const Myclass XXXXX non-const üye fonksiyon

T* 'dan const T*'a dönüşüm var.
const T* 'den T*'a dönüşüm yok.

const overloading
pratik olarak:
const nesne için const üye fonksiyon
const olmayan nesne için const olmayan üye fonksiyon çağırılır.

3. Bir sınıfın const üye fonksiyonu sınıfın non-const üye fonksiyonunu çağıramaz

sentax açısından const fonksiyon kullanılırsa:
derleyici gizli parametreyi Myclass *p ====> const Myclass *p yapar !!!

const correctness : const olması gereken herşey const olacak, birtane bile istisna olmayacak

this (pointer,hangi nesne için çağrılmışssa o nesnenin adresi)

üye fonksiyonun sınıfın içerisinde tanımlanması ayrı bir sentax özelliği
=>> sınıfın inline fonksiyonları

h file içerisinde tanımlanmayan fonksiyon cpp'de tanımlanamaz, sentax hatası

mutable (değiştirilebilir), immutable (değiştirilemez)

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

void func(Data &);   // mutator , nesneyi değiştiren
void func(const Data&);  // accessor, nesneyi değiştirmeyecek
```

const üye fonksiyonlar:

```
class Myclass
{
public:
    void func(M y c l a s s * p, int);
    // M y c l a s s  gizli parametredir, bu aslında görünürde yok fakat derleyiciye
göre var.
    // gizli olmasa const olup olmamasına bakacaktık
    // üye fonksiyonlar için:
    // 1) non-const fonksiyonlar
    // 2) const üye fonksiyonlar
private:
    int m_x;
};
```

```
class Myclass
{
public:
    void func(int);  // const olmayan üye fonksiyon
    void foo()const;   // const üye fonksiyon  // sınıfın veri elemanlarını
değiştiremez
private:
    int m_x;
};
```

sentax açısından üye fonksiyonda const kullanılırsa:
derleyici gizli parametreyi Myclass *p ====> const Myclass *p yapar !!!

```
void Myclass::foo()const  // sınıfın veri elemanlarını değiştiremez
{
    m_x = 10;   // sınıfın veri elemanlarına sadece okumak için erişebilir
}
```

derleyici bakışı açısından:

```
void Myclass::foo(const M y c l a s s *p)    // gizli parametre
{
    p->m_x = 10;    //sentax hatası
    auto val = m_x;  // doğru
}
```

1. sınıfın const üye fonksiyonları (hangi nesne için çağırılmışlarsa) o nesnenin veri elemanlarını değiştiremezler

sebebi: const T*'dan T*'a dönüşüm yok

```
class Myclass
{
public:
    void foo();  =>  derleyiciye göre gizli parametre değişkeni void foo(Myclass* );
private:
    int m_x;
};

int main()
{
    const Myclass x; // x değişmeyeceğim diyor
    x.foo();   // hatalı => derleyici sınıf nesnesinin adresini üye fonksiyonun gizli
     // parametre  değişkenine kopyalar foo(Myclass *  =  const Myclass *) ,
    // Myclass* p = &x;   // const'tan normale dönüşüm yok , sentax hatası
}
```

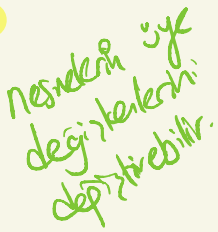
```
class Myclass
{
public:
    void foo()const;
private:
    int m_x;
};

void Myclass::foo()const
{
    m_x = 10;  // hatalı
    // derleyici  ptr->mx = 10;
}
```

```
class Myclass
{
public:
    void foo()const;
private:
    int m_x;
};

void Myclass::foo()const   // benim yorum:
const üye fonksiyon, class'in üye
değişkenlerini değiştiremez
{
    Myclass a;

    a.m_x = 23; // hata yok
}
```



2. const olmayan ya da const olan bir sınıf nesnesi ile sınıfın const üye fonksiyonları çağırılabilirler.
Ancak const bir sınıf nesnesi ile sadece sınıfın const üye fonksiyonları çağırılabilir.
Ancak const bir sınıf nesnesi ile sınıfın non-const üye fonksiyonları çağırılması geçersizdir.

özet:
const Myclass ===> sadece const üye fonksiyon
non-const Myclass ===> const üye fonksiyon
const Myclass XXXXX non-const üye fonksiyon

T* 'dan const T*'a dönüşüm var.
const T* 'den T*'a dönüşüm yok.

const overloading

```
class Myclass
{
public:
    void foo()const;
    void foo();   // overload
private:
    int m_x;
};

int main()
{
    const Myclass cm;
    Myclass m;

    cm.foo();   // void Myclass::foo()const çağırılır
    m.foo();  // void Myclass::foo() çağırılır
}
```

pratik olarak:
const nesne için const üye fonksiyon
const olmayan nesne için const olmayan üye fonksiyon çağırılır.

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

void Myclass::func(int x)
{

}

void Myclass::func()
{
    func(10);   // func ismi blokta bulunamazsa class scope'da aranacak, func()
bulunacak
    //  isim arama biter. Class scope global scope'u maskeledi.
    //  sentax hatası olur. Parametresi olmayan fonksiyona parametresiz argüman
    // gönderiyorsun hatası

    func();  // class'taki func() çağrılır, recursive fonk çağrısı olur.
    // func'ın kendi kendini çağırması
}
```

3. Bir sınıfın const üye fonksiyonu sınıfın non-const üye fonksiyonunu çağıramaz

```
class Myclass{
public:
    void func();
    void foo()const;
};

void Myclass::func()  gizli parametre => Myclass*
{
    foo();  // okay,  gizli parametre => const Myclass*
}  //   const Myclass* = Myclass*

void Myclass::foo()const    gizli parametre => const Myclass*
{
    func();  // sentax hatası , gizli parametre =>  Myclass*
}  //  Myclass* = const Myclass*
```

sınıf içerden dışarıya oluşturmak en büyük hata
sınıf dışarıdan içeriye tasarlanır.
O sınıfı kullanacak onların alacağı hizmetleri belirlenir.

Semantik ve sentaks farkı

```
class Fighter{

public:
    void print()const;
private:
    //
    int m_debug_call_counter{};
    // üye fonksiyon ne kadar çağırılmış onu tutacak
    // bu değerin değişmesi fighter'ın state değişmesi anlamına gelmez

};

void Fighter::print()const
{
    ++m_debug_call_counter;  // semantic sentaks uyuşmazlığı
    // semantic : evet bu const üye fonksiyon, ama  m_debug_call_counter'ı
değiştirmesi
    // normal birşey, çünkü Fighter nesnesinin state'i ile hiçbir alakası yok
    // sentaks : yassak diyo, parametresi const Fighter*
    // böyle bir pointerin gösterdiği nesnenin elemanını değiştiremez
}
```

Çözüm

mutable : semantic sentaks uyuşmazlığı

```
class Fighter{

public:
    void print()const;
private:
    //
    mutable int m_debug_call_counter{};
};

void Fighter::print()const
{
    ++m_debug_call_counter;
}
```

const correctness : const olması gereken herşey const olacak, birtane bile istisna olmayacak

```
class Circle {
public:
    double get_area();  // const yapılmalı !!!
    // yalancı bir fonsiyon
    // nesneyi değiştiriyorum diyor !!

}

void func(const Circle &p)  // üstteki duruma uygun olmaz
{
    auto area = p.get_area();  // sentaks hatası, const bir nesne ile sınıfın const
    // olmayan bir üye fonksiyonu çağırılır
}
```

this pointer

SORU: Üye elemanların nesnenin elemanlarına ulaşımı varken neden this pointer'ına ihtiyaç duyulur ?

Cevap: maskelemeyi aşmak için. Sürekli this kullanımı çok cirkin.

```
class Myclass {
public:
    void func();
    void foo();
private:
    int mx, my;
};

void gf1(Myclass *);  // bu global fonksiyon, Myclass
nesnesinin adresini ister
void gf2(Myclass &);

void Myclass::foo()
{
     // burada gf'ye çağrı yap
    gf1(this);  // this olmasa bu çağrı yapılamaz, çünkü
pointer gizli
    *this;  =>  this bir pointer ise gösterdiği nesne *this'dir
    // diğer dillerde this direk gösterdiği class, C++'da pointer
    gf2(*this);
}
```

chaining (zincirleme)

```
class Myclass {
public:
    Myclass& func();   // hangi nesne için çağrılmışsa o nesnenin adresini döndürsün
    Myclass& foo();
    Myclass& bar();
private:
    int mx, my;
};

Myclass&  Myclass::func()
{
    std::cout << "Myclass::func cagrildi\n";
    // code
    return *this;
}

int main()
{
    Myclass m;
    m.func().foo().bar();
}
```

o nesneyi döndürür

// aynı nesne ise adresleri aynı olmalı

this : oluşturduğu ifade pr value'dur.
*this : lvalue

```
void A::func()
{
    A a;
    this = &a;  // hatalı
    *this = a;  // doğru
}
```

```
class A{

    public:
        void func()const;
        void foo();
};

void A::func()const
{
    this->foo();  // sentax hatası,
    this; // const T*
}

void A::foo()
{
    this;  // T*
}
```

```
class A{

    public:
        A *func()const;
};

A* A::func()const
{
    //
    return this;  // sentax hatası,  çünkü döndürdüğü nesnede const T*'dan T*'a
dönüşüm
}
```

olması gereken:

```
class A{

    public:
        const A *func()const;
};

const A* A::func()const
{
    //
    return this;
}
```

asla ve asla const bir nesneyi tür değiştirme operatörleri ile kullanmayın (const_cast)

yanlış ===> bir fonksiyonun bir tane return deyimi olmalı !!

inline fonksiyonlar
==========

inline expansion: derleyicilerin gerçekleştirdiği en fayda sağlayan optimizasyon tekniklerinden biridir. Derleyici bir noktada bir fonk'nun çağırıldığını görür. Derleyici normalde fonk. giriş çıkış , argümanların kopyalanması vs. gibi işler yapar fonksiyonun derlenmiş kodunu yerleştireyim diyor.

şart: a) fonksiyonun tanımını görmeli
b) verim açısından bir fayda sağlanacak mı ?
c) teknik olarak mümkün mü ?

inline derleyiciye bir öneri , fonksiyonu inline yapmak zorunda değil

```
inline int func(int x)
{
    return x*x  + 5;
}
```