* Tekrar: - Eger tanıtılmadıysa, compiler default constructor default eder. Fakat biz default olmayan bir constructor yazarsak Compiler Default etmez.

* Constructor Initialize List:

```cpp
class Myclass {

public:
    Myclass();
private:
    int mx, my;  // ─> Hayata gelme sırası, bildirilme sırasıdır.
};

//.cpp

Myclass::Myclass() : mx{ 10 }, my{ 20 }
{
}
```
constructor initialize list'de
sıra ÖNEMLİ DEĞİLDİR.

* Class constructor'a gelmeden, eger değer atanmadıysa, default olarak init edilir. (çöp değer)

* Ancak CONST NESNELER Default init yapılamaz.

* Her zaman ilk tercih bu olmalı.(Eger mümkünse) Çünkü efficiency → Default init'leyip atama yapmaması hızlı.

* Sınıfın data member'i const ya da referans ise, constructor initialize list ZORUNLU

* In-class / Default Member Initialization:

```cpp
//in-class initializer
//default member initializer

class Myclass {

private:
    int mx{ 10 };    // ─> Fakat burada Direct-Initialization "( )"
    int my = 20;     //     syntax hatasıdır.
};
```
parantezli
değer atanamaz.

=> Fakat Burada fiilen ilk değer vermiyoruz.

=> Bunla compiler'a örtülü olarak, constructor init. list'e ekletiyoruz. Eger constructorda değer olmazsa.

=> Eger hem default member init, hem constructor init list varsa, Constructor init list dikkate alınır.

* Delegating Constructor:

-> Bir sınıfın constructor'ı data member initlenmek için başka constructorları çağırabilir. Buna delegating constructor denir.

-> Delegating Constructor varsa başka constructor init list kullanılamaz!!
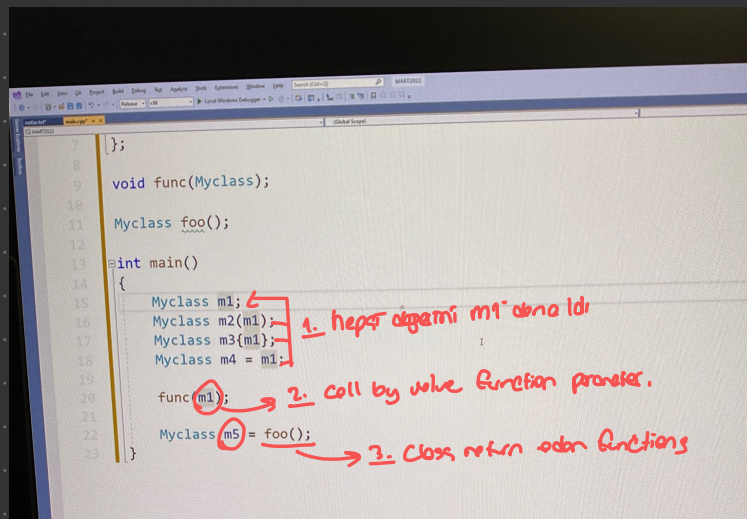
* Special Member Functions:

- defaut ctor
- destructor
- copy ctor ─┐
- move ctro ──┼──→ copy members
- copy assignent ──→ move members
- move assignment

→ Özel koşullarda compiler bunları kendi **default eder**. Biz bildirmesek de.

→ Eğer derleyici sınıfın bir özel üye fonksiyonunu **default edecekse** fakat derleyicinin özel üye fonksiyonu oluşturma sürecinde syntax hatası oluşursa derleyici özel üye fonksiyonu **delete eder**. Priv erişim, const, ref erişimi gibi...
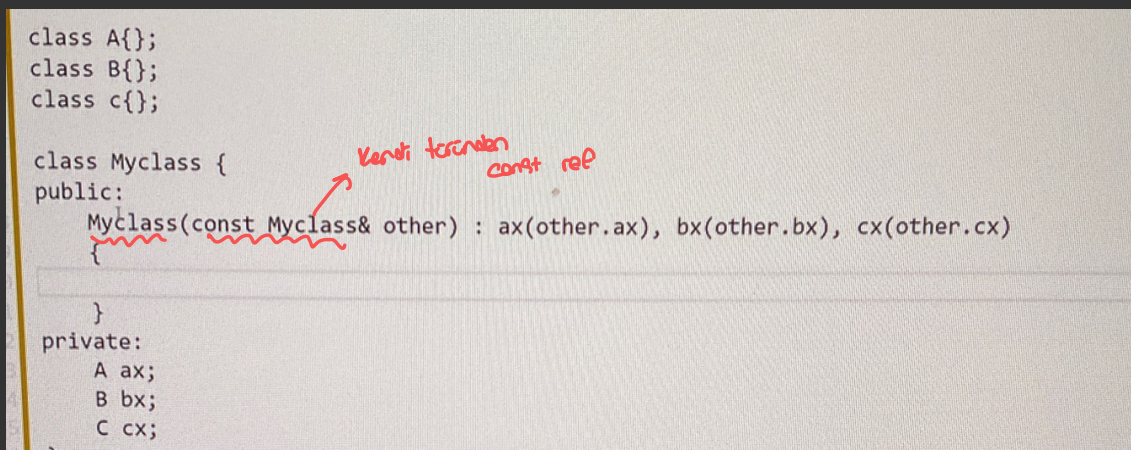
— Copy Constructor:

- Hayata değerini **ayrı türden bir başka sınıf** nesnesinden alarak geliyor.
- Kullanıldığı senaryolar:


```
};

void func(Myclass);

Myclass foo();

int main()
{
    Myclass m1;
    Myclass m2(m1);      1. hayat değerini m1' alana ldı
    Myclass m3{m1};
    Myclass m4 = m1;

    func(m1);      → 2. call by value function paraneter.

    Myclass m5 = foo();
                         → 3. Class return eden functions
}
```

> **Rule of Zero**: Bütün special member fonksiyonların, compiler tarafından **Default edilmesi**

- Non-static, inline, public olarak **default** eder compiler.

```
class A{};
class B{};
class c{};

class Myclass {
public:
    Myclass(const Myclass& other) : ax(other.ax), bx(other.bx), cx(other.cx)
    {

    }
private:
    A ax;
    B bx;
    C cx;
```
Kendi türünden const ref

- Sınıfın veri elemanlarından biri Eğer **POINTERSA / Referanssa** → COPY CTOR, o pointerin değerini kopyala.
  → Yani aynı nesneye işaret eder - Bu dangling pointer'a sebebiyet verebilir.

- Shallow Copy: Kaynağı değil, kaynağı gösteren pointer'in kopyalanması
- Deep Copy: Kaynağın kendisini kopyalar.

- RAII: Resource Aquistion is Initialization
  - → Idiomatik yap.
  - → Bir sınıf nesnesi ısıkunı yerine getirirken kaynak edinmeli.

→ tekrar izle!!! son 15 dk.

```
1274
1275    class Myclass {
1276    public:
1227        Myclass(const Myclass& other : ax(other.ax), bx(other.bx), cx(other.cx)
1278        {
1279        }
1280
1281    private:
128.        A ax;
128         B bx;
1284        C cx;
1285    };
```

=) Shallow Copy
   yapan bir C'tor