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

* Constructor Initialize List:

```
class Myclass {

public:
    Myclass();
private:
    int mx, my;
};

//.cpp

Myclass::Myclass() : mx{ 10 }, my{ 20 }
{
}
```

→ Yazma sırası, bildirime sıradır.

{} yerine () de kullanılabilir!

constructor initialize listesi sıra ÖNEMLİ DEĞİLDİR.

* Class members, class constructor bloğuna girmeden initialize edilir. Eğer bir değer yoksa default init, garbage!

* Ancak CONST NESNELER Default init yapılamaz.

* Her zaman ilk tercih bu olmalı. (Eğer mümkünse) Çünkü efficiency → Default initleyip, atama yapmaktan hızlı.

* Constructor içinde yapılan initialize değil, assignmenttır!

* Sınıfın data member'ı const ya da referans ise, constructor initializer list ZORUNLU

* In-class / Default Member Initialization:

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

class Myclass {

private:
    int mx{ 10 };
    int my = 20;
};
```

Fakat burada Direct-Initialization "()" syntax hatasıdır.

parantez değer atanamaz,

=> Fakat Burada fiilen ilk değer vermiyoruz.

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

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

* Delegating Constructor:

→ Bir sınıfın constructor'ı data member initialize için başka constructorları çağırabilir. Buna delegating constructor denir.

→ Delegating Constructor varsa başka constructor init list kullanılamaz!!

* Special Member Functions:

- default ctor
- destructor
- copy ctor
- move ctor
- copy assignment
- move assignment

→ copy members
→ move members

→ User Declared Special member Functions:

```
class Nec {

public:
    //Nec(); //user-declared (define)
    //Nec() = default;  //user declared - defaulted
    Nec() = delete;
};
```

```
class Nec {

public:
    Nec(int);
    Nec() = default;
};
```

Farklı constructor tanıttığımız için default ctor not Declared oldu!

Şimdi default ctor'u compiler üretebilir edildi.

→ Özel hallerde compiler bunları kendi default eder. Biz bildirmesek de.

→ Eğer derleyici sınıfın bir özel üye fonksiyonunu default ederse fakat derleyicinin özel üye fonksiyonu oluşturma sürecinde syntax hatası oluşursa derleyici özel üye fonksiyonu delete eder. Priv erişim, const, ref erişimi gibi... !!!

```
class Myclass {
private:
    const int x;
};

int main()
{
    Myclass x;
}
```

Const member init edilmediği için syntax hatası

↓

Bu yüzden compiler, default constructor'ı delete etti!

```
class A {
public:
    A(int);
};

class Myclass {
private:
    A m_a;
};
```

- m_a için default ctor çağrılamaz. Not Declared!
- Bu yüzden Myclass sınıfının da Default Ctor'ı deleted!

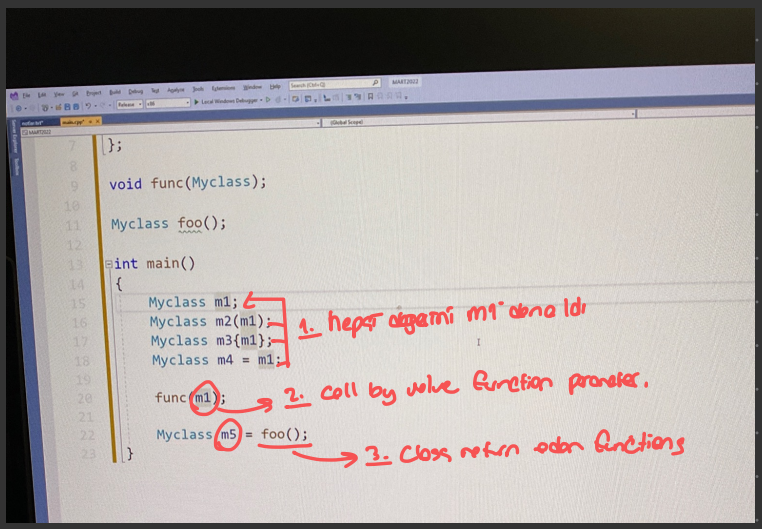
```
class A {
    A();
public:
};

class Myclass {
private:
    Myclass() = default;
    A m_a;
};


int main()
{
    Myclass m;
}
```

* Myclass'ın default constructor'ı deleted!

Bildirmiş olsak bile deleted! Çünkü A'nın constructor'ı private!

- Copy Constructor:

- Kopyala, değerini aynı türden bir başka sınıf nesnesinden alarak geliyor.
- Kullanıldığı senaryolar:



- **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 pointerın değerini kopyala.
  ↳ Yani aynı nesneye işaret eder - Bu dangling pointer'a sebebiyet verebilir.

- Shallow Copy: Kaynağı değil, kaynağı gösteren pointer'ın kopyalanması
- Deep Copy: Kaynağın kendisini kopyalar. Kendimiz yeni bir kaynak yaratırız! Yaratılan bu yeni kaynağa, oluşturduğumuz yeni pointer point eder!
  - → Kaynağı da kopyaladığımız kaynak ile init ederiz!

- RAII: Resource Aquisition is Initialization
  - → Otomatik yapı
  - → Bir sınıf nesnesi ilkelini yerine getirebilmek için, bir kaynak edinmesi gerekliliği durumda, edinilen kaynağın geri verilmesi gerekir!
    - → constructor dosya açar, destructor o dosyayı kapatır

```
class Sentence {
public:
    Sentence(const char* p) : m_len{std::strlen(p)},
    m_p {static_cast<char *>(std::malloc(m_len + 1))}
    {
        if (!m_p) {
            std::cerr << "bellek yetersiz\n";
            std::exit(EXIT_FAILURE);
        }
    }   → std::strcpy(m_p, p);

    void print()const
    {
        std::cout << "[" << m_p << "]\n"
    }

    ~Sentence()
    {
        std::free(m_p);
    }

private:
    std::size_t m_len;
    char* m_p;
};
```

```
void func(Sentence s)
{
    s.print();
}

int main()
{
    Sentence s1 = "Bugun hava cok guzeldi";

    s1.print();
    func(s1);
    (void)getchar();

    s1.print();
}
```

Burada kopyalama var!

Fonksiyon parametresi, sınıf türünden, bu durumda sınıfa gönderilen her nesne, compiler'ın yazdığı copy ctor ile kopyalanır, function scope sonunda da destructor'ı çağırılır.

Fakat func fonksiyonu ile alınan nesne, shallow copy ile bizim nesnemizden kopyalanacak! Aynı kaynağı gösterecekler. Fonksiyon sonunda silinen orijinalden ötürü, bizim asıl sınıfımızın da pointer'ı artık dangling!