Bir sınıfın bir çok constructor'ı olabilir.
Geri dönüş değeri yok.
Argüman gönderilmeden çağrılabilen constructor'a default constructor(special member function, derleyici kendisi yazar) deniyordu.
Default constructor'ı biz yazmaksak derleyici yazar. Biz yazarsak derleyici yazmaz.
Default constructor, special member function'dır.

```
class Nec{  // implicitly default constructor tanımlar
    public:
};

class Nec{  //  default constructor'ı yok !!!
    public:
        Nec(int);
};

class Nec{
public:
    Nec();
    Nec(int);
    void foo();
};

int main()
{
    Nec mynec;
    mynec.Nec();  // sentax hatası, contructor fonksiyonlar isimleri ile çağrılamaz
    // destructor fonksiyonlar isimleri ile çağrılabilir, ama tercih edilmez
}
```

class hayata geldiğinde sınıfın elemanları da initialize ediliyor

```
class Myclass{
public:
    Myclass();
private:
    int mx, my;  // ne zaman hayata geliyorlar ? constructor ana bloğuna girmeden önce
};

// .cpp
Myclass::Myclass()
{
    mx= 10;  // hayata geldikten sonra değer verme, initialize etme değil !!
    my = 20;  // assignment
}
```

Constructor Initializer List: (eskiden kullanılan member initializer list)

```
class Myclass{
public:
    Myclass();
private:
    int mx, my;  // üye değişkenler sınıf içerisinde yapılan bildirim sırasına göre hayata gelirler
};

// .cpp
 Myclass::Myclass() : mx{10}, my{20}   // veya Myclass::Myclass() : mx(10), my(20)
kullanılabilir
{

}
```

```
 Myclass::Myclass() : my{10}, mx{20}
{
  // uygun, hayata gelme sırası önce mx hayata gelecek !! sınıf bildirim sırası önemli

}
```

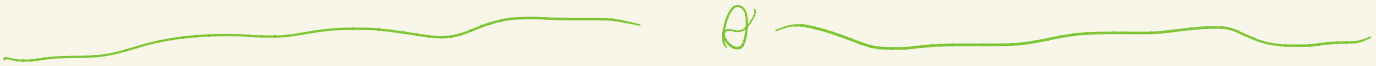
const ve & 'ler default inialize edilemez

Değişkenlere ilk değer vermek zorunda mıyım ? Hayır

```
class Myclass{
public:
    Myclass()
    {

    }
private:
    const int mx;  // const default init. edilemez, hata verir
};
```

çözüm:

```
class Myclass{
public:
    Myclass() : mx{10}
    {

    }
private:
    const int mx;
};
```



```
class Myclass{
public:
    Myclass()
    {

    }
private:
     int& mr;   // default init. olur, sentax hatası
};
```

Bazı durumlarda MIL (member init. list) zorunlu
örneğin sınıfın veri elemanının bir referans olması
örneğin sınıfın veri elemanın const olması

10 kişiden 9'unun yanlış öğrendiği yer

// default member initializer (in-class initializer)

yanıltıcı bir isimlendirme, ilk değer verdiği yok, ilk değeri constructor verir.
sayın derleyici constructorlarından herhangi birinde MIL ile initialize etmezsen sen bunu koda ekleyeceksin.

```
class Myclass{
public:

private:
    int mx = 10;    //  doğru
    int my = 20;   //  doğru
};
```

```
class Myclass{
public:

private:
    int mx (10);   // hatalı
    int my {20};  //  doğru
};
```

```
class Myclass{
public:
    Myclass();
private:
    int mx{20};
};

Myclass::Myclass()  // burayı yazmış kabul ediyoruz  : mx{20}
{
};
```

```
class Myclass{
public:
    Myclass();
private:
    int mx{10};
};

Myclass::Myclass() : mx{20}  // doğrudan 20 değeri ile hayata gelir !!!
{
};
```

eğer ben default constructor kullanmasam ve derleyici bu programının default contructorını default etseydi, derleyici tüm elemanları default initialize eder

```
class Myclass{
public:
    void print()const
    {
        std::cout << mx << "\n";
    }
private:
    int mx;   /// int mx{233}  burada default member initializer
};

int main(){
    Myclass m;  // derleyici tüm elemanları default initialize eder, garbage value ile

    m.print();
}
```

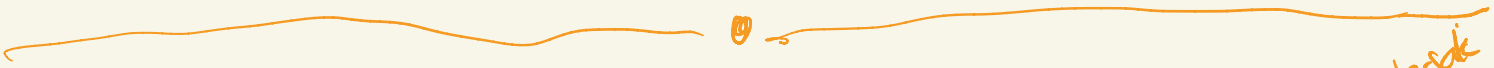
kural : Constructor Initializer List (CIL) birinci seçiminiz olsun

```
class Person{
public :

Person(const char *p) // parametre cstring, null terminated byte stream
{                                   //          beklenir
       m_name = p;
}
private:
    std::string m_name;
};

int main()
{
    Person p{"Oğuzhan Altun"};  // önce m_name'e yapılan default init. önce, sonra Person init ile alır sebebi constructor init list kullanmamamız, eğer kullanılsa bir kere sadece Person init çağırılır
}
```

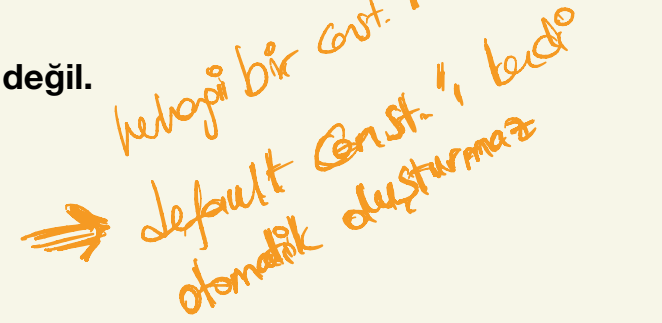
akış : önce string sınıfının const char* parametreli constructor'ı çağrılacak, sonra programın akışı constructor'ın ana bloğuna girecek, default contructor'la hayata getirilmiş string nesnesine atama yapılacak (atama operator fonksiyonu çağrılacak.)

VERiM İÇİN

```
class Person{
public :
    Person(const char *p) : m_name{p}   //  verim yüksek, bir kere init ediyor
    {
        m_name = p;  // atama yapılacak ayrıca, verim az
    }
private:
    std::string m_name;
};

int main()
{
    Person p{"Oğuzhan Altun"};
}
```



Bir sınıfın default constructor'ı olmak zorunda değil.




```
class Nec{
public:
    Nec(int);  //default constructor yok, sorun yok
}
```

fakat senaryoya göre ihtiyaç olabilir

```
class Nec{
public:
    Nec(int);    // böyle constructor tanımlanınca default da tanımlanması zorunlu olur
}

int main()
{
    Nec x; // hata !!  init edilecek fakat default contructor yok
}
```

```
class Nec{
public:
    Nec( ) = delete; // default constructor var ama delete edilmiş yani onu çağırmak sentax hatası
    Nec(int);
}


int main()
{
    Nec mynec;   hata!
}
```

Tavsiye : sınıfınıza default const. yazın. (bilinçli yapılma haricinde)
default const. olmak zorunda değil ama yazılsa iyi olur.

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

int main()
{
    std::vector<Nec> myvec(20); // hata, default const. gerekir !! 20 eleman için
}
```

delagating constructor

C++ geç eklendi.
Bir sınıfın constructor'ı sınıfın bir başka constructurını çağırabilir.

```
class Nec{
public:
    Nec(int x, int y, int z) : m_a{x}, m_b{y}, m_c{z} {}
    Nec(int x) : Nec(x,0,0) {}  // delegating constructor
private:
    int m_a, m_b, m_c
};
```

special member function :
derleyici tarafından kodları yazılabilen ve bizim bilerek isteyerek kodun derleyici tarafından (default yazarak) yazılmasını sağlayabileceğimiz fonksiyonlardır.
==========

default ctor: Derleyici eğer bizim için bir default contructor yazıyorsa sınıfın veri elemanlarını default initialize eder.
destructor
copy ctor
move ctor (C++ 11)
copy assignment
move assignment (C++ 11)

special member function

```
class Nec{
public:
    // Nec();   // user declared  (will be defined)
    // Nec() = default;  // user declared -defaulted // ben bildiriyorum fakat tanımlamasını derleyiciye bırakıyorum
    Nec() = delete; // user declared
private:
    int m_a, m_b, m_c
};
```

default : sadece special member function'larda kullanılır
delete : her fonksiyona kullanılır

```
class Nec{
public:
     Nec(int);  // sadece bu olsa default const. olmaz
    Nec() = default;  // default const.  olur fakat default eklersem ben bildirdim, derleyici sen yaz demektir
};
```

special member function (derleyici tarafından kodları yazılabilen, default sentaxı ile belirtebileceğimiz fonksiyonlar)

derleyicinin bizim için yazacağı constructor sınıfın elemanlarını init. eder.

***Eğer derleyici sınıfın bir özel üye fonksiyonunu default edecek ise fakat derleyicinin özel üye fonksiyonu oluşturması sürecinde bir sentax hatası oluşursa derleyici default edeceği özel üye fonksiyonu delete eder. (yani default contructor delete edilir)**

örnek:

```
class Myclass{

private:
    const int x;
    // sentax hatası yok , default const. var fakat derleyici tarafından delete edilir
};

int main()
{
    Myclass x;
    // sentax hatası olur. Delete edilmiş default const. çağrı yapmak sentax hatası
}
```

// programcılar, Myclass attempting to deleted function hatası alınca delete etmedim ki diye şaşırıyor. Delete etmedim ki diyorlar.

Derleyici nasıl default ediyor default constructor'ı ?
Sınıfın bütün elemanlarını default initialize eder.
Peki const bir nesnenin default initialize edilmesi legal mi ? değil !!
Bu durumda özel üye fonksiyonu sınıfın default constructor'ını delete edecek.

```
class A{

public:
    A(int);
};

class Myclass {

private:
    A m_a;  // hata yok şimdilik, bu sınıfın delete edilmiş default contructor'ı var.
};
```

// Myclass sınıfının constructor'ını yazmadığım için derleyici bu sınıfın default contructor'ını default edecek, nasıl default edecek ?
eleman olan m_a isimli A türünden nesne için default contructor çağıracak
bu geçersiz çünkü A sınıfının default contructor'ı yok .

```
int main()
{
    Myclass m;  // hatalı, delete edilmiş fonksiyona çağrı
}
```

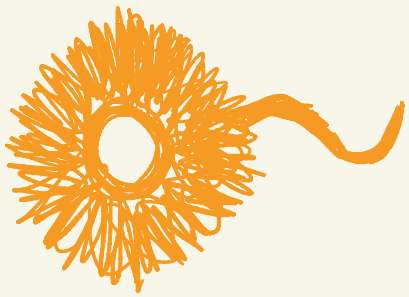
```
class A{

private:
    A();  // default const.  private olursa
};

class Myclass {
private:
    A m_a;
    // default const. çağrı yapar ama private olduğundan default const. delete
eder
};


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

COPY CONSTRUCTOR (kopyalayan kurucu işlev)

Myclass(const Myclass&)

sınıf nesnesi hayata değerini bir başka aynı türden sınıftan alarak hayata başlar. Bu durumda hayata başlayan nesne için çağırılan default const. değil copy constructor.

```
class Myclass{
    //
};

void func(Myclass);

Myclass foo();

int main()
{
    Myclass m1;
    Myclass m2(m1);  // copy constructor
    Myclass m3 {m1}; // copy constructor
    Myclass m4 = m1;  // copy constructor

    func(m1);  // call by value, bir sınıf nesnesi ile çağrı yapılır,  copy constructor çağrılır !!!

    Myclass m5 = foo(); // copy constructor çağrılır !!!
}
```




```
class Myclass{
public:
    Myclass()
    {
        std:: cout << "Myclass default ctor .. this = " << this  << "\n";
    }
    ~Myclass()
    {
        std::cout << "Myclass destructor.. this << this << "\n";
    }
};

void func(Myclass m)
{
        std::cout << " func cagirildi\n";
}
```

```
int main()
{
    Myclass mx;  // main bloğu sonunda yok edilir.  fakat yok edilen nesne fonksiyonun ana parametre değişkeni için ve sonrasında mx için çağırılır.

    func(mx);  // burada fonksiyon copy constructor ile çağırılır, fakat func bloğunun sonunda destructor çağırılır

    (void) getchar();
}
```

baya karışık!!!...

rule of zero (sıfır kuralı): belki sınıfın %90'ı böyle yazılacak .

eğer derleyicinin yazacağı özel üye fonksiyonlarının derleyicinin yazmasına bırakılması ,müdahale edilmemesi.
bu tercih edilmeli !!!!

derleyicinin yazdığı copy constructor
sınıfın
non-static,
public ve
inline üye fonksiyonudur.

```
class A{};
class B{};
class C{};

class Myclass {
public:  // derleyicinin yazdığı copy constructor  // non static veri elemanlarını initalize eder
    Myclass(const Myclass& other) : ax(other.ax), bx(other.bx), cx(other.cx)
    {

    }
private:
    A ax;
    B bx;
    C cx;
};
```

kendimiz yazdığı copy constructor

```
class Date{
public:
    Date(int d, int m, int y) : m_day{d}, m_mon{m}, m_year{y} { }
    void print()const
    {
        std::cout << m_day << '/' << m_mon << '/' << m_year << '\n';
    }
private:
    int m_day, m_mon, m_year;
};

int main()
{
    Date today{27, 4, 2022};
    today.print();

    Date x{today};  // derleyici tarafından yazılan copy constructor'ına güvenerek bunu yazabiliriz
    x.print();
}
```

sınıf handle deyince pointer veya referans anlaşılır.

Pratikte: sınıfınızın veri elemanlarınızdan biri referans veya pointer ise derleyicinin yazdığı copy constructor sadece pointerların değerini kopyalar, pointerların aynı kaynağı göstermesi demek. O kaynak değiştirildiğinde kopyalayan için de değişir.
Örneğin Nesne bir dosyayı kapatır, diğer nesne o dosyayı kullanmak isterse hata olur. Öyle bir copy constructor yazmalıyım ki pointer'ların gösterdiği nesneleri kopyalamalı.

shallow copy (sığ): kaynağı değil de kaynağı gösteren pointer'i kopyala

deep copy (kaynağın kendisinin kopyalanması): kaynağın kendisinin kopyalanması demek. Kendiniz yeni bir kaynak oluşturacaksınız. O kaynak diğer nesnenin kaynağını kopyalayarak oluşturulacak.

Soru: C++'da en sık kullanılan idiomatic yapı nedir ?

RAII acronym (resource acquisition is initialization, kaynak edinimi ilk değer verme ile olur) bir sınıf nesnesi işlevini yerine getirebilmek için bir kaynak ediniyor (bellek alanı v.s), fakat nesnelerin de bir hayatı var
Bu edinilen kaynağı kim geri verecek. Destructor

```
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);
        }
     }

    void print()const
    {
         std::cout << m_p;
    }

    ~Sentence()
    {
        std::free(m_p);
    }
private:
    std::size_t m_len;
    char* m_p;  // heap bellek alanı
};

void func(Sentence s)  // derleyicinin yazdığı copy constr. çağırılır. s1 içindeki pointer dangling
pointer olur.
{
    s.print();
}

int main()
{
    Sentence s1 ="Bugun hava cok guzeldi";
    s1.print();
    func(s1);
    (void) getchar();

    s1.print();
}
```

Anlaması Zor! ☺