

Tekrar: • Shallow Copy ile kopyalanan pointer / referanslar aynı nesneye işaret eder.

```
1287 class MyClass {
1288
1289 public:
1290     MyClass() default;
1291     MyClass(int);
1292 }
1293
```

↳ biz default olmayan bir ctor yazdığımız durumda compiler default ctor oluşturmaz. Ama biz yine de istiyorsak

* Yeni handleleri kopylamak için, compiler copy ctor'u yerine deep copy yapan kendi copy ctor'una. Değer değeri, yeni bir memory alanı oluşturulur.
↳ pointers
refs.

* Copy Assignment: Nesneler birbirine atıldığında

• Bu bir ctor değeri. Return değeri var = *this → MyClass (my adres ref)

```
1301 class MyClass {
1302 public:
1303     MyClass& operator=(const MyClass&);
1304 }
1305
1306
1307
```

• a'ın değeri b'ye atanıyor.
a.operator = (b)

• Bu da shallow copy yapıyor.

```
int main()
{
    Sentence sen_1{ "bugun hava cok guzel!" };

    {
        Sentence sen_2{ "umarim yarin da guzel olur" };
        sen_2 = sen_1;
    }
    (void) getchar();
    sen_1.print();
}
```

→ Bu durumda compiler'ın copy ctor'u

• C++ dilinde atama operatörleri ile oluşturulan ifadelerin değeri intigerleri 1 value expression

• Bir sınıfın copy ctor'ı: bir sınıf nesnesi, kopyası, değerini aynı sınıf türünden alın bir nesne ile çoğaltıldığını gösterir.

• Bir sınıf nesnesine Atama Operatör'ü, aynı türden bir sınıf nesnesi atıldığında, bunu gerçekleştirir. **Assignment**.

→ minör Özet:

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3 #include <iostream>
4 #include <cstring>
5 #include <cstdlib>
6
7
8 class MyClass {
9 public:
10     MyClass(); → default ctor
11     ~MyClass(); → destructor → release resource upon destroy!
12     MyClass(const MyClass&); → copy ctor → deep copy yapmamız gerekir!
13     MyClass& operator=(const MyClass&); → copy assignment → rel
14 }; → shallow copy by default → rel resource and deep copy yapmamız gerekir!
```

* Eğer kopyalayan atama operatör fonksiyonunu kendimiz yazarsak, (ve o klonatır gibi yaparsak copy-swap yaparız)

↳ self assign'a karşı kontrol et

* Big 3 / Rule of 3:

destructor
copy ctor
copy assignment

* Big 5: → Yeni ekler

+ move assignment
+ move constructor.

* Move Semantisi:

• Bir nesnenin yok edilmesine değil, kaynaklarını vermek yerine, onun değerini kopyala gelecekte nesnenin, kaynak kopyalanacak yerine, o kaynağı vermesi.

• R value referanslar önce geldi.

```
void func(const int&);
void func(int&&);
int main()
{
}
```

→ aynı şekilde →

```
class MyClass {
};

void func(const MyClass&);
void func(MyClass &&);
int main()
{
}
```



```
class MyClass {
```

```
public:
```

```
MyClass(const MyClass &);
```

const class ref = l value = copy constructor.

```
MyClass(MyClass &&);
```

l value = move constructor.

```
}
```

```
class MyClass {
```

```
public:
```

```
MyClass(const MyClass &);
```

```
MyClass(MyClass &&);
```

```
MyClass& operator=(const MyClass&);
```

copy assignment

```
MyClass& operator=(MyClass&&); //move assignment
```

move assignment

```
}
```

```
class MyClass {
```

```
};
```

```
int main()
```

```
{
```

```
MyClass m1;
```

```
MyClass m2 = m1;
```

l value of m1 is 700
copy ctor.

```
MyClass m1;
```

```
MyClass m2 = static_cast<MyClass&&>(m1);
```

Bu static cast karte hai ki l value ko move const class mein.

Arre dono hai!

```
MyClass m1;
```

```
MyClass m2 = std::move(m1);
```

l value ko l value mein.

Bu move karte hai, l value ko!!

Note of m2:

```
class MyClass {
```

```
public:
```

```
MyClass(const MyClass &other) : ax(other.ax), bx(other.bx), cx(other.cx)
```

```
{
```

```
}
```

```
MyClass& operator=(const MyClass& other)
```

```
{
```

```
ax = other.ax;
```

```
bx = other.bx;
```

```
cx = other.cx;
```

return *this

```
}
```

```
private:
```

```
A ax;
```

```
B bx;
```

```
C cx;
```

```
}
```

Compiler's copy constructor.

Compiler's copy assignment

*Derleyicinin oluştırdığı Move Semaantiği:

```
//move ctor
Myclass(Myclass &other) : ax(std::move(other.ax)), bx(std::move(other.bx)), cx(std::move(other.cx))
{
}

Myclass& operator=(Myclass &&other)
{
    ax = std::move(other.ax);
    bx = std::move(other.bx);
    cx = std::move(other.cx);

    return *this;
}
```

→ Compiler's copy assignment

Derleyici hangi durumlarda sınıfın hangi özel üye fonksiyonlarını
bir sınıfın bir özel üye fonksiyonu aşağıdaki 3 durumdan birinde olabilir

1. yok

2. user declared (programcı tarafından bildirilmiş)

Myclass();

Myclass() = default; → Ya da hiç tanımlanmış değil.

Myclass() = delete; → kura uygun değil, hata
→ Default = kendi oluşturması

→ derleyicinin varsıfı
3. implicitly declared
defaulted
deleted