→ ilk 1 saat format örneği

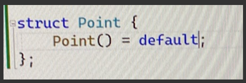
* Aggragate:

→ Birleşik Türler

- Non static data memberlarını doğrudan kullanıma açan sınıflar! Data hiding yok! Invariant kontrol sorumluluğu yok!
- Aynı türden nesnelerin oluşturduğu bir grup / composition!
- std::string bir aggregate değil! Ama string dizi bir aggregate!

*** Aggregate Olma Koşulları:**

1. User declared constructor olamaz ✗

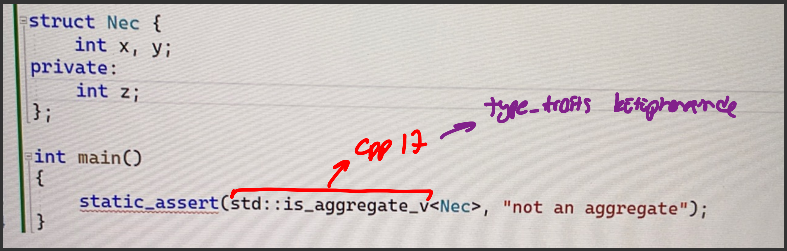


```cpp
struct Point {
    Point() = default;
};
```

→ Bu user declared ctor! User ~~defined~~ provided ctor değil!

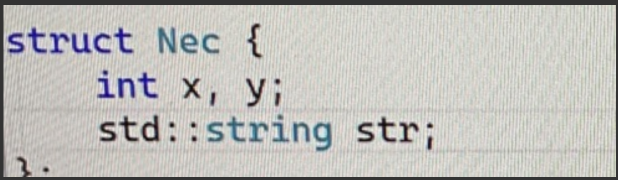
} → Cpp 20'ye kadar aggregate olurdu! Çünkü 20'ye kadar user declared değil user provided olamazdı!

2. Tüm elemanlar public olmalı!



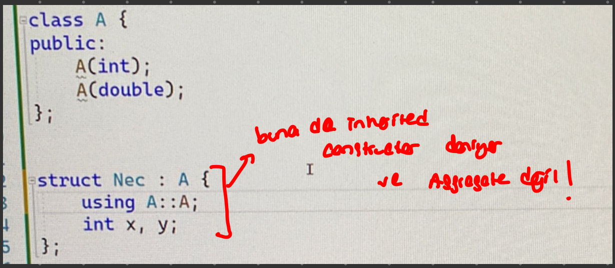
```cpp
struct Nec {
    int x, y;
private:
    int z;
};

int main()
{
    static_assert(std::is_aggregate_v<Nec>, "not an aggregate");
}
```

Cpp 17 → type_traits başlığında



```cpp
struct Nec {
    int x, y;
    std::string str;
};
```

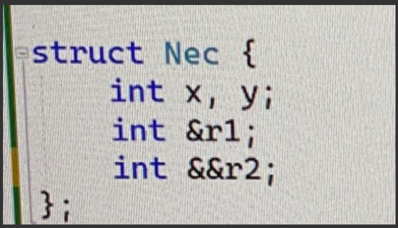
→ Nec bir aggregate. Aggregate olması için, elemanlarının da aggregate olması GEREKMEZ! Sadece public olmalı! Cpp 17 ile artık public inheritance ile elde edilen sınıflar da aggregate olabilir, ve yine base class aggregate olmak zorunda değil!



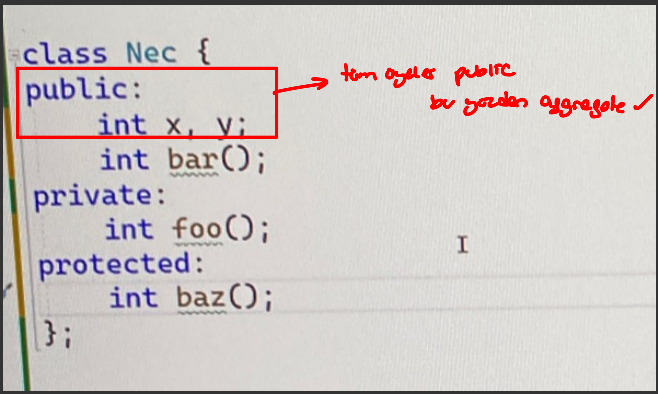
```cpp
class A {
public:
    A(int);
    A(double);
};

struct Nec : A {
    using A::A;
    int x, y;
};
```

buna da inherited constructor deniyor ve Aggregate değil!

3. Aggregateların, referans elemanları olabilir / member functionları olabilir → Public bir class da olabilir. (Struct olmak zorunda değil!)



```cpp
struct Nec {
    int x, y;
    int &r1;
    int &&r2;
};
```



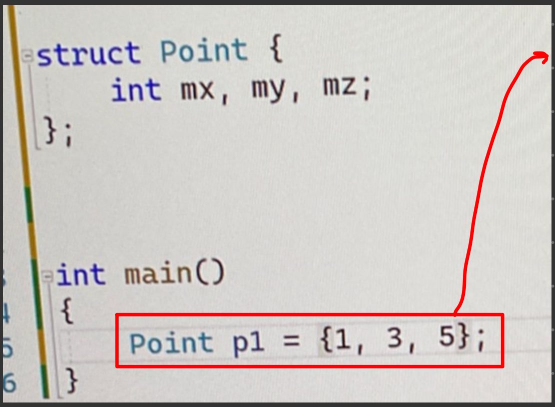
```cpp
class Nec {
public:
    int x, y;
    int bar();
private:
    int foo();
protected:
    int baz();
};
```

→ tüm öğeler public bu yüzden aggregate ✓

4. Virtual function varsa, Aggregate olamaz ✗

5. Static / inline static elemanların olması, Aggregate olmayı BOZMAZ ✓

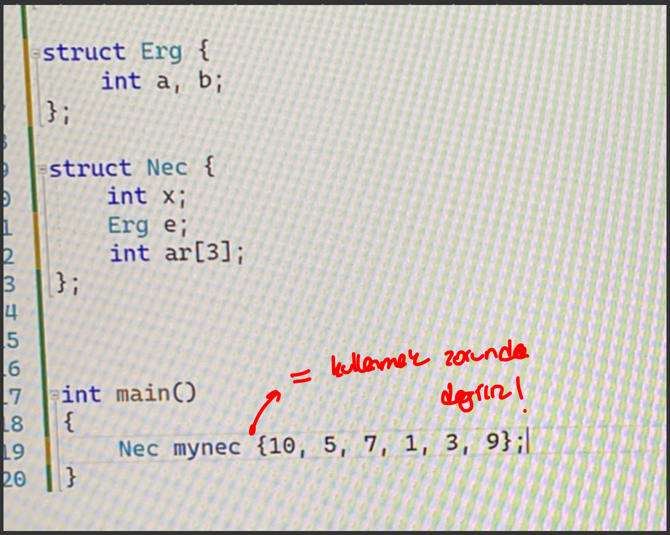
* Aggregate Initialization:



```
struct Point {
    int mx, my, mz;
};


int main()
{
    Point p1 = {1, 3, 5};
}
```

→ Küme parantezi ile ilk değer verilir!

→ Küme parantezi olmadan init yapılamaz!



```
struct Erg {
    int a, b;
};

struct Nec {
    int x;
    Erg e;
    int ar[3];
};


int main()
{
    Nec mynec {10, 5, 7, 1, 3, 9};
}
```

= kullanmak zorunda değiliz!

→ Diğer aggregate typelarda da aynısı geçer, eğer bu init listte tüm elemanları belirtmezsek, belirtilmeyen değerler value initialized!

* Cpp 20 ile Aggregateler:

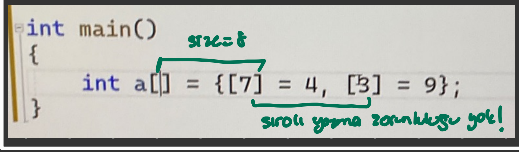
- Önceden user provided constructor aggregateliği bozuyordu. Şimdi user declared olması bozuyor.
- Cpp 20 ile Designated Initializer Syntax geldi!

C99 da da vardı

Ancak En büyük Fark C99 da

Arrayler için de Des. Init. Syntax Geçerli!

* C99 Designated Init:



```
int main()
{
    int a[] = {[7] = 4, [3] = 9};
}
```

size=8

sıralı yazma zorunluluğu yok!

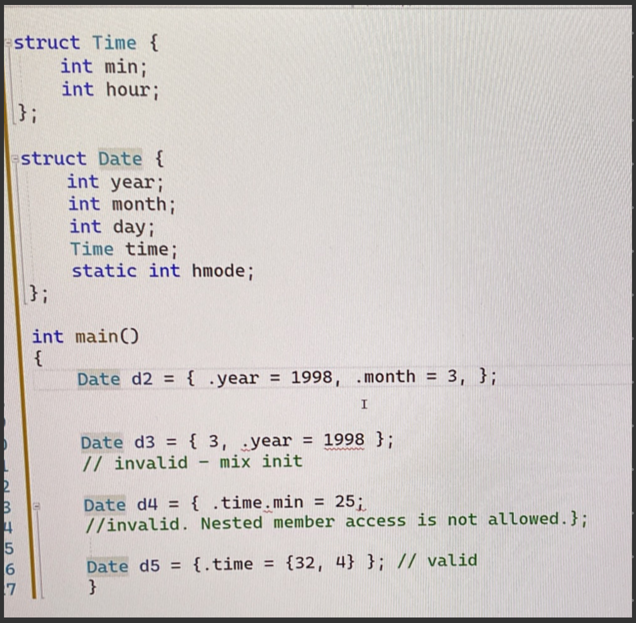
→ Cpp'de Arrayler için yok!

→ Cpp 20 Designated Initializer kuralları: = kullanımı designated initializer'ı etkilemez. Olsa da olur olmasa da

Tüm öğelere değer vermek zorunda değiliz. Değer yoksa value init

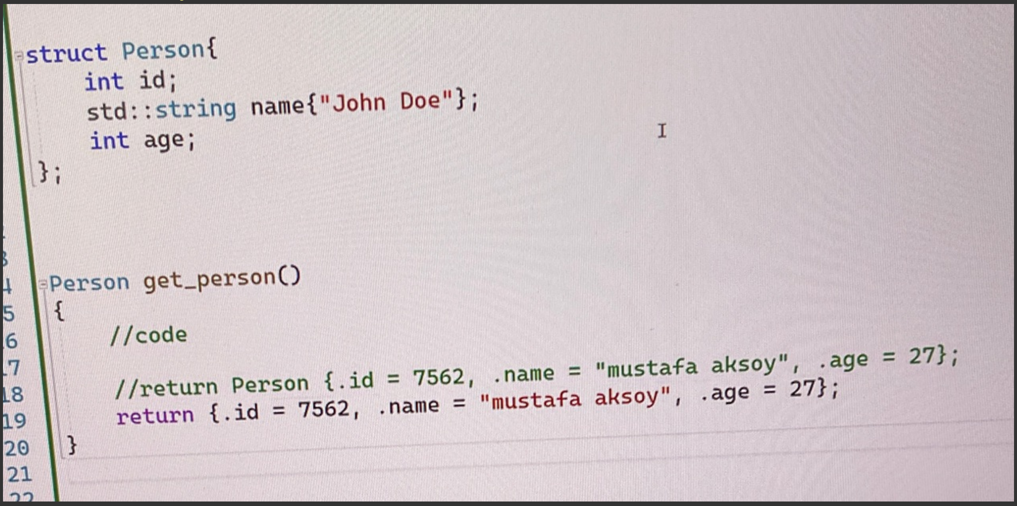
Init ederken sıra bozulamaz! Bildirim sırasıyla değer verilmeli!

Statik veri elemanlarını designated init ile ilk değer vermek syntax hatası!



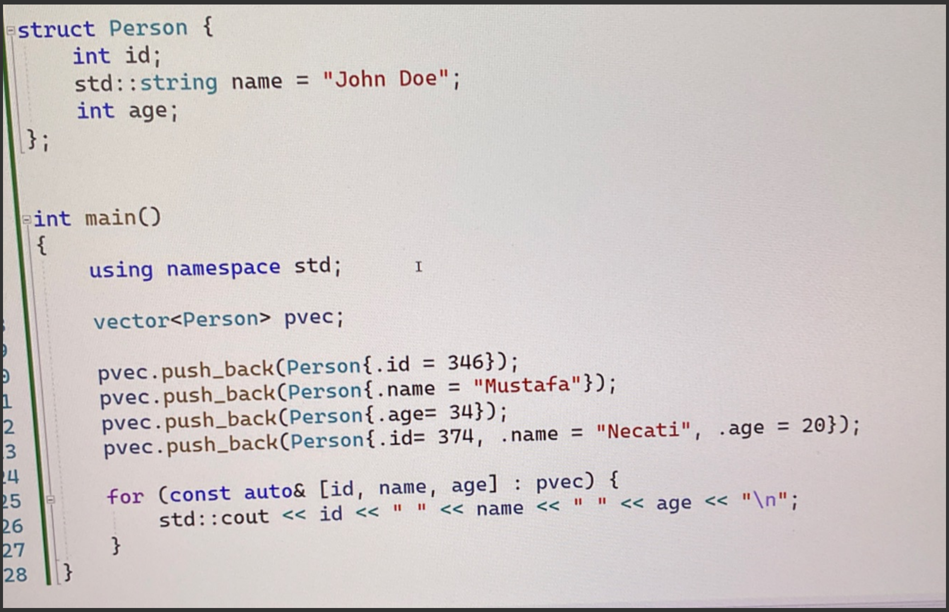
```
struct Time {
    int min;
    int hour;
};

struct Date {
    int year;
    int month;
    int day;
    Time time;
    static int hmode;
};

int main()
{
    Date d2 = { .year = 1998, .month = 3, };


    Date d3 = { 3, .year = 1998 };
    // invalid - mix init

    Date d4 = { .time.min = 25;
    //invalid. Nested member access is not allowed.};

    Date d5 = {.time = {32, 4} }; // valid
}
```

*Designated Initializer as Return Value:



```cpp
struct Person{
    int id;
    std::string name{"John Doe"};
    int age;
};


Person get_person()
{
    //code

    //return Person {.id = 7562, .name = "mustafa aksoy", .age = 27};
    return {.id = 7562, .name = "mustafa aksoy", .age = 27};
}
```

* Aggragate tutan Conteinerlar'ın Designated Initializer ile Deger alması:



```cpp
struct Person {
    int id;
    std::string name = "John Doe";
    int age;
};


int main()
{
    using namespace std;

    vector<Person> pvec;

    pvec.push_back(Person{.id = 346});
    pvec.push_back(Person{.name = "Mustafa"});
    pvec.push_back(Person{.age= 34});
    pvec.push_back(Person{.id= 374, .name = "Necati", .age = 20});

    for (const auto& [id, name, age] : pvec) {
        std::cout << id << " " << name << " " << age << "\n";
    }
}
```