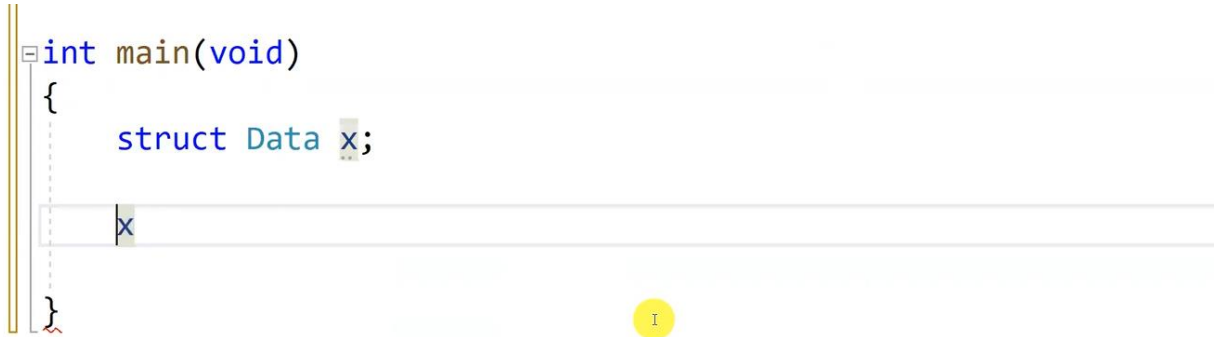
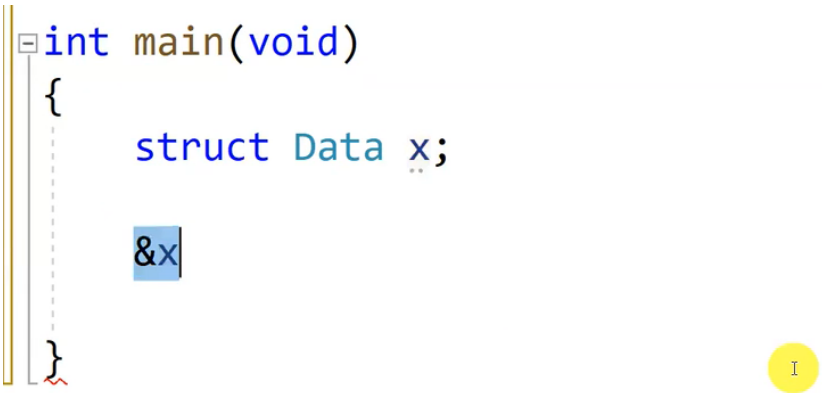
Ders 46

İfadenin türü nedir



Struct data türü kendimiz oluşturduk.

Türü nedir



Struct data * türü adresi

Const ta olabiliyor.

Tek bir bellek bloğunda mı

```c
struct Data{
    int a, b, c;
    double d;
};


int main(void)
{
    struct Data mydata;

    //40 byte
```

Kesinlikle evet tek bir bellek bloğunda ve hatta a b c d nesnelerinide istediğimiz gibi kullanabilir ama bunlara extra yer açılmaz bunlar mydata nın bellek bloğonda olucak.

```
yapı nesnesinin ilk öğesinin adresi aynı olmak zorunda.

yapı nesnelerinin elemanlarına erişim
=====================================

member selection operators

-------------------------------------------------
1   []  ()  .   ->
-------------------------------------------------
```

Örnek syntax hatası

```
struct Data{
    int a, b, c;
    double d;
};

int main(void)
{
    struct Data mydata;

    ///

    mydata.g;
```

Mydata. İle yapının bir nesnesine erişmeliyiz ama içinde g yok bu yüzden sentax hatası

A b c d olsaydı olabilirdi.

```
struct Data{
    int a, b, c;
    double d;
};

int main(void)
{
    struct Data a[10];
    ///

    a[5].
```

Buda legal çünkü buda bir struct nesne alemanı dizinin bütün elemanları nesne elemanı

```c
struct Data{
    int a, b, c;
    double d;
};


int main(void)
{
    struct Data x;
    struct Data* p = &x;

    (*p).b = 10;
```

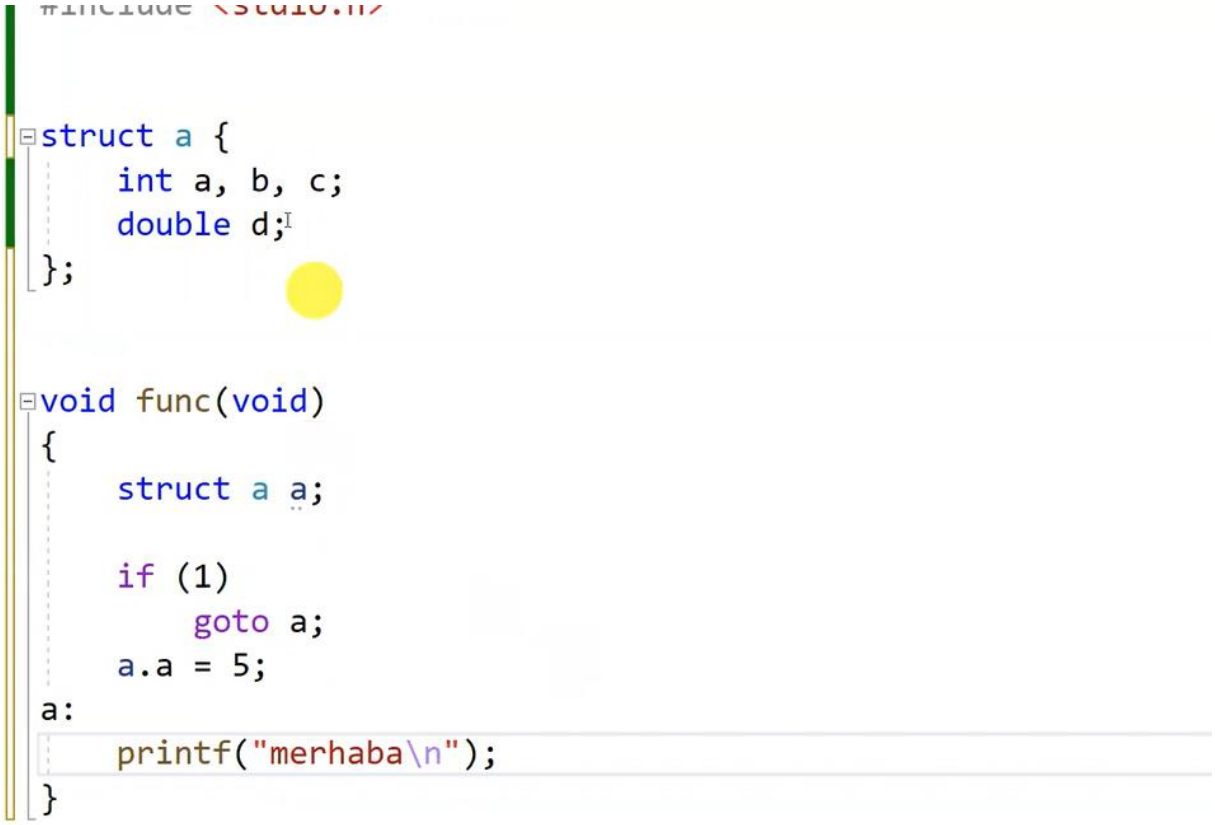
Buda legal çünkü sol operandı l value bir yapı elemanı olmalı.

Normalde karşınıza çıkmayacak bişey

```c
#include <stdio.h>


struct a {
    int a, b, c;
    double d;
};


int main(void)
{
    struct a a;
    a.a = 5;

}
```

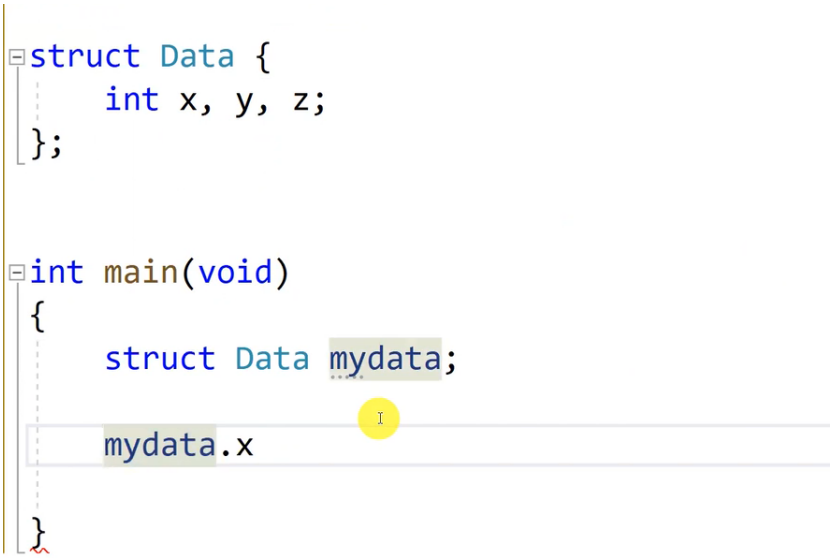
Burada bir sentax hatası yok ama karmaşık bir durum var bu yüzden detaylı anlatmadı.

Örnek puzzle sorusu



Burada hiç biri çakışmıyor ama iyi bir fikir değil

Soru 1



Bu bir ifademi

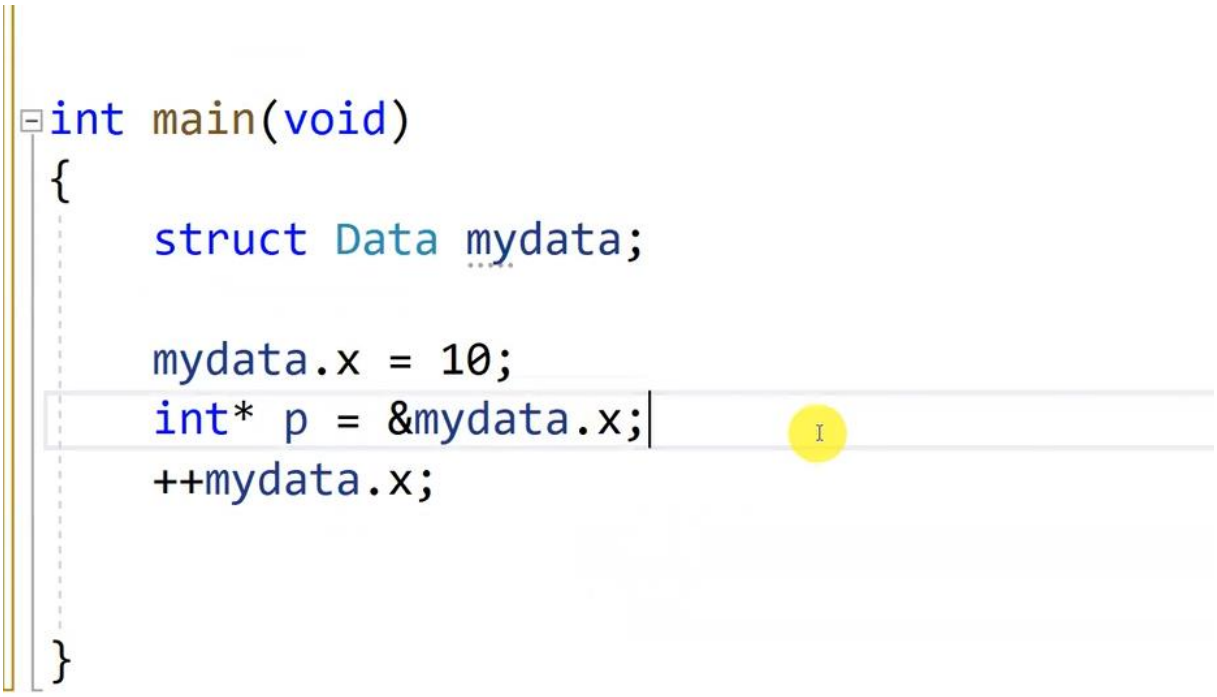
Evet

Bu ifadenin türü nedir.

İnt çünkü artık int nesneye eriştik

Bu ifadenin value kategorisi

L valuedir.



L valur olduğu için bu ifadeyi yukardaki gibi kullanabilirim ve . nokta 1. Öncelik seviyesindeolduğu için yazımındada kolaylık oluyor.

Örnek array dicey

```c
#include <stdio.h>


struct Data {
    int a[10];
    int b[10];
};


int main(void)
{
    struct Data x;

    int* p = x.a;



}
```

x.a yine x yapısının a dizisi demek a decay oluyor

```c
int main(void)
{
    struct Data x;

    int* p = x.a;
    int* p2 = &x.a[0];
```

Aynı ifade operatör öncelik seviyesini çok güzel anlatıyor.

Örnek farklı bir struct türünden atama

```
struct Data {
    int x, y, z;
};

struct Neco {
    int x, y, z;
};

struct Neco b;

int main(void)
{
    struct Data a;

    a = b;

}
```

Herseyleri eiemanları türleri aynı olsa bile bu atama işlemi yapılamıyor

Bu derleyiciye göre farklı bir tür olarak görülüyor sentax hatası.

Örnek aynı tür strct atama

```c
struct Data {
    int x, y, z;
};


int main(void)
{
    struct Data a = { 10, 20 ,30 };
    struct Data b;

    b = a;

    printf("b.x = %d\n", b.x);
    printf("b.y = %d\n", b.y);
    printf("b.z = %d\n", b.z);

}
```

Burada a nın bütün elemanlarının değeri b ye atandı.

Örnek atama operatörü

```c
struct Data {
    int a, b, c;
    double x, y, z;
};


int main(void)
{
    struct Data dx, dy;
    ///code

    dx = dy;

    dx.a = dy.a;
    dx.b = dy.b;



}
```

Bu kodda tek tek atama yapmak ile dx=dy arasında hiçbir fark yok

Hatta dx=dy ile daha iyi optimizasyon yapmaktadır.

Örnek memcpy ile kendimiz yazdık.

```
struct Data {
    int a, b, c;
    double x, y, z;
};

int main(void)
{
    struct Data dx, dy;
    ///code

    dx = dy;
    memcpy(&dx, &dy, sizeof(struct Data));



}
```

Örnek basit dinamik ömürlü structlar

```
struct Data {
    int a, b, c;
};


struct Data* create_data(void)
{
    struct Data* pd = malloc(_Size:sizeof(struct Data));
    //code
    //set the struct Data object
    return pd;
}
```

Basitçe anlattım.

//

Örnek basit

```
struct Point {
    int x, y, z;
};

int main(void)
{
    struct Point p1 = {10, 20, 30};

}
```

Taımlama sırası iledeğer verme sıram aynı aynı dizilerde olduğu gibi

```
struct Point {
    int x, y, z;
};

int main(void)
{
    struct Point p1 = {};

}
```

Boş olması sentaxhatası

```c
struct Point {
    int x, y, z;
};

int main(void)
{
    struct Point p1 = {1, 2, 3, 4};

}
```

Fazla atama ifadesi olursa sentax hatası

```c
struct Point {
    int x, y, z;
};

int main(void)
{
    struct Point p1 = {1};

}
```

Bir tane bile olursa değer vermediğimiz diğerleri 0 olacak.

Tıpkı dizilerde olduğu gibi

Örnek yanının elemanının dzi olması

```c
struct Nec {
    int x, y;
    int a[5];
    double dval;
};

int main(void)
{
    struct Nec nec = { 10, 20, {1, 2, 4, 6, 9}, 4.5 };
```

Çok boyutlu dizilerdeki gibi yapabiliriz. Aslında çok basit

Değer vermediklerimiz yine 0 olucak

Dizinin küme parantezide dahil değer vermediğimiz 0

Soru traling comma

```c
struct Nec {
    int x, y;
    int a[5];
    double dval;
};

int main(void)
{
    struct Nec nec = { 10, 20, {1, 2, 4, 6, 9,}, 4.5, };
```

Dizi için ve yapı için sonda fazla virgül var traling comma yasaldır.

Örnek

```c
struct Nec {
    int x, y;
    int a[5];
    double dval;
};


int main(void)
{
    struct Nec nec = { 10, 20, {1, 2}, 4.5 };

    print_array(nec.a, 5);
```


Print array ile dizisi yazdırdım

Nec.a ile diziye ulaştım ve ilk değer verdiğim 2 si 1 ve 2 geri kalan elemanlar 0 oldu.

Örnek küme parantezi olmadan

```c
struct Nec {
    int x, y;
    int a[5];
    double dval;
};


int main(void)
{
    struct Nec nec = { 10, 20, 1, 2, 4.5 };
```

burada 10 x e 20 y ye gidecek ama kümeparantezi olmadığı için kalan 3 eleman sırayla a dizisine atanacak 4.5 double olsada sıra değişmez kümeparantezi yok çünkü çok boyutlu dizi gibi sırayla atamalar yapılır yine kalanlar 0 olucak.

```
struct Nec {
    int x, y;
    int a[5];
    double dval;
};


int main(void)
{
    struct Nec nec = { 10, 20, 1, 2, 4.5 };

    print_array(nec.a, 5);

    printf("nec.dval = %f\n", nec.dval);
```

Ekrana 1 2 4 0 0 yazar 4.5 int e dönüşür.

Örnek employe

```
struct Employee {
    int m_id;
    char name[40];
    char address[60];
    double wage;

};

int main(void)
{
    struct Employee e1 = { 234, "ertan suluagac", "beylikduzu istanbul", 240.56 };

    char str[100] = "alican";
    const char* p = "mert";
```

Yukardaki gibi bir yapıda kişinin numarası

İsmi

Adresi

Ve ücreti tanımlandı.

Peki ilk değer vermedeki stringler static ömürlümü

Hayır en alttaki pointer mert harici static ömürlü değildir.

Yukardakiler ilk değer verme sentaxı olduğu için char diziye atanır.

Örnek iç içe structlar

```
struct Data{
    int x, y;
    int a[3][3];
    int b[4];
};

struct Neco {
    struct Data ar[3];
};

int main(void)
{

    struct Neco nx = { {{{{}}, {}, {}}
```

Örnek dizideki ilk değer verme

```
int main(void)
{
    int a[10] = { [3] = 6,[8] = 5,[1] = 9 };

}
```

Yapıların sentaxıda benzer şekilde

```c
struct Data {
    int x, y, z;
};

int main(void)
{
    struct Data mydata = {
        .x = 10,
        .y = 20,
        .z = 30,
    };

}
```

burada farklı olarak [] değilde . nokta kullanıyoruz.

.x = 20 , .y= 30 gibi

Verilmeyenler 0 oluyor.

Örnek elemanlarında bazıları dizi ise

```c
struct Data {
    int x, y, z;
    int a[10];
    char name[20];
};

int main(void)
{
    struct Data mydata = {
        .a = { [5] = 3, [9] = 56},
        .name = "mustafa",
        .x = 23
    }
}
```

Burada ilk değer verirken yine . koyduk ve diziye değer verdik hatta dizinin içindede bu yöntemle değer verilebilir örnek a dizisi

```c
struct Data {
    int x, y, z;
};

int main(void)
{
    struct Data d1 = { 3, 5, 8 }, d2 = { 3, 9, 1 };
}
```

```c
struct Data {
    int x, y, z;
};

int main(void)
{
    struct Data ar[3] = {1, 2, 3, 7, 9, 2, 2, 4, 6};
}
```

Legal olsada buda yapılabilir derleyici sırayla elemanları vericekti

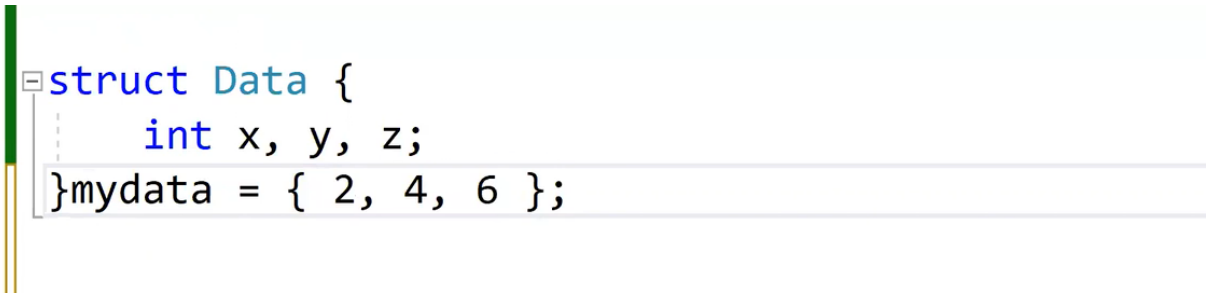
//

Örnek tuhaf tanımlama

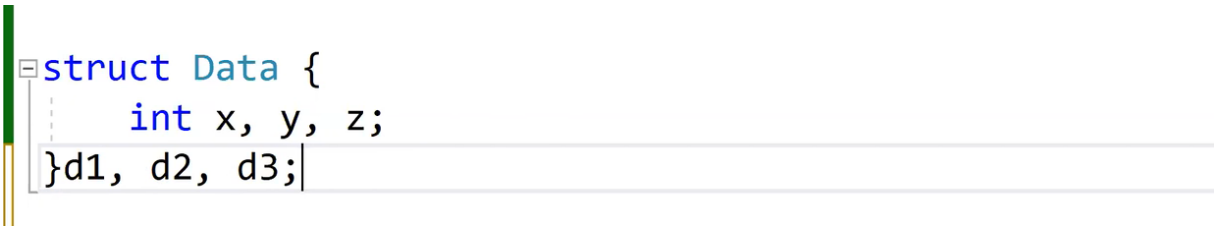
```c
struct Data {
    int x, y, z;
}mydata;
```

Yapı tanımlaması bittikten sonra tanımlama yapılabilir.

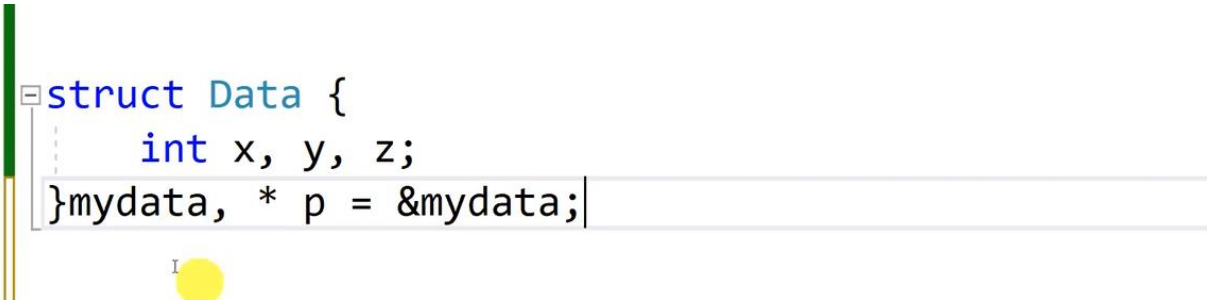
Struct data mydata ; yazmakla aynı anlamda bu



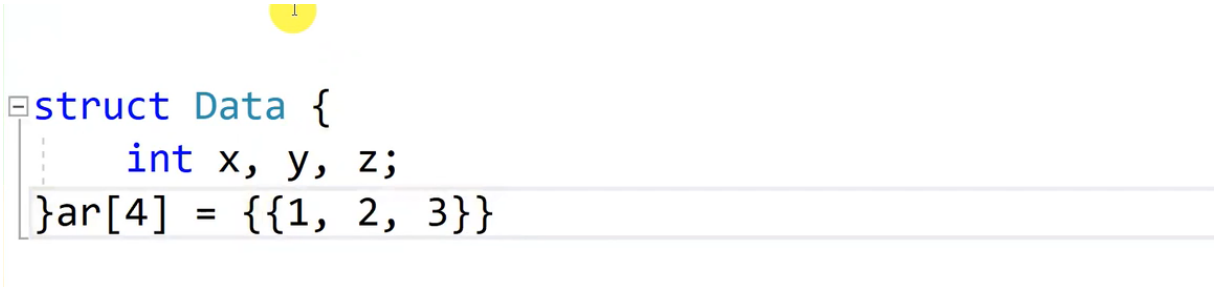
İlk değeride aynı şekilde verebiliriz.



Hatta burada pointer tanımlaması yapbilir.



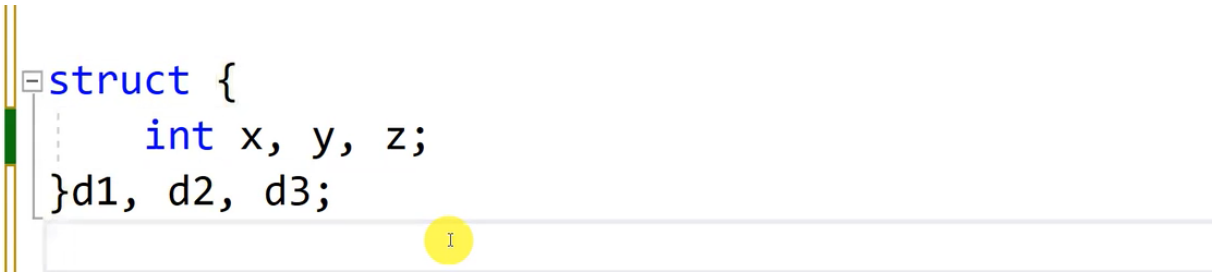
Struct data * türünden pointer oluşturduk.



Bir dizide tanımlayabilirim.

//

Örnek tag ı olmayan struct



Neden isim vermeden yaptık ne işimize yarayacak

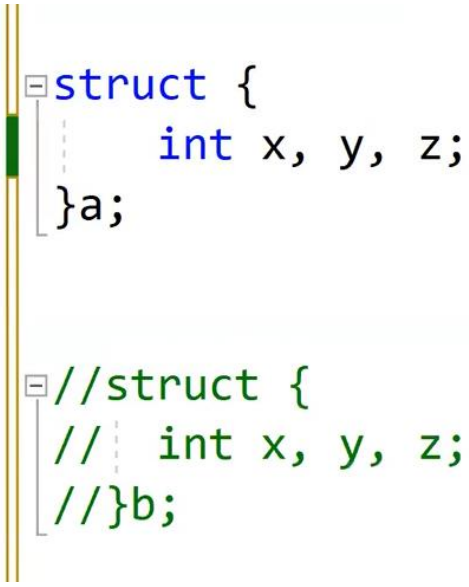
Buradauluşturduğum d1.x diyerek kullanabilirim ama bunu yapmamın ana nedeni bu yapı nesnesini kullanarak yeni bir nesneyapılmasını engellemek

D1 d2d3 dısında bu türden başka bir nesne tanımlanamaz.

//

Örnek tipik müakat sorusu

2 kere aynı boş struct ile tanımlama yapsam geçerli olur mu



Evet geçerli olur.

Peki aşağıdaki atama geçerli olur mu.

```c
struct {
    int x, y, z;
}a;


struct {
    int x, y, z;
}b;

int main(void)
{
    //a = b;

}
```

Hayır geçerli olmaz.

Çünkü derleyici bu iki türü aynı kabul etmiyor.

Bu türden yapılar kullanılabilir.

Atama işleminin geçerli olması için aynı struct türüne ait olması lazım isimleride aynı olmalı.

```c
struct Data{

    int (*p1)(int, int);
    int (*p2)(int);

};
```

Tamamen legal ve sentaxa uygundur.

Eleman fonksiyon olamaz ancak fonksiyon pointer olabilir hatta bu şekilde çok fazla kullanıyoruz.

```
struct Data{

    int (*p1)(int, int);
    int (*p2)(int);
};

int foo(int, int);
int bar(int);

int main(void)
{
    struct Data mydata = { foo, bar };

}
```

//

Soru ilginç örnekler

Legal mi

```
int main(void)
{
    int a[10] = { 0 };
    int b[10] = { 0 };
    ///
    a = b;
```

Kesinlikle hayır çünkü decay den dolayı l value ama a ya atama yapılamıyır.

Bu array decayla ilgili

Her l value atama yapılmaz örnek

```
int main(void)
{
    const int x = 5;

    x = 10;
```

Atama yapmak sentax hatası her l value ifadeye atama yapılmaz.

//

Örnek 2 diziyi yapı ile atama

```

struct Ar10 {
    int a[10];
};

int main(void)
{
    struct Ar10 x = { 3, 4, 6, 7, 9, 2 };
    struct Ar10 y;

    y = x;
```

Normalde 2 dizi birbirine bu şekilde atanamıyor memcpy kullanıyoruz.

Ama bunlaru yukardaki gibi struct elemanına çevirerek bu atama işlemini yapabiliriz bir nevi hile yapıyoruz çünkü aynı türdeki iki struct birbirine atanabilir.

Bir başka önemide yanlışla array dicey olacak durumlardan kurtulma

```c
struct Ar10 {
    int a[10];
};

void func(int* ptr);

int main(void)
{
    struct Ar10 ax;


    func(ax);
```

Normalde dizi adını kullandığımızda array decay olucaktı ama bu şekilde olmuyor bu yüzden fonksiyona verdiğimiz argümanda sentax hatası oldu.

```c
//array wrapper

struct Ar10 {
    int a[10];
};
```

Bunun adı array wrapper (sarmalama)

//

Örnek struct türünden dizi oluşturup karşılaştırılması.

```c
struct Data {
    int a, b, c;
    //
};

int mycmp(const void* vp1, const void* vp2)
{
    const struct Data* p1 = vp1;
    const struct Data* p2 = vp2;

    if (p1->a != p2->a)
        return p1->a - p2->a; //

    if (p1->b != p2->b)
        return p1->b - p2->b; //

    return p1->c - p2->c; //
}
```

```c
int main(void)
{
    struct Data a[100];
    ///code

    qsort(a, 100, sizeof(*a), &mycmp)

}
```

Bu kodu daha önceden yazdık sonuçta qsort bir void türden bağımsız karşılaştırma aracı

İçindeki elemanlarına bakıyoruz.

//

Yapı turu ve typedef bildirimleri.

Bazen standart fonksiyondan hizmet aldığımızda o otomatik olarak typedefbildiriminide kendi yapıyor. Struct türünün direk olarak typedef bildirimini görüyoruz.

Örnek

```c
#include <stdio.h>

int main(void)
{
    FILE* f = fopen("ali.txt", "r");

}
```

Burada struct tütü hazır kütüphaneden gelir ama typedef i kendimiz yazarız

File aslında bir struct türü

```c
/// image_pro.h

struct Image {
    //
    int a, b, c;
};


///
```

//

Örnek eş isim vermek

```c
#include <stdio.h>

/// image_pro.h

struct Image {
    //
    int a, b, c;
};


typedef struct Image Image;

int main(void)
{
    Image x;
    struct Image y;

}
```

Türün kendi isimide es isimi olabilir bu kullanım gayet legal

İsimler farklıda olabilir bu konuda sıkıntı yok.

```c
typedef struct _Image Image;

int main(void)
{
    struct _Image x;
    Image y;

}
```

//

Örnek pointerı typedef yapmak

```
/// image_pro.h

struct Data {
    //
    int a, b, c;
};

typedef struct Data Data;
typedef Data* DataPtr;


int main(void)
{
    Data x;
    DataPtr p = &x;

}
```

Burada typedef bildirimini kullanarak pointer türünün typedefini yaptık.

Aynı typedef içinde 2 farklı nesnede yapabilirim örnek

```
struct Data {
    //
    int a, b, c;
};

typedef int Word, Boolean;

int main(void)
{
    Word x;
    Boolean
}
```

Burada tek typedef ile 2 int tür eş ismi yaptık.

Struct ile bunu yapalım

```c
typedef struct Data Data, *DataPtr;

int main(void)
{
}
```

Aynı seyi yaptık 3 adımda tanımlama

Örnek en sık kullanılan yöntem

```c
struct Data {
    //
    int a, b, c;
}x;
```

3 adımda başına typedef koy

X yerine eş isim yaz

```c
typedef struct Data {
    int a, b, c;
}Data;
```

Daha kolay bir yöntem

```c
typedef struct _Data {
    int a, b, c;
}Data;

int main(void)
{
    struct _Data x = { 3, 6, 7 };
    Data y = { 3, 6, 7 };
```

Genelde _ veya tag ön eki kullanılır.

```c
typedef struct tagData {
    int a, b, c;
}Data;

int main(void)
{
```

Karışıklık olmasın diye