Ders 47

Örnek typedef

Yapı bildirimi ile birlikte yada ayrı olabilir.

```
//necati.h

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

```
//necati.h

struct Nec {
    int a, b, c;
};

typedef struct Nec Nec;
```

Etiket ismi ile typedef ismi aynı olmak zorunda değil genelde _ veya tag kullanılıyor.

```
//necati.h

typedef struct _Nec {
    int a, b, c;
}Nec, *NecPtr
```

```
//necati.h

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

en sık kullanılanlardan biri bu artık struct ile tanımlanmaz farklı bir nesne oluşturmayı engeller.

```
//necati.h

typedef struct {
    int a, b, c;
}*NecPtr;
```

Buda ilginç yapının tür ismi yok ancak * tründen bir değişkeni var

//

Örnek sadece *necptr typedefi var ancak türün ismi yok ne işe yarar.

```
//necati.h

typedef struct {
    int a, b, c;
}*NecPtr;


int main(void)
{
    NecPtr pd = (NecPtr)malloc(sizeof(*pd));
    //
    pd->a =

}
```

Burada kodu iyi anlamak gerekiyor

Bu struct türünün bir ismi olmadığı için statik veya dinamik ömürlü nesne yapamıyırum

Ama dinamik ömürlü bir nesne yapabilirim

Necptr türünden bir değişken tanımladım bu yukardaki yapının * lı hali

Sonra malloc içinde sizeof değerine ihtiyacım var *pd o türü ait bir nesne oldu.

Ama bir sorun var pd çöp değerde ve ben *pd yaptım tanımsız değilmi

Hayır çünkü sizeof operatörü içindeki için işlem kodu üretmez

++x gibi ifade olsaydıda üretmiyecekti hatırla.

Örnek bir bildirim daha

```
typedef struct Nec{
    struct Nec* ptr;
    int a, b, c;
}Nec;
```

Bazı durumlarda yapının içinde kendisine ait pointer değişkeni olsun isteriz

Ama eğer isim yazmasaydım bunu yapamazdım.

```
typedef struct{
    //
    int a, b, c;
}Nec;
```

Henüz bildirimde yapı ismi bilinmediğinden pointerını içine yazamam.

Bunu engeller bu durum.

//

Örnek

Bir tür eş ismi başka türeş isimlerindede kullanılabilir.

```
typedef struct{
    //
    int a, b, c;
}Nec;


typedef Nec* NecPtr;
typedef const Nec* CNecPtr;
```

Yapı türleri adresleri

Örnek pointer

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

int main(void)
{
    struct Point p1 = {1.2, 5.6, 2.3};
    struct Point p2 = {2.3, 3.3, 6.9};
    struct Point* ptr = &p1;

    *ptr = p2; //p1 = p2;

    (*ptr).y = 12;
```

Burada *ptr yi direference ettiğimiz zaman p1 nesnesine erişiriz.

Yapılarda = ile atama yapılabiliyordu hatta dizi sarmalama örneği vermiştik.

Ancak (*ptr).x gibi bir ifade kullanmak herzaman daha masraflı 3 operatör kullanıyoruz.

Bide öncelik meselesi var . * dan daha öncelikle mecbur () kullanıyoruz.

Bu yüzden -> kullanmak daha avantajlı ve kolay

Örnek 2

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

int main(void)
{
    struct Point p = { 1.1, 0.5, 8.9 };

    (&p)->y = 45;
    p.y = 45;
```

bu örneği normalde yapmazdık amayapsakta sentax hatası olmazdı.

Eğer nesne adresi varsa

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

int main(void)
{
    struct Point p = { 1.1, 0.5, 8.9 };
    struct Point* ptr = &p;

    (*ptr).x = 4.5;
    ptr->x = 4.5;
```

Ok kullanırım daha kolay.

//

Zorlu örnekler.

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

struct Point* foo(void);

int main(void)
{
    struct Point a[10] = { 0 };

    a[0].x = 6.6;
    a->x = 6.6;

    a[5].x = 1.2;
    (a + 5)->x = 1.2;

    foo()->x = 1.1;
```

```c
foo()->x = 1.1;
foo()[5].x
```

Bu örnekte a array decaydan dizinin ilk eleman adresi

Bu yüzden -> ile kullanıp yazabilirim.

Ve foo nun döndürdüğü adres point türünden adres

Bunuda ok operatörünün operandı ypabilirim.

Örnek konst

```c
struct Point* foo(void);

int main(void)
{
    struct Point p1 = { 1.2, 4.5, 6.7 };
    struct Point p2 = { 1.2, 4.5, 6.7 };
    struct Point* const ptr = &p1;

    ptr = &p2;
```

```c
struct Point* foo(void);

int main(void)
{
    struct Point p1 = { 1.2, 4.5, 6.7 };
    struct Point p2 = { 1.2, 4.5, 6.7 };
    struct Point const* ptr = &p1;

    (*ptr).y =
}
```

2 ifadede sentax hatası.

Pointer aritmetiği örnek

```c
struct Point {
    double x, y;
};

int main(void)
{
    printf("sizeof(struct Point) = %zu\n", sizeof(struct Point));
    struct Point a[10];

    for (int i = 0; i < 10; ++i) {
        printf("%p  %p\n", &a[i], a + i);
    }


}
```

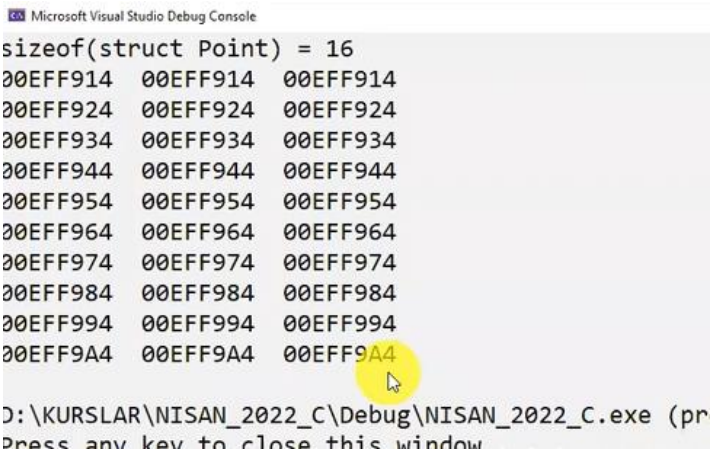
boyutu 16 ve her adres arasında 16 byte var

```c
struct Point {
    double x, y;
};

int main(void)
{
    printf("sizeof(struct Point) = %zu\n", sizeof(struct Point));
    struct Point a[10];
    struct Point* ptr = a;

    for (int i = 0; i < 10; ++i) {
        printf("%p  %p  %p\n", &a[i], a + i, ptr++);
    }


}
```



//

Complete incomplete

```
complete types vs incomplete types

complete
incomplete

============================

bir incomplete type ile (dilin kurallarını çiğnemeden) neler yapabiliriz?
bir incomplete type ile neler yapamayız?
Neleri yapabilmemiz için bir yap türünün "complete type" olması gerekir?
```

Neden bu kadar önemli

Biz su adan kadar complete i gördük

Ama fonksiyon bildirimi gibi yaparsak incomplete olur.

Struct data ;

Gibi

1 incomplete ile neler yaoılır.

Fonksiyon bildirimde kullanabilirim.

```
struct Data;

struct Data foo(struct Data, struct Data*);
```

2 tür eş isminde yer alabilir.

```
typedef struct Data Data;
typedef struct Data* D;
```

Bir fonksiyon türüne eş isim verdim.

```
struct Data;

typedef struct Data* (*Fptr)(struct Data *);
```

//
dakika 2.35.00

Örnek calender time mı bulup bunun hangi yıla ait olduğunu bulmak

```
int main(void)
{
    time_t sec;
    time(&sec);
```

Bu ifade 2 anlama geliyor.

Sen bana bir değişken ver ben onu epoch tan gecen saniye sayısıyla set edeyim

Yada null ver ger dönüş değerim epoch tan gecen saniye sayısı olsun.

```c
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t sec;
    time(&sec);

    struct tm* p = localtime(&sec);

    printf("%d\n", p->tm_year);

}
```

Localtime fonksiyonu set edilen değişkenin değerini değil adresini alır ve bununla işlemler yapar.

Struct tm türü yıl ay gün saniye saat dakika gösterir fonksiyonun geri dönüş değeri bu tür

Şimdi anladım mantıksal bütünlükteki nesnelerin gruplandığı nesne

Tek bir yerde topluyoruz ki gözümüzün önünde olsun ulaşalım.

İnt double gibi düşün unutma.

```c
int main(void)
{
    time_t sec;
    time(&sec);

    struct tm* p = localtime(&sec);

    printf("%02d-%02d-%d %02d:%02d:%2d\n", p->tm_mday,
        p->tm_mon + 1, p->tm_year + 1900,
        p->tm_hour, p->tm_min, p->tm_sec
        );

}
```

Başta yanlış yaptık bilere years 1900 eksik gösterir.