

# Programlamaya Giriş

## HAFTA 5

## Veri Tipi Tanımlama

**Prof. Dr. Cemil ÖZ**

**Doç. Dr. Cüneyt BAYILMIŞ**

**Dr. Öğretim Üyesi Gülüzar ÇİT**

# Konu & İçerik

- **Veri Tipi Tanımlama**
  - Enum
  - Struct
- **Veri Tipi Tanımlama**
- **Tipler İçin Kısa Ad (typedef)**
- **Struct Veri Tipi**
- **Kayan Noktalı Sayılar**
- **Kaynaklar**



# Veri Tipi Tanımlama

- Her programlama dilinde tanımlı temel veri tipleri bulunmaktadır.
- Tanımlı olmayan veri türleri, kullanıcı bazlı tanımlama ile sağlanır.
- Birleştirme ve topluluk gibi iki farklı şekilde yapılabilmektedir.
- Kendi veri tipinizi tanımlama imkanı sunar.

# Veri Tipi Tanımlama...

## ➤ Birleştirme Veri Tipi (Enumerations) - enum

- **enum** veri tipi, değişkenin alabileceği değerlerin belirli/sabit olduğu durumlarda programın daha anlaşılır olmasını sağlar.

```
enum tipAdi { isim listesi (deger1, deger2, ...) } degiskenAdi;
```

# Veri Tipi Tanımlama...

## ➤ Birleştirme Veri Tipi (Enumerations) – enum...

### ➤ ÖRNEK: ⇒ [1]\_enum.cpp

```
enum bolumler { bilgisayar, bilisim, yazilim } bolum;  
  
int main()  
{  
    // Sonuç ekranında Türkçe karakterleri kullanabilmek için  
    setlocale(LC_ALL, "Turkish");  
  
    bolum = bilgisayar;  
    cout << bolum;  
  
    bolum = static_cast<bolumler>(bolum + 1);  
    cout << bolum;  
  
    system("pause");  
  
    return 0;  
}
```

# Veri Tipi Tanımlama...

## ➤ Birleştirme Veri Tipi (Enumerations) – enum...

### ➤ ÖRNEK: ⇒ [1]\_enum.cpp

```
enum yonler { Guney, Kuzey, Dogu, Bati };

int main()
{
    // Sonuç ekranında Türkçe karakterleri kullanabilmek için
    setlocale(LC_ALL, "Turkish");

    enum yonler yon;

    int secim;

    cout << "Yön Giriniz (Güney=0, Kuzey=1, Doğu=2, Batı=3) :";
    cin >> secim;

    yon = static_cast<yonler>(secim);

    switch (yon)
    {
        case Guney: cout << "Guney"; break;
        case Kuzey: cout << "Kuzey"; break;
        case Dogu: cout << "Dogu"; break;
        case Bati: cout << "Bati"; break;
        default: cout << "hatali secim";
    }

    system("pause");

    return 0;
}
```

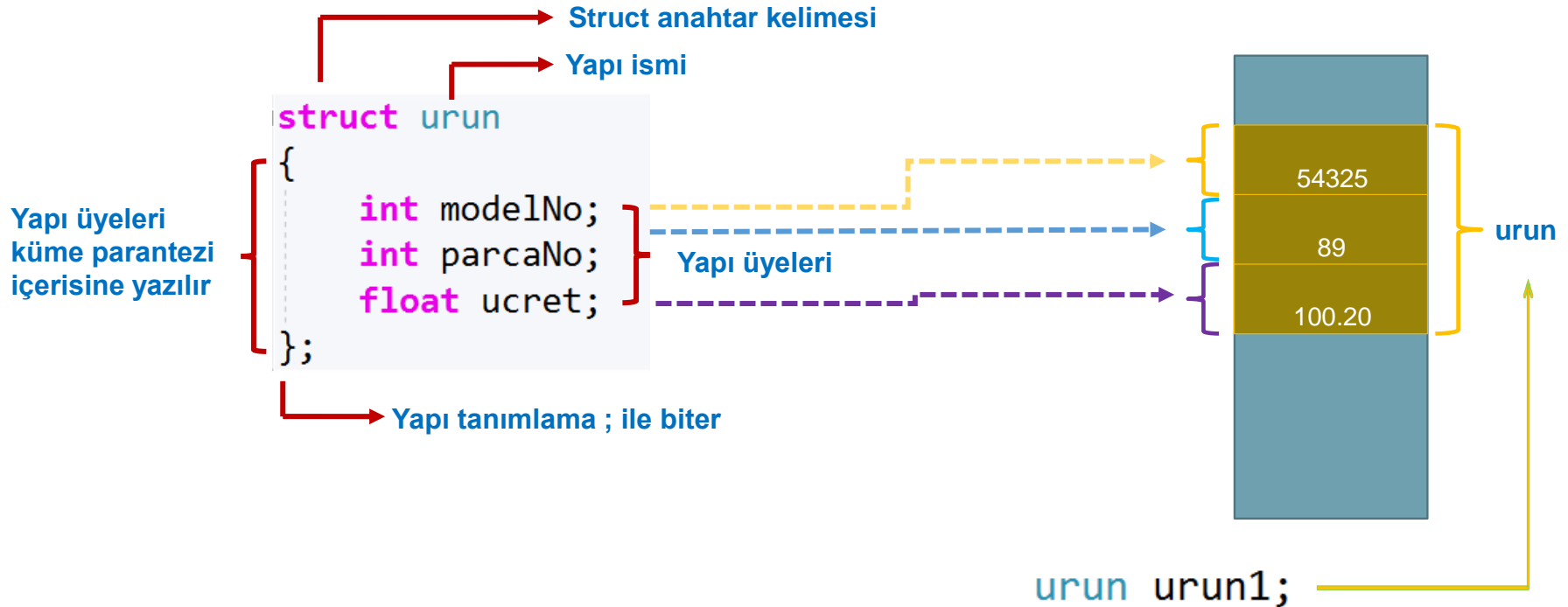
# Veri Tipi Tanımlama...

## ➤ **Yapılar – struct**

- Yapılar ve sınıflar ilişkili fakat farklı tipe de sahip olabilen verileri tutmak için kullanılırlar.
- Yapılar, sınıflar ve diziler statik elemanlardır. Programın çalışma süresi boyunca sabit boyutladır.
- Aynı tipe sahip veri elemanlarının oluşturduğu veri yapılarına (ilişkili veri elemanları topluluğu) dizi denir. Yapılar, diziler ile karıştırılmamalıdır.
- Yapı içerisindeki bileşenlere **ÜYE** denir.
- Her yapı farklı bir isme sahiptir, ancak aynı isimli üyeleri olabilir.

# Veri Tipi Tanımlama...

## ➤ Yapılar – struct...





# Veri Tipi Tanımlama...

## ➤ Yapılar – struct...

### ➤ Yapı tanımlama

- Yapı tanımı bellekte yer ayırmaz. Yapı tanımı ile yapı değişkeni tanımlanmış olmaz.
- Yapı tanımı sadece, yapı değişkenlerinin tanımlandıkları zaman nasıl görüneceklerini gösteren bir modeldir.

### ➤ Yapı değişkenini tanımlama

```
urun urun1;           // urun1 için bellekte yer ayrılır
```

### ➤ Yapı üyelerine değer atama

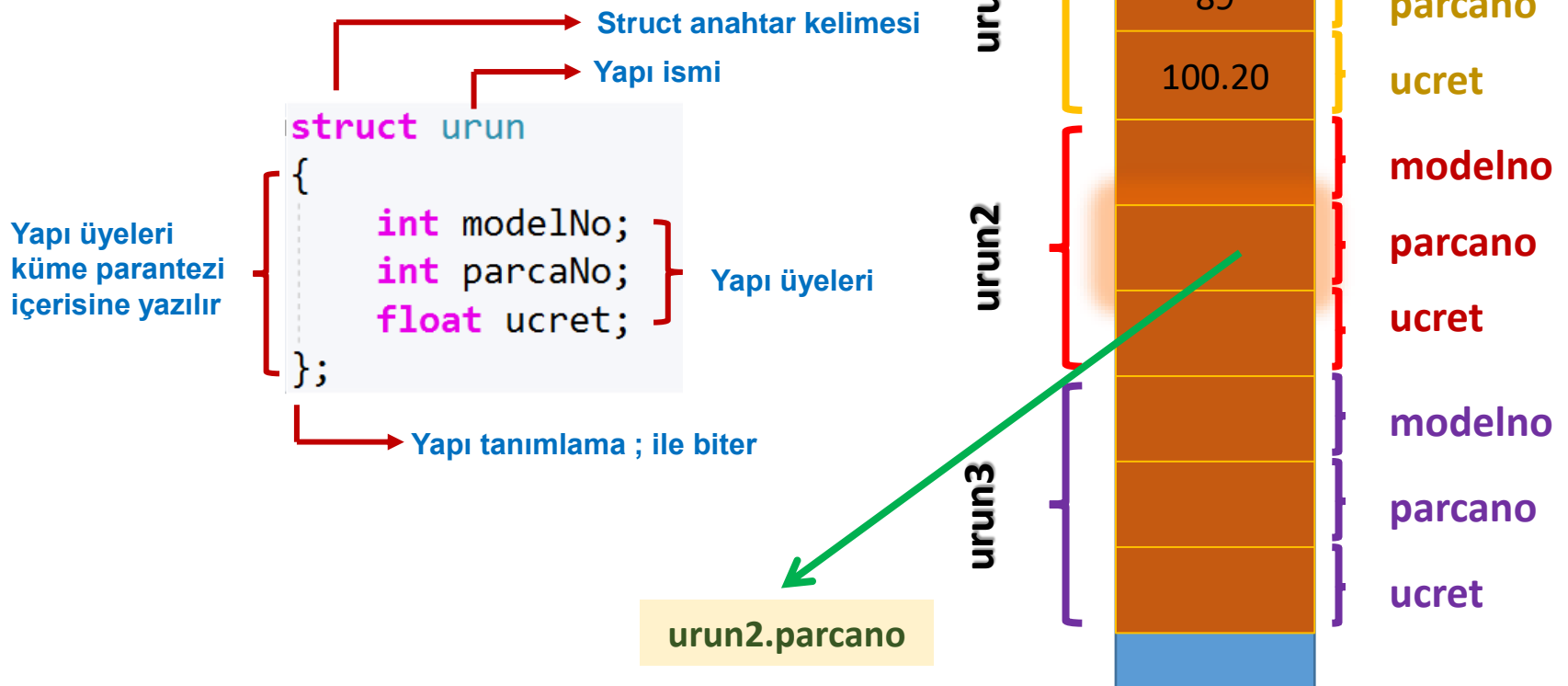
- Yapı içerisindeki üyeye ‘.’ (üye erişim operatörü) ile erişilir.

```
yapiDegiskeni.uyeAdi;
```

```
urun urun1 = { 5235, 98, 153.15f };  
urun1.modelNo = 2016;  
urun urun2;      // urun2 yapı değişkeni tanımla  
urun2 = urun1;   // ilk yapı değişkenindeki değerleri ikinci değişkene ata
```

# Veri Tipi Tanımlama...

## ➤ Yapılar – struct...



# Veri Tipi Tanımlama...

## ➤ Yapılar – struct...

➤ ÖRNEK: ⇒ [3]\_struct.cpp

```
// Yapı Tanımlama
struct urun
{
    int modelNo;
    int parcaNo;
    float ucret;
};
```

```
urun urun1;           // urun1 için bellekte yer ayrılır

urun1 = { 5235, 98, 153.15f };
urun1.modelNo = 2016;
urun urun2;           // urun2 yapı değişkeni tanımla
urun2 = urun1;        // ilk yapı değişkenindeki değerleri ikinci değişkene ata
```

# Veri Tipi Tanımlama...

## ➤Yapılar – struct...

### ➤ÖRNEK: ⇒ [4]\_struct\_karmasik1.cpp

```
struct Karmasik
{
    double gercel;
    double sanal;
};
```

```
Karmasik sayi1, sayi2;
sayi1.gercel = 44.423423423;
sayi1.sanal = 48;
//sayi1.sanal=48e2;
cout << fixed << setprecision(3) << sayi1.gercel
    << ' + ' << sayi1.sanal << "i" << endl;

//cout <<scientific<<setprecision(3)
//    <<sayi1.gercel<<'+ '<<setw(5)<<right
//    <<sayi1.sanal<<'i'<<endl;

cout << sizeof(sayi1) << "      " << sizeof(int) << endl;
```

# Veri Tipi Tanımlama...

## ➤ Yapılar – struct...

### ➤ ÖRNEK: ⇒ [5]\_struct\_karmasik2.cpp

```
Karmasik sayi1, sayi2, sonuc;  
sayi1.gercel = 44.426423423;  
sayi1.sanal = 48;  
  
cout << "2. sayinin gercel ve sanal kisimlarini giriniz:";  
cin >> sayi2.gercel >> sayi2.sanal;  
  
sonuc.gercel = sayi1.gercel + sayi2.gercel;  
sonuc.sanal = sayi1.sanal + sayi2.sanal;  
  
cout << fixed << setprecision(2) << sonuc.gercel  
    << "+" << sonuc.sanal << " i" << endl;
```

# Veri Tipi Tanımlama...

## ➤Yapılar – struct...

### ➤ÖRNEK: ⇒ [6]\_struct\_ogrenci.cpp

```
struct ogrenci {  
    string ad;  
    string soyad;  
    int vize;  
    int final;  
};
```

```
ogrenci ogrNot;                                // yapı değişkeni tanımlama  
  
cout << "Öğrencinin Adı:" << endl;  
cin >> ogrNot.ad;                               // yapı üyesine değer atama  
cout << endl << "Oğrencinin soyadini giriniz \n";  
cin >> ogrNot.soyad;  
cout << endl << "Vize notunu giriniz \n";  
cin >> ogrNot.vize;  
cout << endl << "Final notunu giriniz \n";  
cin >> ogrNot.final;  
cout << endl << "Oğrenci Bilgileri\n";
```

# Veri Tipi Tanımlama...

## ➤ Yapılar – struct...

### ➤ ÖRNEK: ⇒ [7]\_struct\_calisan.cpp

```
if (isci.maas < 600)
    isci.cd = acliksiniri;
else if (isci.maas < 1500)
    isci.cd = ortahalli;
else if (isci.maas > 1500)
    isci.cd = iyidurumda;

switch (isci.cd)
{
case acliksiniri:
    cout << " 1.Calisanin durumu      = \t acliksiniri \n " << endl;
    break;
case ortahalli:
    cout << " 1.Calisanin durumu      = \t ortahalli\n " << endl;
    break;
case iyidurumda:
    cout << " 1.Calisanin durumu      = \t iyidurumda \n" << endl;
    break;
}
```

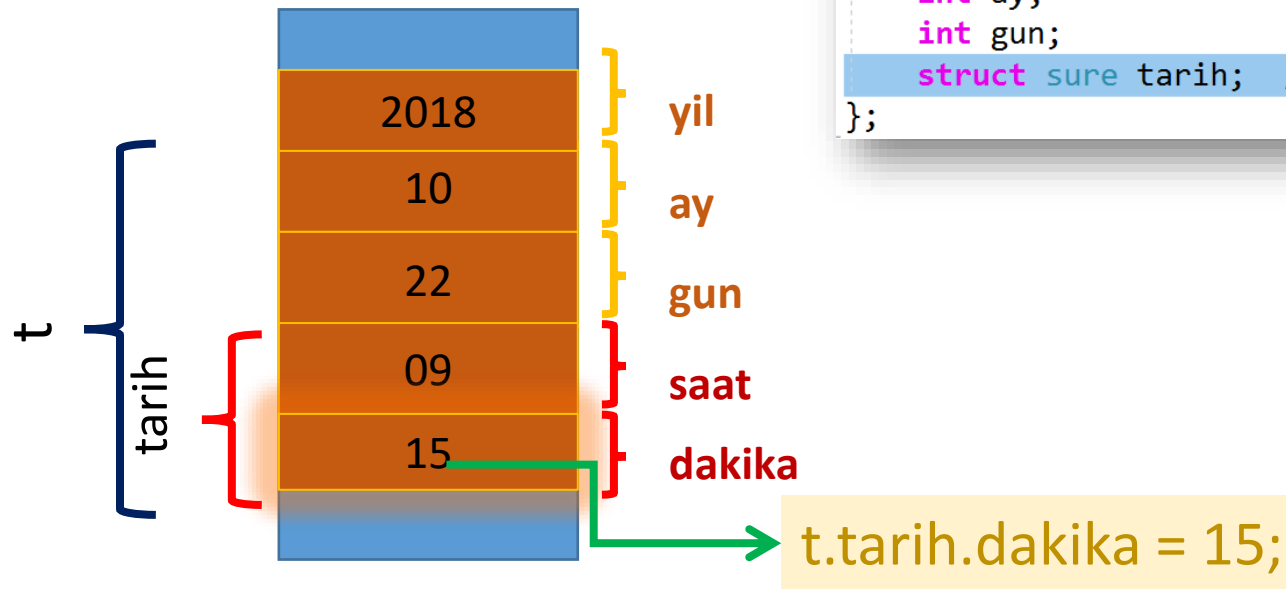
```
enum CalisanDurumu { acliksiniri, ortahalli, iyidurumda };

struct Isci
{
    int isciNo;
    float maas;
    CalisanDurumu cd;
};
```

# Veri Tipi Tanımlama...

## ➤ İç İçe Yapılar – struct...

➤ ÖRNEK: ⇒ [8]\_struct\_tarih.cpp



```
using namespace std;

struct sure {                // yapı tanımlama
    int saat;
    int dakika;
};

struct takvim {              // yapı tanımlama
    int yil;
    int ay;
    int gun;
    struct sure tarih;       // iç içe yapı kullanımı
};
```



# Veri Tipi Tanımlama...

## ➤ İç İçe Yapılar – struct...

➤ ÖRNEK: ⇒ [8]\_struct\_tarih.cpp...

```
takvim t;  
  
cout << "yıl ay gün saat dakika";  
cin >> t.yıl >> t.ay >> t.gun;  
cin >> t.tarih.saad >> t.tarih.dakika;  
  
cout << t.gun << t.ay << t.yıl << '\t'  
      << t.tarih.saad << ':' << t.tarih.dakika;
```

```
//İç içe yapı değişkenine başlangıç değeri atama  
takvim t = { 2018, 10, 22, {9, 15} };
```

# Tipler için Kısa Ad (**typedef**)

- Yazılan kodların genelleştirilmesinde kullanılır. (Farklı tipler için aynı kodların kullanılabilmesi)
- Yazımı karmaşık, uzun ve zor olan tip tanımlarının daha basit ve anlaşılır olmasını sağlamak amacıyla kullanılabilir.

```
typedef veritipi yeniTipAdi;
```

```
typedef    int    tamsayi;  
tamsayi    x, y;    // x ve y int tipindedir
```

# Birlik (union)

- Birlikler yapılara benzer
- Yapıların kullanılmayan üyelerinden doğan verimsizliği ortadan kaldırmak için kullanılır.
- Aynı depo alanını farklı değişkenler ortak kullanırlar.
- Üyeler herhangi bir veri türü olabilir.
- Birlik en az, en büyük alana sahip üyeyi içerecek kadar kapasiteye sahip olmalıdır
- Bir anda sadece bir üyeye erişilebilir.
- Başlangıç değeri ilk üyenin türü ile aynı olmalıdır.
- Başlangıçta sadece ilk üye için atama yapılabilir.

```
union sayi {  
    int x;  
    int y;  
};  
  
union sayi deger = { 10 };    //geçerli  
union sayi deger = { 1.23 };  //geçersiz
```



# Kayan Noktalı Sayılar

- Kayan noktalı sayılar, hassasiyet belirtilmemişse, varsayılan olarak 6 basamak hassasiyetle gösterilirler(32 bitlik işletim sistemleri için)
- Kaç basamak gösterileceğini belirtmek için **fixed** ve **(set)precision** manipulatörleri kullanılır. **fixed** ifadesi kullanılmaz ise toplam basamak sayısı, kullanılır ise virgülden sonraki basamak sayısı belirtilmiş olur.
- Bu manipulatörde **iomanip** kitaplığına ihtiyaç duyar.
- Bilimsel gösterim çok büyük ya da çok küçük sayıların gösteriminde tercih edilir. Sayılar 10 üssü olarak ifade edilirler.

```
// Yazım formatları
cout << fixed;           // cout.setf(ios::fixed);
cout << scientific;      // cout.setf(ios::scientific);
cout << showpoint;       // cout.setf(ios::showpoint);
```

```
// kayan noktalı sayı hassasiyeti
float PI = 3.1415;
cout.precision(2);
cout << PI;
```

# Kayan Noktalı Sayılar...

➤ ÖRNEK: ⇒ [9]\_bicimleme.cpp...

```
double sayi1, sayi2, sayi3;
```

```
sayi1 = 44.426423423;
```

```
sayi2 = 48e-5;
```

```
sayi3 = 566.01;
```

```
cout << setprecision(7) << sayi1 << endl;
```

```
cout << setprecision(5) << sayi1 << endl;
```

```
cout << fixed << setprecision(2) << sayi1 << endl;
```

```
cout << fixed << setprecision(5) << sayi2 << endl;
```

```
cout << scientific << setprecision(3) << sayi3 << endl;
```

```
cout << fixed << setprecision(6) << sayi1 << endl;
```

```
cout << hex << 15 << endl;
```

```
cout << dec << 0xff << endl;
```

```
cout << oct << 8 << endl;
```

```
44.42642
44.426
44.43
0.00048
5.660e+02
44.426423
f
255
10
Press any key to continue . . .
```

# ÖRNEKLER

- olcu\_v1.cpp
- olcu\_v2.cpp
- olcu\_v3.cpp

# KAYNAKLAR

- Deitel, C++ How To Program, Prentice Hall
- Horstmann, C., Budd, T., Big C++, Jhon Wiley & Sons, Inc.
- Robert Lafore, Object Oriented Programming in C++, Macmillan Computer Publishing
- Prof. Dr. Celal ÇEKEN, Programlamaya Giriş Ders Notları
- Prof. Dr. Cemil ÖZ, Programlamaya Giriş Ders Notları