

BM102

Algoritma ve Programlama II

**Yapıcı ve Yıkıcı Fonsiyonlar**  
**(Constructor and Destructor)**

# İçerik



## 1. Yapıcı Fonksiyonlar

1. Yapıcı Fonksiyon Tanımı
2. Üyelere İlk Atama
3. Yapıcı Fonksiyonu Yükleme
4. Yapıcı Fonksiyon Çağırımı
5. Kopya Yapıcı Fonksiyon

## 2. Otomatik Üye Atamaları

## 3. Yıkıcı Fonksiyonlar

## *Çözümlü Sorular*

# Hedefler



- Yapıcı fonksiyon ve diğer fonksiyonlar arasındaki farkları listeleme
- Yapıcı fonksiyonlarla, sınıf üyelerine ilk atama yapma
- Kopya yapıcı fonksiyon yazma
- Varsayılan yapıcı fonksiyonu yazma
- Verilen nesne tanımlarına uygun yapıcı fonksiyonları tanımlama
- Yıkıcı fonksiyonların amacını anlatma
- Yıkıcı fonksiyon tanımı yapma
- Otomatik üye atamanın kaç şekilde yapılabileceğini örneklerle anlatma

# 1. Yapıcı Fonksiyonlar

- *İng. constructor*
- Bir nesnenin tanımında bu nesne üyelerine ilk değer ataması yapmak için kullanılan özel fonksiyonlardır
- Bir nesne tanımlandığında otomatik olarak çalışır.

# 1.1. Yapıcı Fonksiyon Tanımı

- Yapıcı fonksiyon yazarken dikkat edilmesi gereken hususlar:
  - Yapıcı fonksiyonun ismi sınıf ismi ile aynı olmalıdır.
  - Yapıcı fonksiyonların döndürme tipi yoktur.
  - Nesne tanımı yapıldığında otomatik olarak çağırıldıkları için yapıcı fonksiyonları özel olarak çağırmaya gerek yoktur.

# 1.1. Yapıcı Fonksiyon Tanımı...

- Bu örnek programda 5 x 10'luk bir dikdörtgen nesnesi oluşturalım, alanını hesaplayıp yazdıralım.

## (Yapıcı Fonksiyon Kullanmayan Çözüm)

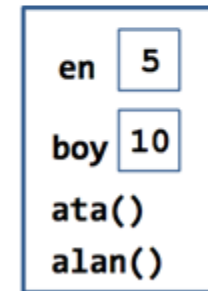
```
#include <iostream>
using namespace std;
class Dikdortgen
{
    int en,boy;
public:
    void ata(int e,int b){en=e; boy=b;}
    int alan() { return en*boy; }
};
int main()
{
    Dikdortgen x;
    x.ata(5,10);
    cout<<"Alan= "<<x.alan()<<endl;
    return 0;
}
```

## Çıktı

Alan= 50



Dikdortgen Sınıf Diyagramı



x

x Nesnesi

# 1.1. Yapıcı Fonksiyon Tanımı...

## (Yapıcı Fonksiyon Kullanan Çözüm)

```
#include <iostream>
using namespace std;
class Dikdortgen
{
    int en,boy;
public:
    Dikdortgen(int e,int b)        // Yapıcı fonksiyon tanımı
    {
        en=e;
        boy=b;}
    int alan()
    {
        return en*boy;
    }
};
int main()
{
    Dikdortgen x(5,10);           // x adlı bir dikdörtgen tanımlanır ve yapıcı çağırılır
    cout<<"Alan= "<<x.alan()<<endl; // x'in alanı yazdırılır
    return 0;
}
```

## Çıktı

Alan= 50

# 1.1. Yapıcı Fonksiyon Tanımı...

- Yapıcı fonksiyon tanımı:

```
Dikdortgen(int e,int b){en=e; boy=b;}
```

- Yapıcı fonksiyon çağırımı:

```
Dikdortgen x(5,10);
```

- Yapıcı fonksiyonun ismi (Dikdortgen) sınıf ismi ile aynıdır.
- Yapıcı fonksiyonun döndürme tipi tanımlanmamıştır.
- `ata()` fonksiyonunda olduğu gibi `e` ve `b` parametrelerini `en` ve `boy` üyelerine atamakla görevlidir.

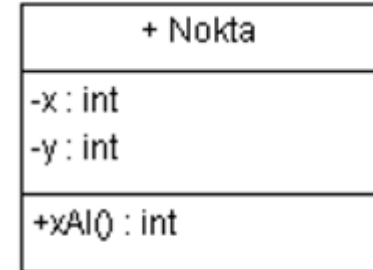
Yapıcı fonksiyonları **public** olarak tanımlamayı unutmamamız gerekir, aksi takdirde otomatik olarak çağırılmaları mümkün olamaz.



# 1.1. Yapıcı Fonksiyon Tanımı...

```
#include <iostream>
using namespace std;
class Nokta
{
    int x,y;
public:
    Nokta(int a=0,int b=0)           // Yapıcı fonksiyon tanımı
    {
        x=a;
        y=b;
    }
    int xAl()
    {
        return x;
    }
};

int main()
{
    Nokta bir(12,6);                // bir adlı nokta tanımlanır ve yapıcı çağırılır
    Nokta iki(25);                  // iki adlı nokta tanımlanır ve yapıcı çağırılır
    Nokta uc;                       // uc adlı nokta tanımlanır ve yapıcı çağırılır
    cout<<"bir'in x koordinati: "<<bir.xAl()<<endl;    // bir'in x'i yazdırılır
    cout<<"iki'nin x koordinati: "<<iki.xAl()<<endl;    // iki'nin x'i yazdırılır
    cout<<"uc'un x koordinati: "<<uc.xAl()<<endl;      // uc'ün x'i yazdırılır
    return 0;
}
```



Nokta Sınıf Diyagramı

## Çıktı

bir'in x koordinati: 12  
iki'nin x koordinati: 25  
uc'un x koordinati: 0

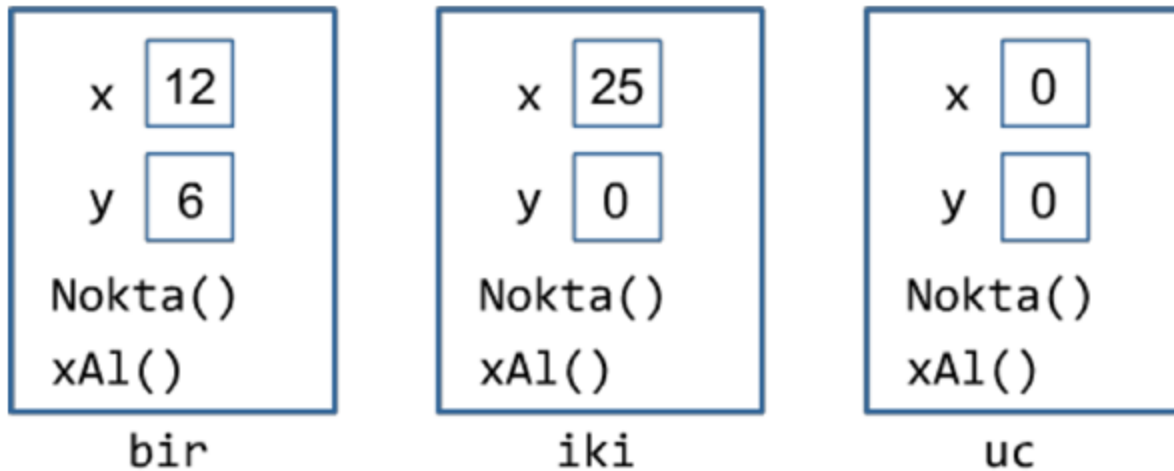
# 1.1. Yapıcı Fonksiyon Tanımı...

Nokta bir(12,6);

Nokta iki(25);

Nokta uc;

bir nesnesi için yapıcı fonksiyona 2 parametre, iki nesnesi için 1 parametre yollanır, uc nesnesi için ise hiç bir parametre gönderilmez. Bu durumda iki ve uc nesneleri için varsayılan değerler kullanılır ve bu nesneler aşağıdaki şekilde oluşur.



bir, iki ve uc Nokta nesneleri

# 1.1. Yapıcı Fonksiyon Tanımı...

- *Varsayılan Yapıcı Fonksiyon* (Default constructor)
- Hiç parametresi olmayan yapıcı fonksiyonlardır

```
Ogrenci(); // Varsayılan yapıcı fonksiyon prototipi
```

Sınıf

Yapıcı

```
Ogrenci::Ogrenci()  
{...} // Varsayılan yapıcı fonksiyon tanımı
```

## 1.2. Üyelere İlk Atama

- *İng. Member initialization*
- Yapıcı fonksiyonlarında atama operatörünü (=) kullanmadan nesne üyelerine ilk değerlerini atamaya *üyelere ilk atama* adı verilir.

Bir sınıfın sabit, referans ve nesne üyelerine sadece ilk atama ile değer verilebilir

# 1.2. Üyelere İlk Atama...

```
class Basit
{
    int i;
    float f;
public:
    Basit(int, float);           // Yapıcı fonksiyon prototipi
};
Basit::Basit(int a, float b)    // Yapıcı fonksiyon tanımı
{
    i=a;
    f=b;
}
int main()
{
    Basit nsn(1, 2.5);          // nsn nesnesi tanımı sırasında yapıcı çağırılır
    return 0;
}

Basit::Basit(int a, float b):i(a), f(b) // İlk atama ile yapıcı fonksiyon tanım
{
}
```

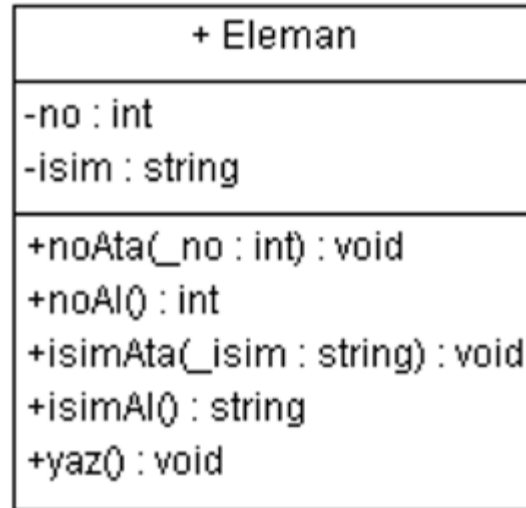
*= operatörü ile atama*

*Üyelere İlk atama*

*İlk atama yöntemi*

## 1.3. Yapıcı Fonksiyonunu Yükleme

- Yapıcı fonksiyonlar, farklı parametrelerle birden fazla tanımlanabilir.



Eleman Sınıf Diyagramı

# 1.3. Yapıcı Fonksiyonunu Yükleme...

```
#include <iostream>
#include <string>
using namespace std;
class Eleman
{
    int no;
    string isim;
    public:
        Eleman(){}; // 1. yapıcı fonksiyon tanımı
        Eleman(int,string); // 2. yapıcı fonksiyon prototipi
        void noAta(int _no){no=_no;}
        int noAl(){return no;}
        void isimAta(string _isim){isim=_isim;}
        string isimAl(){return isim;}
        void yaz()
        {cout<<"Isim: "<<isim<<" No: "<<no<<endl;}
};
Eleman::Eleman(int i,string s) // 2. yapıcı fonksiyon tanımı
{
    no=i;
    isim=s;
}
```

```
int main() {
    Eleman e1; // 1. yapıcı çağırılır
    Eleman e2(123,"Ali"); // 2. yapıcı çağırılır
    int no;
    string isim;
    cout<<"Elemanın numara ve ismini girin:";
    cin>>no>>isim;
    e1.noAta(no);
    e1.isimAta(isim);
    e1.yaz();
    e2.yaz();
    return 0;
}
```

## Çıktı

```
Elemanın numara ve ismini girin:432 Ahmet
Isim: Ahmet No: 432
Isim: Ali No: 123
```

## 1.3. Yapıcı Fonksiyonunu Yükleme...

- Eğer bir sınıda hiçbir yapıcı fonksiyon tanımlanmamışsa, derleyici otomatik olarak içi boş bir varsayılan yapıcı fonksiyon tanımlar.
- Ama eğer sınıda başka bir yapıcı fonksiyon tanımı yapıldıysa, varsayılan fonksiyon otomatik olarak yaraalmaz.



# 1.3. Yapıcı Fonksiyonunu Yükleme...

```
class Eleman
{
    int no;
    string isim;
};
```

→ Eleman e; ✓

```
class Eleman
{
    int no;
    string isim;
public:
    Eleman(int _no,string _isim)
        :no(_no),isim(_isim);
};
```

→ Eleman e; ✗

```
class Eleman
{
    int no;
    string isim;
public:
    Eleman(){}
    Eleman(int _no,string _isim)
        :no(_no),isim(_isim);
};
```

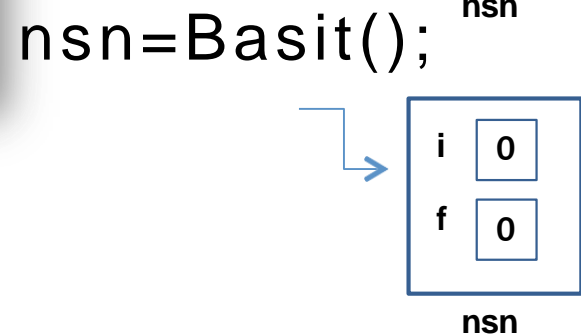
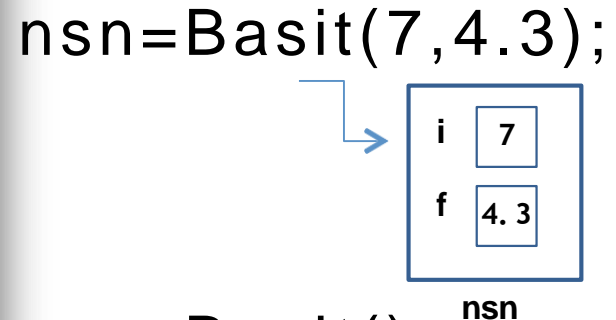
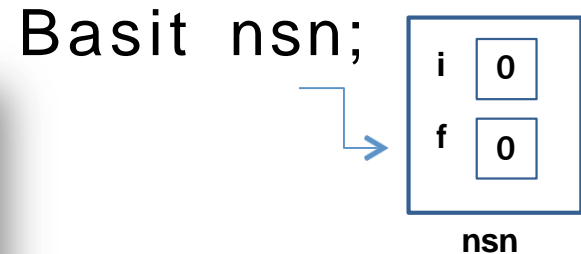
→ Eleman e; ✓

## 1.4. Yapıcı Fonksiyon Çağırımı

- Yapıcı fonksiyonlar nesne tanımı yapıldığı sırada otomatik olarak çağırılır
- Ayrıca yapıcı fonksiyon istenildiğinde de çağırılabilir

# 1.4. Yapıcı Fonksiyon Çağırımı...

```
class Basit
{
    int i;
    float f;
public:
    Basit();           // Yapıcı fonksiyonlar
    Basit::Basit(int a,float b):i(a),f(b) {}
};
Basit::Basit()
{
    i=0;
    f=0.0;
}
```



# 1.5. Kopya Yapıcı Fonksiyonlar

- *İng. Copy constructor*
- Aynı sınıf tipinde başka bir nesneyi parametre olarak alan yapıcı fonksiyonlardır
- Parametre olarak gelen nesneyi, içinde bulunduğu nesneye kopyalar.

# 1.5. Kopya Yapıcı Fonksiyonlar...

```
#include <iostream>
using namespace std;
class A
{
    int no;
public:
    A();                // Varsayılan yapıcı fonksiyon prototipi
    A(A &par);          // Kopya yapıcı fonksiyon prototipi
};
A::A()                 // Varsayılan yapıcı fonksiyon tanımı
{
    no=0;
    cout<<"Varsayılan Yapıcı"<<endl;
}
A::A(A &par)           // Kopya yapıcı fonksiyon tanımı
{
    no=par.no;
    cout<<"Kopya Yapıcı"<<endl;
}
```

```
int main()
{
    A a1;               // Varsayılan yapıcı çağırılır
    A a2(a1);           // Kopya yapıcı çağırılır
    A a3=a2;            // Kopya yapıcı çağırılır
    return 0;
}
```

## Çıktı

```
Varsayılan Yapıcı
Kopya Yapıcı
Kopya Yapıcı
```

## 2. Otomatik Üye Atamaları

- Aynı tipte bir nesne başka bir nesneye atanırsa, bu nesnenin tüm üyeleri ikinci nesnenin üyelerine otomatik olarak atanır.
  - Tanımlama sırasında bir nesne bir başka nesneye atanırsa
  - Bir fonksiyona parametre olarak bir nesne yollanırsa
  - Bir fonksiyondan bir nesne döndürülürse

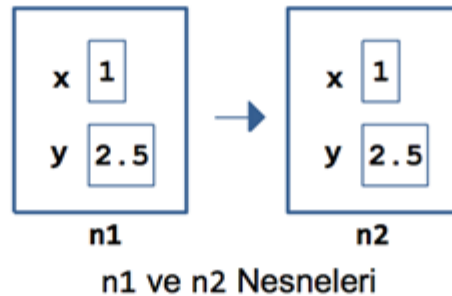
## 2. Otomatik Üye Atamaları

- Otomatik üye ataması üç şekilde gerçekleşir.
  - Tanımlama sırasında bir nesne bir başka nesneye atanırsa
  - Bir fonksiyona parametre olarak bir nesne yollanırsa
  - Bir fonksiyondan bir nesne döndürülürse

# 2. Otomatik Üye Atamaları...

*Tanımlama sırasında bir nesnenin bir başka nesneye atanma durumu:*

```
#include <iostream>
using namespace std;
class Bir
{
    int x;
    float y;
public:
    Bir(int _x,float _y):x(_x),y(_y){}
};
int main()
{
    Bir n1(1,2.5);
    Bir n2=n1; // n1 nesnesinin tüm üyeleri n2 nesnesine otomatik olarak kopyalanır
    return 0;
}
```

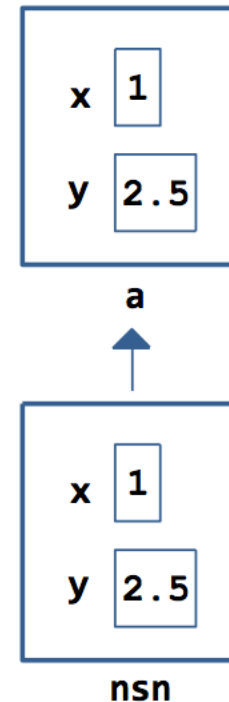




## 2. Otomatik Üye Atamaları...

*Bir fonksiyona parametre olarak bir nesne yollanma durumu:*

```
class Bir
{
    int x;
    float y;
public:
    Bir(int _x,float _y):x(_x),y(_y){}
};
void fon(Bir a)
{
}
int main()
{
    Bir nsn(1,2.5);
    fon(nsn);           // nsn nesnesi a nesnesine yollanır
    return 0;
}
```

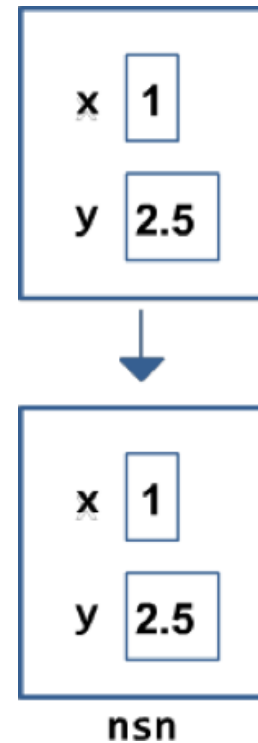


nsn Nesnesinin Parametre Olarak Gönderilmesi

## 2. Otomatik Üye Atamaları...

*Bir fonksiyondan bir nesneyi döndürme durumu:*

```
class Bir
{
    int x;
    float y;
public:
    Bir(int _x,float _y):x(_x),y(_y){}
    Bir(){}
};
Bir fon()
{
    return Bir(1,2.5);
}
int main()
{
    Bir nsn;
    nsn=fon(); // fon()'un döndürdüğü nesne nsn'e atanır
    return 0;
}
```



nsn Nesnesinin Döndürülmesi

## 2. Otomatik Üye Atamaları...

- Aşağıdaki işlemde görüldüğü gibi ikirasyonel sayıyı çarpan bir program yazalım.

1 ← Pay  
3 ← Payda

| + Rasyonel  |
|---|
| +pay : int<br>+payda : int  |
| +payAl() : int<br>+paydaAl() : int<br>+carp(r : Rasyonel&) : Rasyonel |

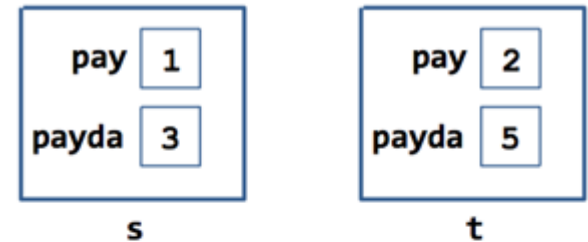
Rasyonel Sınıf Diyagramı

## 2. Otomatik Üve Atamaları...

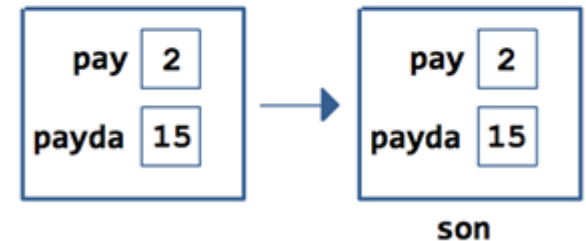
```
#include <iostream>
using namespace std;
class Rasyonel
{
    int pay,payda;
public:
    Rasyonel(int x,int y):pay(x),payda(y){} //Yapıcı fonksiyon tanımı
    int payAl(){return pay;} //payı döndür
    int paydaAl(){return payda;} //paydayı döndür
    Rasyonel carp(Rasyonel&); //carp() fonksiyonu prototipi
};
Rasyonel Rasyonel::carp(Rasyonel& r) //carp() fonksiyonu tanımı
{
    int a=r.payAl();
    int b=r.paydaAl();
    return Rasyonel(pay*a,payda*b); //İsimsiz sonuc nesnesini döndürür
}
int main()
{
    Rasyonel s(1,3),t(2,5); //1/3 ve 2/5 sayılarını tutan nesneler oluşturulur.
    Rasyonel son=s.carp(t); //Çarpım sonucu son nesnesine atanır.
    cout<<s.payAl()<<"/"<<s.paydaAl();
    cout<<" * "<<t.payAl()<<"/"<<t.paydaAl();
    cout<<" = "<< son.payAl()<<"/"<<son.paydaAl()<<endl;
    return 0;
}
```

**Çıktı**

$$1/3 * 2/5 = 2/15$$



s ve t Nesneleri



son Nesnesi

# 3. Yıkıcı Fonksiyonlar

- *İng. Destructor*
- Yıkıcı fonksiyonlar ise bir nesne programda yok edilmeden hemen önce otomatik olarak çağırılan üye fonksiyonlardır.
- Nesnenin yaraaldığı blok bittiğinde o nesne yok edilir ve yok olmadan hemen önce yıkıcı fonksiyon otomatik olarak çağılır.
- Yıkıcı fonksiyonu yazarken şu noktalara dikkat etmemiz gerekir:
  - Yıkıcı fonksiyon ismi ~ işareti ile başlar ve sınıfla aynı ismi taşır
  - Yıkıcı fonksiyonlara yükleme yapılmaz
  - Parametreleri ve döndürme tipleri yoktur
  - Otomatik olarak çağınırlar

# 3. Yıkıcı Fonksiyonlar...

```
#include <iostream>
using namespace std;
class Eleman
{
    int no;
public:
    Eleman (int=0);           // Yapıcı fonksiyon prototipi
    ~Eleman();               // Yıkıcı fonksiyon prototipi
};

Eleman::~~Eleman()           // Yıkıcı fonksiyon tanımı

{
    cout<<"Yıkıcı fonksiyon"<<endl;
    cout<<"No: "<<no<<endl; }
};

Eleman::Eleman (int x)       // Yapıcı fonksiyon tanımı
{
    no=x;
    cout<<"Yapıcı fonksiyon"<<endl;
}

int main()
{
    Eleman kisi1(123),kisi2;
    return 0;
}
```

## Çıktı

```
Yapıcı fonksiyon
Yapıcı fonksiyon
Yıkıcı fonksiyon
No: 0
Yıkıcı fonksiyon
No: 123
```

# Çözümlü Sorular

## Soru

Ulke adında bir sınıf tanımı yapılmıştır. Bu sınıftan aşağıda verilen nesnelerin yaratılması istenmektedir. Buna göre Ulke sınıfını ve bu sınıfa ait gerekli yapıcı fonksiyon tanımlarını yapınız.


- a. `Ulke turkiye("Ankara","TR");` *//başkent ve uluslararası plaka*
- b. `Ulke almanya;`
- c. `Ulke ingiltere("Londra");`
- d. `Ulke kosova(2000000);` *//nüfusu*
- e. `Ulke ıspanya("Madrid","E","İspanyolca");`
- f. `Ulke estonya(372);` *//telefon kodu*



# Çözümlü Sorular

## Cevap

```
class Ulke
{
    string baskent, plaka, dil;
    long nufus;
    int telefon;
public:
    Ulke(string b, string p, string d)
        :baskent(b),plaka(p),dil(d){}
    Ulke(string b, string p):baskent(b),plaka(p){}
    Ulke(string b):baskent(b){}
    Ulke(){}
    Ulke(long n):nufus(n){}
    Ulke(int t):telefon(t){}
};
```





# Çözümlü Sorular

## Soru

Aşağıda isim ve telefon numaralarını içeren TelefonRehberi tipinde rehber1 nesnesi oluşturulmuştur. Bu rehberdeki bilgileri de içerecek şekilde yeni bir nesne oluşturmak istenmektedir. Kopya yapıcı fonksiyonları kullanarak rehber1 nesnesinin içeriğini kopyalayarak rehber2 nesnesini yaratan ve yeni nesnenin içeriğini aşağıda verilen örnekteki gibi ekrana yazdıran bir program yazınız.

```
string isimler[] = {"Aydan Sener","Fatma Girik",  
                  "Filiz Akin","Hulya Kocyigit",  
                  "Turkan Soray"};  
  
long telefonlar[] =  
    {1111111,2222222,3333333,4444444,5555555};  
TelefonRehberi rehber1(isimler,telefonlar);  
TelefonRehberi rehber2(rehber1);  
rehber2.yazdir();
```

### Örnek Çıktı

```
Aydan Sener 1111111  
Fatma Girik 2222222  
Filiz Akin 3333333  
Hulya Kocyigit 4444444  
Turkan Soray 5555555
```



# Çözümlü Sorular

## Cevap

```
#include <iostream>
using namespace std;
class TelefonRehberi
{
    string *isimler;
    long *telefonlar;
public:
    TelefonRehberi(string *_isimler, long *_telefonlar)
        :isimler(_isimler),telefonlar(_telefonlar){}
    TelefonRehberi(TelefonRehberi &rehber);
    void yazdir();
};
TelefonRehberi::TelefonRehberi(TelefonRehberi &rehber)
{
    isimler = rehber.isimler;
    telefonlar = rehber.telefonlar;
}
void TelefonRehberi::yazdir()
{
    for(int i=0;i<5; i++)
        cout<<isimler[i]<<" "<<telefonlar[i]<<endl;
}
int main()
{
    string isimler[] = {"Aydan Sener","Fatma Girik",
        "Filiz Akin","Hulya Kocyigit","Turkan Soray"};
    long telefonlar[] = {1111111,2222222,3333333,4444444,5555555};
    TelefonRehberi rehber1(isimler,telefonlar);
    TelefonRehberi rehber2(rehber1);
    rehber2.yazdir();
    return 0;
}
```

