

## 9. Kurucu ve yok ediciler

### Kurucular(Constructor)

Kurucu fonksiyonlar sınıfların üye verilerine(değişkenlerine) başlangıç değerlerini vermek için kullanılan özel fonksiyonlardır. Kurucu fonksiyonlar nesneler tanımlandıkları anda otomatik olarak çalışan fonksiyonlardır. Kurucu fonksiyonlar sınıf bildirimleri yapılırken sınıf isimleri ile aynı ismi alırlar. Bu fonksiyonlarda diğer fonksiyonlarda olduğu gibi parametrelili veya parametrelili olarak tanımlanabilirler. Aynı sınıf için birden fazla kurucu fonksiyon tanımlanabilir. Bu durumda kurucu fonksiyonların aşırı yüklenmesi durumu ortaya çıkar. Aşırı yüklenmiş fonksiyonlar bilindiği gibi parametrelerin sayısına, veya türüne göre hangi fonksiyonun çağrılacağına karar verilir. Aşağıda iki tür kurucu fonksiyon prototipi görülmektedir.

```
Dortgen::Dortgen() ;           //parametresiz kurucu
Dortgen::Dortgen(int,int,int,int); //parametrelili kurucu
```

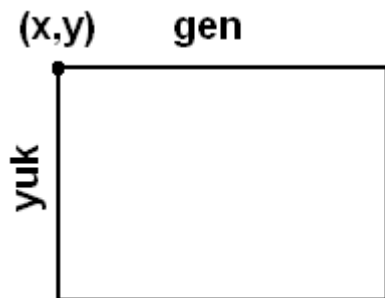
Kurucu fonksiyonların dönüş tipi yoktur. Geri dönüş tipi olarak herhangi bir tip (void bile) belirtilmez. Kurucu fonksiyonlar public olarak tanımlanmalıdır.

### Yok ediciler(Destructors)

Yok edici fonksiyonlar Nesneler yok edildiklerinde, bellekten atıldıklarında otomatik olarak çalışan fonksiyonlardır. Yok edici fonksiyonlarda sınıf ismi ile aynı ismi alırlar ancak önlerine '~' işareti konulur. Yok edici fonksiyonlar sınıf üyeleri için dinamik olarak bellekten yer alındığında, alınan yerleri geri vermek için kullanılır. Aşağıda yok edici fonksiyon prototipi görülmektedir. Yok ediciler parametre almazlar ve geriye değer döndürmezler. Bir sınıfta sadece bir adet yok edici fonksiyon olabilir.

```
Dortgen::~~Dortgen();
```

Şimdi daha önceki derslerimizde kullandığımız Dörtgen sınıfını biraz değiştirelim. Dörtgen sınıfı aşağıdaki resimde görüldüğü gibi özellikleri belirlensin. **x**, **y** değişkenleri dörtgenin üst köşe koordinatını, **yuk** değişkeni yüksekliği ve **gen** değişkeni de genişliği belirtsin.



```

#include <iostream>
#include <conio.h>
using namespace std;

class Dortgen {
    int x, y,gen,yuk ;
public:
    Dortgen() ;    //parametresiz kurucu
    Dortgen(int,int,int,int);    //parametrelili kurucu
    ~Dortgen();    //yok edici fonksiyon
    void DegerVer(int,int,int,int);
    int Alan (void);
};

//Yok edici fonksiyon
Dortgen::~Dortgen() {
    cout<<"yok edici calisti"<<endl;    //Yok edici bu örnekte hiçbir işe yararamıyor.
                                         //Sadece ekrana mesaj veriyor
}

//parametresiz kurucu fonksiyonu
Dortgen::Dortgen() {
    x = 5;
    y = 5;
    gen=10;
    yuk=3;
    cout<<"Parametresiz kurucu calisti"<<endl;
}

//parametrelili kurucu fonksiyonu
Dortgen::Dortgen(int a, int b, int g, int h) {
    x = a;
    y = b;
    gen=g;
    yuk=h;
    cout<<"Parametrelili kurucu calisti"<<endl;
}

```

```

void Dortgen::DegerVer(int a, int b, int g, int h) {
x = a;
y = b;
gen=g;
yuk=h;

}

int Dortgen::Alan (void) {
    return gen*yuk;
}

int main ()
{
    {
        Dortgen D1,D2(3,7,10,4);    //Bu satırda D1 ve D2 nesneleri tanımlanıyor
                                    // Önce D1 için paramertresiz kurucu, Sonra
                                    // D2 nesnesi için parametrelili kurucu çalışıyor

        //D1.DegerVer(3,4,7,9);    // Daha sonra D1 ve D2 nesnelerinin verilerine
        //D2.DegerVer(7,2,6,60);    //değer verilebilir

        cout << "Alan1: " << D1.Alan()<<'\n';
        cout << "Alan2: " << D2.Alan()<<endl;
    }

                                    //Burada D1 ve D2 için yok ediciler çalıştı

    getch();
    return 0;
}

```

```

Parameresiz kurucu calisti
Parametrelili kurucu calisti
Alan1: 30
Alan2: 40
yok edici calisti
yok edici calisti

```

Aşağıdaki programda kurucu ve yok edici fonksiyonların işlevleri daha iyi anlaşılmaktadır. Bu programda Dortgen sınıfının üye verileri int \*x, \*y,\*gen,\*yuk ; şeklinde pointer olarak tanımlanmakta ve kurucu fonksiyonlarda bu pointerlere dinamik olarak ayrılan yerler atanmaktadır.Daha sonra yok edici fonksiyon ile de ayrılan yerler geri verilmektedir.

//Bir başka örnek

#include <iostream>

#include <conio.h>

using namespace std;

```
class Dortgen {
    int *x, *y,*gen,*yuk ;
public:
    Dortgen() ;    //parametresiz kurucu
    Dortgen(int,int,int,int);    //parametreli kurucu
    ~Dortgen();    //yok edici fonksiyon
    void DegerVer(int,int,int,int);
    int Alan (void);
};
```

//Yokedici fonksiyon

```
Dortgen::~Dortgen() {
    delete x;
    delete y;
    delete gen,yuk;
    cout<<"yok edici calisti"<<endl;
}
```

//parametresiz kurucu fonksiyonu

```
Dortgen::Dortgen() {
    x = new int;    //Dinamik olarak bellekten yer alınıyor
    y = new int;
    gen=new int;
    yuk=new int;
    *x=10;
    *y=20;
    *gen=5;
    *yuk=6;
```

```

        cout<<"Parameresiz kurucu calisti"<<endl;
    }
    //parametrelili kurucu fonksiyonu
    Dortgen::Dortgen(int a, int b, int g, int h) {
        x = new int; //Dinamik olarak bellekten yer aliniyor
        y = new int;
        gen=new int;
        yuk=new int;

        *x = a;
        *y = b;
        *gen=g;
        *yuk=h;
        cout<<"Parametrelili kurucu calisti"<<endl;
    }

    void Dortgen::DegerVer(int a, int b, int g, int h) {
        *x = a;
        *y = b;
        *gen=g;
        *yuk=h;
    }

    int Dortgen::Alan (void) {
        return *gen*(*yuk);
    }

    int main ()
    {
        {
            Dortgen D1,D2(3,7,10,4);
            Dortgen *D3=new Dortgen;

            D1.DegerVer(3,4,7,9);
            D2.DegerVer(7,2,6,60);
            D3->DegerVer(5,5,12,3); //D3 pointer nesne

            cout << "Alan1: " << D1.Alan()<<'\n';
            cout << "Alan2: " << D2.Alan()<<endl;
            cout << "Alan3: " << D3->Alan()<<endl;

            delete D3;
        }

        getch();
    }

```

```
    return 0;
}
```

### Kurucu fonksiyonlarda ilk değer ataması

Kurucu fonksiyonlarda nesnelerin verilerine ilk değerlerini atamak için atama '=' operatörün kullanılmaktadır. Ancak sabit verilere değer atamak için atama '=' operatörü kullanılmaz. Aşağıdaki örneği inceleyiniz.

```
class C{
    int s;           // sabit olmayan veri
    const float pi; // sabit veri

public:
    C() { // Kurucu fonksiyon
        s = 0;      // Doğru, çünkü s sabit değil
        pi = 3.14; // HATA! pi sabit veri
    }
};
```

Aşağıdaki gibi bir yazım da derleme hatasına neden olur.

```
class C{
    int s=0;           // sabit olmayan veri
    const float pi=3.14; // HATA! S sabit veri

}
```

Bu problem kurucu fonksiyonlarda ilk değer atama yapısı (constructor initializer) kullanılarak çözülür.

Kurucularda verilere ilk değer atamak için kurucu fonksiyonun adından sonra iki nokta üst üste (:) konur, ardından ilk değer atanacak verinin adı ve parantez içinde atanacak değer yazılır.

```
class C{
    int s;           // sabit olmayan veri
    const float pi; // sabit veri

public:
    C():pi(3.14) { // Kurucu fonksiyon pi ye 3.14 değeri veriliyor.
        s = 0;      // Doğru, çünkü s sabit değil
    }
};
```

Tüm üye verilere değer atamak için bu yapı kullanılabilir. Eğer kurucu fonksiyonun başka bir görevi yoksa gövdesi boş kalabilir.

```
C():pi(3.14), s(0)
{ } //gövde boş kalabilir
```

## Kopyalama Kurucusu

Kopyalama kurucusu özel bir kurucu fonksiyondur. Daha önceden tanımlanmış var olan bir nesnenin verilerini yeni oluşturulan nesneye kopyalarlar. Böylece yeni oluşturulan nesne var olan nesnenin bir kopyası olur. Bu fonksiyonlar parametre olarak kendisi ile aynı sınıftan bir nesneye referans alır.

```
#include <iostream>
#include <conio.h>
using namespace std;

class Dortgen {
public:
    int x, y;
    Dortgen(); //Kurucu
    Dortgen(const Dortgen &); //Kopyalama kurucusu
    void DegerVer(int,int);
    int Alan (void);
};

Dortgen::Dortgen() { //parametresiz kurucu
    x = 5;
    y = 5;
}

Dortgen::Dortgen(const Dortgen &nesne) { //Kopyalama kurucusu
    x = nesne.x;
    y = nesne.y;
}

void Dortgen::DegerVer(int a, int b) {
    x = a;
    y = b;
}

int Dortgen::Alan (void) {
    return x*y;
}
```

```

int main ()
{
    Dortgen D1;

    D1.DegerVer(9,10);

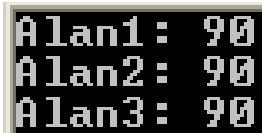
    Dortgen D2=D1;
    Dortgen D3(D1); //Kopyalama Kurucusu calisti

    D1=D2;

    cout << "Alan1: " << D1.Alan()<<'\n';
    cout << "Alan2: " << D2.Alan()<<endl;
    cout << "Alan3: " << D3.Alan()<<endl;

    getch();
    return 0;
}

```



```

Alan1: 90
Alan2: 90
Alan3: 90

```