# BAĞLANTILI LİSTELER

## Genel Bilgiler

Bağlantılı liste, aynı kümeye ait veri parçalarının birbirlerine, bellek üzerinde, sanal olarak bağlanmasıyla oluşturulur. Tüm veriler bir tren katarı gibi birbirine bağlı parçalardan oluşur.

# Genel Bilgiler

Bağlantılı liste veri yapısında birisi veri, diğeri bağlantı bilgisi olmak üzere temelde iki kısım bulunur. Veri kısmında, o uygulama için gerekli bir veya birkaç bilgi bulunabilir, işaretçi olarak adlandırılan bağlantı kısmında ise bağlantının nereye yapıldığını gösteren bir veya birkaç adres bilgisi bulunabilir.



# Bağlantılı Liste Tipleri

Bağlantılar, veri parçalarının geliş/giriliş sırasına göre olabileceği gibi belirli bir anahtar sözcüğe göre sıralı özellikte de yapılabilir. Eğer bağlantılar;

belirli bir anahtar sözcüğe göre sıralı özellikte yapılıyorsa sıralı bağlantı liste (ordered link list),

geliş sırasına göre yapılıyorsa, bağlantılı liste (link list) olarak adlandırılır.

## Bağlantılı Liste Kavramları (LinkList)

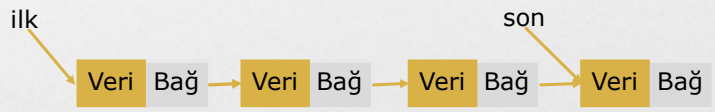
Bağlantı listesindeki veri kısmı uygulamanın gereksinimine göre bir karakterlik olabileceği gibi bir string, bir tamsayı, bir kesirli sayı veya bunların karışımından oluşan bir veri kümesi olabilir.

Bağ kısmı ise bağlantılı liste üzerinde bir sonraki veri parçasının yerini işaret eden bir adres veya indis bilgisidir. Uygulama ve problemin gereksinimine göre bağ kısmı da birkaç parçadan oluşabilir.



## Bağlantılı Liste Kavramları (LinkList)

Şekilde en yalın bağlantılı liste gösterilmiştir; burada, bağlantılar tek yönlü olup liste üzerinde baştan sona doğru bir yönde hareket edilebilir.



## Bağlantılı Liste Kavramları (LinkList)

Bağlantılı liste üzerinde arama işlem yapılırken başlangıç noktası ilk değişkendir. Liste üzerindeki ara düğümlere doğrudan erişilemez; hep, bu ilk adlı değişken üzerinden başlanmalıdır.

Dolayısıyla bir bağlantılı liste üzerinde arama yapmanın maliyeti en kötü durumda O( n) olur.



## Bağlantılı Liste (Linked List)

Bağlantılı listede elemanları birbirini izleyen bellek adreslerinde veya dizi gözlerinde olmayıp listeye ekleme sırasındaki boş yere bağlıdır.

Bağlantılı listeler dizi üzerinde tutulsalar dahi bağlantı bilgisi gerekmektedir.

## Liste Uzunluğu (Length of List)

Listedeki art arda bağlı elamanların/düğümlerin sayısıdır. Eğer elaman sayısı sıfır ise boş liste (empty üst) olarak adlandırılır ve Φ karakteriyle veya {} şeklinde gösterilir.

Program tasarımı sırasında içerisine bir şeyler yazılmamış olan bir çift aç-kapa parantez "()" bos küme anlamında kullanılır.



## Alt liste (Sublist):

Listenin, genel özellikleri aynı kalmak koşuluyla, üzerinde hareket edilebilecek belirli bir parçasıdır.

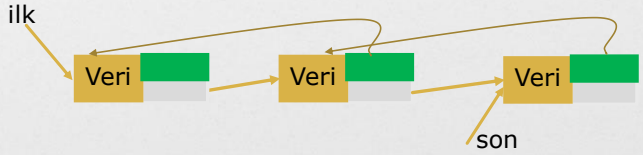
## Tek Yönlü Bağlantılı Liste (One-Linked List)

Düğümler arasında tek yönlü bağlantı vardır; bu durumda listenin başından sonuna doğru hareket edilebilir. Liste üzerindeki işlemler listenin başından başlar. Ekleme, arama, listeleme gibi işlemlerin karmaşıklığı *O(n)* olur. Listenin tutulması için iki tane işaretçi kullanılırsa ekleme karmaşıklığı *0(l),* arama ve listeleme karmaşıklığı da *O( n )* olur.



## Çift Yönlü Bağlantılı Liste (Double-Linked List)

Düğümler arasındaki bağlantı; bir düğüm hem bir sonraki hem de bir önceki düğümü işaret eder. Dolayısıyla iki yönlü, hem listenin sonuna hem de başına doğru hareket edilebilir. Tek yönlü bağlantıya göre daha esnek yapısı vardır ve algoritmaların geliştirilmesi daha kolay olur denilebilir. İki İşaretçi değişken kullanılırsa, arama ve üsteleme maliyetleri yine *O( n )* olurken ekleme maliyeti *0(l)* olur.

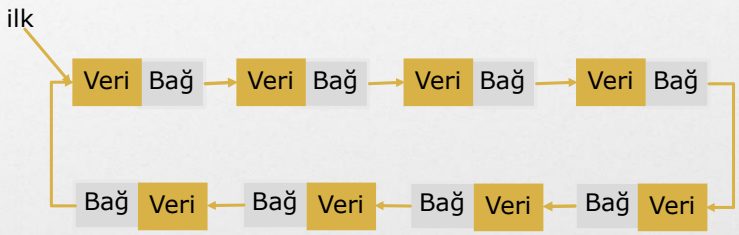


## Çevrimsel Bağlantılı Liste (Circular Linked List):

Çevrimsel bağlantılı listelerde düğümler arasında çevrimsel bir bağlantı vardır; eğer *n* elemanlı bir bağlantılı listede *n* adım hareket edilirse, yine, listenin başladığı noktaya dönülür. Çevrimli bağlantılı listede düğümlerden birinin adresini bilmek yeterlidir; buradan tüm liste kayıtlarına erişilebilir. Uygulama istenen fonksiyonların gerçekleştirimi kolaylaştırdığı durumlarda iki yönlü çevrimsel bağlantılı liste oluşturulabilir.



## Çevrimsel Bağlantılı Liste (Circular Linked List):



## Karma Bağlantılı Liste (Hybrid Linked List)

Eğer gerekiyorsa, algoritma tasarımının kolaylaştırılması amacıyla, çeşitli bağlantılı listelerin bir araya getirilmesiyle oluşturulan listeler karma bağlantılı liste olarak adlandırılır.



## Bağlantılı Listelerin Bellekte Tutulma Biçimleri

Genel olarak bağlantılı listeler dizi üzerinde kurulabilir veya *malloc ( )* gibi fonksiyonla bellek alanları dinamik olarak istenip kurulabilir. Bağlantılı listenin dinamik bellek yöntemiyle kullanılması daha yaygındır denilebilir. Bu durumda işaretçi değişkenler kullanılır.

## Bağlantılı Listelerin Bellekte Tutulma Biçimleri

- Bağlantılı liste boş iken her iki işaretçi değişken NULL değerine sahiptirler;
- *malloc ()* ile yer istendiğinde alınan yerin başlangıç adresi p gibi geçici bir işaretçi değişkende tutulur,
- ilk ekleme yapıldığında ilk ve son işaretçileri aynı elemanı gösterir;
- daha sonraki eklemeler farklı değerlere sahip olurlar.



## Bağlantılı Listelerin Bellekte Tutulma Biçimleri

Listeden bir eleman silinmesi/çıkarılması gerektiğinde

- önce bağlantı koparılır;
- bağlantı, silinmek istenenin bir önceki ve bir sonraki arasında yeniden kurulur,
- silinecek elemanın işgal ettiği yer *free (p)* gibi bir fonksiyonla bellek yönetim birimine iade edilir.

## Bağlantılı Listelerin Bellekte Tutulma Biçimleri

Dizilere, programın başında kaç elemanlı olacağı belirtilerek bildiriliyorsa statik dizi, programın yürütülmesi sırasında işletim sisteminden isteniyorsa dinamik dizi denilir. Ancak buradaki dinamiklik ayrık bellek alanında olan dinamiklikten farklıdır; burada, dizinin tüm elemanlarının sığacağı bellek alanı önceden istenir ve dizi dolduğunda yeni eklemeler yapılamaz. Ekleme yapılabilmesi dinamik tablo üreten özel fonksiyon kullanılmalıdır.



## Bağlantılı Listelerin Bellekte Tutulma Biçimleri

Bağlantılı liste dizi üzerinde tutulursa, program tasarımı sırasında dizi gözlerine erişim aşağıda verildiği gibi gerçekleştirilir.

```
strcpy(LISTE[0].string, "baba");
scanf("%s", LISTE[0].string);
LISTE[0].arka=2;
puts(LISTE[6].string);
printf("%d",    LİSTE[6].arka];
if(ilk==0)
```

| | | |
|---|---|---|
| 0 | baba | 2 |
| 1 | koş | 6 |
| 2 | bana | 3 |
| 3 | top | 4 |
| 4 | al | 1 |
| 5 | koş | - |
| 6 | ali | 5 |

## Bağlantılı Listelerin Bellekte Tutulma Biçimleri

Bağlantılı liste ayrık bellek alanları üzerinde tutulursa, program tasarımı sırasında dizi gözlerine erişim aşağıdaki verildiği gibi gerçekleştirilir:

```
Strcpy(ilk->string,    "baba");
scanf("%s",ilk->string) ;
ilk->arka=2;
puts(son->string);
printf("%d", son->arka);
if(ilk==NULL)
p=ilk;
q=ilk->arka;
```



## Dizi üzerinde mi? ayrık bellek alanlarında mı tutulmalıdır ?

Eğer liste boyu önceden belirliyse ve fazla büyümeyeceği düşünülüyorsa bağlantılı liste dizi üzerinde tutulabilir.

Ancak liste boyunun ne olacağı önceden bilinmiyorsa ve liste uzunluğu çok büyük olabilecekse bağlantılı liste kullanılmalıdır.

Her iki yöntemde de elemanları üzerinde dolaşma ardışıl olarak yapılacağı için işlemlerin zaman karmaşıklıkları aynı olur.

## Ayrık Alanlarda Bağlantılı Liste Uygulaması

```
struct liste{
  Elemanptr bas;
  Elemanptr son;
  };
  typedef struct liste Liste;
  typedef Liste* Listeptr;
  Listeptr yeni_liste(){
     Listeptr liste;
     Liste = malloc(sizeof(Liste));
     liste->bas == null;
     liste->son == null;
  return liste;
}
```

Aşağıda her bir kayıtın içerisinde bir tane bilgi ve bir tane bağlantı bilgisi olan veri yapısının ve kayıt adlı yeni bir veri türü oluşturulması:

```
public class Liste{
  Eleman bas;
  Eleman son;
  public Liste (){
     bas == null;
     son == null;
  }
}
```



Maliyet O(1)

*Kaynak: Olcay Taner Yıldız, C&&Java ile Veri Yapılarına Giriş,*

## Ayrık Alanlarda Bağlantılı Liste Uygulaması (Başına eleman ekleme)

```
void liste_basina_ekle(Listeptr I,
Elemanptr yeni){
if (I->son == NULL)
    I->son == yeni;
  yeni->ileri = I->bas;
  I->bas = yeni;
}
```

```
void listeBasinaEkle(Eleman  yeni){
if (son == NULL)
son == yeni;
yeni.ileri = bas;
bas = yeni;
}
```

*Kaynak: Olcay Taner Yıldız, C&&Java ile Veri Yapılarına Giriş,*



Maliyet O(1)

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Sona eleman ekleme)

```
void listeye_ekle(Listeptr I, Elemanptr yeni){
if (I->bas == NULL)
    I->bas == yeni;
else
    I->son->ileri = yeni;
  I->son = yeni;
}
```

```
void listeyeEkle(Eleman  yeni){
if (bas == NULL)
   bas == yeni;
else
   son.ileri = yeni;
son = yeni;
}
```



Maliyet O(N)

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Ortaya eleman ekleme)

```
void liste_orta_ekle(Elemanptr yeni, Elemanptr once){
     yeni->ileri =once->ileri;
      once->ileri = yeni;
}

void listeyOrtaEkle(Eleman  yeni, Eleman once){
     yeni.ileri =once.ileri;
      once.ileri = yeni;
}
```



Maliyet O(1)

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Listede arama)

```
Elemanptr liste_ara(Liste I, int deger){
  Elemanptr tmp;
  tmp = I->bas;
  while (tmp){
      if (tmp->icerik == deger)
      return tmp;
      tmp = tmp->ileri;
      }
  return NULL;
```

```
Eleman listeAra(int deger){
  Eleman tmp;
  tmp = bas;
  while (tmp! = null){
      if (tmp.icerik == deger)
      return tmp;
      tmp = tmp.ileri;
      }
  return NULL;
```



Maliyet O(N)

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Listenin ilk elemanını silme)

```
void liste_basi_sil(Listeptr I){
  I->bas = I->bas->ileri;
  if (I->bas == NULL)
      I->son = NULL;
 }
```

```
void listeBasiSil(){
  bas = bas.ileri;
  if (bas == null)
      son = null;
 }
```



Maliyet O(1)

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Listenin son elemanını silme)

```
void liste_sonu_sil(Listeptr I){
Elemanptr tmp.once;
tmp = I->bas;
once = NULL;
while (tmp != I->son){
        once = tmp;
        tmp = tmp->ileri;
  }
  if (once == NULL)
        I->bas = NULL;
  else
        once ->ileri = NULL;
  I->son = once;
}
```

```
void listeSonuSil(){
Eleman tmp.once;
tmp = bas;
once = NULL;
while (tmp != son){
        once = tmp;
        tmp = tmp.ileri;
  }
  if (once == null)
        bas = null;
  else
        once.ileri = null;
  son = once;
}
```



**Maliyet O(N)**

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Listenin ortasındaki elemanı silme)

```
void listeden_sil(Listeptr I, Elemanptr son){
Elemanptr tmp, elemanonce;
tmp = I->bas;
elemanonce = NULL;
while (tmp != I->son){
        elemanonce = tmp;
        tmp = tmp->ileri;
  }
  Elemanonce->ileri =son -> ileri;
}
```

```
void listedenSil(Eleman son){
Eleman tmp, elemanonce;
tmp = bas;
elemanonce = null;
while (tmp != son){
        elemanonce = tmp;
        tmp = tmp.ileri;
  }
  Elemanonce.ileri =son.ileri;
}
```



**Maliyet O(N)**

# Ayrık Alanlarda Bağlantılı Liste Uygulaması (Listedeki eleman sayısını bulma)

```
int eleman_sayisi(Listeptr I){
int sayac = 0;
Elemanptr tmp;
tmp = I->bas;
while (tmp != I->son){
      tmp = tmp->ileri;
      sayac++;
     }
 return sayac;
}
```

```
int elemanSayisi(){
int sayac = 0;
Eleman tmp;
tmp = bas;
while (tmp != null){
      tmp = tmp.ileri;
      sayac++;
     }
 return sayac;
}
```

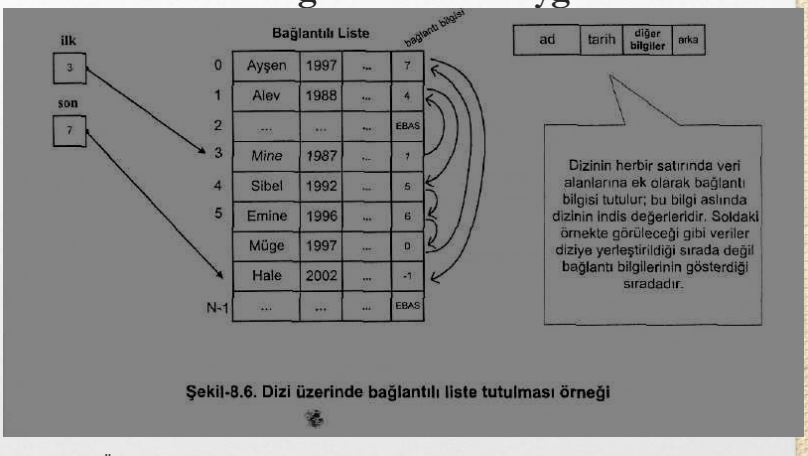
Maliyet O(N)



# Dizi Üzerinde Bağlantılı Liste Uygulaması

Dizi üzerinde bağlantılı liste tutulması, listenin herbir düğümünün bir dizi satırında tutulmasına dayanır. Ancak, üste elemanları dizilerde olduğu gibi art arda gelmezler; bir elemandan sonra hangisinin geldiği bir bağlantı bilgisi üzerinden anlaşılır.

## Dizi Üzerinde Bağlantılı Liste Uygulaması




**Şekil-8.6. Dizi üzerinde bağlantılı liste tutulması örneği**



## Dizi Üzerinde Bağlantılı Liste Uygulaması

Eleman ekleme algoritması kaba-kodu

```
k←boş dizi gözü bul
if (boş dizi gözü yok ise)
    dizide yer yok mesajı ver;
if (halihazırda düğüm varsa)
    var olanların sonuna ekle;
    (son=k)
else
    listenin ilk elemanı olarak ekle;
    (ilk=k, son=k)
```

# Dizi Üzerinde Bağlantılı Liste Uygulaması

**Fonksiyon-8.6. Dizi üzerinde bağlantılı listeye düğüm ekleme**

```
int  ekle(char  isim[],  unsigned  tarih,  char mesaj[])
{
int  k;
for(k=0;  D[k].arka  !=EBAS  && k<N;  k++),-  /* burası boş döngü */
If(k==N)
return  -1;
/* verinin ilgili alanlara yerleştirilmesi */
strcpy (D [k] . isini,   isim);
D[k]. tarih=tarih;
strcpy(D[k].mesaj,  mesaj);
 D[k].arka=-1;
if (ilk! =-1)  {   /* üstede halihazırda düğüm var ise */
D[son].arka=k;
son=k;
 }
else  {   /* ilk eklenen düğüm ise burası yürütülür */
ilk=k;
son=k; }
return  0;
}
```



# Dizi Üzerinde Bağlantılı Liste Uygulaması

**Fonksiyon-8.7. Dizi üzerinde bağlantılı listede dolaşma/listeleme**

```
int listele()
{
        unsigned int g;

        if(ilk==-1)      /* liste boş mu? */
                return -1;      /* liste boş */
        g=ilk;
        while(g=! -1) {   /* liste sonuna gelene kadar dön */
                yazEkrana(g); /* verileri ekrana yazan fonksiyon */
                g=D[g].arka;
        }
        return 0;      /* listeleme tamam */
}
```

# Dizi Üzerinde Bağlantılı Liste Uygulaması

Fonksiyon-8.8. Dizi üzerinde bağlantılı listede arama

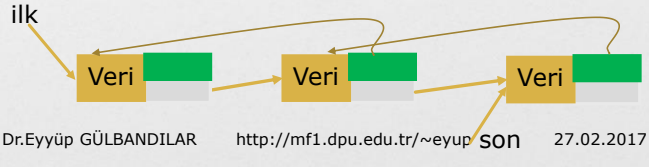
```
int ara(char isim[])
{
        int g;

        g=ilk;
        while(g=! -1) {
                if(!strcmp(D[g].isim, isim))
                        return g;     /* aranan bulundu */
                g=D[g].arka;
        }
        return -1;     /* aranan yok */
}
```



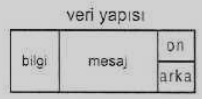
## İki Yönlü Bağlantılı Liste Uygulaması (double linked list)

İki-yönlü bağlantılı listede (double linked list) Şekil'de görüldüğü gibi iki tane bağlantı bilgisi vardır; birisi o düğümün arkasındaki diğeri önündeki düğümü işaret eder. Dolayısıyla liste üzerinde iki yönlü hareket edilebilir; bu, araya düğüm ekleme ve aradan düğüm silme gibi işlemlerin daha kolay yapılmasını sağlar.



## İki Yönlü Bağlantılı Liste Uygulaması (double linked list)

```
typedef struct topluluk {
    int bilgi;
    char mesaj[100];
    struct topluluk *on;
    struct topluluk *arka;
}BLISTE2;
```






## İki Yönlü Bağlantılı Liste Uygulaması (tanımlama)

```
struct cifteleman{
    int icerik;
    struct cifteleman* ileri,
    struct cifteleman* geri,
};
typedef struct cifteleman Cifteleman;
typedef Cifteleman* Ciftelemanptr;
  Ciftelemanptr yeni_cifteleman(int icerik){
  Ciftelemanptr eleman;
  eleman = malloc(sizeof(Cifteleman));
  eleman->icerik = icerik;
  eleman->ileri = NULL;
  eleman->geri = NULL;
  return eleman;
}
```

```
public class CiftEleman{
    int icerik;
    CiftEleman ileri,
    CiftEleman geri,
  public CiftEleman(int icerik){
  this.icerik = icerik;
  ileri = NULL;
  geri = NULL;
  }
}
```

**Maliyet O(1)**

## İki Yönlü Bağlantılı Liste Uygulaması (liste başına eleman ekleme)

```
void liste_basina_ekle(Ciftlisteptr I,
Ciftelemanptr yeni){
  if (I->son==NULL)
     I->son=yeni;
  else
     I->bas->geri=yeni;
  yeni->ileri=I->bas;
  I->bas=yeni;
}
```

```
void listeBasinaEkle( CiftEleman yeni){
  if (son==NULL)
     son=yeni;
  else
     bas.geri=yeni;
  yeni.ileri=bas;
  bas=yeni;
}
```

**Maliyet O(1)**



## İki Yönlü Bağlantılı Liste Uygulaması (liste sonuna eleman ekleme)

```
void listeye_ekle(Ciftlisteptr I, Ciftelemanptr
yeni){
  if (I->bas==NULL)
     I->bas=yeni;
  else
     I->son->ileri=yeni;
  yeni->ileri=I->son;
  I->son=yeni;
}
```

```
void listeyeEkle( CiftelEman yeni){
  if (bas==NULL)
     bas=yeni;
  else
     son.ileri=yeni;
  yeni.geri=son;
  son=yeni;
}
```

**Maliyet O(1)**

## İki Yönlü Bağlantılı Liste Uygulaması
## (liste ortasına eleman ekleme)

```
void liste_orta_ekle(Ciftelemanptr yeni,
Ciftelemanptr once){
      yeni->ileri = once->ileri;
      yeni->geri = once;
      once->ileri = geri->yeni;
   once->ileri = yeni;
}
```

```
void listeOrtaEkle(CiftEleman yeni,
CiftEleman once){
      yeni.ileri = once.ileri;
      yeni.geri = once;
      once.ileri = geri.yeni;
   once.ileri = yeni;
}
```

**Maliyet O(1)**



## İki Yönlü Bağlantılı Liste Uygulaması
## (liste ilk elemanını silme)

```
void liste_basi_sil(Ciftlisteptr I){
      I->bas = I->bas->ileri;
      if (I->bas==NULL)
         I->son = NULL;
      else
         I->bas->geri = NULL;
}
```

```
void listeBasiSil(){
      bas=bas.ileri;
      if (bas = null)
         son = null;
      else
         bas.geri = null;
}
```

**Maliyet O(1)**

## İki Yönlü Bağlantılı Liste Uygulaması (liste son elemanını silme)

```
void liste_sonu_sil(Ciftlisteptr I){
    I->son = I->son->geri;
    if (I->son==NULL)
        I->bas = NULL;
    else
        I->son->ileri = NULL;
}
```

```
void listeSonuSil(){
    son=son.geri;
    if (son = null)
        bas = null;
    else
        son.ileri = null;
}
```

**Maliyet O(1)**



## İki Yönlü Bağlantılı Liste Uygulaması (listenin ortasındaki elemanı silme)

```
void listeden_sil(Ciftelemanptr s){
    s->ileri->geri->s->geri;
    s->geri->ileri = s->ileri;
}
```

```
void listedenSil(Cifteleman s){
    s.ileri.geri=s.geri;
    s.geri.ileri = s.ileri;
}
```

**Maliyet O(1)**

## Dairesel Bağlı Liste Uygulamaları (Eleman ekleme)

```
struct liste_ekle {
(Ciftelemanptr bas)
}
typedef struct daireliste Daireliste;
typedef Daireliste*Dairelisteptr;
Dairelisteptr yeni_Daireliste(){
    Dairelisteptr liste;
    liste=malloc(sizeof(Daireliste));
    liste->bas=NULL
}
```

```
public class DaireListeden{
    cifteleman bas;
    public Dairesel.Liste(){
    bas=null;
    }
}
```

**Maliyet O(1)**



## Dairesel Bağlı Liste Uygulamaları (eleman ekleme)

```
struct listeEkle (Dairelisteptr I,
Ciftelemanptr yeni) {
if (I->bas==NULL){
    yeni->ileri=yeni;
    yeni->geri=yeni;
    }else{
    yeni->ileri=I->bas;
    yeni->geri=I->bas->geri;
    I->bas->geri->ileri=yeni;
    I->bas->geri=yeni;
    }
    I->bas=yeni;
}
```

```
struct liste.ekle (CiftEleman yeni) {
if (bas==NULL){
    yeni.ileri=yeni;
    yeni.geri=yeni;
    }else{
    yeni.ileri=bas;
    yeni.geri = bas->geri;
    bas.geri.ileri=yeni;
    bas.geri=yeni;
    }
    bas = yeni;
}
```

**Maliyet O(1)**

## Dairesel Bağlı Liste Uygulamaları (ilk eleman silme)

```
void liste.sil(Dairelisteptr I){
if (I->bas->ileri=I->bas){
      I->bas=NULL;
      }else{
      I->bas->geri->ileri=I->bas->ileri;
      I->bas->ileri->geri=I->bas->geri;
      I->bas = I->bas->=ileri;
      }
}
```

```
void liste.sil(){
if (bas.ileri=bas)
      bas=NULL;
      }
      else{
      bas.geri.ileri=bas.ileri = bas.geri;
      bas==bas.ileri;
      }
}
```



**Maliyet O(1)**