



# YMT219

## Veri Yapıları

Yrd.Doç.Dr. Erkan TANYILDIZI

# Listeler

Yrd. Doç. Dr. Aybars UĞUR,  
Yrd.Doç.Dr. M. Ali Akcayol ders  
notları ve yabancı kaynak  
derlemelerinden hazırlanmıştır.

# LİSTELER

- Günlük hayatta listeler; alışveriş listeleri, davetiye, telefon listeleri vs. kullanılır.
- Programlama açısından liste; aralarında doğrusal ilişki olan veriler topluluğu olarak görülebilir.
- Veri yapılarında değişik biçimlerde listeler kullanılmakta ve üzerlerinde değişik işlemler yapılmaktadır.

# Doğrusal Listeler (Diziler)

- Diziler(arrays), doğrusal listeleri oluşturan yapılardır. Bu yapıların özellikleri şöyle sıralanabilir:
- Doğrusal listelerde süreklilik vardır. Dizi veri yapısını ele alırsak bu veri yapısında elemanlar aynı türden olup bellekte art arda saklanırlar.
- Dizi elemanları arasında başka elemanlar bulunamaz. Diziye eleman eklemek gerektiğinde (dizinin sonu hariç) dizi elemanlarının yer değiştirmesi gerekir.
- Dizi program başında tanımlanır ve ayrılacak bellek alanı belirtilir. Program çalışırken eleman sayısı arttırılamaz veya eksiltilemez.

## Doğrusal Listeler (Diziler)

- Dizinin boyutu baştan çok büyük tanımlandığında kullanılmayan alanlar oluşabilir.
- Diziye eleman ekleme veya çıkarmada o elemandan sonraki tüm elemanların yerleri değişir. Bu işlem zaman kaybına neden olur.
- Dizi sıralanmak istendiğinde de elemanlar yer değiştireceğinden karmaşıklık artabilir ve çalışma zamanı fazlalaşır.

# BAĞLI LİSTELER

- Bellekte elemanları ardışık olarak bulunmayan listelere **bağlı liste** denir.
- Bağlı listelerde her eleman kendinden sonraki elemanın nerede olduğu bilgisini tutar. İlk elemanın yeri ise yapı türünden bir göstericide tutulur. Böylece bağlı listenin tüm elemanlarına ulaşılabilir.
- Bağlı liste dizisinin her elemanı bir yapı nesnesidir. Bu yapı nesnesinin bazı üyeleri bağlı liste elemanlarının değerlerini veya taşıyacakları diğer bilgileri tutarken, bir üyesi ise kendinden sonraki bağlı liste elemanı olan yapı nesnesinin adres bilgisini tutar.

# BAĞLI LİSTELER

- Bağlantılı liste yapıları iki boyutlu dizi yapısına benzemektedir. Aynı zamanda bir boyutlu dizinin özelliklerini de taşımaktadır.
- Bu veri yapısında bir boyutlu dizilerde olduğu gibi silinen veri alanları hala listede yer almakta veri silindiği halde listenin boyu kısaltmamaktadır.
- Eleman eklemede de listenin boyutu yetmediğinde kapasiteyi arttırmak gerekmektedir. Bu durumda istenildiği zaman boyutun büyütülebilmesi ve eleman çıkarıldığında listenin boyutunun kendiliğinden küçülmesi için yeni bir veri yapısına ihtiyaç vardır.

# BAĞLI LİSTELER

- Doğrusal veri yapılarında dinamik bir yaklaşım yoktur. İstenildiğinde bellek alanı alınamaz ya da eldeki bellek alanları iade edilemez. Bağlantılı listeler dinamik veri yapıları olup yukarıdaki işlemlerin yapılmasına olanak verir. Bağlantılı listelerde düğüm ismi verilen bellek büyüklükleri kullanılır.
- Bağlantılı listeler çeşitli tiplerde kullanılmaktadır;
  - Tek yönlü doğrusal bağlı liste
  - İki yönlü doğrusal bağlı liste
  - Tek yönlü dairesel bağlı liste
  - İki yönlü dairesel bağlı liste



# BAĞLI LİSTELER

- Örnek: C dilinde bağlı liste yapısı

```
struct ListNode  
{ int data;  
  struct ListNode *next;  
}
```

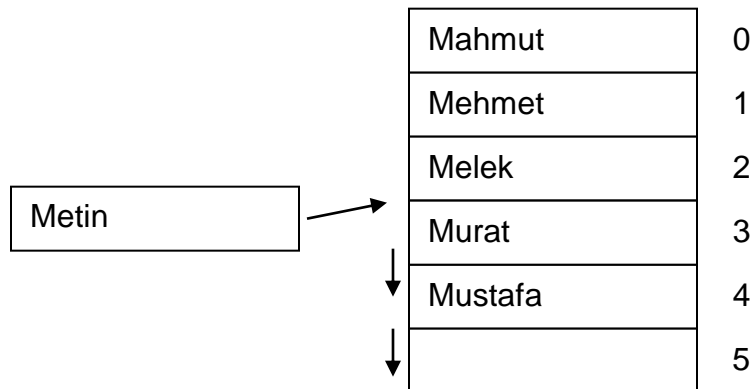
- Bağlı listenin ilk elemanının adresi **global** bir göstericide tutulabilir. Son elemanına ilişkin gösterici ise **NULL** adresi olarak bırakılır. Bağlı liste elemanları **malloc** gibi dinamik bellek fonksiyonları ile oluşturulur.

- Örnek: Java dilinde bağlı liste yapısı

```
public class ListNode  
{  
  int data;  
  public ListNode next;  
}
```

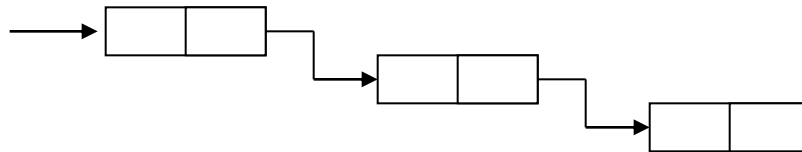
# BAĞLI LİSTELER

- **Bağlı Listelerle Dizilerin Karşılaştırılması:**
- Diziler;
  - Boyut değiştirme zordur
  - Yeni bir eleman ekleme zordur
  - Bir elemanı silme zordur
  - Dizinin tüm elemanları için hafızada yer ayrılır
- Bağlı listeler ile bu problemler çözülebilir.



# BAĞLI LİSTELER

- Ayrıca,
  - Her dizi elamanı için ayrı hafıza alanı ayrılır.
  - Bilgi kavramsal olarak sıralıdır ancak hafızada bulunduğu yer sıralı değildir.
  - Her bir eleman (node) bir sonrakini gösterir.



# BAĞLI LİSTELER

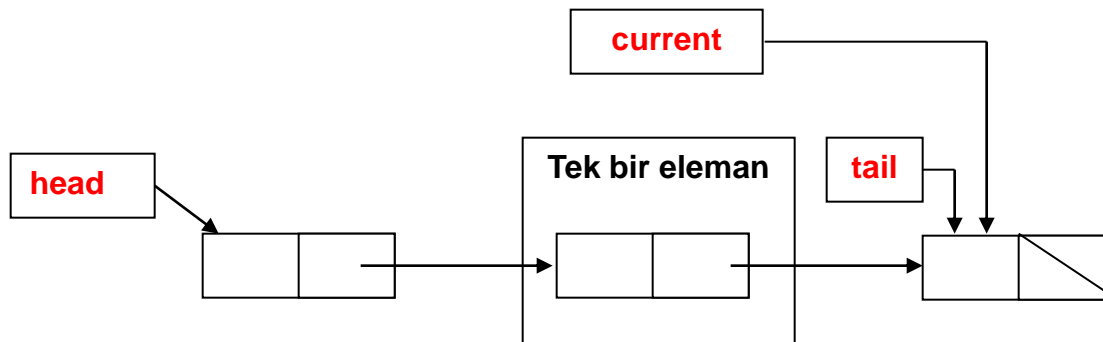
- Listeler;
  - Listedeki her bir eleman **data** (veri) ve **link** (bağlantı) kısmından oluşur. Data kısmı içerisinde saklanan bilgiyi ifade eder. Link kısmı ise kendisinden sonraki elamanı işaret eder.

```
public class ListNode  
{  
    int data;  
    public ListNode sonraki;  
}
```



# BAĞLI LİSTELER

- Listede bir başlangıç (**head**) elemanı, birde sonuncu (**tail**) elemanı vardır.
- Listede aktif (**current**) eleman şu anda bilgilerine ulaşabileceğimiz elemandır.

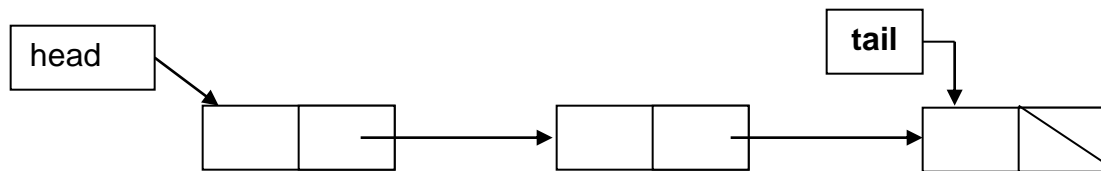


# Bağlı Liste İşlemleri

- Listeler ile yapılan işlemler,
  - Listeye eleman ekleme
    - Başa
    - Sona
    - Sıralı
  - Listeden eleman silme
    - Baştan
    - Sondan
    - Tümünü
    - Belirlenen bilgiye sahip elemanı
    - İstenen sıradaki elemanı
  - Arama
  - Listeleme
    - İstenen bilgiye göre
  - Kontrol
    - Boş liste
    - Liste boyutu

# Tek Yönlü Bağlı Listeler

- Listedeki elemanlar arasında sadece tek bir bağ vardır. Bu tür bağlı listelerde hareket yönü sadece listenin başından sonuna doğrudur.

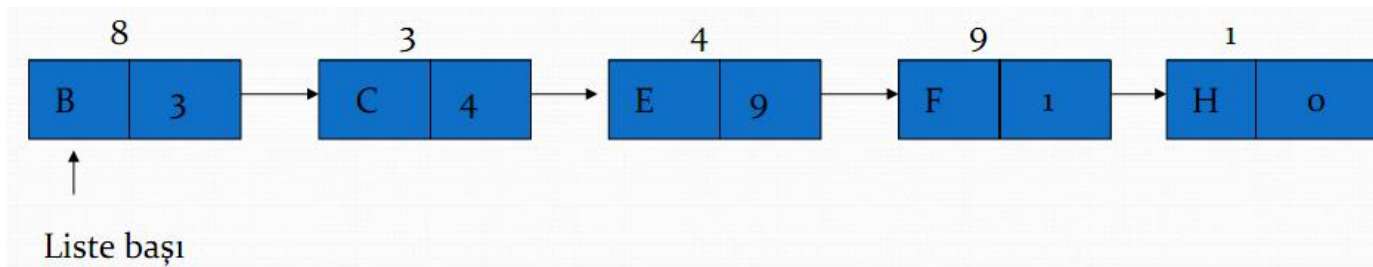


# Tek Yönlü Bağlı Liste

Adres	Veri alanı	bağ
1	H	0
2		
3	C	4
4	E	8
5		
6		
7	B	3
8	F	1

Liste başı →

- Bağlı bir listeye eleman eklemek için önce liste tanımlanır.
- Bunun için listede tutulacak verinin türü ve listenin eleman sayısı verilir. Aşağıda verilen listeyi ele alarak bu listeye 'G' harfini eklemek isteyelim.
- Önce listede bu veri için boş bir alan bulunur ve bağ alanlarında güncelleme yapılır. 'G' verisini 6.sıraya yazarsak 'F' nin bağı 'G' yi, 'G' nin bağı da 'H' yi gösterecek şekilde bağ alanları değiştirilir.



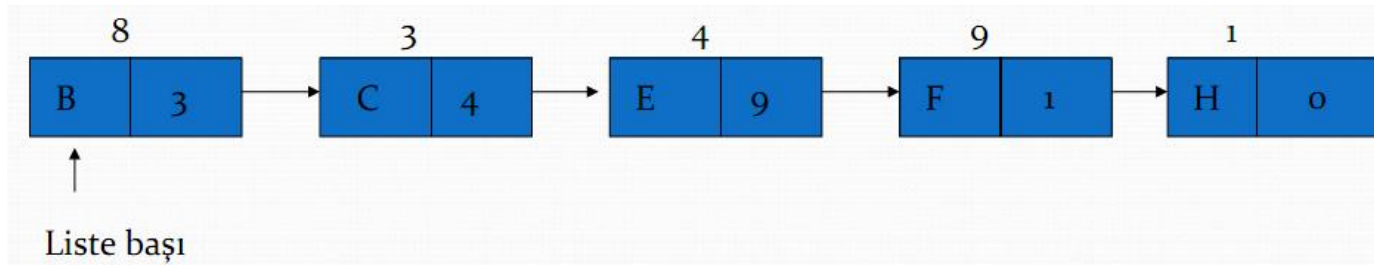


# Tek Yönlü Bağlı Liste

Liste başı

Adres	Veri alanı	bağ
1	H	o
2		
3	C	4
4	E	8
5		
6		
7	B	3
8	F	1

- Boşlar listesinden boş bir düğüm alınarak düğümün adresi bir değişkene atanır.



$X=5$

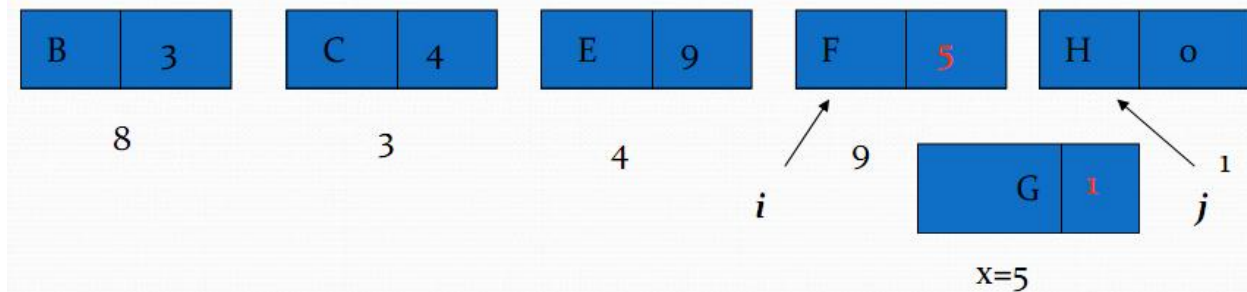
# Tek Yönlü Bağlı Liste

- Yeni düğümün veri alanına saklanması gereken 'G' değeri yazılır.



x=5

- Yeni düğümün listedeki yeri bulunur. Bunun için düğümün arasına gireceği düğümlerin adresi belirlenerek bunların adresleri **i, j** değişkenlerine atanır.



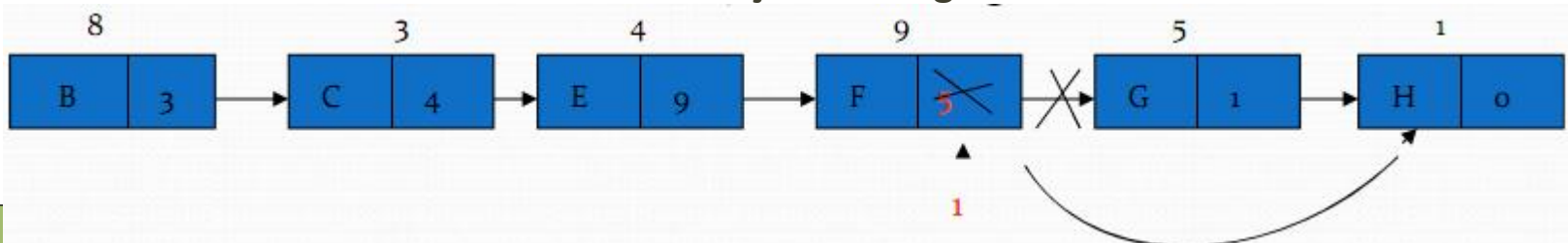
- x** adresli düğümün bağ alanına **j** düğümünün adresi yazılır. Bu şekilde yeni düğüm daha önce **i** düğümü tarafından işaretlenen **j** düğümünü gösterir. **i**, düğümünün bağ alanına da **x** düğümünün adresi yazılır ve yeni düğüm listeye eklenmiş olur.

## Tek Yönlü Bağlı Listeler: Düğüm Ekleme –Kaba Kod

- **Algorithm insert (newElement)**
- **// bağlantılı listeye eklenecek yeni düğümü yarat**
- **newNode = allocateNewNode (newElement)**
- **// listenin başına yeni düğümü ekle**
- **// yeni düğüm adresi, listeye yeni eklenme noktası olsun**
- **newNode** ← nextElement header
- **header** ← address (newNode)

# Tek Yönlü Bağlı Liste

- Bağlı bir listeden eleman silme; önce listeden çıkarılmak istenen eleman bulunur. Eğer silinecek eleman listede yoksa bu durum bir mesaj ile bildirilir. Eleman bulunduğunda bağ alanlarında güncelleme yapılarak eleman listeden silinir.
- Örneğimizdeki oluşturulan listeden 'G' düğümünü çıkarmak isteyelim. Bunun için yapılacak işlem, 'G' elemanından önce gelen 'F' elemanının bağ alanına çıkarılacak olan 'G' elemanının bağ alanındaki adresi yazmaktır.
- Bu durumda 'F' düğümü , 'H' düğümünü göstereceği için 'G' düğümü listeden çıkarılmış olur. 'G' elemanından sonra gelen düğümler değişmediği için herhangi bir kaydırma işlemine gerek kalmaz. Bu işlem dizi kullanılarak yapılsa idi, G den sonra gelen elemanların bir eleman sola çekilmesi gerekirdi.



## Düğüm Silme –Kaba Kod

- **Algorithm delete(element)**
- previousNode null
- nextNode header
- **while nextNode != null and nextNode.element != element**
- **do**
- previousNode nextNode // bir önceki düğümün referansını tut
- nextNode nextNode-->nextElement // bir sonraki düğüme ilerle
- **end while** // yukarıdaki döngü eleman bulunduğunda veya liste tamamen tarandığı halde eleman bulunamadığında sonlanır
- **If nextNode != null** // nextNode ile gösterilen düğüm silinecek düğümdür.
- previousNode-->nextElement = nextElement-->nextElement // Önceki düğüm sonraki ile yer değiştirir
- deallocate memory used by nextElement
- **else** // eğer bir düğüm bulunamadıysa bu bölüme gelinir
- // yapılacak bir silme işlemi yok

## Liste Boyutu –Kaba Kod

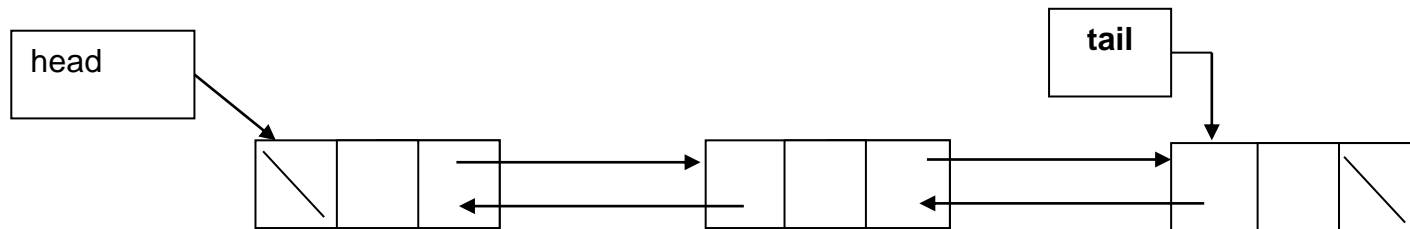
- **Algorithm traverseList()**
- **nextNode header**
- **while nextNode != null and nextNode.element != element**
- **do**
- **// Bir sonraki düğüm null/0 olana veya bir sonraki**
- **// bileşen bulunamayana kadar yapılacak işlemler**
- **someOperation(nextNode) // bir sonraki düğüme ilerle**
- **nextNode nextNode-->nextElement**
- **end while**

## Bağlantılı Listelerde Arama –Kaba Kod

- **Algorithm find(element)**
- **nextNode header**
- **while nextNode != null and nextNode.element != element**
- **do**
- **// sonraki elemana (düğüm) git**
- **nextNode nextNode-->nextElement**
- **end while**
- **// yukarıdaki döngü eleman bulunduğunda veya liste tamamen**
- **//tarandığı halde eleman bulunamadığında sonlanır**
- **If nextNode != null If nextNode != null**
- **return nextNodeelement**
- **else**
- **// bir sonraki bileşen (düğüm) bulunulamıyor ise ELSE bloğuna girilir.**
- **return null**

## İki Yönlü Bağlı Listeler:

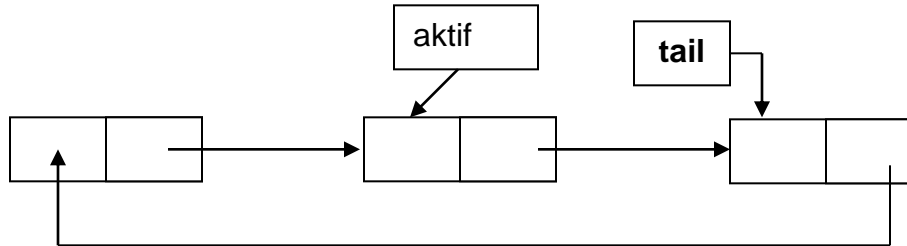
- Listedeki elemanlar arasında iki yönlü bağ vardır. Elemanın bağlantı bilgisi bölümünde iki gösterici bulunur. Bu göstericinin biri kendisinden sonra gelen elemanı diğeri ise kendisinden önce gelen elemanın adres bilgisini tutar.
- Bu sayede listenin hem başından sonuna hem de listenin sonundan başına doğru hareket edilebilir. Bu yöntem daha esnek bir yapıya sahip olduğundan bazı problemlerin çözümünde daha işlevsel olabilmektedir.



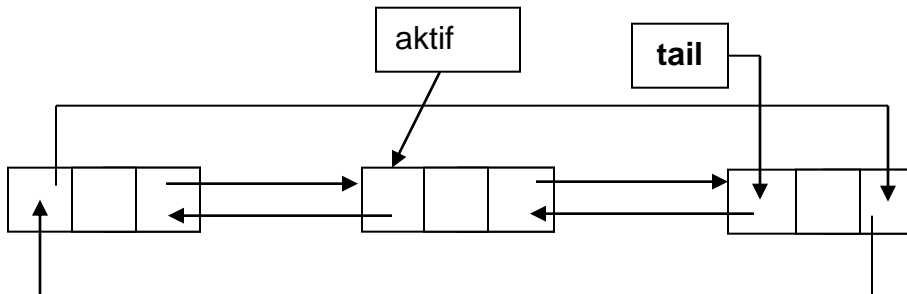


## Dairesel Bağlı Listeler:

- Tek Yönlü Dairesel Bağlı Listeler** : Listedeki elemanlar arasında tek yönlü bağ vardır. Tek yönlü bağlı listelerden tek farkı ise son elemanın göstericisi ilk listenin ilk elemanının adresini göstermesidir. Bu sayede eğer listedeki elemanlardan birinin adresini biliyorsak listedeki bütün elemanlara erişebiliriz.



- İki Yönlü Dairesel Bağlı Listeler** : Hem dairesellik hem de çift bağıllık özelliklerine sahip listelerdir. İlk düğümden önceki düğüm son, son düğümden sonraki düğüm de ilk düğümdür.



# Örnekler- C++

- **Örnek 1: Tek yönlü bağlı liste**

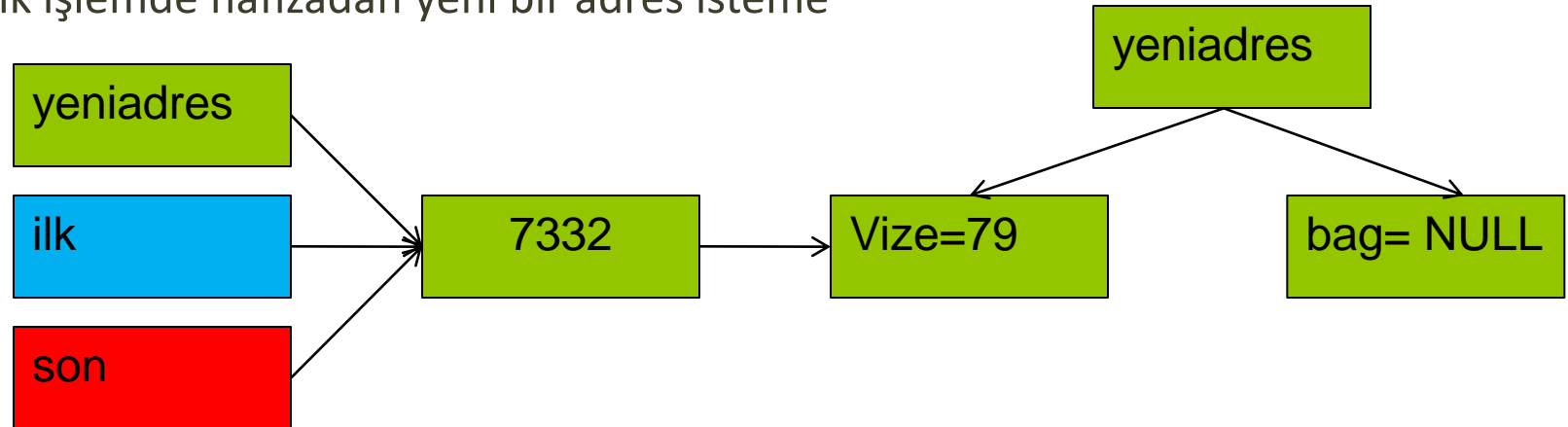
- `#include <stdio.h>`
- `#include <stdlib.h>`
- `main() {`
- `struct notlar {`
- `int vize1;`
- `struct notlar *bag;`
- `} *yeniadres, *ilk, *son;`
- `yeniadres=(struct notlar*) malloc(sizeof(struct notlar));`
- `printf("Yeniadres:%d\n",yeniadres);`
- `ilk=yeniadres;`
- `son=yeniadres;`
  
- `yeniadres->vize1=79;`
- `yeniadres->bag=NULL;`

## C++

- `printf("ilk:%d\n",ilk); //ilk elemanın adresini tutar`
- `printf("son:%d\n",son); //Son elemanın adresini tutar`
- `printf("T1___yeniadres->vize1:%d yeniadres->bag:%d\n",yeniadres->vize1, yeniadres->bag);`
- `//Hafızadan tekrar yer iste`
- `yeniadres=(struct notlar*) malloc(sizeof(struct notlar));`
- `printf("Yeniadres:%d\n",yeniadres);`
- `son->bag=yeniadres;`
- `yeniadres->vize1=95;`
- `yeniadres->bag=NULL;`
- `son=yeniadres;`
- `printf("ilk:%d\n",ilk);`
- `printf("son:%d\n",son);`
- `printf("T2___yeniadres->vize1:%d yeniadres->bag:%d\n", yeniadres->vize1, yeniadres->bag);}`

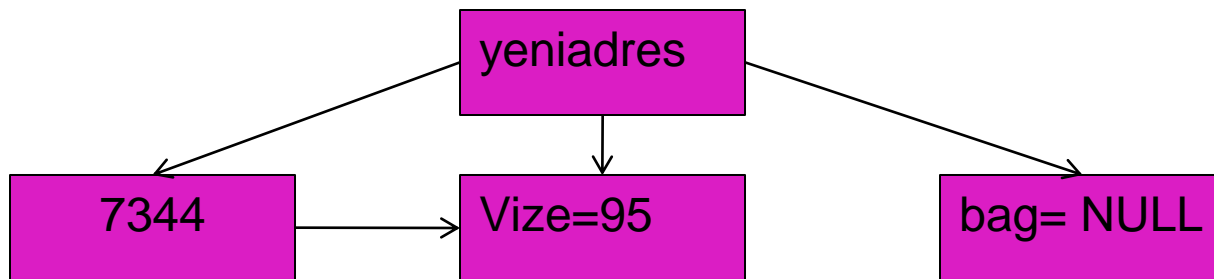
# BAĞLI LİSTELER

- İlk işlemde hafızadan yeni bir adres isteme

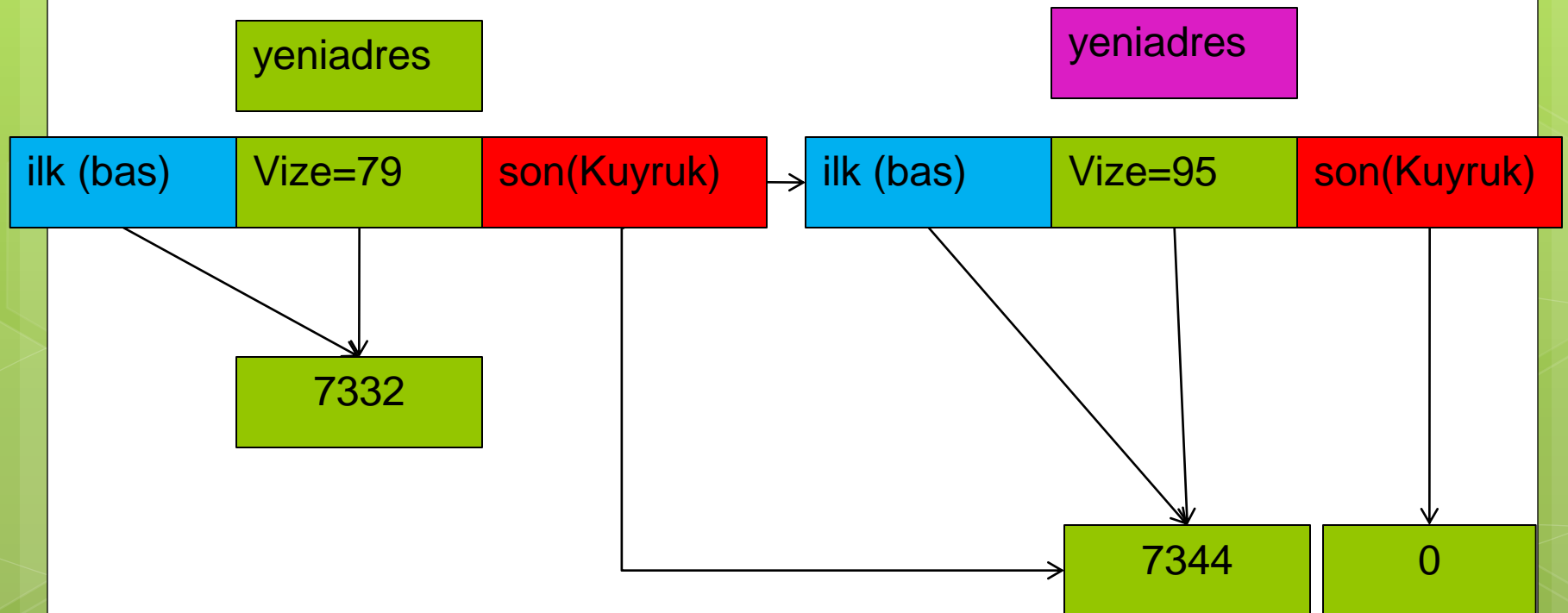


# BAĞLI LİSTELER

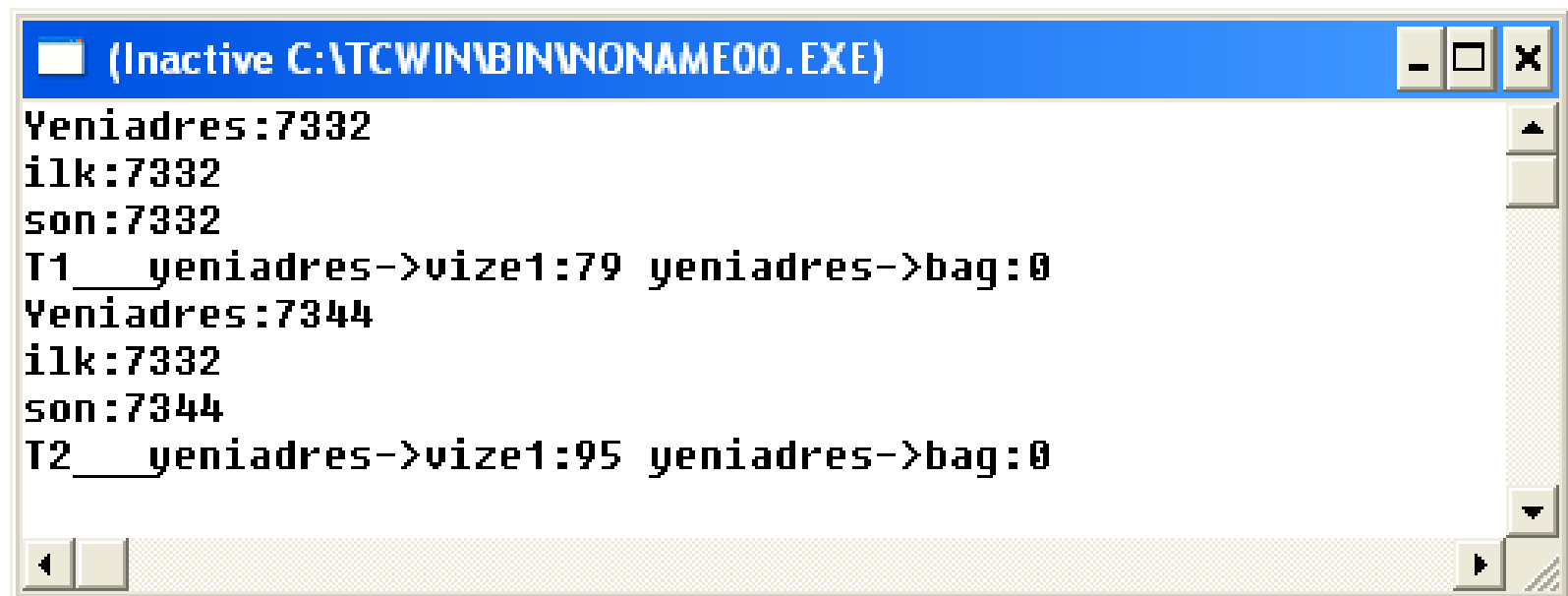
- İkinci işlemde hafızadan yeni bir adres isteme



# BAĞLI LİSTELER



# C++



```
(Inactive C:\TCWIN\BIN\WONAME00.EXE)
Yeniadres:7332
ilk:7332
son:7332
T1___yeniadres->vize1:79 yeniadres->bag:0
Yeniadres:7344
ilk:7332
son:7344
T2___yeniadres->vize1:95 yeniadres->bag:0
```

# Örnekler - C++

- **Örnek: 2- Çift yönlü bağlı liste ile film adlarının saklanması**

- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#include <string.h>`
- `#define TSIZE 45`
- `struct film {`
- `char baslik[TSIZE];`
- `int rating;`
- `struct film * sonraki; };`
- `int main(void) {`
- `struct film * ilk = NULL;`
- `struct film * onceki, * simdiki;`
- `char input[TSIZE];`
- `puts("Filimin basligini girin (sonlandirmek icin enter (boşluk) girin):");`



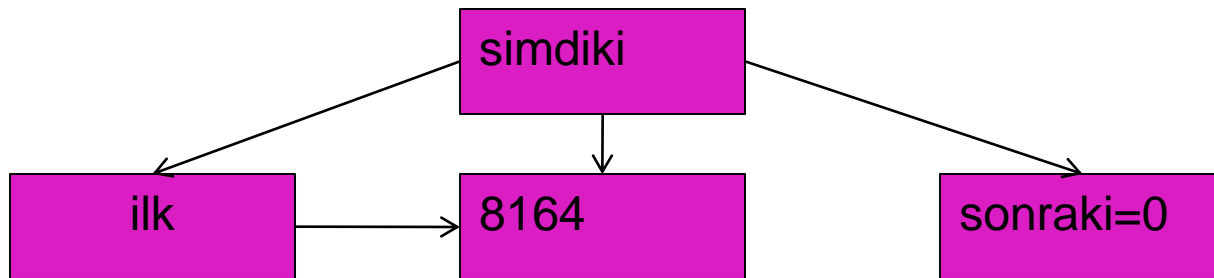
## C++

- while (gets(input) != NULL && input[0] != '\0') {
- simdiki = (struct filim \*) malloc(sizeof(struct filim));
- if (ilk == NULL)     ilk = simdiki;
- else     onceki->sonraki = simdiki;
- 
- simdiki->sonraki = NULL;
- strcpy(simdiki->baslik, input);
- puts("Ratingi girin <0-10>:");
- scanf("%d", &simdiki->rating);
- while(getchar() != '\n')     continue;
- puts("Filimin basligini girin (sonlandirmak icin bosluk girin):");
- onceki = simdiki;
- }

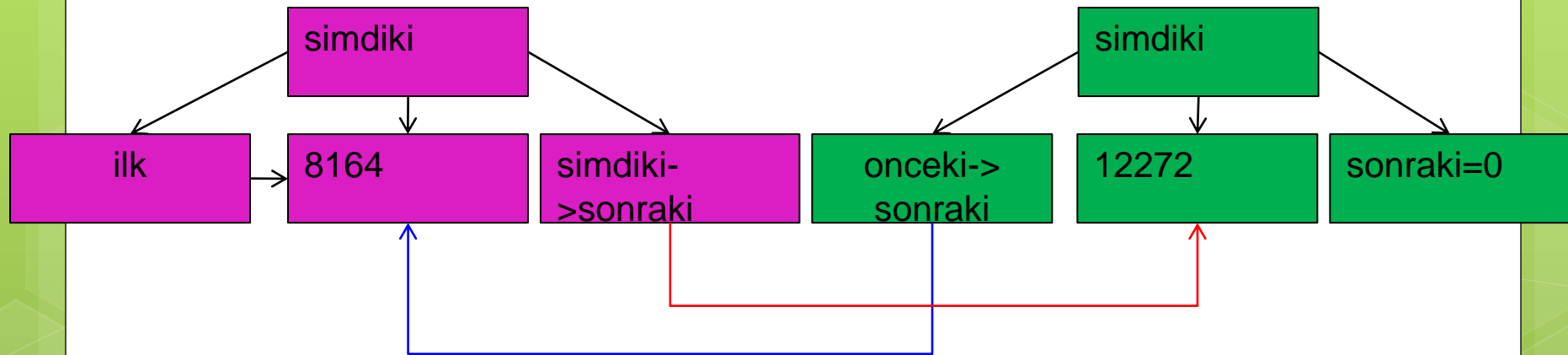
# C++

- if (ilk == NULL)
- printf("Veri girilmemistir. ");
- else
- printf ("Filim Listesi:\n");
- simdiki = ilk;
- while (simdiki != NULL)
- {
- printf("Filim: %s Rating: %d\n", simdiki->baslik, simdiki->rating);
- simdiki = simdiki->sonraki;
- }
- simdiki = ilk;
- }

# LİSTELER ve BAĞLI LİSTELER

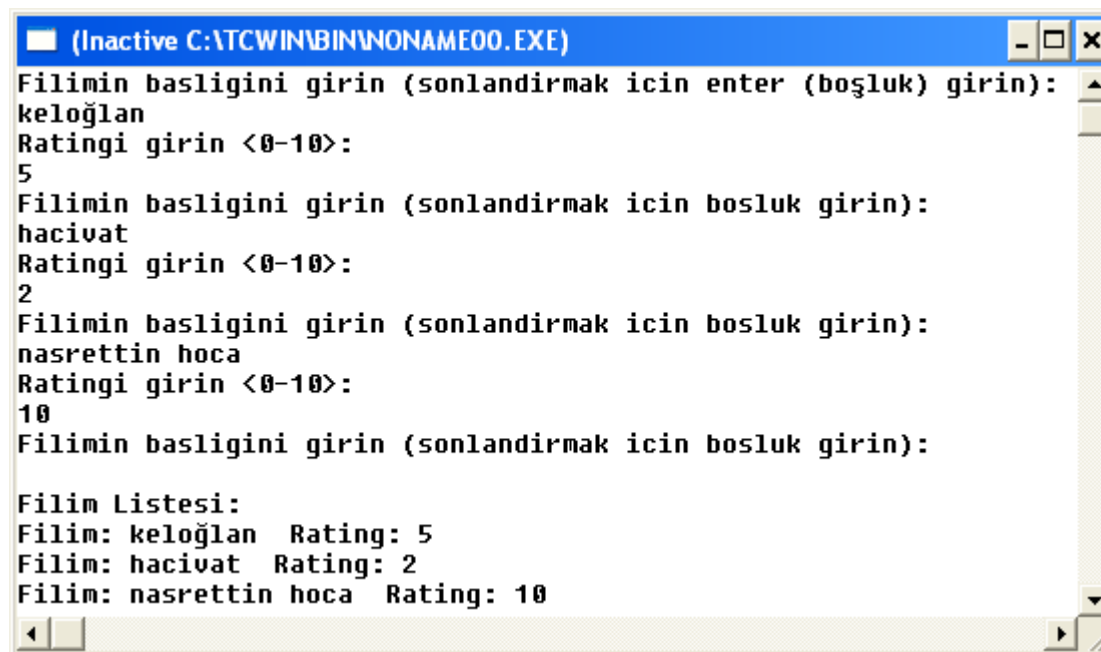


# LİSTELER ve BAĞLI LİSTELER



# LİSTELER ve BAĞLI LİSTELER

## Örnekler



```
(Inactive C:\TCWIN\BIN\WONAME00.EXE)
Filimin basligini girin (sonlandirmak icin enter (bosluk) girin):
keloglan
Ratingi girin <0-10>:
5
Filimin basligini girin (sonlandirmak icin bosluk girin):
hacivat
Ratingi girin <0-10>:
2
Filimin basligini girin (sonlandirmak icin bosluk girin):
nasrettin hoca
Ratingi girin <0-10>:
10
Filimin basligini girin (sonlandirmak icin bosluk girin):

Filim Listesi:
Filim: keloğlan   Rating: 5
Filim: hacivat   Rating: 2
Filim: nasrettin hoca   Rating: 10
```

## Java Programlama Dilinde Bağlı Liste Örneği: Bağlı Listenin Düğüm Yapısı

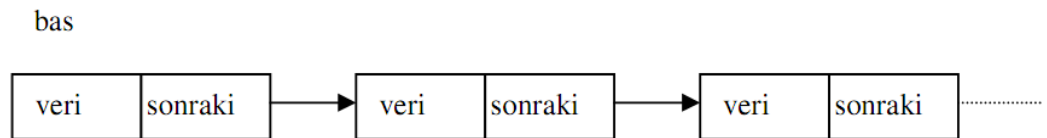
Düğüm

veri	sonraki
------	---------

- `class Dugum`
- `{`
- `public int veri; // Değişik tiplerde çoğaltılabilir`
- `public Dugum sonraki; // Sonraki düğümün adresi`
- `public Dugum (int gelenVeri) // Yapıcı metot`
- `{ veri = gelenVeri; } // Düğüm yaratılırken değerini aktarır`
- `public void yazdir() // Düğümün verisini yazdırır`
- `{ System.out.print(" "+veri); }`
- `}`
- `}`

# Java-Bağlı Liste ve Bağlı Listede Ekleme Yapısı

- `class BListe`
- `{`
- `private Dugum bas;`     *// Listenin ilk düğümünün adresini tutar*
- `public BListe()`     *// Bir BListe nesnesi yaratıldığında*
- `{`
- `bas = null;`     *// boş liste olarak açılır.*
- `}`
  
- `public void basaEkle(int yeniEleman)` *// Liste başına eleman ekler*
- `{`
- `Dugum yeniDugum = new Dugum(yeniEleman);`
- `yeniDugum.sonraki = bas;`
- `bas = yeniDugum;`
- `}`



# Java-Bağlı Listeye Arama Yapısı

- `public Dugum bul (int anahtar) {`
- `//Listede anahtar değerini bulur`
- `Dugum etkin = bas;`
- `while(etkin.veri != anahtar)`
- `{`
- `if(etkin.sonraki==null)`
- `return null;`
- `else`
- `etkin = etkin.sonraki;`
- `};`
- `return etkin; }`



# Java-Bağlı Listede Silme Yapısı

```
○ public Dugum sil(int anahtar)
○ {
○     // Verilen anahtar değerindeki düğümü siler
○     Dugum etkin = bas;
○     Dugum onceki = bas;
○     while(etkin.veri!=anahtar)
○     {
○         if(etkin.sonraki==null)      return null;
○         else      { onceki = etkin; etkin = etkin.sonraki; }
○     }
○     if(etkin==bas)      bas = bas.sonraki;
○     else      onceki.sonraki = etkin.sonraki;
○     return etkin;
○ }
```

## Java - Bağlı Listede Listeleme Yapısı

- `public void listele()`
- `{`
- `System.out.println();`
- `System.out.print("Bastan Sona Liste : ");`
- `Dugum etkin = bas;`
- `while(etkin!=null)`
- `{ etkin.yazdir(); etkin=etkin.sonraki; }`
- `}`
- `}`

# Java - Bağlı Liste Örneği

- // Bir bağlı liste oluşturarak, Bliste ve Dugum sınıflarını metotlarıyla birlikte test eden sınıf
  - `class BlisteTest {`
  - `public static void main(String args[]) {`
  - `Bliste liste = new Bliste(); // liste adlı bir bağlı liste nesnesi oluşturur.`
  - `liste.basaEkle(9);`
  - `for(int i=8; i>=1; --i) liste.basaEkle(i);`
  - `liste.listele(); int deger = 5; Dugum d = liste.bul(deger);`
  - `if(d==null) System.out.println("\n"+deger+" Listede Yok");`
  - `else System.out.println("\n"+deger+" Bulundu");`
  - `Dugum s = liste.sil(5);`
  - `liste.listele();`
  - `}`
  - `}`
- Ekran Çıktısı :
- // Bastan Sona Liste : 1 2 3 4 5 6 7 8 9
- // 5 Bulundu
- // Bastan Sona Liste : 1 2 3 4 6 7 8 9

# Java Programlama Dilinde Bağlı Liste Örneği ve Kullanımı

- Ödev:
- Verilen örnekte
- Ekleme,
  - Baştan ekleme
  - İstenilen sırada ekleme
  - Sondan ekleme yapıları
- Silme
  - Baştan silme
  - İstenilen sırada silme
  - Sondan silme
- yapıları sizin tarafınızdan bu örnek üzerinde oluşturulacaktır ve ders hocasına teslim edilecektir.

# Ödev

- 2-
- K adet ders için N adet öğrencinin numara ve harf ortalamalarını bağlı dizilerle gösteriniz.
- Her bir node öğrenci numarası, dersin kodu, dersin harf ortalaması bilgilerini bulunduracak.
- Her öğrencinin numarası headNode içindeki bilgi olacak.
- Her dersin kodu headNode içindeki bilgi olacak.
- Her bir node aynı derste bir sonraki öğrenciyi gösterecek.
- Her bir node aynı kişinin diğer dersini gösterecek.
- Program Java veya C# Windows application olarak hazırlanacak ve aşağıdaki işlemleri butonlarla yapacak.
- 1- Bir öğrenciye yeni bir ders ekleme
- 2- Bir derse yeni bir öğrenci ekleme
- 3- Bir öğrencinin bir dersini silme
- 4- Bir derste bir öğrenciyi silme
- 5- Bir derste tüm öğrencileri numara sırasına göre sıralı listeleme
- 6- Bir öğrencinin aldığı tüm dersleri ders koduna göre sıralı listeleme

# Ödev

- **3-** Java veya C# ile dairesel bağlı liste uygulamasını gerçekleştiriniz.
- Ekleme,
  - Baştan ekleme
  - İstenilen sırada ekleme
  - Sondan ekleme yapıları
- Silme
  - Baştan silme
  - İstenilen sırada silme
  - Sondan silme