

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;  
}
```

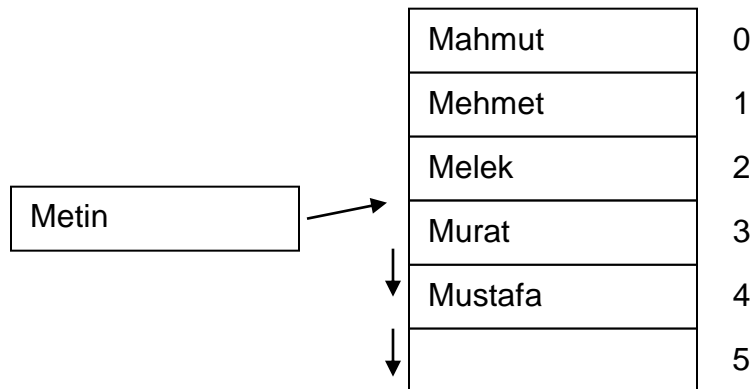
- Bağlı listenin ilk elemanının adresi **global** bir göstericide tutulabilir. Son elemanına ilişkin gösterici ise **NULL** adresi olarak bırakılır. Bağlı liste elemanları **malloc** gibi dinamik bellek fonksiyonları ile oluşturulur.

- Örnek: Java dilinde bağlı liste yapısı

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

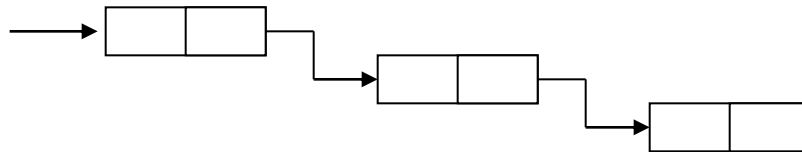

BAĞLI LİSTELER

- **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

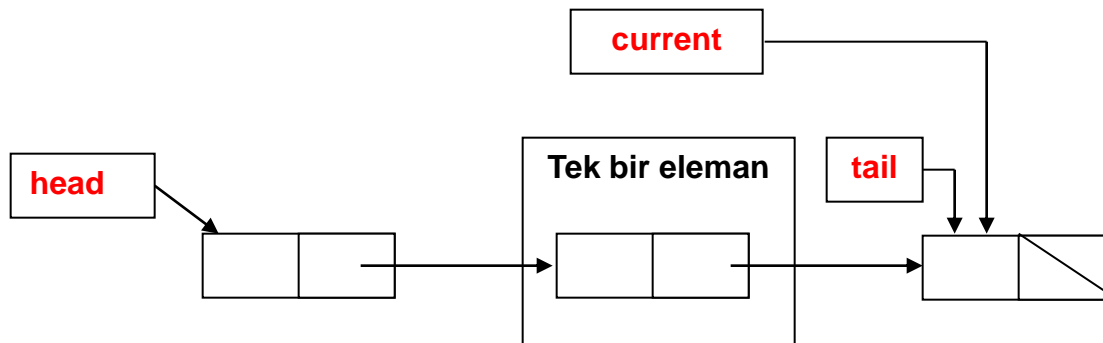
- Listeler;
 - Listedeki her bir eleman **data** (veri) ve **link** (bağlantı) kısmından oluşur. Data kısmı içerisinde saklanan bilgiyi ifade eder. Link kısmı ise kendisinden sonraki elemanı işaret eder.

```
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.

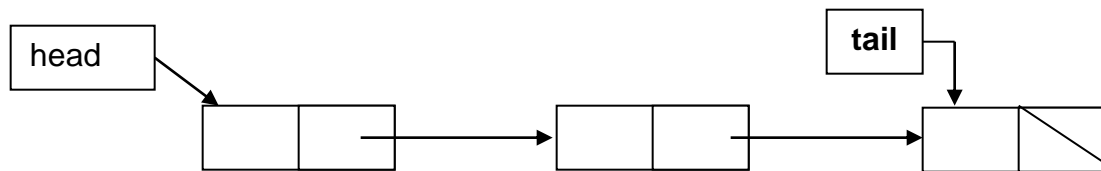


Bağlı Liste İşlemleri

- 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.

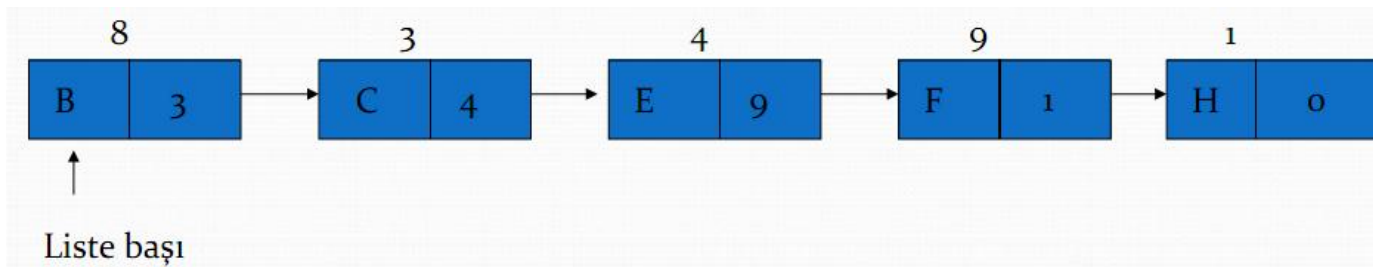


Tek Yönlü Bağlı Liste

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.

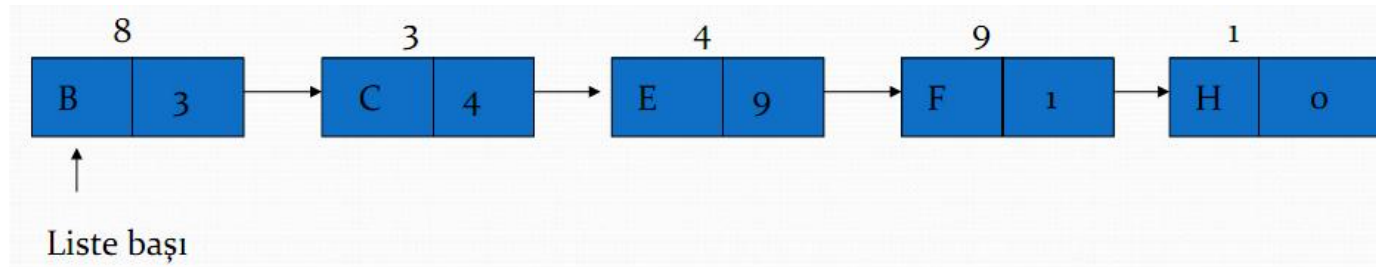


Tek Yönlü Bağlı Liste

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

Liste başı →

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



$X=5$

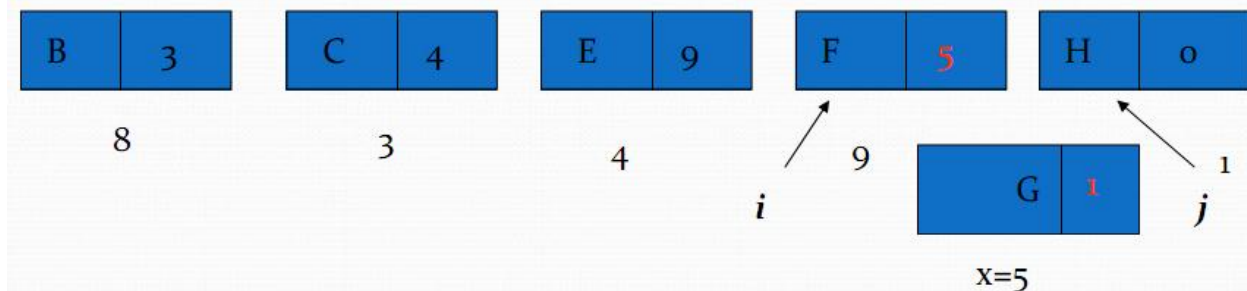
Tek Yönlü Bağlı Liste

- 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.



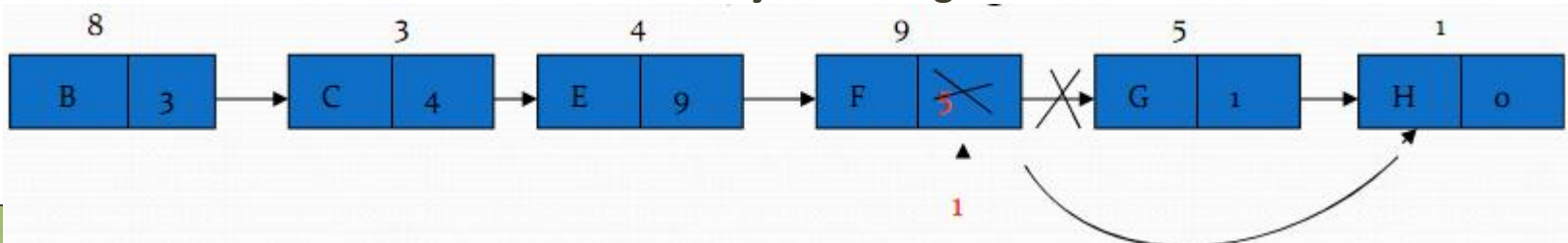
- x** adresli düğümün bağ alanına **j** düğümünün adresi yazılır. Bu şekilde yeni düğüm daha önce **i** düğümü tarafından işaretlenen **j** düğümünü gösterir. **i**, düğümünün bağ alanına da **x** düğümünün adresi yazılır ve yeni düğüm listeye eklenmiş olur.

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

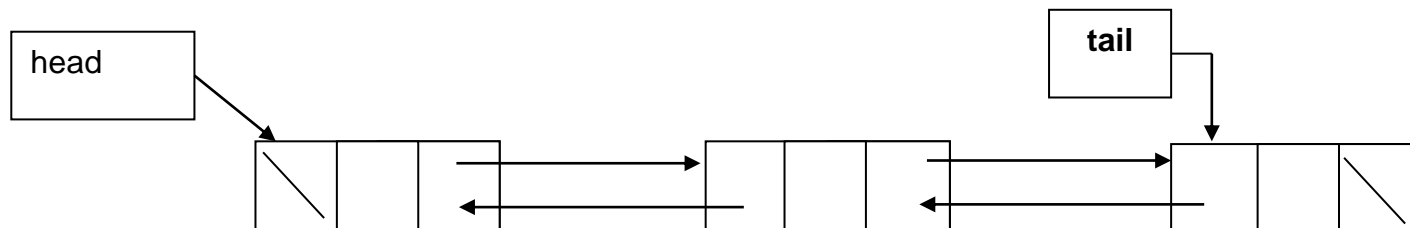
- **Algorithm traverseList()**
- **nextNode header**
- **while nextNode != null and nextNode.element != element**
- **do**
- **// Bir sonraki düğüm null/0 olana veya bir sonraki**
- **// bileşen bulunamayana kadar yapılacak işlemler**
- **someOperation(nextNode) // bir sonraki düğüme ilerle**
- **nextNode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

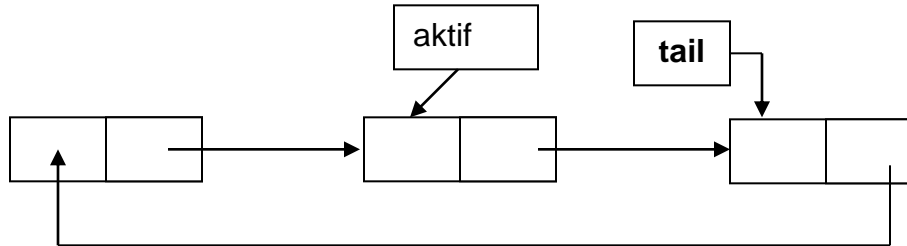
İki Yönlü Bağlı Listeler:

- 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.

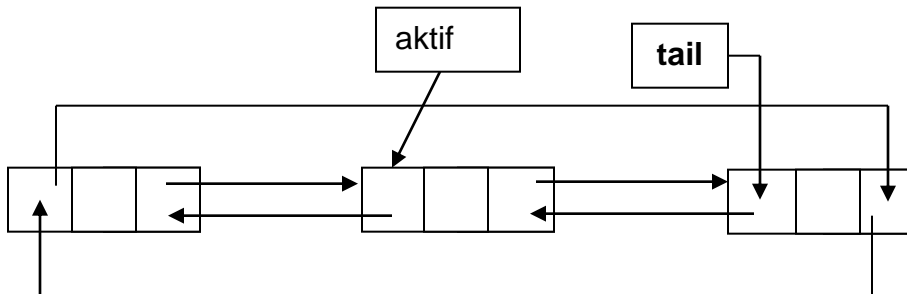


Dairesel Bağlı Listeler:

- 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 bağlilik 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;`

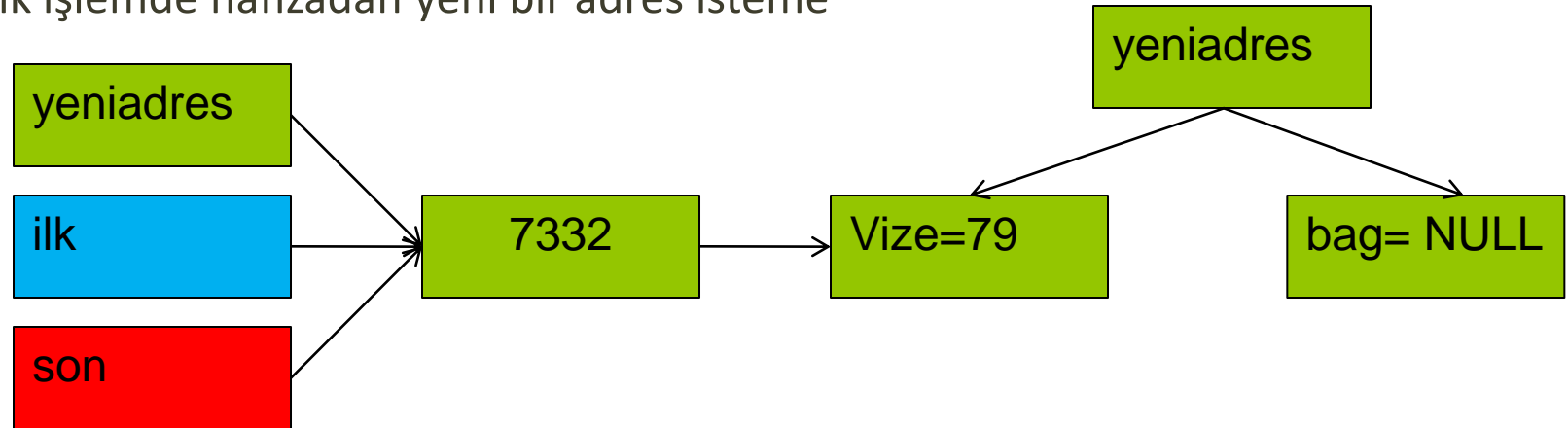
- `yeniadres->vize1=79;`
- `yeniadres->bag=NULL;`

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);}`

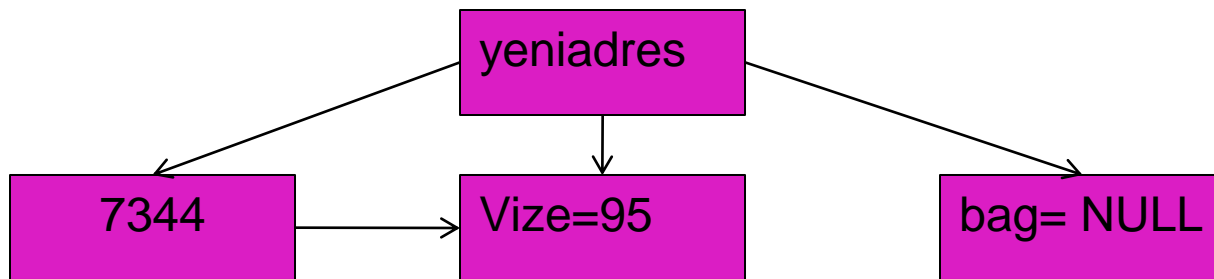
BAĞLI LİSTELER

- İlk işlemde hafızadan yeni bir adres isteme

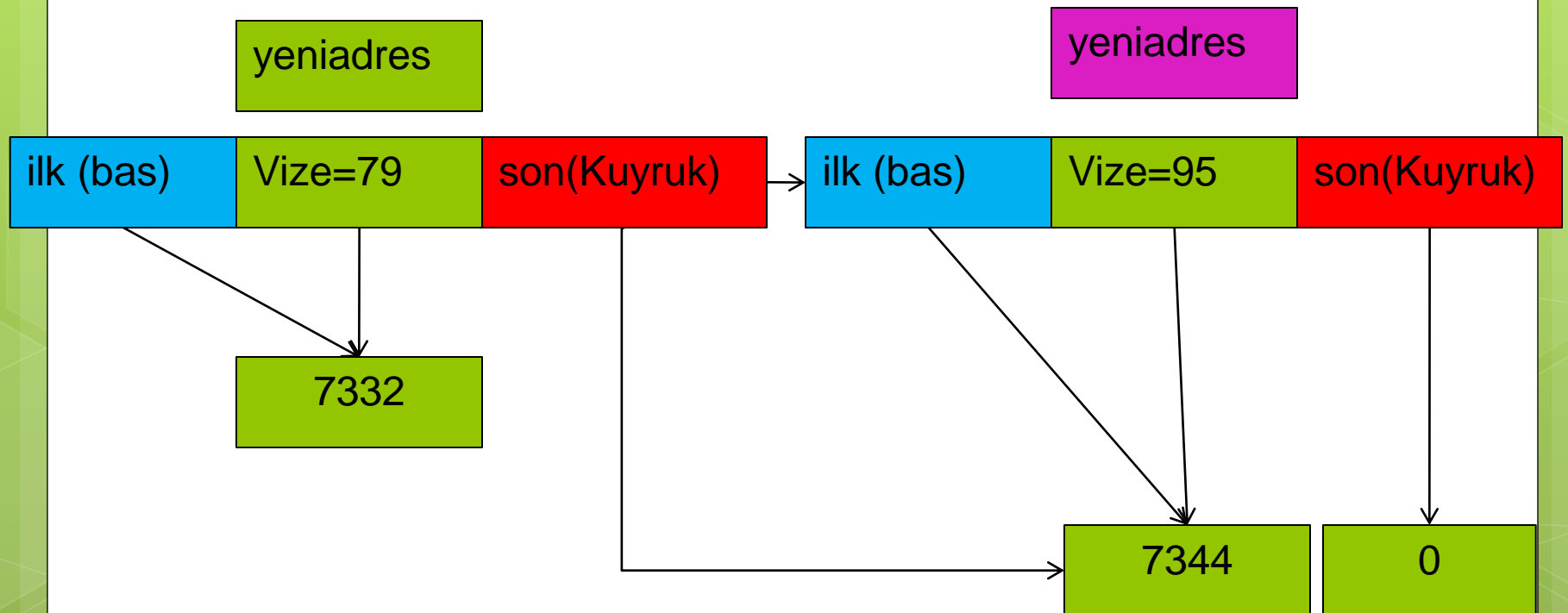


BAĞLI LİSTELER

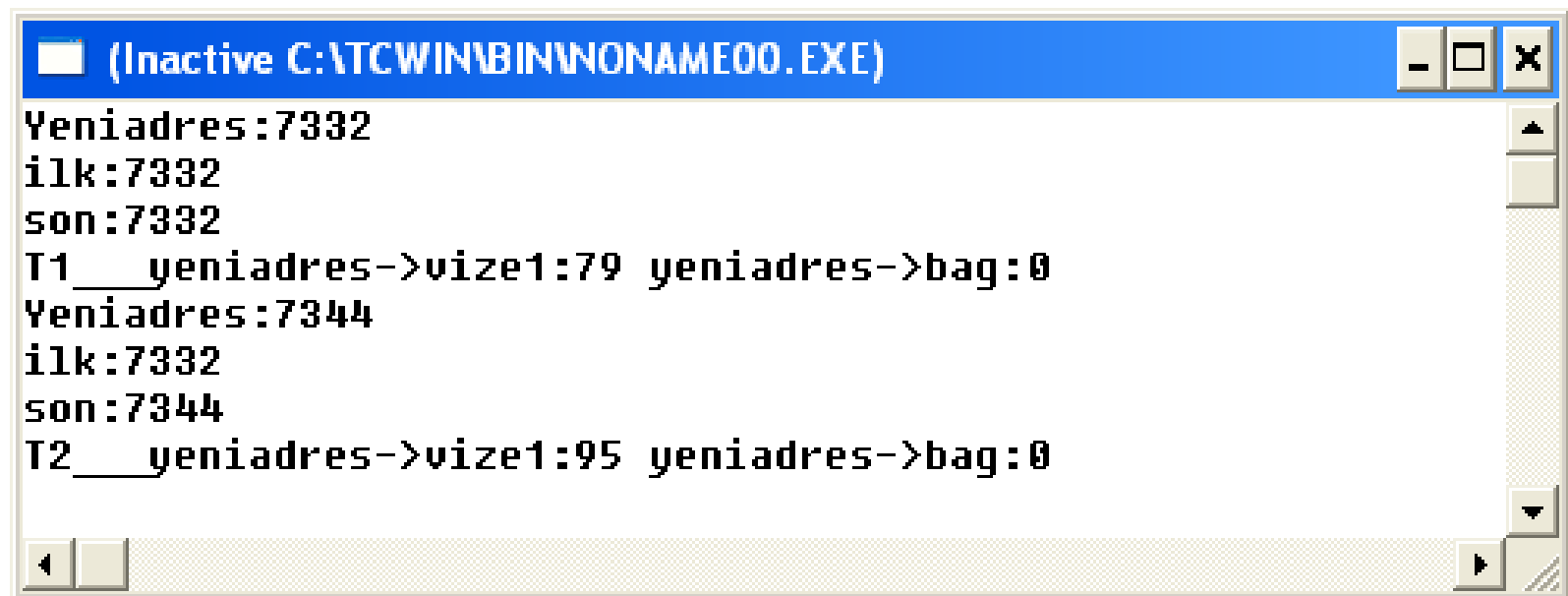
- İkinci işlemde hafızadan yeni bir adres isteme



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):");`

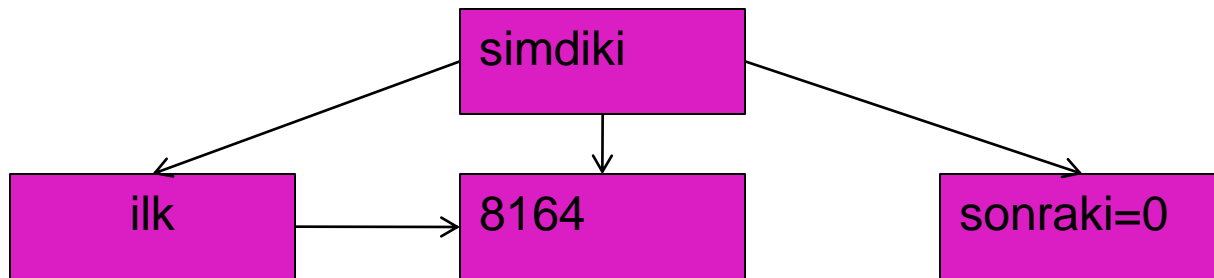
C++

- 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;
- }

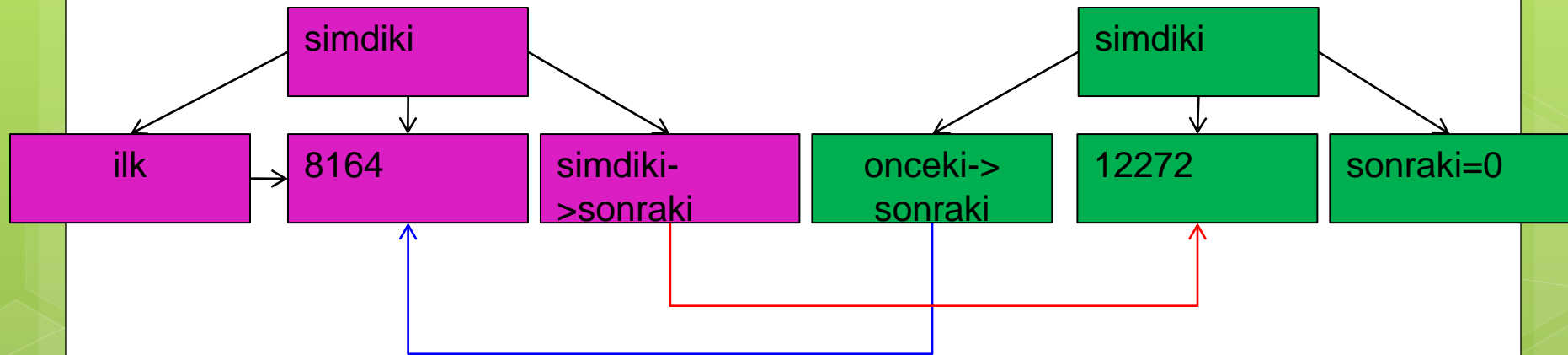
C++

- 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

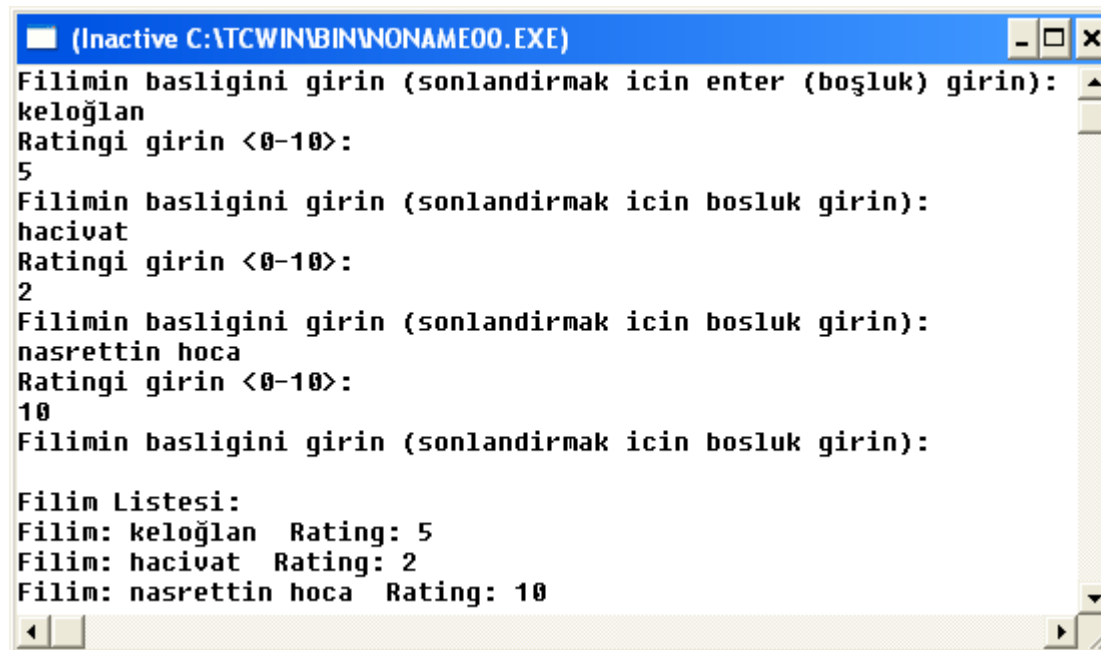


LİSTELER ve BAĞLI LİSTELER



LİSTELER ve BAĞLI LİSTELER

Örnekler



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

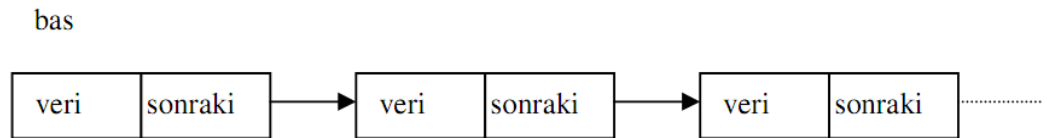
veri	sonraki
------	---------

- `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



Yığıt (Stack)

Yığıt/Yığın (Stack)

- Son giren ilk çıkar (**Last In First Out-LIFO**) veya İlk giren son çıkar (**First-in-Last-out FILO**) mantığıyla çalışır.
- Eleman ekleme çıkarmaların en üstten (top) yapıldığı veri yapısına yığıt (stack) adı verilir.
- Bir eleman ekleneceğinde yığıtın en üstüne konulur. Bir eleman çıkarılacağı zaman yığıtın en üstündeki eleman çıkarılır.
- Bu eleman da yığıttaki elemanlar içindeki en son eklenen elemandır. Bu nedenle yığıtlara LIFO (Last In First Out Son giren ilk çıkar) listesi de denilir.

Yığıt/Yığın (Stack)

- Yığın yapısını gerçekleştirmek için 2 yol vardır.
 - Dizi kullanmak
 - Bağlantılı liste kullanmak
- empty stack:** Boş yığıt
- push (koy):**Yığıta eleman ekleme.
- pop (al):**Yığıttan eleman çıkarma



Yığın İşlemleri

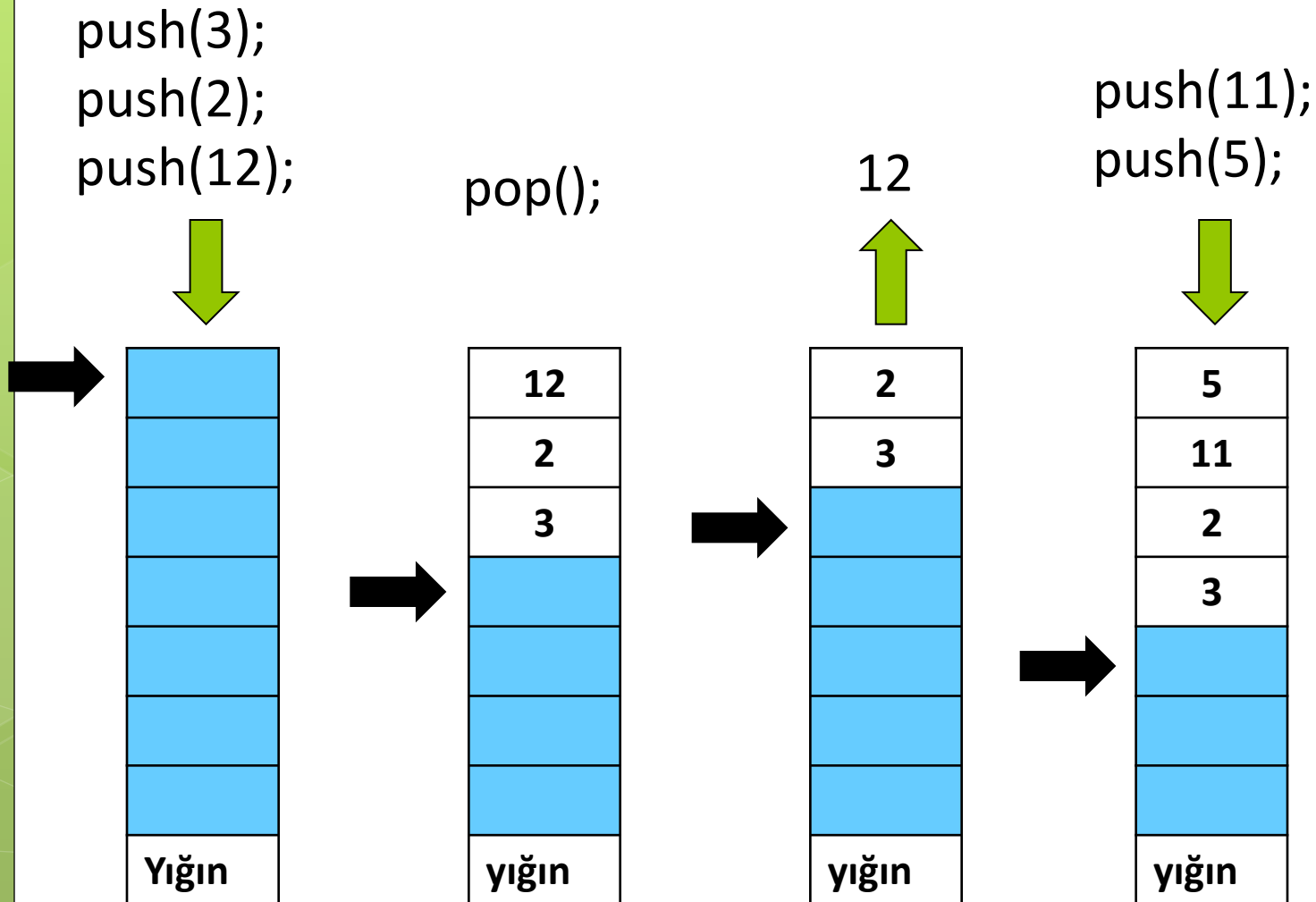
○ Ana yığın işlemleri:

- **push(nesne):** yeni bir nesne ekler
 - Girdi: Nesne Çıktı: Yok
- **pop():** en son eklenen nesneyi çıkarıp geri döndürür.
 - Girdi: Yok Çıktı: Nesne

○ Yardımcı yığın işlemleri:

- **top():** en son eklenen nesneyi çıkarmadan geri döndürür.
 - Girdi: Yok Çıktı: Nesne
- **size():** depolanan nesne sayısını geri döndürür.
 - Girdi: Yok Çıktı: Tamsayı
- **isEmpty():** yığında nesne bulunup bulunmadığı bilgisi geri döner.
 - Girdi: Yok Çıktı: Boolean

Yığın (stack) Yapısı



Yığıt/Yığın (Stack)

- Örnek kullanım yerleri
 - Yazılım uygulamalarındaki Undo işlemleri stack ile yapılır. Undo işlemi için LIFO yapısı kullanılır.
 - Web browser'lardaki Back butonu (önceki sayfaya) stack kullanır. Buradada LIFO yapısı kullanılır.
 - Matematiksel işlemlerdeki operatörler (+,*,/, - gibi) ve operandlar için stack kullanılabilir.
 - Yazım kontrolündeki parantezlerin kontrolünde stack kullanılabilir.

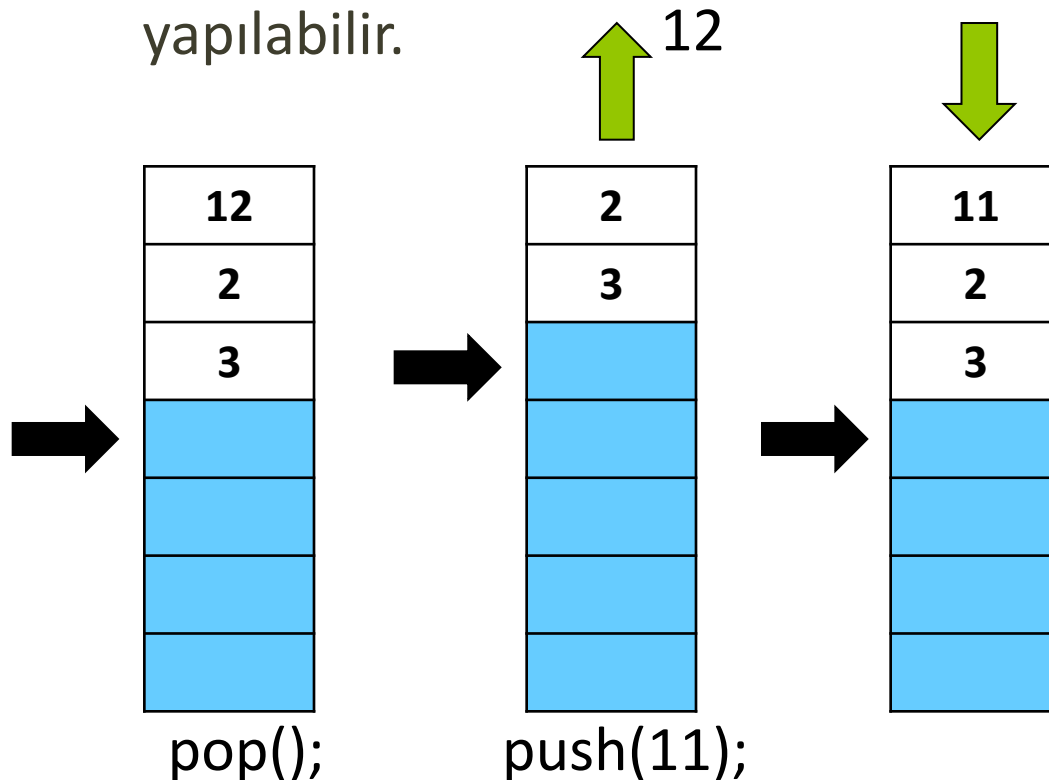
Yığıt/Yığın (Stack)

○ Örnek : Yığına ekleme ve çıkarma

İşlem	Yığıt (tepe)	Çıktı
○ push("M");	M	
○ push("A");	MA	
○ push("L");	MAL	
○ push("T");	MALT	
○ pop();	MAL	T
○ push("E");	MALE	T
○ pop();	MAL	TE
○ push("P");	MALP	TE
○ pop();	MAL	TEP
○ push("E");	MALE	TEP
○ pop();	MAL	TEPE

Dizi Tabanlı Yığın

- Bir yığının gerçekleştirmenin en kolay yolu dizi kullanmaktır. Yığın yapısı dizi üzerinde en fazla N tane eleman tutacak şekilde yapılabilir.



Dizi Tabanlı Yığın

- Nesneleri soldan sağa doğru ekleriz. Bir değişken en üstteki nesnenin index bilgisini izler. Eleman çıkarılırken bu index değeri alınır.

Algorithm *size()*

return $t + 1$

Algorithm *pop()*

if *isEmpty()* then

throw *EmptyStackException*

else

$t \leftarrow t - 1$

return $S[t + 1]$



Dizi Tabanlı Yığın

- Yığın nesnelerinin saklandığı dizi dolabilir. Bu durumda push işlemi aşağıdaki mesajı verir.
- FullStackException (DoluYığınİstinası)**
 - Dizi tabanlı yaklaşımın sınırlamasıdır.

```
Algorithm push(o)  
  if  $t = S.length - 1$  then  
    throw FullStackException  
  else  
     $t \leftarrow t + 1$   
     $S[t] \leftarrow o$ 
```



Başarım ve Sınırlamalar

- Başarım

- n yığındaki nesne sayısı olsun
- Kullanılan alan $O(n)$
- Her bir işlem $O(1)$ zamanda gerçekleşir.

- Sınırlamalar

- Yığının en üst sayısı önceden tanımlanmalıdır ve değiştirilemez.
- Dolu bir yığına yeni bir nesne eklemeye çalışmak istisnai durumlara sebep olabilir.

Büyüyebilir Dizi Tabanlı Yığın Yaklaşımı

- **push** işlemi esnasında dizi dolu ise bir istisnai durum bildirimi geri dönmektense yığının tutulduğu dizi daha büyük bir dizi ile yer değiştirilir.
- Yeni dizi ne kadar büyüklükte olmalı?
 - **Artımlı strateji:** yığın büyüklüğü sabit bir **c** değeri kadar arttırılır.
 - **İkiye katlama stratejisi:** önceki dizi boyutu iki kat arttırılır
- Bağlı liste yapılarını kullanarak yığın işlemini gerçekleştirmek bu tür problemlerin önüne geçmede yararlı olur.

```
Algorithm push(o)
  if  $t = S.length - 1$  then
     $A \leftarrow$  new array of
      size ...
    for  $i \leftarrow 0$  to  $t$  do
       $A[i] \leftarrow S[i]$ 
     $S \leftarrow A$ 
   $t \leftarrow t + 1$ 
   $S[t] \leftarrow o$ 
```

Yığın ve Operasyonları

```
public class Yigin {  
  
    int kapasite=100; // maksimum eleman sayısı  
    int S[]; //Yığın elemanları - pozitif tam sayı  
    int p; // eleman sayısı  
  
    public Yigin(){ // yapıcı yordam  
        s[] = new int[kapasite];  
        p = 0;  
    }  
  
    int koy(int item);  
    int al();  
    int ust();  
    boolean bosmu();  
    boolean dolumu();  
}
```

Yığın Operasyonları – bosmu, dolumu

```
// yığın boşsa true döndür  
public boolean bosmu() {  
    if (p < 1) return true;  
    else return false;  
} //bitti-bosmu
```

```
// Yığın doluysa true döndür  
public boolean dolumu() {  
    if (p == kapasite-1) return true;  
    else return false;  
} // bitti-dolumu
```

Yığın Operasyonları: koy

```
// Yığının üstüne yine bir eleman koy
// Başarılı ise 0 başarısız ise -1 döndürür.
int koy(int yeni){

    if (dolumu()){
        // Yığın dolu. Yeni eleman eklenemez.
        return -1;
    }

    S[p] = yeni;
    p++;

    return 0;
} /bitti-koy
```

Yığın Operasyonları: ust

```
// Yığının en üstündeki sayıyı döndürür
// Yığın boşsa, -1 döndürür
public int ust(){
    if (bosmu()){
        // Yığın başsa hata dönder
        System.out.println("Stack underflow");
        return -1;
    }

    return S[p-1];
}
```

Stack Operations: al

```
// En üsteki elemanı dönder.  
// Yığın boşsa -1 dönder.  
public int al(){  
    if (bosmu()){  
        // Yığın boşsa hata dönder  
        System.out.println("Stack underflow");  
        return -1;  
    }  
  
    int id = p-1; // en üsteki elemanın yeri  
    p--;         // elemanı sil  
  
    return S[id];  
}
```

Yığın Kullanım Örneği

```
public static void main(String[] args){
    Yigin y = new Yigin();

    if (y.bosmu())
        System.out.println("Yığın boş");

    y.koy(49);    y.koy(23);

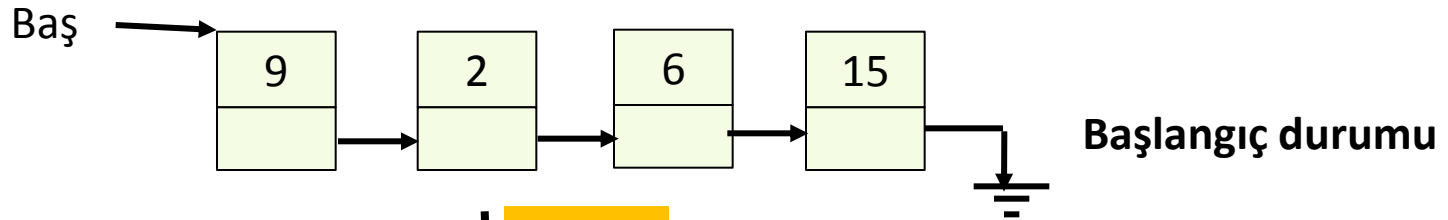
    System.out.println("Yığının ilk elemanı: "+ y.al());

    y.koy(44);    y.koy(22);

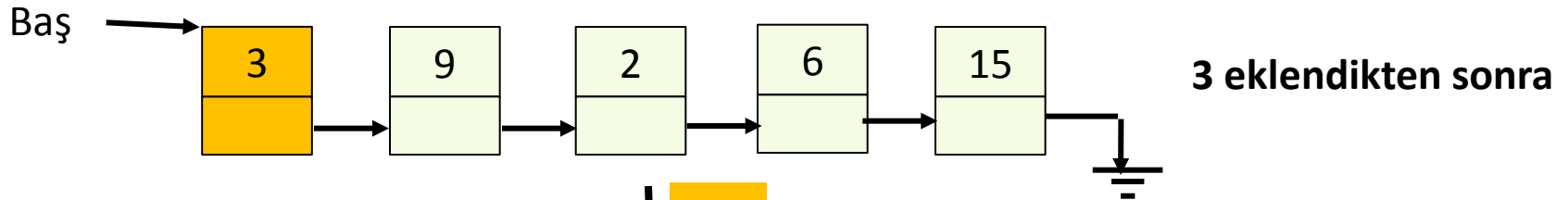
    System.out.println("Yığının ilk elemanı: "+ y.al());
    System.out.println("Yığının ilk elemanı: "+ y.al());
    System.out.println("Yığının ilk elemanı: "+ y.ust());
    System.out.println("Yığının ilk elemanı: "+ y.al());

    if (y.bosmu()) System.out.println("Yığın boş");
}
```

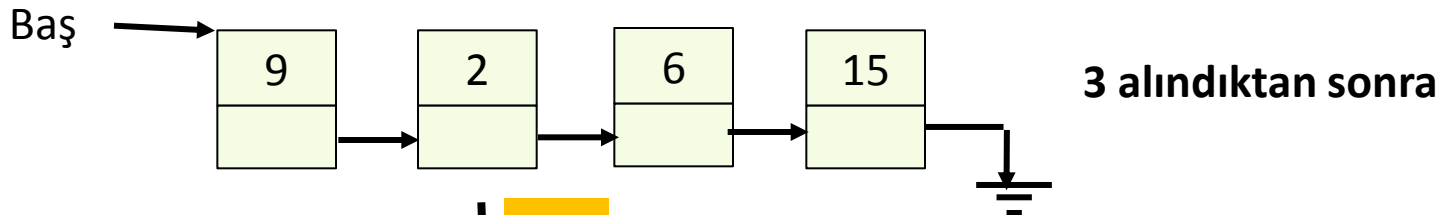
Yığın- Bağlantılı Liste Gerçekleştirimi



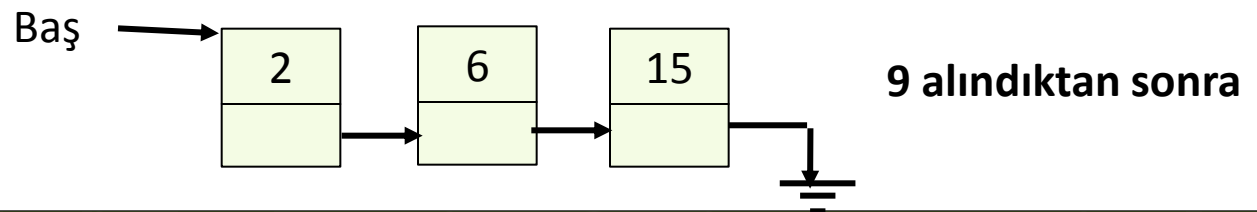
↓ koy(3)



↓ al()



↓ al()



Bağlantılı Liste Gerçekleştirimi

```
public class YiginDugumu {  
    int eleman;  
    YiginDugumu sonraki;  
  
    YiginDugumu(int e){  
        eleman = e; sonraki = NULL;  
    }  
}
```

```
public class Yigin {  
    private YiginDugumu ust;  
  
    public Yigin() {ust = null;}  
  
    void koy(int eleman);  
    int al();  
    int ust();  
    boolean bosmu();  
};
```

Yığın Operasyonları: koy, bosmu

```
// Yığına yeni eleman ekle
public void koy(int eleman){
    YiginDugumu x = new YiginDugumu(eleman);
    x.sonraki = ust;
    ust = x;
}

// Yığın boşsa true döndür
public boolean bosmu(){
    if (ust == NULL)
        return true;
    else
        return false;
}
```

Yığın Operasyonları: ust

```
// Yığının ilk elemanını döndür
public int ust(){
    if (bosmu()){
        System.out.println("Stack underflow"); // Boş yığın
        return -1; // Hata
    }

    return ust.eleman;
} //bitti-ust
```

Yığın Operasyonları: Al()

```
// Yığının en üst elemanın siler ve döndürür.  
public int Al(){  
    if (bosmu()){  
        System.out.println("Stack underflow"); // Boş yığın.  
        return -1; // Hata  
    }  
  
    YiginDugumu temp = ust;  
  
    // Bir sonraki elemana geç  
    ust = ust.sonraki;  
  
    return temp.eleman;  
} //bitti-al
```

Yığın Kullanım Örneği

```
public static void main(String[] args){
    Yigin y = new Yigin();

    if (y.bosmu())
        System.out.println("Yığın boş");

    y.koy(49);    y.koy(23);

    System.out.println("Yığının ilk elemanı: "+ y.al());

    y.koy(44);    y.koy(22);

    System.out.println("Yığının ilk elemanı: "+ y.al());
    System.out.println("Yığının ilk elemanı: "+ y.al());
    System.out.println("Yığının ilk elemanı: "+ y.ust());
    System.out.println("Yığının ilk elemanı: "+ y.al());

    if (y.bosmu()) System.out.println("Yığın boş");
}
```

Örnekler –Java

- Java'da hazır Stack (yığıt) sınıfı da bulunmaktadır. Aşağıdaki örnekte String'ler, oluşturulan **s** yığıtına yerleştirilerek ters sırada listelenmektedir.
- `import java.util.*;`
- `public class StackTest`
- `{`
- `public static void main(String args[])`
- `{ String str[] = { "Bilgisayar", "Dolap", "Masa", "Sandalye", "Sıra" };`
- `Stack s = new Stack();`
- `for(int i=0; i < t.length; ++i) s.push(str[i]);`
- `while(!s.empty()) System.out.println(s.pop());`
- `}`
- `}`

Uygulama Ödevi

- Derleyici/kelime işlemciler
 - Derleyicileri düşünecek olursak yazdığımız ifadede ki parantezlerin aynı olup olmadığını kontrol ederler.
 - Örneğin: $2*(i + 5*(7 - j / (4 * k))$ ifadesinde parantez eksikliği var. ")"
 - Yığın kullanarak ifadedeki parantezlerin eşit sayıda olup olmadığını kontrol eden programı yazınız.



Uygulama Ödevi

- Yığın kullanarak parantez kontrol:
 - 1) Boş bir yığın oluştur ve sembolleri okumaya başla
 - 2) Eğer sembol başlangıç sembolü ise ('(', '[', '{') Yığına koy
 - 3) Eğer sembol kapanış sembolü ise (')', ']', '}')
 - I. Eğer yığın boşsa hata raporu döndür
 - II. Değilse
 - Yığından al
 - Eğer alınan sembol ile başlangıç sembolü aynı değilse hata gönder
 - 4) İfade bitti ve yığın dolu ise hata döndür.

ÖRNEKLER

C# Programlama Dilinde Bağlı Liste Örneği ve Kullanımı-TEK YÖNLÜ BAĞLI LİSTE Düğüm Yapısı

- `public class ListNode`
- `{`
- `public string numara, adSoyad;`
- `public double ortalama;`
- `public ListNode sonraki;`
- `public ListNode(string numara, string adSoyad, double ortalama)`
- `{`
- `this.numara = numara; ;`
- `this.adSoyad = adSoyad;`
- `this.ortalama = ortalama;`
- `}`
- `}`

C# -TEK YÖNLÜ BAĞLI LİSTE Yapısı

- `public class LinkedList`
- `{`
- `public ListNode headNode, tailNode;`
- `public LinkedList()`
- `{`
- `headNode = new ListNode("head","",o);`
- `tailNode = new ListNode("tail","",o);`
- `headNode.sonraki = tailNode;`
- `tailNode.sonraki = tailNode;`
- `}`
- `}`

C# -TEK YÖNLÜ BAĞLI LİSTE

Başa Düğüm Ekleme

- `public void Ekle(LinkedList bL, ListNode IN, string yer)`
- `{`
- `ListNode aktif = bL.headNode;`
- `IN.sonraki = aktif.sonraki;`
- `aktif.sonraki = IN;`
- `}`

C# -TEK YÖNLÜ BAĞLI LİSTE

Sıraya Düğüm Ekleme

- `public void Ekle(LinkedList bL, ListNode IN, string yer)`
- `{`
- `ListNode aktif = bL.headNode;`
- `while ((aktif.sonraki != bL.tailNode) &&`
- `(string.Compare(aktif.sonraki.numara, IN.numara) < 0))`
- `aktif = aktif.sonraki;`
- `IN.sonraki = aktif.sonraki;`
- `aktif.sonraki = IN;`
- `}`

C# -TEK YÖNLÜ BAĞLI LİSTE

Sona Düğüm Ekleme

- `public void Ekle(LinkedList bL, ListNode lN, string yer)`
- `{`
- `ListNode aktif = bL.headNode;`
- `while (aktif.sonraki != bL.tailNode)`
- `aktif = aktif.sonraki;`
- `lN.sonraki = aktif.sonraki;`
- `aktif.sonraki = lN;`
- `}`

C# -TEK YÖNLÜ BAĞLI LİSTE

Düğüm Ekleme

- `public void Ekle(LinkedList bL, ListNode IN, string yer)`
- `{ ListNode aktif = bL.headNode;`
- `if (yer == "BASA")`
- `{ IN.sonraki = aktif.sonraki;`
- `aktif.sonraki = IN; }`
- `else if (yer == "SIRAYA")`
- `{`
- `while ((aktif.sonraki!= bL.tailNode) && (string.Compare(aktif.sonraki.numara, IN.numara)<0))`
- `aktif = aktif.sonraki;`
- `IN.sonraki = aktif.sonraki;`
- `aktif.sonraki = IN;`
- `}}`
- `else if (yer == "SONA")`
- `{ while (aktif.sonraki != bL.tailNode)`
- `aktif = aktif.sonraki;`
- `IN.sonraki = aktif.sonraki;`
- `aktif.sonraki = IN;`
- `} }`

C# -TEK YÖNLÜ BAĞLI LİSTE

Düğüm Silme

- **//Baştan düğüm silme**
- if (rBBas.Checked)
- `bagliListe.headNode.sonraki = bagliListe.headNode.sonraki.sonraki;`
- **//Sondan Düğüm silme**
- else if (rBSon.Checked)
- {
- `ListNode aktif = bagliListe.headNode;`
- while (aktif.sonraki.sonraki != bagliListe.tailNode)
- `aktif = aktif.sonraki;`
- `aktif.sonraki = bagliListe.tailNode;`
- }

C# -TEK YÖNLÜ BAĞLI LİSTE

Düğüm Silme

- **//Tümünü Silme**
- else if (rBTumu.Checked)
- {
- ListNode aktif = bagliListe.headNode;
- while (aktif.sonraki != bagliListe.tailNode)
- aktif.sonraki = aktif.sonraki.sonraki;
- }
- **//Aranan kişiyi Silme**
- else if (rBKisi.Checked)
- {
- ListNode aktif = bagliListe.headNode;
- while((aktif.sonraki!=bagliListe.tailNode) &&(aktif.sonraki.numara!=tBNum.Text))
- aktif = aktif.sonraki;
- aktif.sonraki = aktif.sonraki.sonraki;
- }

C# -TEK YÖNLÜ BAĞLI LİSTE

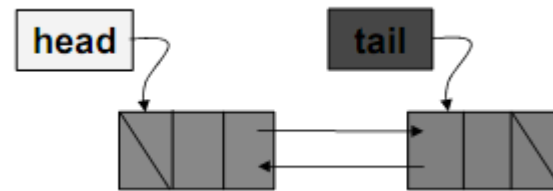
Düğüm Silme

o //İstenilen Sırada Silme

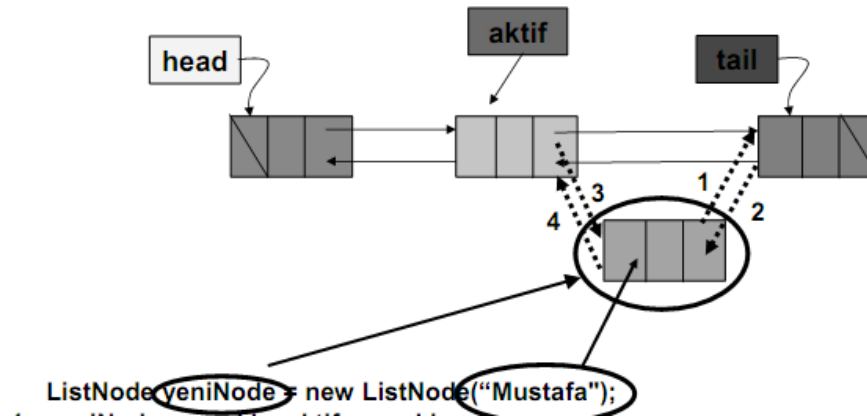
- o else if (rBSira.Checked)
- o { int Sira = Convert.ToInt16(tBSil.Text); int i = 1;
- o if (Sira != 0)
- o { ListNode aktif = bagliListe.headNode;
- o while ((aktif.sonraki != bagliListe.tailNode) && (i < Sira))
- o { aktif = aktif.sonraki; i++; }
- o if ((aktif.sonraki == bagliListe.tailNode) && ((i-1) < Sira))
- o MessageBox.Show("Listedeki eleman sayısından büyük değer girildi !" ,
- o "Hata !", MessageBoxButtons.OK);
- o else if (!(aktif.sonraki == bagliListe.tailNode))
- o { aktif.sonraki = aktif.sonraki.sonraki; }
- o }
- o }

C# Programlama Dilinde İKİ YÖNLÜ BAĞLI LİSTE

- İKİ YÖNLÜ BAĞLI LİSTE
- `public class ListNode {`
- `public string adSoyad;`
- `public ListNode onceki, sonraki;`
- `public ListNode(string adSoyad)`
- `{ this.adSoyad = adSoyad; }`
- `}`
- `public class LinkedList {`
- `public ListNode headNode, tailNode;`
- `public LinkedList()`
- `{`
- `headNode = new ListNode("head");`
- `tailNode = new ListNode("tail");`
- `headNode.onceki = headNode;`
- `headNode.sonraki = tailNode;`
- `tailNode.onceki = headNode;`
- `tailNode.sonraki = tailNode;`
- `}`
- `}`

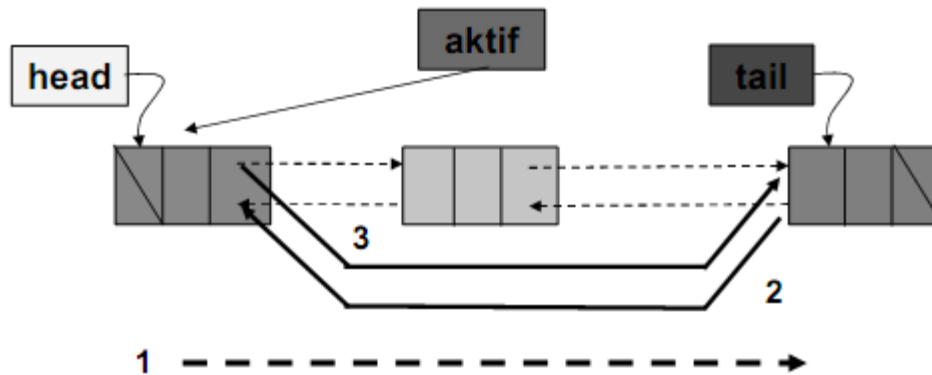


C# Programlama Dilinde Bağlı Liste Örneği ve Kullanımı



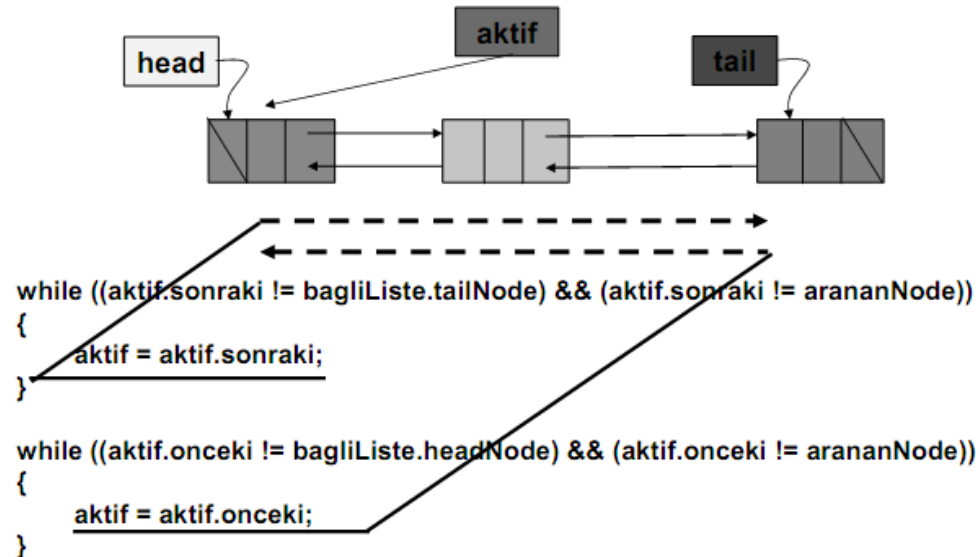
- Eleman ekleme
- `ListNode yeniNode = new ListNode("Mustafa");`
- `yeniNode.sonraki = aktif.sonraki;`
- `aktif.sonraki.onceki = yeniNode;`
- `aktif.sonraki = yeniNode;`
- `yeniNode.onceki = aktif;`

C# Programlama Dilinde Bağlı Liste Örneği ve Kullanımı



- Eleman silme
- while ((aktif.sonraki != bagliListe.tailNode) && (aktif.sonraki != silinecekNode))
- {
- aktif = aktif.sonraki;
- }
- aktif.sonraki.sonraki.onceki = aktif;
- aktif.sonraki = aktif.sonraki.sonraki;

C# Programlama Dilinde Bağlı Liste Örneği ve Kullanımı



- Eleman arama
- while ((aktif.sonraki != bagliListe.tailNode) && (aktif.sonraki != arananNode))
- { aktif = aktif.sonraki;}
- while ((aktif.onceki != bagliListe.headNode) && (aktif.onceki != arananNode))
- { aktif = aktif.onceki; }

C++ Dairesel Bağlı listeler ile oyun

- #include "stdio.h"
- #include "conio.h"
- #include "stdlib.h"
- #include "string.h"
- #include "iostream.h"
- //OYUNCULARIN İSİMLERİ-----
- char *isimler[10]= {"HACER", "GULTEN","IDRIS", "HASAN","HATİCE",
"CEMIL","BELMA", "CANSU","EBRU","NALAN"};
- //-----
- typedef struct veri{
- char adi[20];
- struct veri *arka;
- } BLISTE;
- BLISTE *ilk=NULL,*son=NULL;
- //-----

Örnekler-Dairesel Bağlı listeler ile oyun

```
○ int ekle(BLISTE *sayigeldi)
○ {
○   if(ilk==NULL)
○   {
○     ilk=sayigeldi;
○     son=ilk;
○     ilk->arka=ilk;
○   }
○   else
○   {
○     son->arka=sayigeldi;
○     son=sayigeldi;
○     son->arka=ilk;
○   }
○   return 0;
○ }
○ //-----
```


Örnekler-Dairesel Bağlı listeler ile oyun

- `int oyuncular()`
- `{ BLISTE *p; int x=3,y=3,i=0; clrscr();`
- `gotoxy(x,y);printf("DAIRESEL BAGLI LISTE");y+=2;`
- `gotoxy(x,y);printf("ODEVIN KONUSU : Dairesel Bagli Liste Yapisini
Kullanarak Oyun Tasarlama");`
- `y+=3; gotoxy(x,y);printf("Oyuncular : "); p=ilk;`
- `while(p->arka!=ilk) { printf("%s\n ",p->adi); p=p->arka; i++; }`
- `printf("%s\n ",p->adi);`
- `y+=13;`
- `gotoxy(x,y);printf("TOPLAM OYUNCU SAYISI : %d ",i); getch();`
- `return 0;}`
- `//-----`

Örnekler-Dairesel Bağlı listeler ile oyun

- `int oyundan_at(BLISTE *onceki,BLISTE *silinecek) {`
- `if(ilk->arka==ilk) { //listede bir eleman varsa`
- `clrscr();`
- `printf("Oyunda Su an Sadece 1 kisi var");`
- `printf("OYUNU KAZANAN = %s",ilk->adi); }`
- `else if(onceki->arka==ilk) //silinecek ilk eleman mı?`
- `{ onceki->arka=ilk->arka; ilk=ilk->arka;`
- `printf(" Oyundan Cıkıyor....%s",onceki->adi);`
- `free(silinecek); oyuncular(); }`
- `else if(silinecek==son)//silinecek son eleman mı?`
- `{ onceki->arka=ilk; son=onceki; free(silinecek); oyuncular(); }`
- `else //silinecek aradan mı?`
- `{ onceki->arka=silinecek->arka; free(silinecek); oyuncular(); }`
- `return 0;}`
- `//-----`

Örnekler-Dairesel Bağlı listeler ile oyun

- void main(void) {
- clrscr(); randomize();
- BLISTE *yeni;int i,x=3,y=3;
- for(i=0;i<10;i++) {
- yeni=(BLISTE *)malloc(sizeof(BLISTE));
- if(!yeni) { clrscr();
- printf("Oyuncuları Ekleme Yapacak Kadar Bos Alan Yok Kusura Bakmayiniz");
- exit(0); }
- else { strcpy(yeni->adi,isimler[i]); ekle(yeni); }
- }
- clrscr();x=3;y=3; int soylene;
- oyuncular();
- yeni=ilk; BLISTE *p; BLISTE *bironceki;

Örnekler-Dairesel Bağlı listeler ile oyun

- do {
- printf("\n\nSAYIN :%s ---> Bir Sayı Soyler misiniz ? ",ilk->adi);
- scanf("%d",&soylenen);
- for(i=0;i<soylenen;i++) {
- yeni->adi; bironceki=yeni; yeni=yeni->arka; }
- //bağı koparılacak olan.. adresi=yeni, bir oncekinin adresi bironcekinde, tahmini p ile adreslenen kişi yapacak
- p=yeni->arka;//sayıyı tahmin edecek kişi
- oyundan_at(bironceki,yeni); yeni=p;
- } while(ilk!=son);
- if(ilk==son) { clrscr(); x=5; y=5;
- gotoxy(x,y); printf("Oyunda Su an Sadece 1 kisi var");
- gotoxy(x,y);printf("OYUNU KAZANAN = %s",ilk->adi); y+=2;
- gotoxy(x,y);printf("Oyun bitmistir...programi sonlandırmak için herhangi bir tusa basınız."); y+=2; getch(); exit(0); }
- getch(); }

Örnekler -Java

- Örnek : Java'da dizi kullanarak karakter yığıtı sınıfı oluşturma ve kullanma.
- `import java.io.*;`
- `class StackChar`
- `{`
- `private int maxSize; private char[] stackArray; private int top;`
- `public StackChar(int max)`
- `{ maxSize = max; stackArray = new char[maxSize]; top= -1; }`
- `public void push(char j) { stackArray[++top] = j; }`
- `public char pop() { return stackArray[top--]; }`
- `public boolean isEmpty() { return top== -1; }`
- `}`

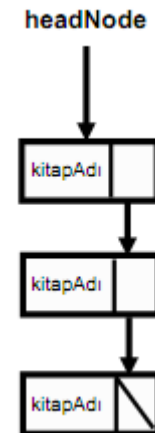
Örnekler –Java

```
○ class Reverse
○ {
○   public static void main(String args[])
○   {
○     StackChar y = new StackChar(100);
○     String str = "Merhaba";
○     for(int i=0; i<str.length(); ++i)
○       y.push(str.charAt(i));
○     while(! y.isEmpty() ) System.out.println(y.pop());
○   }
○ }
```

Örnekler –C#

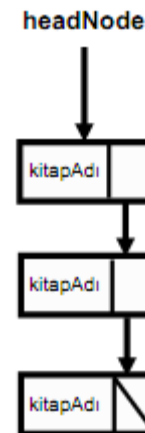
- **Örnek:** Bağlı liste ile yığıt oluşturma
- `class stackNodeC`
- `{`
- `public string kitapAdi; public stackNodeC sonraki;`
- `public stackNodeC(string kitapAdi) { this.kitapAdi = kitapAdi; }`
- `}`

- `class stackC`
- `{`
- `public stackNodeC headNode;`
- `public stackC(string kitapAdi)`
- `{`
- `this.headNode = new stackNodeC(kitapAdi);`
- `this.headNode.sonraki = headNode;`
- `}`
- `}`
- `stackC kitapYigin = new stackC("");`
-



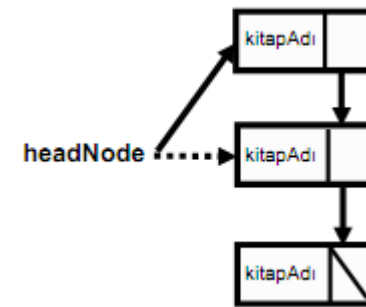
Örnekler –C#

- /* Stack işlemleri :
- boş yığın `stackSize() == 0` eleman sayısı= `stackSize()`
- eleman ekleme= `push(kitapAdi)` eleman alma= `pop()` */
- `public int stackSize()`
- {
- `stackNodeC aktif = new stackNodeC("");`
- `aktif = kitapYigin.headNode;`
- `int i = 0;`
- `while (aktif.sonraki != aktif)`
- {
- `aktif = aktif.sonraki; i++;`
- }
- `return i;`
- }



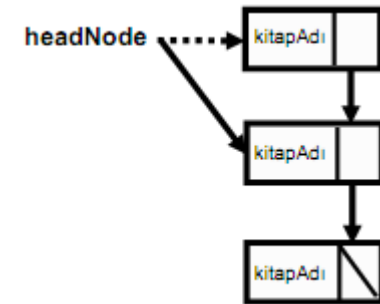
Örnekler –C#

- // Stack işlemleri (eleman ekleme)
- public void push(string kitapAdi)
- {
- if (stackSize() >= MaxSize)
- MessageBox.Show ("Yığın maksimum elemana sahip ! Yeni
- eleman eklenemez !", "Dikkat");
- else
- { stackNodeC yeniNode = new stackNodeC(tBKitapAdi.Text);
- yeniNode.sonraki = kitapYigin.headNode;
- kitapYigin.headNode = yeniNode;
- IStackSize.Text = "Stack Size =
- "+Convert.ToString(stackSize());
- }
- }



Örnekler –C#

- **// Stack işlemleri (eleman alma)**
- `public void pop()`
- `{`
- `if (stackSize() == 0)`
- `{`
- `MessageBox.Show("Yığında eleman yok !", "Dikkat");`
- `}`
- `else`
- `{`
- `kitapYigin.headNode = kitapYigin.headNode.sonraki;`
- `}`
- `}`



Örnekler –C++

- **Örnek:** Yığıt (Stack) kullanarak bağlı liste işlemleri. Girilen cümleyi kelimeleri bozmadan tersten yazdır.
- `#include <stdio.h>`
- `#include <string.h>`
- `#include <conio.h>`
- `#include <stdlib.h>`
- `struct kelimeler`
- `{ char kelime[50];`
- `struct kelimeler *onceki;`
- `} *tepe,*yenieleman,*yedek;`

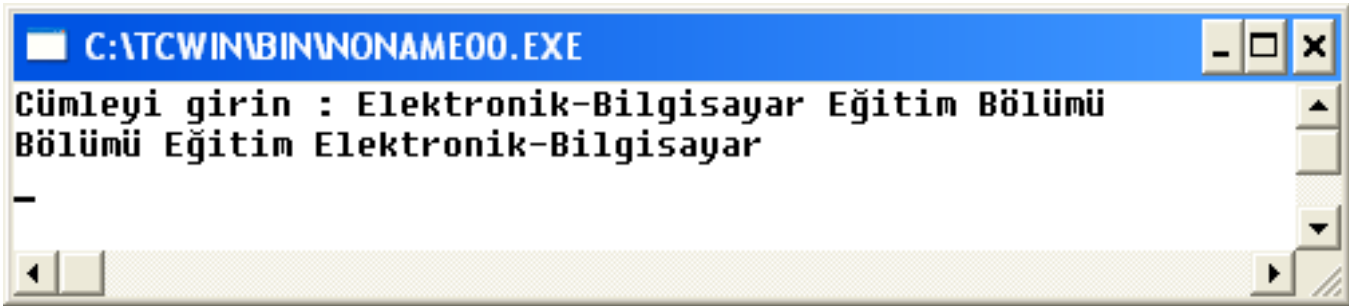
Örnekler –C++

- `void push(char *gelen) {`
- `yenieleman =(struct kelimeler *) malloc(sizeof(struct kelimeler));`
- `strcpy(yenieleman->kelime, gelen);`
- `yenieleman->onceki = tepe;`
- `tepe = yenieleman; }`
- `int pop(char *giden) {`
- `if (tepe != NULL) {`
- `yedek = tepe;`
- `strcpy(giden, tepe->kelime);`
- `tepe = tepe->onceki; free(yedek); return 0; }`
- `else`
- `return 1; /* yığın boş */`

Örnekler –C++

```
○ main() {  
○   char *yenikelime, *cumle;  
○   tepe = NULL;  
○  
○   printf("Cümleyi girin : "); gets(cumle);  
○   yenikelime = strtok(cumle, " "); // cumle değişkenini boşluğa göre  
    ayırma.  
○  
○   while (yenikelime) {  
○       push (yenikelime);  
○       yenikelime = strtok(NULL, " ");  
○   }  
○   yenikelime = (char *) malloc(20); /* yenikelime NULL idi, yer açalım*/  
○   while (!pop(yenikelime)) printf("%s ", yenikelime);  
○   printf("\n");  
○   getch();  
○ }
```

Örnekler –C++



```
C:\TCWIN\BIN\WONAME00.EXE
Cümleyi girin : Elektronik-Bilgisayar Eğitim Bölümü
Bölümü Eğitim Elektronik-Bilgisayar
-
```