

Yığın (Stack) devam

Infix, Postfix, & Prefix
Gösterimleri

Yığın Kullanımı - Infix Gösterimi

- Genellikle cebirsel işlemleri şu şekilde ifade ederiz: $a + b$
- Buna infix gösterim adı verilir, çünkü operatör (“+”) ifade içindedir.
- Problem: Daha karmaşık cebirsel ifadelerde parantezlere ve öncelik kurallarına ihtiyaç duyulması.
- Örneğin:
 - $a + b * c = (a + b) * c ?$
 - $= a + (b * c) ?$

Infix, Postfix, & Prefix Gösterimleri

- Herhangi bir yere operatör koymamamızın önünde bir engel yoktur.
- **Operatör Önde (Prefix) : + a b**
 - Biçim: işlem işlenen işlenen (operator operand operand) şeklindedir: + 2 7
 - İşlem sağdan sola doğru ilerler. Öncelik (parantez) yoktur.
- **Operatör Arada (Infix) : a + b**
 - Biçim: işlenen işlem işlenen (operand operator operand) şeklindedir: 2 + 7
 - İşlem öncelik sırasına göre ve soldan sağa doğru ilerler.
- **Operatör Sonda (Postfix) : a b +**
 - Biçim: işlenen işlenen işlem (operand operand operator) şeklindedir: 2 7 +
 - İşlem soldan sağa doğru ilerler. Öncelik (parantez) yoktur.

Prefix, Postfix : Diğer İsimleri

- Prefix gösterimi Polanyalı (**Polish**) bir mantıkçı olan **Lukasiewicz**, tarafından tanıtıldığı için “**Polish gösterim**” olarak da isimlendirilir.
- Postfix gösterim ise ters **Polish** gösterim “**reverse Polish notation**” veya **RPN** olarak da isimlendirilebilir.

Neden Infix, Prefix, Postfix ?

- **Soru:** infix gösterimde çalışmayla herşey yolunda iken neden böyle “aykırı”, “doğal olmayan” bir gösterim şekli tercih edilsin.?
- **Cevap:** postfix and prefix gösterimler ile parantez kullanılmasına gerek yoktur !
- Bir çok compilerlar programlardaki infix notasyonunda yazili expressionlari postfix notasyonuna çevirerek bellekte saklar.

Infix, Prefix, Postfix İşlemleri

- **İşlem önceliği (büyükten küçüğe)**

- Parantez
- Üs Alma
- Çarpma /Bölme
- Toplama/Çıkarma

- Parantezsiz ve aynı önceliğe sahip işlemcilerde soldan sağa doğru yapılır (üs hariç).
- Üs almada sağdan sola doğrudur. $A-B+C$ 'de öncelik $(A-B)+C$ şeklindedir. A^B^C 'de ise $A^(B^C)$ şeklindedir. (parantezler öncelik belirtmek için konulmuştur)

Parantez -Infix

- $2+3*5$ işlemini gerçekleştiriniz.
- $+$ önce ise:
 - $(2+3)*5 = 5*5 = 25$
- $*$ önce ise:
 - $2+(3*5) = 2+15 = 17$
- Infix gösterim paranteze ihtiyaç duyar.

Prefix Gösterim

- $* + 2 3 5 =$

- $= * \underline{+ 2 3} 5$

- $= * \underline{5 5} = 25$

- $+ 2 * 3 5 =$

- $= + 2 \underline{* 3 5}$

- $= + 2 \underline{15} = 17$

- Paranteze ihtiyaç yok!

Postfix Gösterim

- $2\ 3 + 5\ * =$

- $= \underline{2\ 3} + 5\ *$

- $= \underline{5\ 5}\ * = 25$

- $2\ 3\ 5\ * + =$

- $= 2\ \underline{3\ 5}\ * +$

- $= \underline{2\ 15}\ + = 17$

- **Sonuç:**


- Infix işlem sıralarının düzenlenmesi için paranteze ihtiyaç duyan tek gösterim şeklidir


Tamamen Parantezli Anlatım


- TPA gösterimde her operatör ve işlenenini çevreleyen parantezlerden oluşan tam bir set vardır.
- Hangisi tam parantezli gösterim?
 - $(A + B) * C$
 - $((A + B) * C)$
 - $((A + B) * (C))((A + B) (C))$

Infix'ten Prefix'e Dönüşüm

- Her bir operatörü kendi işlenenlerinin soluna taşı ve parantezleri kaldır. :

$$((A + B) * (C + D))$$


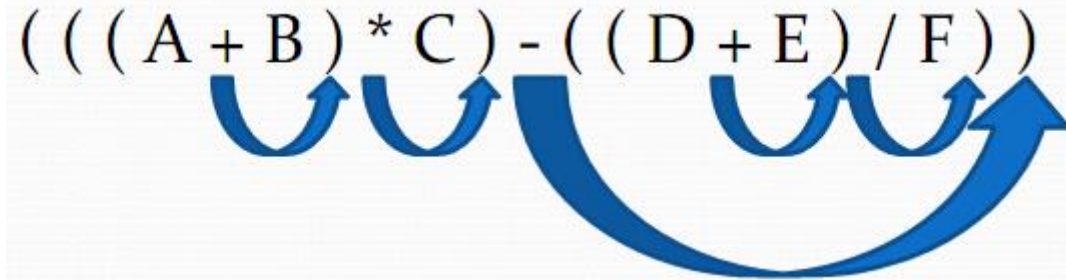
$$(+ A B * (C + D))$$


$$* + A B (C + D)$$


$$* + A B + C D$$

- İşlenenlerin sırasında bir değişiklik olmadı!

Infix'ten Postfix'e Dönüşüm



- $((AB+* C) - ((D + E) / F))$
- $(AB+C* - ((D + E) / F))$
- $AB+C* ((D + E) / F)-$
- $AB+C* (DE+ / F)-$
- $A B + C * D E + F / -$
- İşlenenlerin sırası değişmedi!
- Operatörler değerlendirme sırasına göre!

Infix, Prefix, Postfix

- Aşağıda verilen işlemlerde işleyişe bakınız

Infix	Postfix	Prefix
$A+B-C$	$AB+C-$	$-+ABC$
$(A+B)*(C-D)$	$AB+CD-*$	$*+AB-CD$
$A^B*C-D+E/F/(G+H)$	$AB^C*D-EF/GH+/+$	$+-*^ABCD//EF+GH$
$((A+B)*C-(D-E))^(F+G)$	$AB+C*DE-FG+^$	$^-*+ABC-DE+FG$
$A-B/(C*D^E)$	$ABCDE^*/-$	$-A/B*C^DE$

Infix, Prefix, Postfix İşlemleri

- Örnek: Parantezsiz operatör arada ifadenin operatör sonda hale çevrilmesi : $a + b * c - d$

<u>Okunan</u>	<u>Yığıt</u>	<u>Çıktı /Operatör sonda ifade</u>
a		a
+	+	a
b	+	a b
*	+ *	a b
c	+ *	a b c
-	+ *	a b c
	+	a b c *
	-	a b c * +
d	-	a b c * + d -

Operatör Sonda (Postfix) İfadenin İşlenişi:

Örnek: a b c * + d - ifadesini a=2 b=3 c=5 d=10 --> 2 3 5 * + 10 -

<u>Okunan</u>	<u>Yığıt</u>	<u>Hesaplanan</u>
2	2	
3	2 3	
5	2 3 5	
*	2	islem=* pop1=5 pop2=3
	2 15	3 * 5=15
+	17	islem=+ pop1=15 pop2=2
		2 + 15 =17
10	17 10	
-	7	islem=- pop1=10 pop2=17
		17 - 10= 7
		a b c * + d -

Infix, Prefix, Postfix İşlemleri

- Örnek: Parantezli operatör arada ifadenin operatör sonda hale çevrilmesi. Infix ifade: $(2 + 8) / (4 - 3)$

Okunan	Yığıt	Hesaplanan
((
2	(2
+	(+	2
8	(+	2 8
)		2 8 +
/	/	2 8 +
(/(2 8 +
4	/(2 8 + 4
-	/(-	2 8 + 4
3	/(-	2 8 + 4 3
)	/	2 8 + 4 3 -
		2 8 + 4 3 -
		2 8 + 4 3 - /

Örnek: Java

- postfix olarak yazılmış ifadeyi hesaplayarak sonucu bulan bir program yazınız.
- Örnek olarak "5 3 2 - / 4 7 * + "ifadesinin sonucunu bulunuz.
- Sonuç = 33 olarak bulunacak.

Örnek: Java

- `import java.awt.*;`
- `import java.io.*;`
- `import java.util.*;`

- `public class yigin {`
- `public int top=0;`

- `public void push(int x, int a[], int top)`
- `{ a[top] = x; ++top; this.top=top; }`

- `public int pop(int a[], int top)`
- `{ --top; this.top=top; return a[top]; }`
-

Örnek:

- `public static void main(String[] args) {`
- `yigin metot = new yigin();`
- `String str=""; int x = 0 , a=0,b=0; int stk[]=new int [120];`
- `Scanner oku = new Scanner(System.in);`
- `System.out.println("\n Postfix ifadeyi giriniz\n");`
- `str=oku.nextLine();`
- `for(int i=0;i<str.length();i++)`
- `{ if(str.charAt(i) == '+')`
- `{ b = metot.pop(stk, metot.top);`
- `a = metot.pop(stk, metot.top);`
- `metot.push(a + b, stk, metot.top);`
- `System.out.println("\n a+b =" + (a+b));`
- `}`

Örnek:

- `else if(str.charAt(i) == '-')`
- `{ b = metot.pop(stk,metot.top); a = metot.pop(stk, metot.top);`
- `metot.push(a - b, stk,metot.top);`
- `System.out.println("\n a-b =" + (a-b));`
- `}`
- `else if(str.charAt(i) == '/')`
- `{ b = metot.pop(stk, metot.top); a = metot.pop(stk, metot.top);`
- `metot.push(a / b, stk, metot.top);`
- `System.out.println("\na/b= " + a/b);`
- `}`
- `else if(str.charAt(i) == '*')`
- `{ b = metot.pop(stk, metot.top); a = metot.pop(stk, metot.top);`
- `metot.push(a * b, stk, metot.top);`
- `System.out.println("\n a*b= " + a*b);`
- `}`
-
-
-

Örnek:

- `if(str.charAt(i) >= '0' && str.charAt(i) <= '9')`
- `{ x =str.charAt(i)- '0'; metot.push(x, stk, metot.top); }`
- `}`
- `System.out.println("\n Postfix ifadesinin işlem sonucu="+ metot.pop(stk, metot.top));`
- `}`
- Çıktı:
- Postfix ifadeyi giriniz
- `532-/47*+`
- `a-b =1`
- `a/b= 5`
- `a*b= 28`
- `a+b =33`
- Postfix ifadesinin işlem sonucu=33

Infix, Prefix, Postfix İşlemleri

- Örnek: Aşağıda boş bırakılan alanları tamamlayınız

Infix	Prefix	Postfix
2x(3+5)-7^(2+1)		
	++x23^-5721	
		235+7-^2x1+
2x3+5-7^2+1		

Infix, Prefix, Postfix İşlemleri

○ Cevap

○ Infix

○ $2x(3+5)-7^{(2+1)}$

○ $(2x3)+(5-7)^2+1$

○ $2^{(3+5-7)}x2+1$

○ $2x3+5-7^2+1$

Prefix

$-x2+35^7+21$

$++x23^5-721$

$+x^2-+35721$

$+--+x235^721$

Postfix

$235+x721+^--$

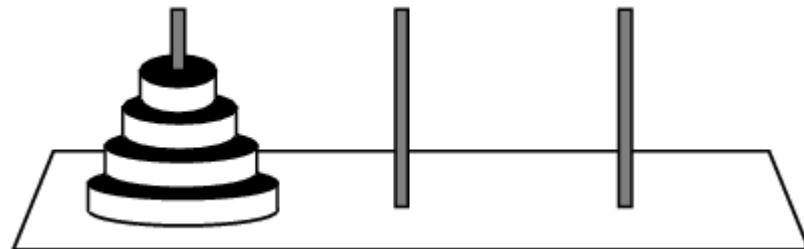
$23x57-2^+1+$

$235+7-^2x1+$

$23x5+72^--1+$

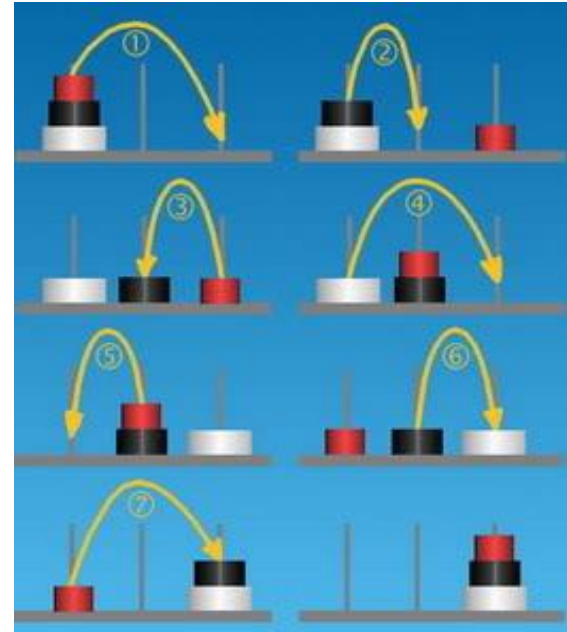
Hanoi Kuleleri Yığın temelli Çözüm

- **Verilen:** üç iğne
 - İlk iğnede en küçük disk en üstte olacak şekilde yerleştirilmiş farklı büyüklükte disk kümesi.
- **Amaç:** diskleri en soldan en sağa taşımak.
- **Şartlar:** aynı anda sadece tek disk taşınabilir.
- Bir disk boş bir iğneye veya daha büyük bir diskin üzerine taşınabilir.



Hanoi Kuleleri Yığın temelli Çözüm

- Problem karmaşıklığı 2^n
- 64 altın disk
- 1 taşıma işlemi 1 saniye sürsün:
- 18446744073709551616 sn
- 593.066.617.596,15 yıl
- Dünyanın sonuna 600.000.000.000 yıl var 😊



Hanoi Kuleleri– Özyinelemeli Çözüm-Java

```

○ package hanoikuleleri;
○ import java.util.*;
○ public class Hanoikuleleri {

○ public static void main(String[] args)
○ {
○   System.out.print("n değerini giriniz : ");
○   Scanner klavye = new Scanner(System.in);   int n = klavye.nextInt();
○   tasi(n, 'A', 'B', 'C');
○ }
○ public static void tasi(int n, char A, char B, char C)
○ {if(n==1) System.out.println(A + " --> " + B);
○   else
○   {
○     tasi(n-1, A, C, B);      tasi(1, A, B, C);      tasi(n-1, C, B, A); }
○   return;
○ }

```

Ödev

- 1-Infix'ten Prefix ve Postfix'e Çevirin

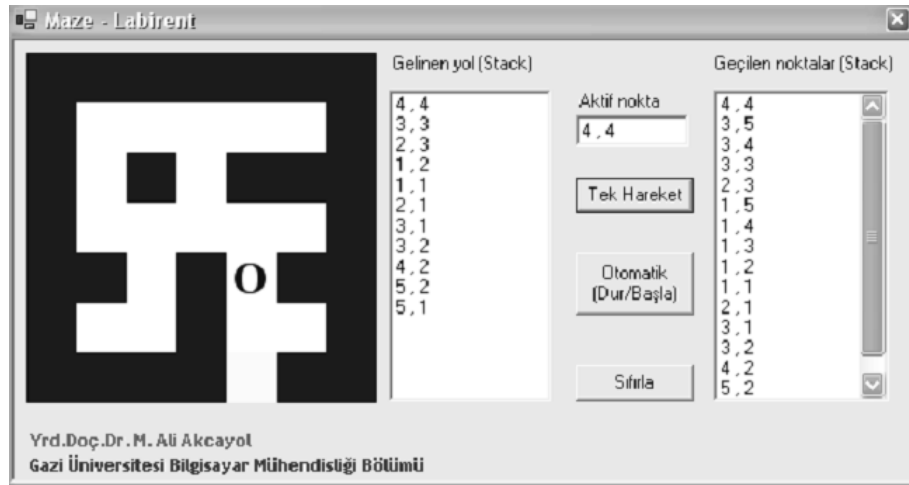
- x
- $x + y$
- $(x + y) - z$
- $w * ((x + y) - z)$
- $(2 * a) / ((a + b) * (a - c))$


- 2-Postfix'ten Infix'e Çevirin

- $3 r -$
- $1 3 r - +$
- $s t * 1 3 r - + +$
- $v w x y z * - + *$

Ödev

- 3- postfix olarak yazılmış ifadeyi hesaplayarak sonucu bulan programı Java/C# bağlı liste yapısı ile yazınız.
- 4-Verilen Maze (Labirent) uygulamasını -başlangıç olarak istediğimiz noktadan başlayarak çıkışa ulaşmasını sağlayınız.





Kuyruk (Queue)

Kuyruk (Queue)



- Kuyruklar, eleman eklemelerinin **sondan** (rear) ve eleman çıkarmalarının **baştan** (front) yapıldığı doğrusal veri yapılarıdır.
- Bir eleman ekleneceği zaman kuyruğun sonuna eklenir.
- Bir eleman çıkarılacağı zaman kuyrukta bulunan ilk eleman çıkarılır.
- Bu nedenle kuyruklara **FIFO** (First In First Out-ilk giren ilk çıkar) veya **LILO** (Last-in-Last-out-Son giren son çıkar) listeleri de denilmektedir.

Kuyruk (Queue)

- Gerçek yaşamda banklarda, duraklarda, otoyollarda kuyruklar oluşmaktadır. Kuyruğu ilk olarak girenler işlemlerini ilk olarak tamamlayıp kuyruktan çıkarlar.
- İşletim sistemleri bünyesinde öncelik kuyruğu, yazıcı kuyruğu gibi birçok uygulama alanı vardır.
- Kuyruk modeli, program tasarımında birçok yerde kullanılır. Örneğin, iki birimi arasında iletişim kanalı, ara bellek/tampon bellek oluşturmada bu modele başvurulur.

Kuyruk (queue)

- Ardışıl nesneleri saklarlar
- Ekleme ve silme FIFO'ya göre gerçekleşir.
- Ekleme işlemi kuyruk arkasına yapılırken, çıkarma işlemi ise kuyruk önünden yapılır.
- **Ana Kuyruk İşlemleri:**
- İki tane temel işlem yapılabilir ;
 - **enqueue (object)**, bir nesneyi kuyruğun en sonuna ekler (**insert**).
 - **object dequeue ()**,
Kuyruk başındaki nesneyi getirir ve kuyruktan çıkarır (**remove** veya **delete**).

Kuyruk Veri Yapısı Bileşenleri

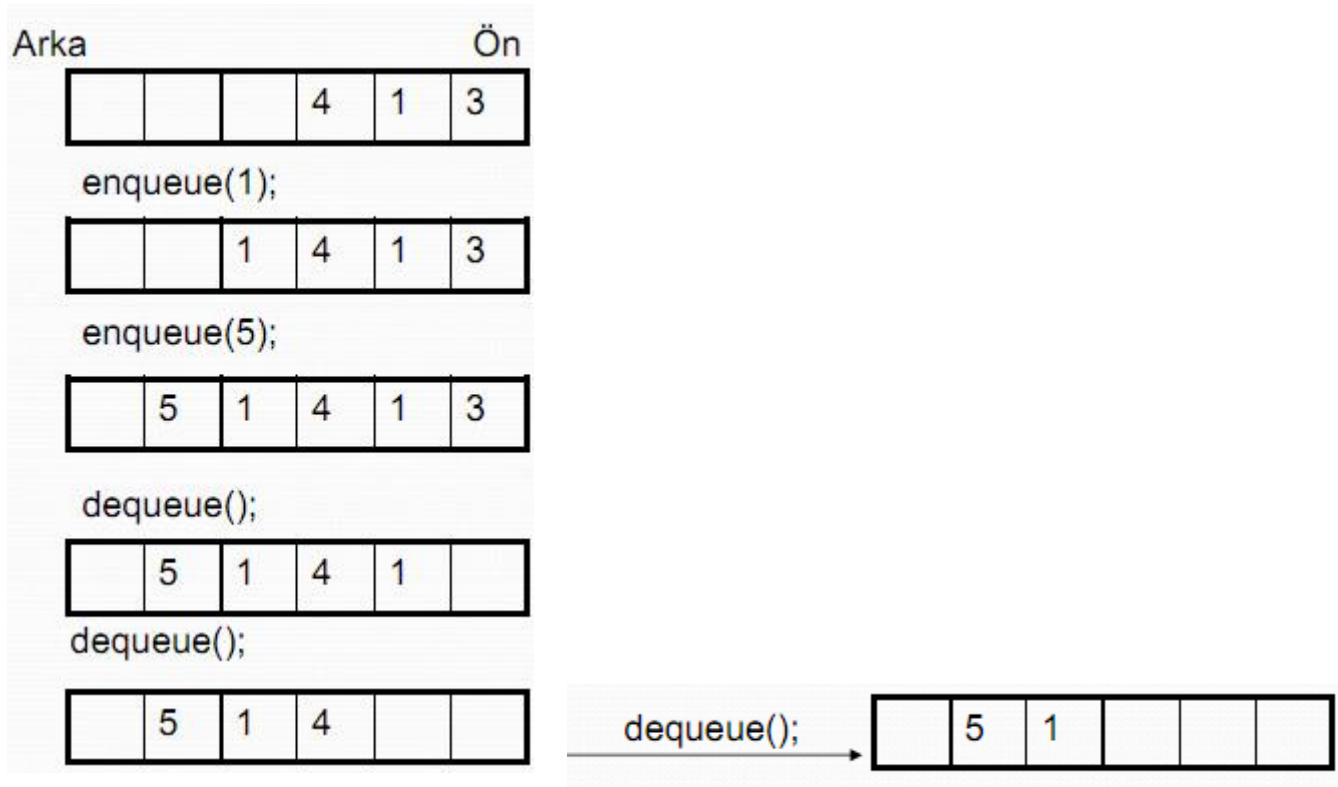
- Yardımcı kuyruk işlemleri:

- `object front()` (getHead/getFront): kuyruk başındaki nesneyi kuyruktan çıkarmadan geri döndürür.
- `integer size()`: kuyrukta saklanan nesne sayısını geri döner
- `boolean isEmpty()`: Kuyrukta nesne olup olmadığını kontrol eder.

- İstisnai Durumlar (Exceptions)

- Boş bir kuyruktan çıkarma işlemi yapılmak istendiğinde veya ilk nesne geri döndürülmek istendiğinde **EmptyQueueException** oluşur.

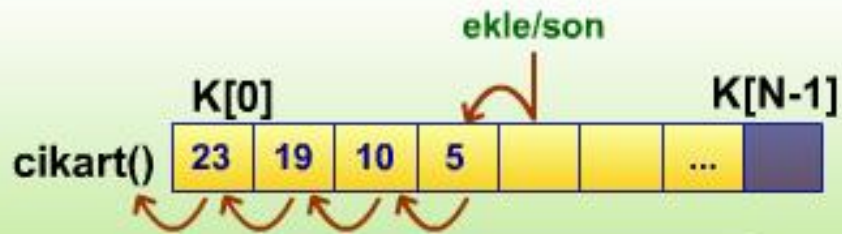
Kuyruk Ekleme/Çıkarma



Kuyruk Tasarımı

- Kuyruk tasarımı çeşitli şekillerde gerçekleştirilebilir. Biri, belki de en yalın olanı, bir dizi ve bir indis değişkeni kullanılmasıdır.
- Dizi gözlerinde yığına atılan veriler tutulurken indis değişkeni kuyruğa eklenen son veriyi işaret eder.
- Kuyruktan alma/çıkartma işlemi her zaman için dizinin başı olan 0 indisli gözden yapılır.
- Kuyruk tasarımı için, genel olarak, üç değişik çözüm şekli vardır:
 - **Dizi Üzerinde Kaydırmalı (QueueAsArray) (Bir indis Değişkenli)**
 - **Dizi Üzerinde Çevrimsel (QueueAsCircularArray) (İki indis Değişkenli)**
 - **Bağlantılı Liste (QueueAsLinkedList) ile**

Kuyruk Tasarımı



a) Kaydırmalı kuyruk



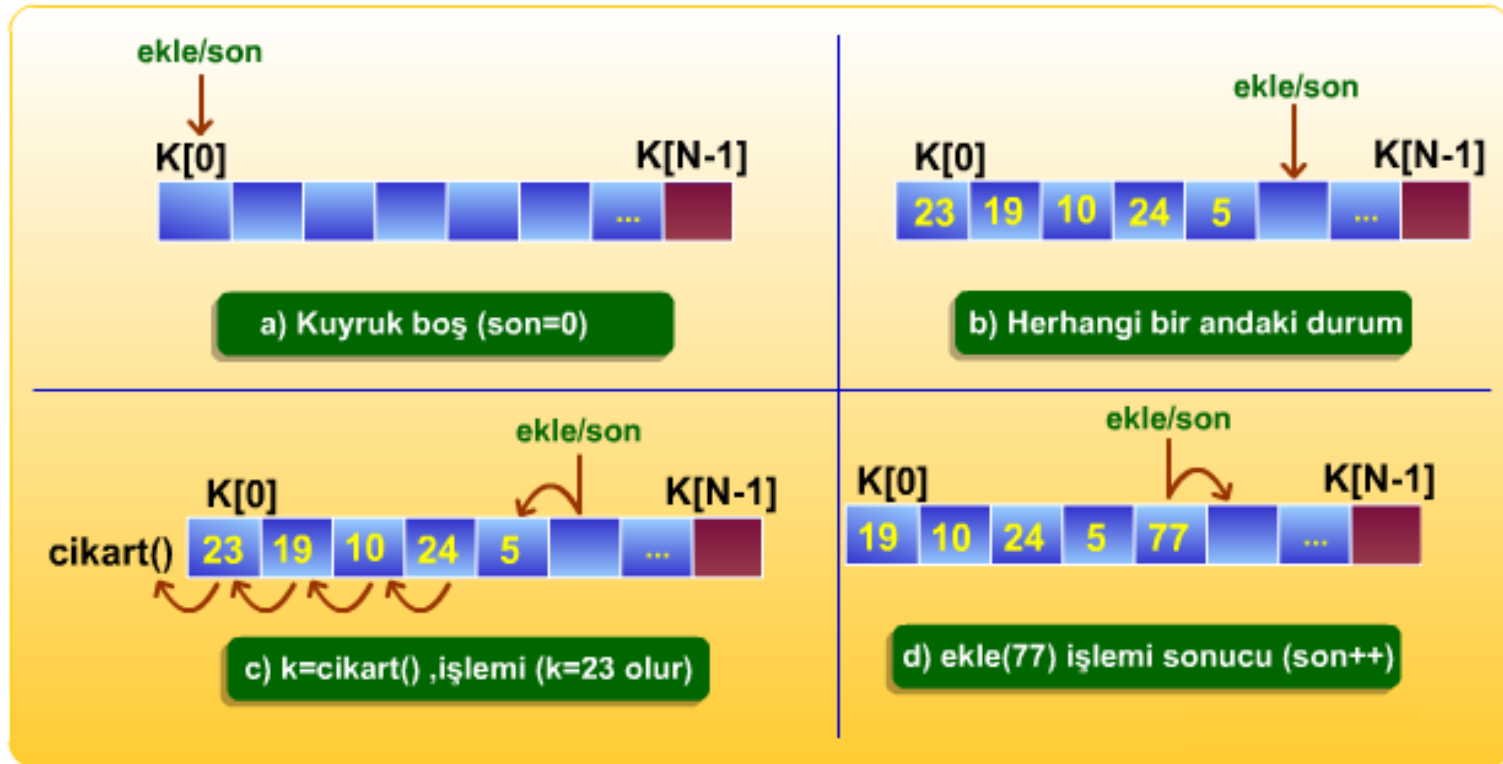
c) Bağlantılı listeli kuyruk



b) Çevrimsel kuyruk

Dizi Üzerinde Kaydırmalı Kuyruk

- N uzunluktaki bir dizi üzerinde kaydırmalı kuyruk yapısının davranışı şekilde gösterilmiştir;



Dizi Üzerinde Kaydırmalı Kuyruk

- a)'da kuyruğun boş hali,
- b)'de ise herhangi bir andaki tipik hali görülmektedir. Kuyruktan çıkartma işlemi,
- c)'de görüldüğü gibi dizinin ilk gözünden, yani $K[0]$ elemanı üzerinden yapılır; eğer kuyrukta birden fazla veri varsa, geride olanlarda bir öne kaydırılır.
- Kuyruğa ekleme işlemi çıkartma işleminden daha az maliyetle yapılır; eklenecek veri doğrudan ekle/son adlı indis değişkenin gösterdiği bir sonraki göze yapılır.

Dizi Üzerinde Kaydırmalı Kuyruk Kaba Kodu

Dizi üzerinde kaydırmalı kuyruk-ekleme işlemi kaba-kodu

```
if(Kuyrukta yer yoksa)  
    Kuyruk dolu mesajını yaz ve EKES değerini gönder;  
else  
    son'u bir sonraki göz için arttır;  
    Veriyi K kuyruğunda son ile gösterilen göze ekle;  
}
```

Dizi üzerinde kaydırmalı kuyruk-çıkartma işlemi kaba-kodu

```
if(Kuyrukta veri yoksa)  
    Kuyruk boş mesajını yaz ve EKES değerini gönder;  
else  
    K kuyruğundaki K[0] verisi al;  
    Kuyruktaki verileri öne kaydır, K[0]'i K[1], K[1]'i K[2]...K[son-2]'i K[son-1].  
    son'u bir önceki göz için azalt;  
    Veriyi gönder;  
}
```

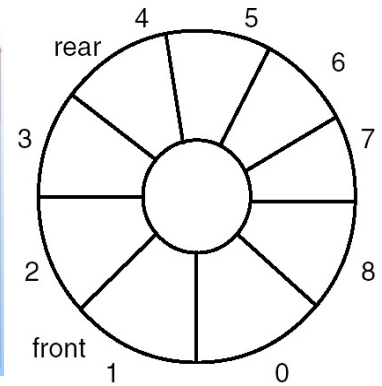
Dizi Üzerinde Çevrimsel Kuyruk

- Bu çözüm şeklinde dizi elemanlarına erişim doğrusal değil de çevrimsel yapılır; yani dizinin son elemanına ulaşıldığında bir sonraki göz dizinin ilk elemanı olacak şekilde indis değeri hesaplanır.
- Böylece kuyruk için tanımlanan dizi üzerinde sanki bir halka bellek varmış gibi dolaşılır.
- Kuyruktaki sıradaki eleman alındığında bu işaretçinin değeri bir sonraki gözü gösterecek biçimde arttırılır.
- Arttırma işleminin de, dizinin sonuna gelindiğinde başına gidecek biçimde çevrimsel olması sağlanır.
- Aşağıda dizi üzerinde çevrimsel kuyruk modeli için ekleme ve çıkartma işlemleri kaba-kodları verilmiştir:

Dizi Üzerinde Çevrimsel Kuyruk

Dizi üzerinde çevrimsel kuyruk - ekleme işlemi kaba-kodu

```
if(Kuyrukta yer yoksa)
    Kuyruk dolu mesajını yaz ve EKES değerini gönder;
else
    son'u çevrimsel şekilde bir sonraki göz için ayarla;
    Veriyi K kuyruğuna ekle;
}
```



- Yukarıda görüldüğü gibi ilk önce ekleme yapılması için kuyrukta yer olup olmadığına bakılmakta, yer varsa son adlı ekleme sayacı çevrimsel erişime imkan verecek şekilde arttırılmakta ve işaret ettiği yere veri koyulmaktadır.

Dizi Üzerinde Çevrimsel Kuyruk

- Aşağıda ise alma işlemi kaba kodu verilmiştir. Görüleceği gibi ilk önce kuyrukta veri olup olmadığı sınanmakta ve veri varsa ilk adlı işaretçinin gösterdiği gözdeki veri alınıyor, ilk adlı işaretçi çevrimsel olarak sıradaki veriyi gösterecek şekilde ayarlanıyor ve veri gönderiliyor.

Dizi üzerinde çevrimsel kuyruk - çıkartma işlemi kaba-kodu

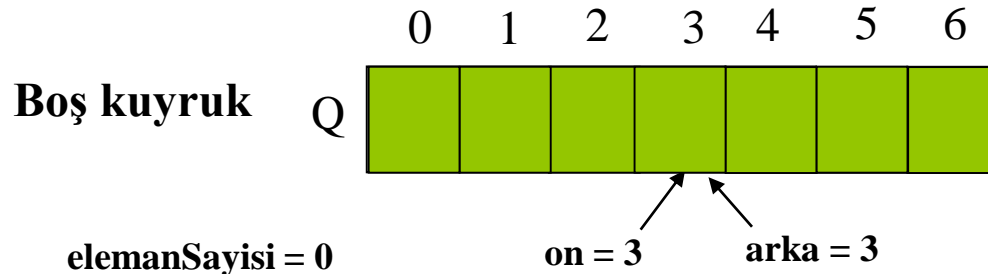
```
if(Kuyrukta yer yoksa)
    Kuyruk boş mesajını yaz ve EKES değerini gönder;
else
    K kuyruğundan ilk indisli yerdeki veriyi al;
    ilk'i çevrimsel şekilde bir sonraki göz için ayarla;
    Kuyruktan alınan veriyi gönder;
}
```

Dizi Kullanılarak Gerçekleştirim

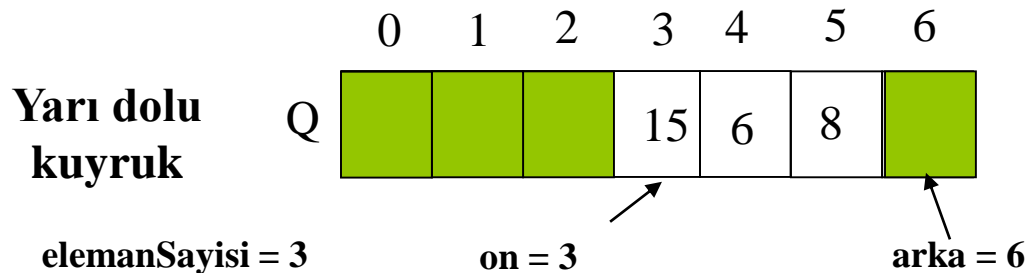
- **N** boyutlu bir dizi kullanılır.
- **ön** kuyruğun ilk elemanını tutar. Dizide ilk elemanın kaçınıcı indisten başlayacağını belirtir.
- **arka** kuyrukta son elemandan sonraki ilk boşluğu tutar.
- **elemanSayisi** kuyruktaki eleman sayısını tutar.
- **Boş kuyruk** eleman sayısının sıfır olduğu durumdur.
- **Dolu kuyruk** eleman sayısının N 'ye eşit olduğu durumdur.

Dizi Kullanarak Gerçekleştirim

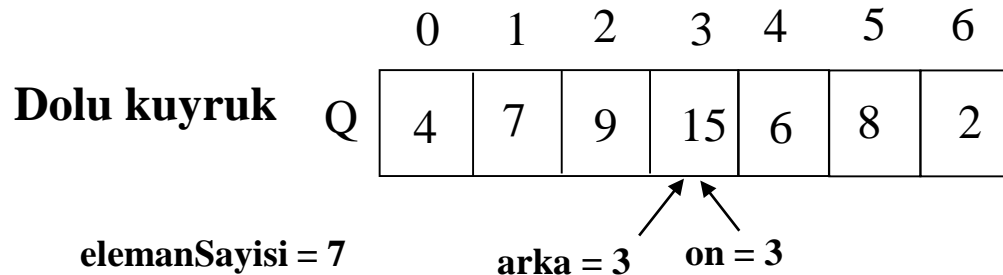
- Kuyruğu N boyutlu bir dizi (`int K[N]`) ve 3 değişken (`int on`, `int arka`, `int elemanSayisi`) ile gerçekleştirebiliriz.



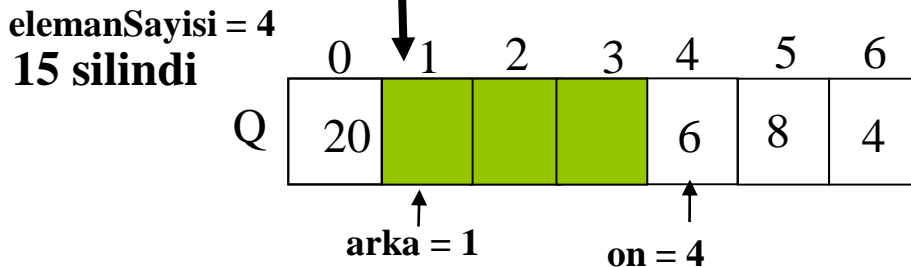
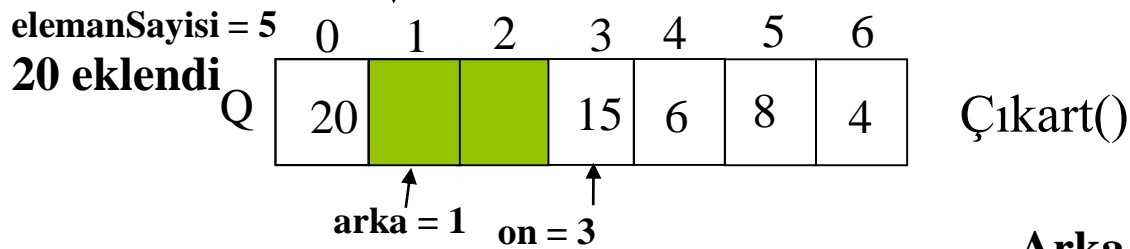
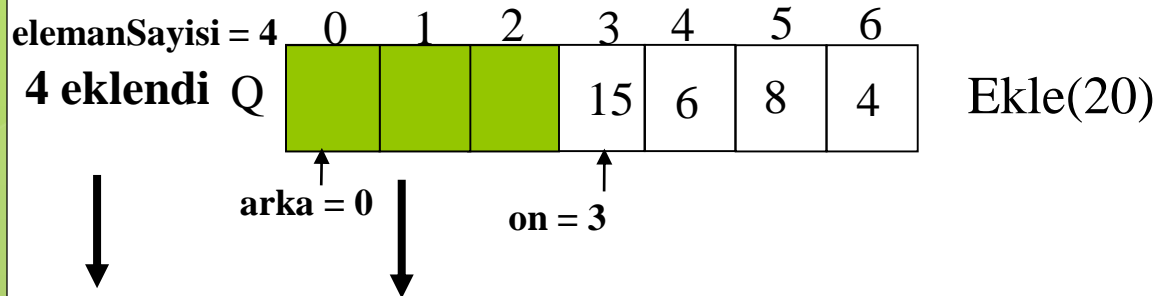
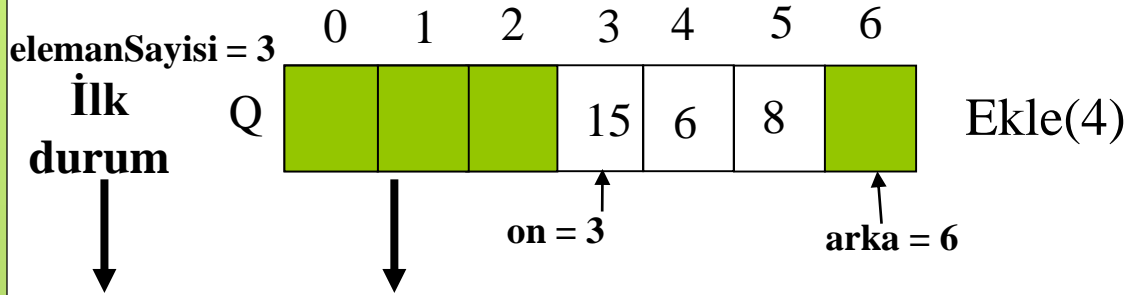
- on ve arka birbirlerine eşit ve elemanSayisi = 0 → Boş kuyruk



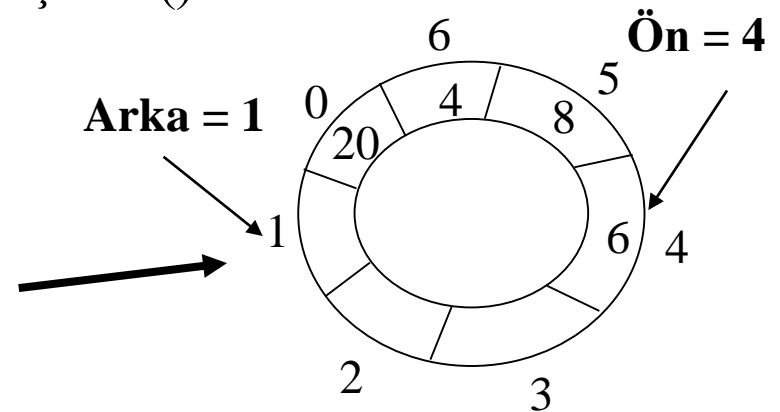
- on kuyruktaki ilk elemanı tutar
- arka son elemandan sonraki ilk boş yeri tutar.



- on ve arka birbirlerine eşit ve elemanSayisi=7 → Dolu kuyruk.



Kuyruğun kavramsal görünümü:
Döngüsel dizi



Dizi Üzerinde Çevrimsel Kuyruk- Java

```
public class kuyruk {  
  
    private int K[N];      // kuyruk elemanlarını tutan dizi  
    private int on;        // kuyruğun başı  
    private int arka;      // kuyruğun sonu  
    private int elemanSayisi; // kuyruktaki eleman sayısı  
  
    public kuyruk() ;  
  
    public boolean bosmu() ;  
    public boolean dolumu() ;  
    public int ekle(int item) ;  
    public int cikart() ;  
};
```

Dizi Üzerinde Çevrimisel Kuyruk- Java

```
// yapıcı yordam
public Kuyruk(){
    on = arka = elemanSayisi = 0;
}

// Kuyruk boşsa true döndür
public boolean bosmu(){
    return elemanSayisi == 0;
}

// Kuyruk doluysa true döndür
public boolean dolumu(){
    return elemanSayisi == N;
}
```

Dizi Üzerinde Çevrimsel Kuyruk- Java

```
// Kuyruğa yeni bir eleman ekle
// Başarılı olursa 0 başarısız olursa -1 döndür
public int ekle(int yeni){
    if (dolumu()){
        System.out.println("Kuyruk dolu.");
        return -1;
    }

    K[arka] = yeni;    // Yeni elemanı sona koy
    arka++; if (arka == N) arka = 0;
    elemanSayisi++;

    return 0;
}
```


Dizi Üzerinde Çevrimsel Kuyruk- Java

```
// Kuyruğun önündeki elemanı çıkart ve döndür.  
// Kuyruk boşsa -1 döndür  
public int cikart(){  
    int id = -1;  
  
    if (bosmu()){  
        System.out.println("Kuyruk boş");  
        return -1;  
    }  
  
    id = on;    // ilk elemanın olduğu yer  
    on++; if (on == N) on = 0;  
    elemanSayisi--;  
  
    return K[id]; // elemanı döndür  
}
```

```
public static void main(String[] args){
    Kuyruk k;

    if (k.bosmu())
        System.out.println("Kuyruk boş");

    k.ekle(49);
    k.ekle(23);

    System.out.println("Kuyruğun önü: "+ k.cikart());
    k.ekle(44);
    k.ekle(22);

    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());

    if (k.bosmu())
        System.out.println("Kuyruk boş");
}
```

Dizi Gerçekleştirimi: Son Söz

- Kuyruğu N-1 elemanlı bir dizi (`int K[N]`) ve 2 tane değişken (`int on`, `int arka`) ile gerçekleştirebiliriz.
- Aşağıdakileri nasıl tanımlayabileceğimizi düşünün.
 - Boş kuyruk
 - Dolu kuyruk

Boş kuyruk Q



`on = 3`

`arka = 3`

- On ve arka birbirine eşit ise boş kuyruk

Dolu kuyruk Q



`arka = 2`

`on = 3`

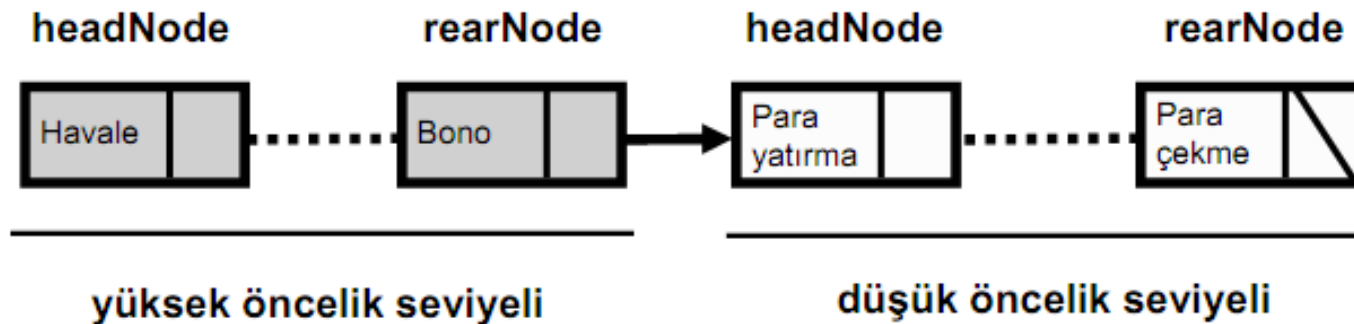
- On ve arka arasında 1 tane boşluk varsa dolu kuyruk

Öncelikli Kuyruklar (Priority Queues)

- Öncelikli kuyruk uygulamasında kuyruğa girecek verilerin veya nesnelerin birer öncelik değeri vardır ve ekleme bu öncelik değerine bakılarak yapılır.
- Eğer eklenecek verinin önceliği en küçük ise, yani en önceliksiz ise, doğrudan kuyruğun sonuna eklenir; diğer durumlarda kuyruk üzerinde arama yapılarak kendi önceliği içerisinde sona eklenir.
- Örneğin bazı çok prosesli işletim sistemlerinde bir proses işleme kuyruğu vardır ve yürütülme için hazır olan proseslerin listesi bir kuyrukta tutulur. Eğer proseslerin birbirlerine göre herhangi bir önceliği yoksa, prosesler kuyruğa ekleniş sırasına göre işlemci üzerinde yürütülürler; ilk gelen proses ilk yürütülür. Ancak, proseslerin birbirlerine göre bir önceliği varsa, yani aynı anda beklemekte olan proseslerden bazıları daha sonra kuyruğa eklenmiş olsalar dahi diğerlerine göre ivedi olarak yürütülmesi gerekebilir.
- Öncelikli kuyruk oluşturmak için bağlantılı listeyle, kümeleme ağacına dayalı ve araya sokma sıralaması yöntemiyle sağlanan çözümler olmaktadır.

Öncelikli Kuyruklar (Priority Queues)

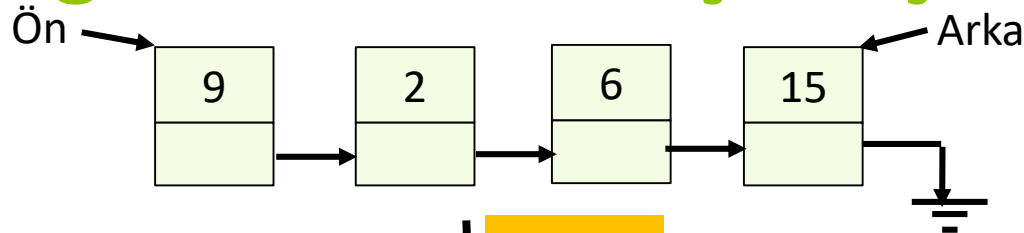
- Kuyruktaki işler kendi arasında önceliklerine göre seviyelendirilebilir.
- Her öncelik seviyesinin headNode ve rearNode'u vardır.
- Elaman alınırken en yüksek seviyeye ait elemanların ilk geleni öncelikli alınır.
- Yüksek öncelik seviyeli grubun son elemanı düşük öncelik seviyeli grubun ilk elemanından daha önceliklidir.



Bağlantılı Liste ile Kuyruk Tasarımı

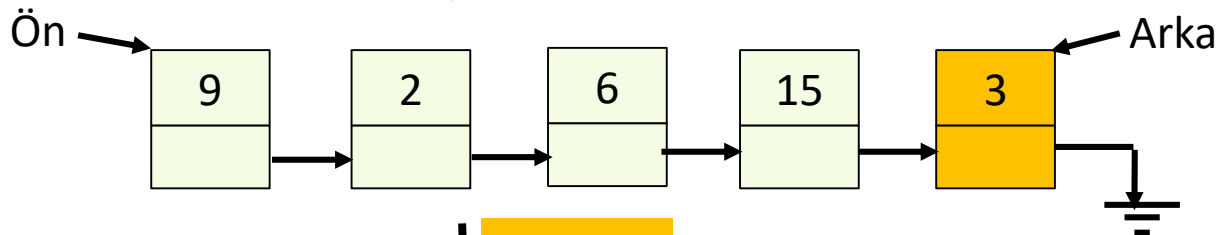
- Ekleme (add) işlemi, son adresindeki verinin arkasına eklenir.
- Çıkarma (get) işlemi, ilk verinin alınması ve kuyruktan çıkarılması ile yapılır. bellekte yer olduğu sürece kuyruk uzayabilir.

Bağlantılı Liste Gerçekleştirimi



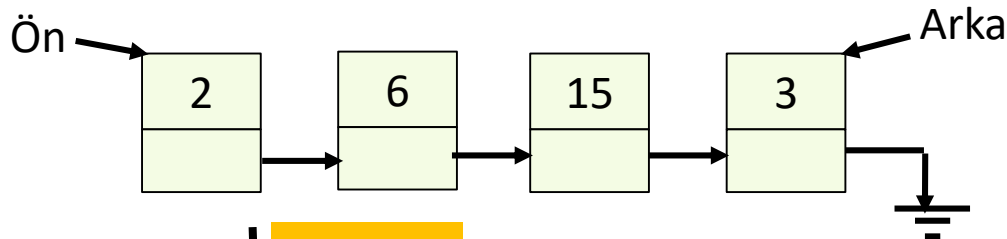
Başlangıç durumu

↓ ekle(3)



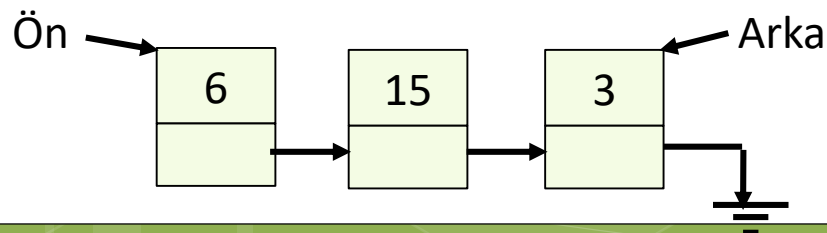
3 eklendikten sonra

↓ cikart()



9 çıkartıldıktan sonra

↓ cikart()



2 çıkartıldıktan sonra

```
public class KuyrukDugumu {  
    public int eleman;  
    public KuyrukDugumu sonraki;  
  
    public KuyrukDugumu(int e){  
        eleman = e; sonraki = null;  
    }  
}  
  
public class Kuyruk{  
    private KuyrukDugumu on; // Kuyruğun önu  
    private KuyrukDugumu arka; // Kuyruğun arkası  
  
    public Kuyruk() {  
        on = arka = null;  
    }  
    public boolean bosmu() { ... }  
    public void ekle(int eleman) { ... }  
    public int cikart() { ... }  
}
```


Haftalık Ödev

- 1- Banka kuyruğu örneğini ele alıp Banka işlemlerini kendi içerisinde üç öncelik grubuna ayırınız. Her yeni gelen kişiyi iş seçimine göre otomatik olarak ait olduğu grubun en sonuna ekleyiniz (enqueue). Her eleman alımında (dequeue) ise kuyruktaki en yüksek seviyeli grubun ilk elemanını alınız. Kuyruğu Dairesel bağlı listeler ile gerçekleştiriniz.

Haftalık Ödev

- 2- Aşağıda verilen uygulamayı C# veya Javada yapınız

Uygulama programı (Stacks ve Queues):

The application window is titled "Stacks and Queues" and is divided into two main sections: Stack (Kitap Yığını) and Queue (Banka Kuyruğu).

Stack (Kitap Yığını) Section:


- Üst (Top):** A list box containing the following items: Wireless Internet, C# How to Program, Algorithms, Automata, Computer Organization, and Data Structures.
- Kitap Adı (Push):** A text input field containing "Wireless Network" and a "Push" button.
- Kitap Adı (Üst):** A text input field containing "Wireless Network" and a "Pop" button.
- Max Size:** A numeric input field set to 10.
- Stack Size = 6:** A label indicating the current size of the stack.
- Alt (Bottom):** A button labeled "Örnek Stack (Maze - Labirent)".

Queue (Banka Kuyruğu) Section:


- Ad - İş süresi - Bekleme süresi:** A label for the queue entries.
- Baş (Front):** A list box containing the following entries: Dilek - 5 - 0, Mustafa - 3 - 5, Mehmet - 8 - 8, Fatma - 10 - 16, Füsün - 2 - 26, and Fethi - 7 - 28.
- Kişi Adı (Enqueue):** A text input field containing "Fethi" and an "Enqueue" button.
- İş süresi (dk):** A numeric input field set to 7.
- Kişi Adı (Baştaki):** A text input field and a "Dequeue" button.
- Max Size:** A numeric input field set to 10.
- Queue Size = 6:** A label indicating the current size of the queue.
- Son (Rear):** A label indicating the rear of the queue.

Footer:

Yrd.Doç.Dr. M. Ali Akcayol
Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü



Örnek Uygulamalar



Örnek Uygulamalar Java-Yığıt

Yığıt (stack) yapısının dinamik dizi ile gerçekleştirimi

- Stack.java
- public interface Stack<T>
- { public boolean empty(); // yığıt boş mu test eder
- public T top(); // tepedeki elemanı ver (silme)
- public T pop(); // tepedeki elemanı sil ve ver
- public void push(T item); // item'i yığıtın tepesine ekle
- public void clear(); // Yığıtı boşalt }

Yığıt (stack) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayStack.java`
- `public class ArrayStack<T> implements Stack<T>`
- `{`
- `private static final int DEFAULT_SIZE = 10;`
- `private T[] array; // yığıt elemanlarını tutan dizi`
- `private int top; // son eklenen elemanın indisi`
- `public ArrayStack() // kurucu sınıf`
- `{ array = (T[]) new Object[DEFAULT_SIZE]; top = -1; }`
- `public boolean empty() // yığıt boş mu test eder`
- `{ return (top== -1); }`

Yığıt (stack) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayStack.java`
- `public T top() // tepedeki elemanı ver (silme)`
- `{ if (empty()) return null;`
- `return array[top];`
- `}`
- `public T pop() // tepedeki elemanı sil ve ver`
- `{ if (empty()) return null;`
- `return array[top--];`
- `}`

Yığıt (stack) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayStack.java`
- `public void push(T item) // item'i yığıtın tepesine ekle`
- `{ if (top+1==array.length)`
- `{ T[] newArray = (T[]) new Object[array.length * 2];`
- `for (int i=0; i<array.length; i++)`
- `newArray[i] = array[i];`
- `array = newArray;`
- `}`
- `array[++top] = item;`
- `}`
- `public void clear() // Yığıtı boşalt`
- `{ top = -1; }`
- `}`

Yığıt (stack) yapısının dinamik dizi ile gerçekleştirimi

- TestArrayStack.java
- public class TestArrayStack
- {
- public static void main(String[] args)
- { Stack<String> yigit = new ArrayStack<String>();
- yigit.push("a"); yigit.push("b");
- System.out.println("Yigittaki son eleman: " + yigit.top());
- yigit.push("c");
- String s = yigit.pop();
- System.out.println("Yigittan silinen eleman: " + s);
- yigit.push("d");
- }
- }

Yığıt (stack) yapısının ArrayList ile gerçekleştirimi

- Stack.java // Daha önce verildi
- ArrayListStack.java
- import java.util.*;
- public class ArrayListStack<T> implements Stack<T>
- { private ArrayList<T> array; // yığıt elemanlarını tutan dizi
- public ArrayListStack() // kurucu sınıf
- { array = new ArrayList<T>(); }
- public boolean empty() // yığıt boş mu test eder
- { return array.size()==0; }
-
- public T top() // tepedeki elemanı ver (silme)
- { if (empty()) return null;
- return array.get(array.size()-1);
- }
-

Yığıt (stack) yapısının ArrayList ile gerçekleştirimi

- ArrayListStack.java
- `public T pop() // tepedeki elemanı sil ve ver`
- `{`
- `if (empty()) return null;`
- `return array.remove(array.size()-1);`
- `}`
- `public void push(T item) // item'i yığıtın tepesine ekle`
- `{ array.add(item); }`
- `public void clear() // Yığıtı boşalt`
- `{ array.clear(); }`
- `}`

Yığıt (stack) yapısının ArrayList ile gerçekleştirimi

- TestArrayListStack.java
- public class TestArrayListStack
- {
- public static void main(String[] args)
- {
- Stack<String> yigit = new ArrayStack<String>();
- yigit.push("a");
- yigit.push("b");
- System.out.println("Yigittaki son eleman: " + yigit.top());
- yigit.push("c");
- String s = yigit.pop();
- System.out.println("Yigittan silinen eleman: " + s);
- yigit.push("d");
- }
- }

Yığıt (stack) yapısının bağlantılı liste ile gerçekleştirimi

- Stack.java // Daha önce verildi
- LinkedListStack.java
- public class LinkedListStack<T> implements Stack<T>
- { private Node<T> top = null; // yığıtın tepesindeki eleman
- public boolean empty() // yığıt boş mu test eder
- { return top==null; }
- public T top() // tepedeki elemanı ver (silme)
- { if (empty()) return null;
- return top.data;
- }
- public T pop() // tepedeki elemanı sil ve ver
- { if (empty()) return null;
- T temp = top.data;
- top = top.next;
- return temp;
- }

Yığın (stack) yapısının bağlantılı liste ile gerçekleştirimi

- LinkedListStack.java
- public void push(T item) // item'i yığının tepesine ekle
- { Node<T> newNode = new Node<T>();
- newNode.data = item;
- newNode.next = top;
- top = newNode;
- }
- public void clear() // Yığını boşalt
- { top = null; }
- // Yığın elemanlarını tutan liste elemanı (inner class)
- class Node<T>
- { public T data;
- public Node<T> next;
- }
- }

Yığıt (stack) yapısının bağlantılı liste ile gerçekleştirimi


- TestLinkedListStack.java
- public class TestLinkedListStack
- { public static void main(String[] args)
- { Stack<Character> yigit = new LinkedListStack<Character>();
- yigit.push('a');
- yigit.push('b');
- yigit.push('c');
- System.out.println("Yigittaki elemanlar cikariliyor: ");
- while (!yigit.empty())
- {
- System.out.println(yigit.pop());
- }
- }
- }

Hanoi Kuleleri– Özyinelemeli Çözüm- C#

- `using System;`
- `using System.Collections.Generic;`
- `using System.Text;`
- `namespace Hanoi`
- `{ class Program`
- `{ static void Main(string[] args)`
- `{ int x; char from='A', to='B', help='C';`
- `do {`
- `try`
- `{ Console.Write(" input number of disk: "); x = Int32.Parse(Console.ReadLine()); }`
- `catch (FormatException e) { x = -10; }`
- `} while(x== -10 || x>10);`

Hanoi Kuleleri– Özyinelemeli Çözüm- C#

- `Console.WriteLine("\n from = A, to = B, help = C\n");`
- `hanoi(x, from, to, help); Console.Read();`
- `}`
- `static void hanoi(int x, char from, char to, char help)`
- `{ if (x > 0)`
- `{ hanoi(x - 1, from, help, to);`
- `move(x, from, to);`
- `hanoi(x - 1, help, to, from);`
- `}`
- `}`
- `static void move(int x, char from, char to)`
- `{ Console.WriteLine(" move disk "+x+" from "+from+" to "+to); }`
- `}`
- `}`



Örnek Uygulamalar **JAVA-Kuyruk**

Kuyruk Tasarımı ve Kullanımı -Java

- `class Kuyruk`
- `{ private int boyut; private int[] kuyrukDizi;`
- `private int bas; private int son; private int elemanSayisi;`
-
- `public Kuyruk(int s)`
- `{ boyut = s; kuyrukDizi = new int[boyut];`
- `bas = 0; son = -1; elemanSayisi = 0;`
- `}`
- `public void ekle(int j) // Kuyrugun sonuna eleman ekler`
- `{`
- `if (son==boyut-1) son = -1;`
- `kuyrukDizi[++son] = j; elemanSayisi++;`
- `}`

Kuyruk Tasarımı ve Kullanımı -Java

- `public int cikar()`
- `{`
- `int temp = kuyrukDizi[bas++];`
- `if(bas==boyut) bas=0;`
- `elemanSayisi--; return temp;`
- `}`
- `public boolean bosMu() { return(elemanSayisi==0); }`
- `}`

- `public class KuyrukTest {`
- `public static void main(String args[])`
- `{ Kuyruk k = new Kuyruk(25); k.ekle(1); k.ekle(2);`
- `System.out.println(k.cikar()); k.ekle(3);`
- `for(int i=4; i<10; ++i) k.ekle(i);`
- `while(!k.bosMu()) System.out.println(k.cikar());`
- `}`
- `}`

Kuyruk (queue) yapısının dinamik dizi ile gerçekleştirimi

- Queue.java
- public interface Queue<T>
- {
- public boolean empty(); // kuyruk boş mu test eder
- public T getFront(); // kuyruğun önündeki elemanı ver (silme)
- public T dequeue(); // kuyruğun önündeki elemanı sil ve ver
- public void enqueue(T item); // item'i kuyruğun arkasına ekle
- public void clear(); // kuyruğu boşalt
- }

Kuyruk (queue) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayQueue.java`
- `public class ArrayQueue<T> implements Queue<T>`
- `{ private static final int DEFAULT_SIZE = 10;`
- `private T[] array; // kuyruk elemanlarını tutan dizi`
- `private int size; // kuyruktaki eleman sayısı`
- `private int front; // en önce eklenen elemanın indisi`
- `private int back; // en son eklenen elemanın indisi`
- `public ArrayQueue() // kurucu sınıf`
- `{ array = (T[]) new Object[DEFAULT_SIZE]; clear(); }`
- `public boolean empty() // kuyruk boş mu test eder`
- `{ return size==0; }`

Kuyruk (queue) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayQueue.java`
- `public T getFront() // kuyruğun önündeki elemanı ver (silme)`
- `{`
- `if (empty()) return null;`
- `return array[front];`
- `}`
- `public T dequeue() // kuyruğun önündeki elemanı sil ve ver`
- `{`
- `if (empty()) return null;`
- `size--;`
- `T temp = array[front];`
- `front = increment(front);`
- `return temp;`
- `}`

Kuyruk (queue) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayQueue.java`
- `public void enqueue(T item) // item'i kuyruğun arkasına ekle`
- `{`
- `if (size==array.length) // kuyruk dolu`
- `{`
- `T[] newArray = (T[]) new Object[array.length * 2];`
- `for (int i=0; i<size; i++, front=increment(front))`
- `newArray[i] = array[front];`
- `array = newArray;`
- `}`
- `back = increment(back);`
- `array[back] = item;`
- `size++;`
- `}`
-

Kuyruk (queue) yapısının dinamik dizi ile gerçekleştirimi

- `ArrayQueue.java`
- `private int increment(int i)`
- `{`
- `i++;`
- `if (i==array.length) i=0;`
- `return i;`
- `}`
- `public void clear() // kuyrukı boşalt`
- `{`
- `size = 0;`
- `front = 0;`
- `back = -1;`
- `}`
- `}`

Kuyruk (queue) yapısının dinamik dizi ile gerçekleştirimi

- TestArrayQueue.java
- public class TestArrayQueue
- { public static void main(String[] args)
- { Queue<Integer> kuyruk = new ArrayQueue<Integer>(); int say=1;
- for (int i=1; i<=15; i++)
- kuyruk.enqueue(i);
- for (int i=1; i<=10; i++)
- System.out.println((say++) + ".kuyruk elemani: " + kuyruk.dequeue());
- for (int i=11; i<=25; i++)
- kuyruk.enqueue(i);
- say=1;
- while (!kuyruk.empty())
- System.out.println((say++) + ".kuyruk elemani: " + kuyruk.dequeue());
- }
- }

Kuyruk (queue) yapısının ArrayList ile gerçekleştirimi

- Queue.java // daha önce verilmişti
- ArrayListQueue.java
- import java.util.ArrayList;
- public class ArrayListQueue<T> implements Queue<T>
- { private ArrayList<T> array; // kuyruk elemanlarını tutan dizi
- public ArrayListQueue() // kurucu sınıf
- { array = new ArrayList<T>(); }
-
- public boolean empty() // kuyruk boş mu test eder
- { return array.size()==0; }
- public T getFront() // kuyruğun önündeki elemanı ver (silme)
- { if (empty()) return null;
- return array.get(0);
- }

Kuyruk (queue) yapısının ArrayList ile gerçekleştirimi

- ArrayListQueue.java
- `public T dequeue() // kuyruğun önündeki elemanı sil ve ver`
- `{`
- `if (empty()) return null;`
- `return array.remove(0);`
- `}`
- `public void enqueue(T item) // item'i kuyruğun arkasına ekle`
- `{`
- `array.add(item);`
- `}`
- `public void clear() // kuyruğu boşalt`
- `{`
- `array.clear();`
- `}`
- `}`

Kuyruk (queue) yapısının ArrayList ile gerçekleştirimi

- TestArrayListQueue.java
- public class TestArrayListQueue
- {
- public static void main(String[] args)
- {
- Queue<Integer> kuyruk = new ArrayListQueue<Integer>();
-
- for (int i=1; i<=25; i++)
- kuyruk.enqueue(i);
-
- int say=1;
- while (!kuyruk.empty())
- System.out.println((say++) + ".kuyruk elemani: " + kuyruk.dequeue());
- }
- }

Kuyruk (queue) yapısının bağlantılı liste ile gerçekleştirimi

- Queue.java // daha önce verilmişti
- LinkedListQueue.java
- public class LinkedListQueue<T> implements Queue<T>
- {
- private Node<T> front, back; // kuyruk başı ve sonu
- public LinkedListQueue() // kurucu sınıf
- { clear(); }
-
- public boolean empty() // kuyruk boş mu test eder
- { return front==null; }
-
- public T getFront() // kuyruğun önündeki elemanı ver (silme)
- { if (empty()) return null;
- return front.data;
- }

Kuyruk (queue) yapısının bağlantılı liste ile gerçekleştirimi

- `public T dequeue() // kuyruğun önündeki elemanı sil ve ver`
- `{ if (empty()) return null;`
- `T temp = front.data; front = front.next; return temp;`
- `}`
- `public void enqueue(T item) // item'i kuyruğun arkasına ekle`
- `{ if (empty()) front = back = new Node<T>(item, null);`
- `else back = back.next = new Node<T>(item, null);`
- `}`
- `public void clear() { front = back = null; } // kuyruğu boşalt`
- `// Kuyruk elemanlarını tutan liste elemanı (inner class)`
- `private class Node<T>`
- `{ private T data; private Node<T> next;`
- `public Node(T data, Node next) { this.data = data; this.next = next; }`
- `}`
- `}`

Kuyruk (queue) yapısının bağlantılı liste ile gerçekleştirimi

- TestLinkedListQueue.java
- public class TestLinkedListQueue
- { public static void main(String[] args)
- { Queue<Integer> kuyruk = new ArrayListQueue<Integer>();
- for (int j=1, i=1; i<=20; i++)
- { if ((int)(Math.random()*2) == 1)
- { System.out.println("Kuyruga ekle: " + j); kuyruk.enqueue(j++); }
- else
- { System.out.println("Kuyrugun onundeki "
- + kuyruk.dequeue() + " cikarildi..");
- }
- }
- }
- }

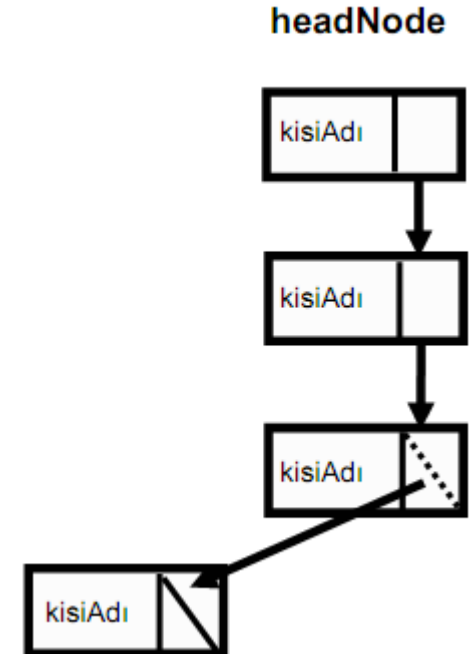
Bağlı Liste ile Kuyruk gerçekleştirimi C#

- **// Queue oluşturma :**
- `class queueNodeC {`
- `public string kisiAdi; public queueNodeC sonraki;`
- `public queueNodeC(string kisiAdi) { this.kisiAdi = kisiAdi; }`
- `}`
- `class queueC`
- `{`
- `public int size; public queueNodeC headNode;`
- `public queueC(string kisiAdi)`
- `{`
- `this.headNode = new queueNodeC(kisiAdi);`
- `this.headNode.sonraki = headNode; this.size = 0;`
- `}`
- `}`
- `queueC kisiKuyruk = new queueC("");`

Bağlı Liste ile Kuyruk gerçekleştirimi C#

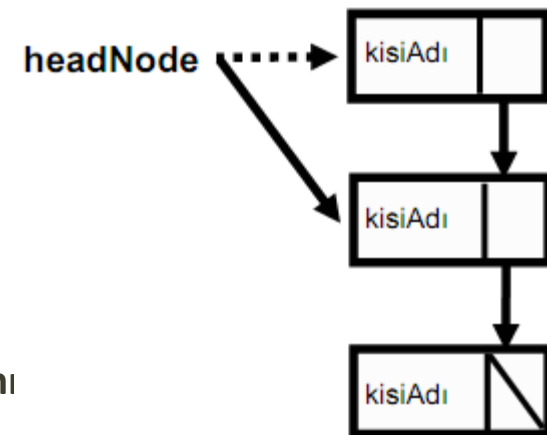
Örnek (C#):

- **/* Queue işlemleri :**
- **boş kuyruk size == 0**
- **eleman sayısı= size**
- **eleman ekleme= enqueue(kisiAdi)**
- **eleman alma= dequeue() */**
- `public void enqueue(kisiAdi)`
- `{`
- `queueNodeC yeniNode = new queueNodeC(kisiAdi);`
- `queueNodeC aktif = kisiKuyruk.headNode;`
- `while (aktif.sonraki != aktif)`
- `{ aktif = aktif.sonraki; }`
- `aktif.sonraki = yeniNode;`
- `yeniNode.sonraki = yeniNode;`
- `kisiKuyruk.size++;`
- `}`



Bağlı Liste ile Kuyruk gerçekleştirimi C#

- `// Queue işlemleri (eleman alma)`
- `public void dequeue()`
- `{`
- `queueNodeC yeniNode = new queueNodeC(" ");`
- `yeniNode = kisiKuyruk.headNode;`
- `kisiKuyruk.headNode = kisiKuyruk.headNode.soni`
- `kisiKuyruk.size--;`
- `}`



Kuyruk oluşturma, ekleme ve silme C++

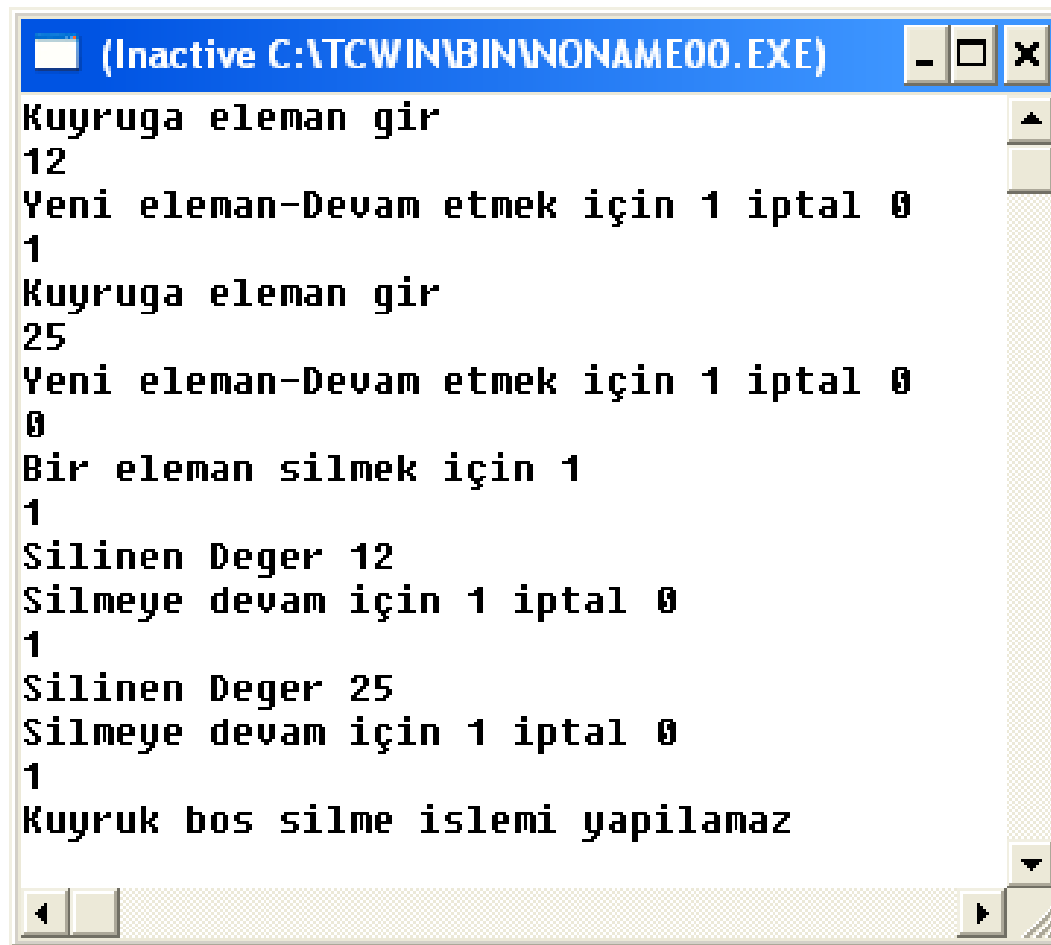
- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#define boyut 10 /*Maksimum kuyruk uzunluğu */`

- `void ekle (int kuyruk[], int *arka, int deger)`
- `{`
- `if(*arka < boyut-1) {`
- `*arka= *arka +1;`
- `kuyruk[*arka] = deger; }`
- `else`
- `{ printf("Kuyruk doldu daha fazla ilave edilemez\n"); }`
- `}`

- `void sil (int kuyruk[], int * arka, int * deger)`
- `{ int i;`
- `if(*arka<0) {printf("Kuyruk bos silme islemi yapilamaz\n");}`
- `*deger = kuyruk[0];`
- `for(i=1;i<=*arka;i++)`
- `{ kuyruk[i-1]=kuyruk[i]; }`
- `*arka=*arka-1;`
- `}`

Kuyruk oluşturma, ekleme ve silme C++

- `main() { int kuyruk[boyut]; int arka, n, deger; arka=(-1);`
- `do {`
- `do { printf("Kuyruga eleman gir\n"); scanf("%d",°er);`
- `ekle(kuyruk,&arka,deger); //arka değeri arttı`
- `printf("Yeni eleman-Devam etmek için 1 iptal 0 \n"); scanf("%d",&n);`
- `} while(n == 1);`
- `printf("Bir eleman silmek için 1\n"); scanf("%d",&n);`
- `while(n == 1) {`
- `sil(kuyruk,&arka,°er);`
- `printf("Silinen Deger %d\n",deger); printf("Silmeye devam için 1 iptal 0\n");`
- `scanf("%d",&n); }`
- `printf("Eleman girmek icin 1 iptal icin 0\n"); scanf("%d",&n);`
- `} while(n == 1); }`



```
(Inactive C:\TCWIN\BIN\WONAME00.EXE)
Kuyruga eleman gir
12
Yeni eleman-Devam etmek için 1 iptal 0
1
Kuyruga eleman gir
25
Yeni eleman-Devam etmek için 1 iptal 0
0
Bir eleman silmek için 1
1
Silinen Deger 12
Silmeye devam için 1 iptal 0
1
Silinen Deger 25
Silmeye devam için 1 iptal 0
1
Kuyruk bos silme islemi yapilamaz
```

Dizi Üzerinde Çevrimsel Kuyruk- C++

- `#include <stdio.h>`
- `#define N 500 /*Maksimum kuyruk uzunluğu */`
- `#include <stdlib.h>`
- `typedef struct kuyrukyapisi`
- `{ int bas; int son ; int sayac; int D[N]; } KUYRUK ;`
-
- `KUYRUK *M;`
- `void baslangic (KUYRUK *K)`
- `{ K->bas=0;`
- `K->sayac=0;`
- `K->son=0;`
- `}`
-

Dizi Üzerinde Çevrimsel Kuyruk- C++

- void ekle(int veri, KUYRUK *K)
- {
- if(K->sayac>N-1) { printf("Kuyruk dolu"); }
- /*Doğrusal erişimli bir diziye çevrimsel erişim yapılabilmesi için dizi indisi son gözden sonra ilk gözü işaret edebilmesi için artık bölme işlemiyle indis=(indis+1)%N yapılır. */
- K->son=(K->son+1)%N;
- K->D[K->son]=veri;
- K->sayac++;
- }
- void silme(KUYRUK *K,int *deger)
- {
- if((K->sayac)<=0) {printf("Kuyruk bos silme islemi yapilamaz\n");}
- *deger = K->D[K->bas]=0;
- K->bas=(K->bas+1)%N;
- K->sayac--;
- }

Dizi Üzerinde Çevrimsel Kuyruk- C++

```

○ main() {
○ int veri,n,deger;
○ M=(KUYRUK *)malloc(sizeof(KUYRUK));
○ baslangic (M);
○ do {
○     do { printf("Kuyruğa eleman gir\n");   scanf("%d",&veri); ekle (veri,M);
○         printf("Yeni eleman-Devam etmek için 1 iptal 0 \n"); scanf("%d",&n);
○     } while(n == 1);
○     printf("Bir eleman silmek için 1\n"); scanf("%d",&n);
○     while( n == 1) { silme (M,&deger);
○         printf("Silinen Deger %d\n",deger); printf("Silmeye devam için 1 iptal 0\n");
○         scanf("%d",&n);    }
○     printf("Eleman girmek için 1 iptal için 0\n");   scanf("%d",&n);
○ } while(n == 1);
○ }

```