# Veri Yapıları

## Kuyruk (QUEUE)

Dr. Sinan TUNCEL

## Kuyruk (Qeueu)

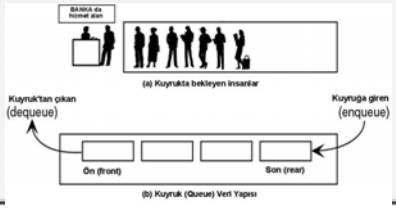
Kuyrugun çalısma mantıgını anlatan gündelik hayatta karsılastıgımız birkaç örnek;

· Bir sinema gisesinden bilet almak için bekleyen insanlar,
· Bir bankamatik'ten para çekmek için bekleyen insanlar,
· Bir durakta otobüse binmek için bekleyen insanlar,

Gerçek yasamda kuyruğa ilk olarak girenler işlemlerini ilk olarak tamamlayıp kuyruktan çıkarlar. Veri yapılarındaki kuyruklar bu tür veri yapılarının simülasyonunda kullanılmaktadır. Ayrıca işlemci, yazıcı, disk gibi kaynaklar üzerindeki işlemlerin yürütülmesinde ve bilgisayar ağlarında paketlerin yönlendirilmesinde de kuyruklardan yararlanılmaktadır.



Kuyruklar, eleman eklemelerin sondan (rear) ve eleman çıkarmaların bastan (front) yapıldığı veri yapılarıdır. Bir eleman ekleneceği zaman kuyruğun sonuna eklenir. Bir eleman çıkarılacağı zaman kuyrukta bulunan ilk eleman çıkarılır. Bu eleman da kuyruktaki elemanlar içinde ilk eklenen elemandır. Bu nedenle kuyruklara **FIFO (First-In First-Out = ilk giren ilk çıkar)** listeleri de denilmektedir.






## Kuyruk İslemleri

**createQueue :**Boş bir kuyruk oluşturulur.

**enqueue (q,x) :** q kuyruğunun sonuna x elemanını ekler **(append veya insert)**

**dequeu(q) :** q kuyruğunun basındaki elemanı siler (dequeue)

**queueEmpty :** Kuyruğun boş olup olmadığı bilgisi verilir.
**queueFull :**Kuyruğun dolu olup olmadığı bilgisi verilir.



## Kuyruk İşlemleri ve Tanımları

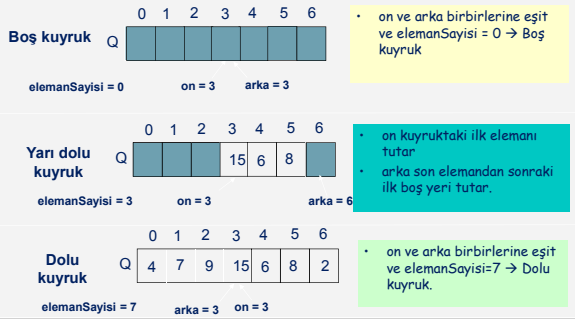
**insert(q,x) :** q kuyruğunun sonuna x elemanını ekler. (enqueue)

**x=remove(q) :** q kuyruğunun başındaki elemanı silerek x'e atar. (dequeue)

| operation | front | rear | 1 | 2 | 3 | 4 | element | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | | | | | | Queue is empty |
| 1. add 12 | 1 | 2 | 12 | | | | 12 | |
| 2. add 15 | 1 | 3 | 12 | 15 | | | 15 | |
| 3. add 17 | 1 | 4 | 12 | 15 | 17 | | 17 | |
| 4. delete | 2 | 4 | | 15 | 17 | | 12 | |
| 5. delete | 3 | 4 | | | 17 | | 15 | |
| 6. add 10 | 3 | 5 | | | 17 | 10 | 10 | Queue is full |
| 7. add 5 | 3 | 5 | | | 17 | 10 | | Error |

## Dizi Kullanarak

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



## Dairesel kuyruk

Dizinin başında kullanılmayan gözler olduğu zaman gelen elemanlar kuyruğun sonuna eklendiği için boş gözler gerksiz yer işgal edecektir.

Bu problemi önlemenin bir yolu, kuyruktan bir eleman çıktığında dizide bulunan bütün elemanları öne doğru birer kaydırmaktır. Fakat hareketin fazla olduğu bir kuyrukta bu yöntem sistemin yavaş çalışmasına sebep olacaktır. Bunun için, kullanılabilecek en uygun yöntem **dairesel kuyruk** yapısı kullanmaktır.

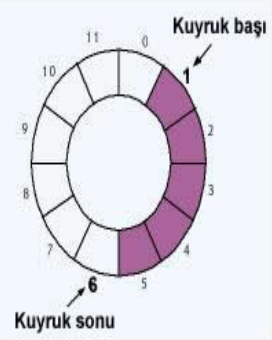


## Dairesel kuyruk

Kuyruk başını ve kuyruk sonunu gösteren adreslerin dairesel olarak ilerlediği kuyruk yapısıdır.

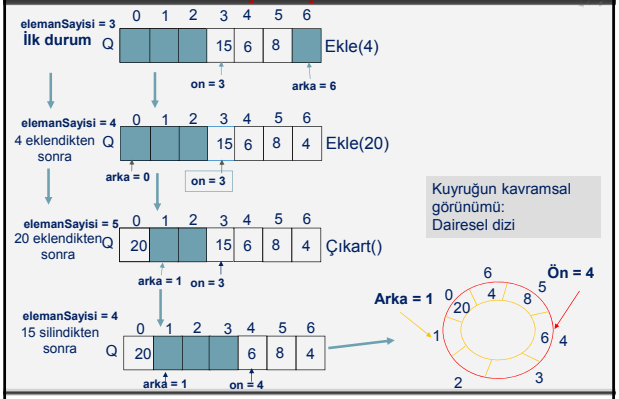
Burada n. Adresten sonra 0. adres gelebilmektedir.

Dairesel kuyruğa bir eleman eklendiğinde kuyruğun sonunu gösteren adres 1 artar. Dairesel kuyruktan bir eleman silindiğinde kuyruğun başını gösteren adres 1 artar. Her iki adres de n. adresi gösterdiğinde bir sonraki adımda 0. adresi göstereceklerdir.






## Dizi Kullanarak Gerçekleştirim






## Örnek

```
#include <iostream.h>
#define MAXSIZE 200
int q[MAXSIZE];
int front, rear;
void add(int);
int del();
bool dolu();
main(){
   int istek=1,i,num;
   front =0;
   rear = 0;
   while(istek ==1){
      cout<<"MENU: 1.kuyruğa ekle       2.kuyruktan sil ";
      cin>>istek;
      switch(istek){
         case 1:
            cout<<" veri gir... ";
            cin>>num;
            add(num);
            break;
         case 2: i=del();
            cout<<"silme fonksiyonundan  "<<i;break;
         default: cout<<"geçersiz seçim";
      }
   cout<<" tekrar  ( 1 evet), diger tuslar çık) ";
   cin>>istek;
   }
}
```

```
void add(int a){
    if(rear>MAXSIZE){
                cout<<"kuyruk boş";
                return;
                }else{
                q[rear]=a;
                  rear++;
                cout<<"son="<<rear<<" ve ilk="<<front;
    }
}
int del(){
   int a;
   if(dolu()){
                cout<<"kuyruk boş";
                return(0);
                }else{
                a=q[front];
                front++;
                }
                return(a);
}
bool dolu(){
if(front == rear) return true;
else return false;
}
```



Öncelikli kuyruklar, temel kuyruk işlemlerinin sonuçlarını elemanların gerçek sırasının belirlendiği veri yapılarıdır.

Azalan ve artan sırada olmak üzere iki tür öncelik kuyruğu vardır. Artan öncelik kuyruklarında elemanlar herhangi bir yere eklenebilir ama sadece en küçük eleman çıkarılabilir.

apq, artan öncelik kuyrugu olmak üzere

pqinsert(apq,x) x elemanını kuyruga ekler ve

pqmindelete(apq) en küçük elemanı kuyruktan çıkararak degerini döndürür.

Azalan öncelik kuyrugu ise artan öncelik kuyrugunun tam tersidir.



Artan öncelik kuyruğunda önce en küçük eleman, sonra ikinci küçük eleman sırayla çıkarılacağından dolayı elemanlar kuyruktan artan sırayla çıkmaktadırlar. Birkaç eleman çıkarıldıktan sonra ise daha küçük bir eleman eklenirse doğal olarak kuyruktan çıkarıldığında önceki elemanlardan daha küçük bir eleman çıkmış olacaktır.

Öncelik kuyruklarında sadece sayılar veya karakterler degil karmasık yapılar da olabilir. Örnek olarak telefon rehberi listesi, soyad, ad, adres ve telefon numarası gibi elemanlardan olusmaktadır ve soyada göre sıralıdır.

Öncelik kuyruklarındaki elemanların sırasının elemanların alanlarından birisine göre olması gerekmez. Elemanın kuyruga eklenme zamanı gibi elemanların alanları ile ilgili olmayan dıssal bir degere göre de sıralı olabilirler.

Öncelik kuyruklarının gerçeklestiriminde dizi kullanımı etkin bir yöntem degildir.

İyi çalışmalar…

13