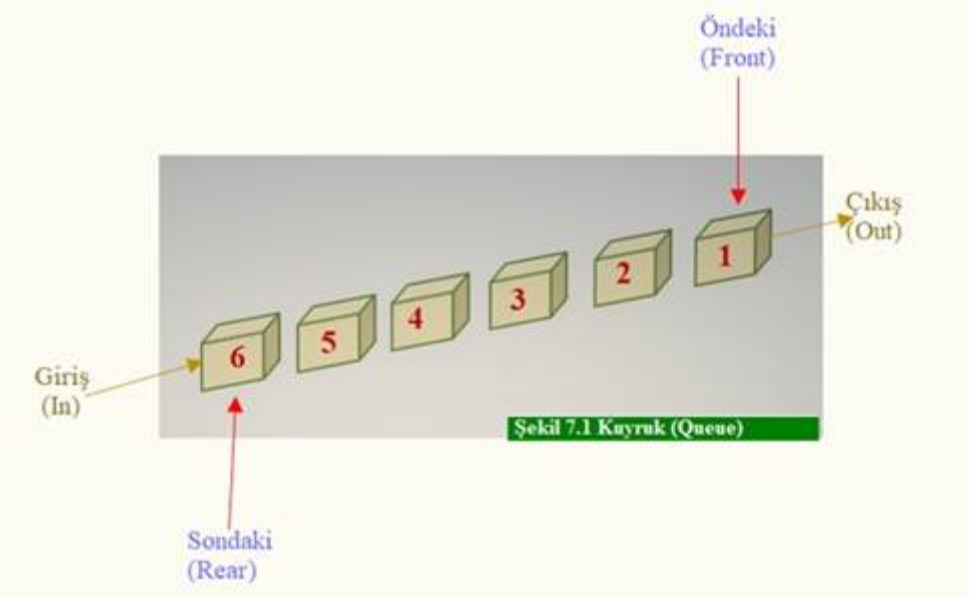
7. KUYRUK (QUEUE) VERİ YAPISI

Giriş

Kuyruk veri yapısı günlük hatta en çok karşılaşılan bir yapıdır. Üniversitesinin yemekhanesinde öğrencilerin yemek almak için oluşturdukları kuyruk, markette satın alınan ürünlerin ücretini ödemek için müşterilerin kasaların önünde oluşturdukları kuyruk, hastanede hastaların muayene olmak için doktorun odası önünde oluşturdukları kuyruk bu yapıya verilecek iyi örneklerdendir. Bu yapıların ortak özelliği, sırada bekleyen kişiler için sunulan hizmete erişebilecek tek bir nokta bulunmasıdır. Kuyruk yapısını tam olarak anlayabilmek için doktorun odası önünde bekleyen hastaları düşünelim. Sıraya ilk giren hasta (Kuyruğun önündeki hasta), ilk muayene olacak kişidir ve o muayene olduktan sonra sıra bir sonraki hastaya gelir. Bu esnada muayene olmak için hastaneye yeni gelen bir hasta olduğunda bu hasta kuyruğun en sonuna geçmek zorundadır. Muayene olmak için doktorun odası önünde bekleyen hastalar düşünüldüğünde en son muayene olacak hasta kuyruğa en son eklenen hasta olacağını tahmin etmek zor değildir. Yukarıda açıklanan özellikleri ile kuyruk veri yapıları bazı özellikleriyle yığın (stack) veri yapılarına benzer. Örneğin kuyruk veri yapıları da yığın veri yapıları gibi sınırlı erişimli bir veri yapısıdır. Her iki veri yapısı da doğrusaldır.

Kuyruk veri yapısının tanımı, “Eleman ekleme işleminin sondan *(rear),* eleman çıkarma işleminin baştan *(front)* yapıldığı doğrusal bir veri yapısıdır.” şeklinde verilebilir. Başka bir ifadeyle kuyruk veri yapıları, önce gelen elemana ilk hizmetin verildiği veri yapılarıdır. Yığın veri yapısından farklı olarak (*6. Bölümde açıklandığı gibi yığın veri yapıları son giren ilk çıkar prensibine göre çalışır - LIFO*) ilk gelen – ilk çıkar (First In First Out – FIFO) prensibine göre çalışır.



Literatürde Bir elemanın kuyruğa girmesi *enqueue,* bir elemanın kuyruktan silinmesi *dequeue* olarak kullanılır. Kuyruk veri yapısında enqueue işlemleri kuyruğun arkasından, dequeue işlemleri kuyruğun önünden yapılır. Kuyruk veri yapısında dolu bir kuyruğa eleman eklemek *overflow* hatası döndürürken boş bir kuyruktan eleman silmek *underflow* hatası döndürür.

Kuyruk yapısının, bilgisayarda kullanılmasına örnek olarak işletim sistemlerinin proseslere bilgisayarın fiziksel kaynaklarını paylaştırılmasını verebiliriz. Örneğin prosesler işlemciyi ilk gelen ilk hizmet alır yaklaşımı ile kullanılır. Benzer şekilde giriş çıkış ünitelerinin kullandırılması da aynı mantıkla çalıştırılır.

Bundan başka bilgisayarda yazdırma işlemlerinde de kuyruk yapısı kullanılır. Çünkü yazıcılar görece bilgisayarlara göre daha yavaştır. Dolayısıyla bilgisayarda bir darboğaz oluşturmamak için yazdırma işleri kuyruğa alınır ve sırayla yazdırılır. Bu tam bir kuyruk uygulamasıdır.

Yine bilgisayarlarda kuyruk uygulamasına başka bir örnek, klavye ve fare işlemlerinin ardışık olarak okunmasını gösterebiliriz.

Kuyruk yapıları da yığınlara benzer biçimde diziler ve bağlı listeler kullanılarak oluşturulup çalıştırılabilir. Kavram basit olsa da, bir kuyruğu programlamak, bir yığını programlamak kadar basit değildir. Hastanede muayene olmak için bekleyen hasta örneğine geri dönelim ve bir

hastanın muayene olup kuyruktan ayrıldığını varsayalım. Bu durumda sıradaki herkes bir öne hareket etmek zorundadır, Şimdi, bir seferde yalnızca bir kişinin hareket edebildiğini varsayalım. Böylece, ikinci kişi birinci kişinin bıraktığı boşluğu doldurmak için öne çıkar ve ardından üçüncü kişi, ikinci kişinin bıraktığı boşluğu doldurmak için öne çıkar ve bu böyle devam eder. Şimdi, herkes öne çıkana kadar hiç kimsenin sıraya eklenemeyeceğini hayal edin. Çizginin çok yavaş hareket edeceğini görebilirsiniz. Çalışan bir kuyruğu programlamak zor değildir, ancak hızlı bir kuyruk oluşturmak oldukça büyük bir öneridir.

Bu bölümde dizi kullanarak kuyruk yapısına örnek verilirken iki ayrı yaklaşım kullanılacaktır. Bu yaklaşımlardan birincisinde dizi kullanarak bir kuyruk oluşturulacak, bu örnekte herhangi bir iyileştirme önerilmeyecek, dizi kullanılarak önerilecek ikinci örnekte yukarıdaki paragrafta açıklanan eksiklikleri giderilecek, dairesel bir kuyruk uygulaması geliştirilecektir. Son olarak bağlı liste kullanarak geliştirilen bir kuyruk veri yapısı örneği verilecektir.

7.1. Kuyrukların (Queue) Dizi (Array) Uygulaması

Bu başlık altında C programlama dilinde dizi kullanarak kuyruk işlemlerinin nasıl yapılacağı anlatılacaktır. Dizi ile kuyruk oluşturmak için yapılması gereken iki temel işlem vardır. Bu işlemlerden birincisi kuyruğa eleman eklemek, ikincisi ise kuyruktan eleman çıkarmaktır.

7.1.1. Kuyruğa Eleman Ekleme

Kuyrukların dizi uygulamasında başlangıçta kuyrukta herhangi bir eleman yoktur. Bu durumda *front* ve *rear* değişkenleri aynı adresi göstermektedir (front=0, rear=0). Fakat uygulama çalışırken her zaman başlangıç konumunda olunmayabilir. Bu durumda kuyruğa önceden veriler girilmiş demektir. Kuyruğa önceden eleman girilmiş ise (kuyruk boş değilse) önce kuyruğa yeni eleman eklemek için yer olup olmadığı kontrol edilir. Eğer yer yoksa kullanıcıya bu doğrultuda mesaj verilip ekleme fonksiyonunun çalışması sonlandırılır. Bu kontrollerin sonunda kuyrukta eleman girilmesi için yer olduğu anlaşılırsa rear değişkeninin değeri bir arttırılır ve işaret ettiği yere yeni eleman eklenir. 7.1.2 ‘de eleman ekleme fonksiyonu verilmiştir.

7.1.2. Kuyruğa Eleman Ekleyen Fonksiyon

```
void elemanEkle(int kyr[],int sayi){
    if(rear==-1)
    {
        front=rear=0;
        kyr[rear]=sayi;
    }
    else if(rear==MAX-1){
        printf("\n Kuyruk Dolu\n");
        return;
    }
    else{
        rear++;
        kyr[rear]=sayi;
    }
    printf("\nEleman Eklendi..");
}
```

/* rear ve front değişkenlerinin başlangıç değerleri -1 olarak verilmiştir. Fonksiyon çağırıldığında rear==-1 ise kuyrukta herhangi bir eleman yoktur. rear ve front değerleri sıfır yapılarak kuruğa ilk eleman eklenir. */

/*rear==MAX-1 ise Kuyruk doludur. mesaj verilip fonksiyondan çıkılır */

/*Yukarıdaki iki koşul sağlanmıyorsa kuyrukta daha eleman vardır. bu durumda rear bir arttırılır ve iaret ettiği yere eleman eklenir. */

Fonksiyon 7.1 Yığına Eleman Ekleme (Enqueue)

Eğer kuyruk bir dizi üzerinde oluşturuluyorsa bu kuyruğa eleman eklemek için aşağıdaki adımlar takip edilebilir:

1. **Adım:** *rear* değerinin eksi bire eşit olup olmadığı kontrol edilir.

```
if(rear==-1)
```

Bu eşitlik sağlanıyorsa *rear* ve *front* değişkenlerinin değerlerine sıfır atanır ve bu indisin işaret ettiği yere fonksiyon çağırılırken gönderilen *sayi* değeri atanır.

2. **Adım:** Yukarıdaki eşitlik sağlanmıyorsa kuyruğun kapasitesinin dolu olup olmadığı kontrol edilir.

```
else if(rear==MAX-1)
```

Kuyruk dolu ise kullanıcıya mesaj verilip fonksiyondan çıkılır.

3. **Adım:** 1 ve 2. Adımlardaki koşul sağlanmıyorsa kuyrukta yer vardır. *rear* değeri bir arttırılır ve fonksiyonun çağırılması sırasında fonksiyona gönderilen *sayi* değeri *rear* 'in işaret ettiği yere atanır.

7.1.3. Kuyruktan Eleman Silme

Kuyruktan eleman silinmesi için, önce kuyruğun boş olup olmadığı kontrol edilir. Eğer boş ise kullanıcıya bu doğrultuda bir mesaj verilir ve fonksiyondan çıkılır. Eğer kuyruk boş değil ve fakat kuyrukta sadece bir eleman varsa (*front == rear ise*) o zaman front değişkeninin gösterdiği adresteki eleman kuyruktan çıkarılır. *rear* ve *front* değişkenlerinin değeri

-1 (eksi bir) yapılır. Bu şekilde başlangıç şartlarına dönülmüş olur. Yukarıdaki açıklanan iki karşılaştırmanın sonucu *false* ise o zaman *front* değişkenin gösterdiği adresteki eleman kuyruktan çıkartılır ve *front* değişkeni bir arttırılır.

7.1.4. Kuyruktan Eleman Silme Fonksiyonu

```
void elemanSil(int kyr[])
{
    int sayi;
    if(front==-1){
        printf("Kuyruk Boş");
        return;
    }
    else if(front==rear){
        sayi=kyr[front];
        front=rear=-1;
    }
    else{
        sayi=kyr[front];
        front++;
    }
    printf("\nEleman Silindi: %d.",sayi);
}
```

/*front değişkeninin değeri -1 ise kuyrukta herhangi bir eleman yoktur. Dolayısı ile bu durumda kullanıcıya mesaj verip fonksiyondan çıkılır. */

/*front ==rear ise kuyrukta bir eleman vardır. Bu durumda elema kuyruktan silinir ve front ve rear değişkenlerine -1 atanır */

/* Yukarıdaki iki koşulun sağlanmadığı durumda silinecek eleman sayı değşkenine atanıp front değişkeni bir arttırılır. */

Fonksiyon 7.2 Kuyruktan eleman silme fonk.

Eğer kuyruk bir dizi üzerinde oluşturulmuş ise bu kuyruktan eleman çıkarmak için aşağıdaki adımlar takip edilebilir:

1. Adım: Kuyruğun boş olup olamadığını kontrol et. Kuyruk boş ise (

```
if(front==-1)
```

) kullanıcıya **IMAGE** şeklinde mesaj ver ve fonksiyonun çalışmasını sonlandır.

2. Adım: Yukarıdaki eşitlik sağlanmıyorsa

```
(front==rear)
```

olup olmadığını kontrol et. Eşit ise *front* değişkenin gösterdiği adresteki elemanı kuyruktan çıkar *front* ve *rear* değişkenlerinin değerlerini bir yap.

3. Adım: 1 ve 2. Adımlardaki koşullar sağlanmıyorsa front değişkeninin gösterdiği adresteki elemanı kuyruktan çıkar. Front değişkeninin değerini bir arttır.

4. 6.2.1. Yığınların Dizi Uygulamasında Yığına Eleman Ekleme (*push(eklEleman)*)

7.1.5. Kuyruktaki Elemanları Ekrana Yazdırmak.

```
void yazdir(int kyr[])
{       int i;

    if(rear==-1) {
     printf("\nKuyruk Boş .\n");
      return; }

    for(i=front;i<=rear;i++){
        printf("%d  ",kyr[i]);
    }

    printf("\n");
}
```

/* Kuyruğun Boş olup olmadığını kontrol et. Kuyruk boş ise fonksiyondan çık */

/* döngü içerisinde kuyruğun ilk elemanından son elamanına kadar bütün elemanları yazdır.

Fonksiyon 7.3 Kuyruk Yadırma Fonksiyonu

7.1.6. Kuyrukların Dizi Uygulaması (C programı)

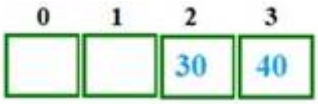
```
#include <stdio.h>
#include <locale.h>
#define MAX 4
int kuyruk[MAX],front=-1,rear=-1;
void elemanEkle(int kyr[],int sayi){
    if(rear==-1){
        front=rear=0;
        kyr[rear]=sayi;
    }
    else if(rear==MAX-1){
        printf("\n Kuyruk Dolu\n");
        return;
    }
    else{
        rear++;
        kyr[rear]=sayi;
    }
    printf("\nEleman Eklendi..");
}
void yazdir(int kyr[]){
      int i;
    if(rear==-1) {
     printf("\nKuyruk Boþ.\n");
      return; }
    for(i=front;i<=rear;i++){
        printf("%d  ",kyr[i]);
    }
     printf("\n");
}
void elemanSil(int kyr[]){
    int sayi;
    if(front==-1){
        printf("Kuyruk Boş");
        return;
    }
    else if(front==rear){
        sayi=kyr[front];
        front=rear=-1;
    }
    else{
        sayi=kyr[front];
        front++;
    }
    printf("\nEleman Silindi: %d.",sayi);
}
```

```
int main(
    setlocale(LC_ALL,"Turkish");
    int sayi,secim;
while(1){
    printf("\n\n Kuyruktaki Elemanlar :\n\n");
    yazdir(kuyruk);
    printf("\n 1. Eleman Ekle");
    printf("\n 2. Eleman Sil");
    printf("\n 3. Yazdir");
    printf("\n 9. Çıkış\n");
    printf("\n    Seçiminiz : ");
    scanf("%d",&secim);
    switch(secim){
     case 9:
         return 0;
         break;
     case 1:
         printf("Eklenecek Elemanı Giriniz :");
         scanf("%d",&sayi);
         elemanEkle(kuyruk,sayi);
         break;
     case 2:elemanSil(kuyruk);
         break;
     case 3:yazdir(kuyruk);
         break;
     default:
         printf("\nGeçersiz Seçim\n");
         break;
    } }
return 0;
}
```

Kuyruk Dizi Uygulaması (C programı)

Yukarıda verilen C programı çalıştırıldığında aşağıdaki durumlarla karşılaşılmıştır.

7.2. Dairesel Kuyruk

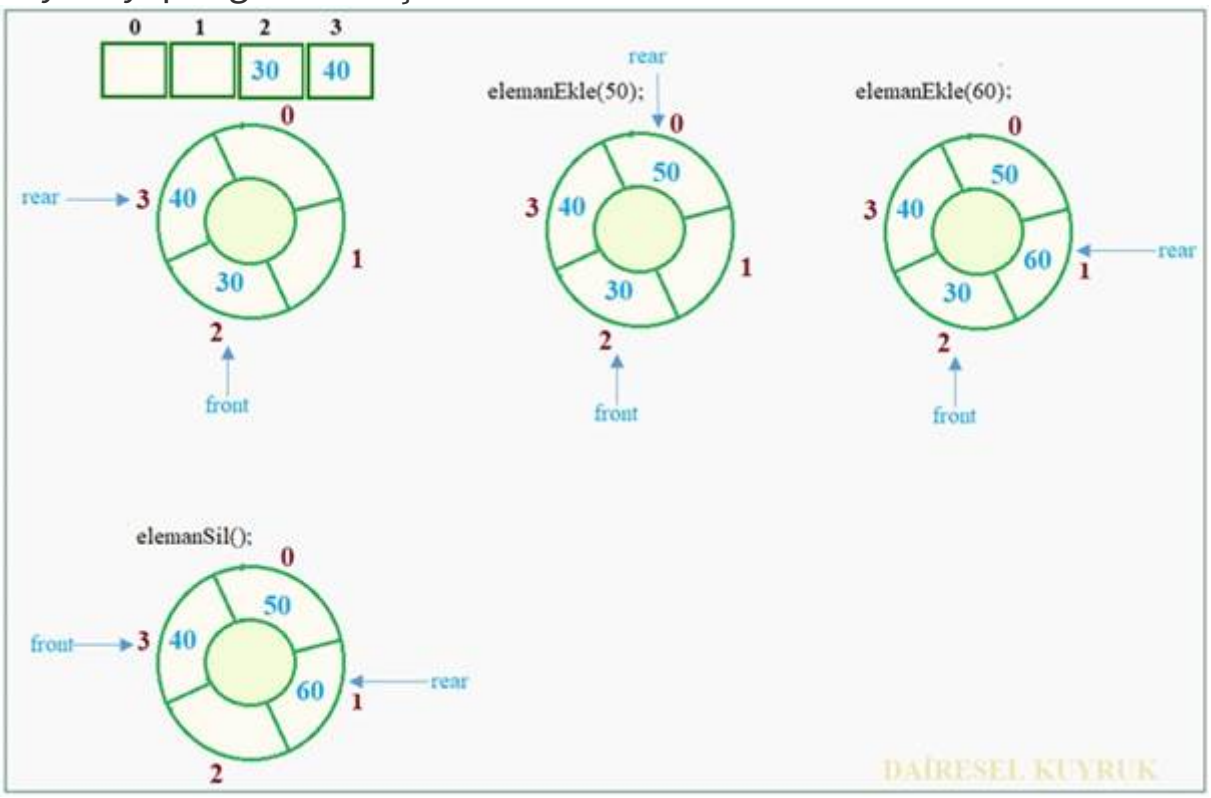


Yukarıda verilen kuyruk örneğinde [0] ve [1] indisi ile işaret edilen adresler boş, [2], [3] indisleri ile işaret edilen adresler doludur. Şekilden de görülebileceği gibi kuyruğun sonu dolu olduğundan ve yeni gelecek eleman da sona gelmesi gerektiğinden, 7.1'de verilen programda bir iyileştirme yapılmadan, kuyrukta boş yer olmasına rağmen yeni eleman eklenemez.

Bu problemin çözümü için iki farklı yöntem uygulanabilir. Birinci yöntem, kuyruğun başındaki eleman silindikten sonra, ikinci sıradaki elemanı silinen elemanın yerine, üçüncü sıradaki elemanı ikinci sıradaki elemanın yerine kaydırmak şeklinde bir çözüm üretmektir ve her silme işleminden sonra kuyruktaki bütün elemanların her defasında yer değiştirmesi sonucunu doğurur ve yavaş çalışır.

Problemin çözümü için ikinci yöntem dairesel kuyruk yapısı oluşturmaktır. Dairesel kuyruk yapısı, kuyruğun son elemanına ulaşıldığında, bu uca kuyruğun baş tarafında boşalan hücreler varsa bunları ekleyerek ve front değeri N-1 olduğunda buradaki elemanı sildikten sonra front değerini baştaki boş hücreleri işaret edecek şekilde güncelleyerek oluşturulabilir. Bu durumda kuyruktaki veriler bir tekerleğin üzerinde hareket ediyormuş gibi davranır.

Dairesel Kuyruk, işlemlerin FIFO (İlk Giren İlk Çıkar) prensibine göre gerçekleştirildiği ve son konumun bir daire oluşturmak için ilk konuma bağlandığı doğrusal bir veri yapısıdır. Bellek yönetimi ve CPU işlemlerini planlamak gibi önemli kullanım alanları vardır. Aşağıda şekilde dairesel kuyruk yapısı gösterilmiştir.



7.2.1. Kuyruğa Eleman Ekleyen Fonksiyon

```
void elemanEkle(int sayi){
if((front == 0 && rear == MAX-1) || (front == rear+1)){
printf("Kuyruk Dolu\n");
return;
}
if(front == -1){
front = 0;
rear = 0;
}
else{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
kuyruk[rear] = sayi;
}
```

Fonksiyon 7.4 Dairesel kuyruk-Eleman Ekle

1. **Adım:** Dairesel sıranın (kuyruğun) dolu olup olmadığını kontrol edin. Kuyruk dolu ise kullanıcıya 'Kuyruk Dolu' mesajı verip fonksiyondan çıkın.

2. **Adım:** Eğer kuyruk boş ise *front=0, rear=0* ataması yapın.

Değilse

Eğer

```
(rear == MAX-1)
```

ise

```
rear = 0;
```

Atamasını yap. ***Bu atama ile kuyruk dairesel hale gelir*.**

Değilse

```
rear = rear+1;
```

Atamasını yap.

3. **Adım:**

```
kuyruk[rear] = sayi;
```

atamasını yap.

7.2.2. Kuyruktan Eleman Silen Fonksiyon

```
void elemanSil()
{
if(front == -1)
{
printf("Kuyruk Bos\n");
return ;
}
printf("Kuyruktan cikan eleman : %d \n",kuyruk[front]);
if(front == rear)
{
front = rear=-1;
}
else
{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
}
```

Fonksiyon 7.5 Dairesel Kuyruk- Eleman silme

1. Adım: Dairesel sıranın (kuyruğun) boş olup olmadığını kontrol edin (

```
if(front == -1)
```

). Kuyruk boş ise kullanıcıya 'Kuyruk Boş' mesajı verip fonksiyondan çıkın.

2. Adım:

```
kuyruk[front]
```

elemanını kuyruktan çıkartın.

3. Adım: Eğer

```
(front == rear)
```

ise

```
front = rear=-1;
```

Atamasını yapın. Kuyruk boş hale gelir.

Değilse

Eğer

```
(front == MAX-1)
```

ise

```
front = 0;
```

atamasını yapın. ***Bu atama ile kuyruk dairesel hale gelir.***

Değilse

```
front = front+1;
```

atamasını yap.

7.1.6. Dairesel Kuyruk (C programı)

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 4
int kuyruk[MAX];
int front = -1;
int rear = -1;
void elemanEkle(int);
void elemanSil();
void yazdir();
int main(){
int secim,sayi;
do{
printf("\n1.Eleman Ekle\n");
printf("2.Eleman Sil\n");
printf("3.Yazdir\n");
printf("4.Cikis\n");
printf("Seciminiz : ");
scanf("%d",&secim);
switch(secim){
case 1 :
printf("Kuyruga Eklenecek Eleman : ");
scanf("%d", &sayi);
elemanEkle(sayi);
yazdir();
break;
case 2 :
elemanSil();
yazdir();
break;
case 3:
yazdir();
break;
case 4:
break;
default;
printf("Hatali Secim");
}
}while(secim!=4);
return 0;
}

void elemanEkle(int sayi){
if((front==0&&rear==MAX-1)||(front==rear+1)){
printf("Kuyruk Dolu\n");
return;
}
if(front == -1){
front = 0;
rear = 0;
}
else{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
kuyruk[rear] = sayi;
}
void elemanSil(){
if(front == -1){
printf("Kuyruk Bos\n");
return ;
}
printf("Kuyruktan cikan eleman:%d\n",kuyruk[front]);
if(front == rear){
front = rear=-1;
}
else{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
}
void yazdir(){
    printf("\n");
    if(front == -1){
        printf("\n Kuyrukta Silinecek Eleman Yok");
        return;
    }
    for(int i = front; i != rear; i = (i+1)%MAX){
        if(i == front && i == rear){
            printf(" %d", kuyruk[i]);
        }
        else if(i == front){
            printf(" %d", kuyruk[i]);
        }
        else
            printf(" %d", kuyruk[i]);
  }
   printf(" %d", kuyruk[rear]); }
```

DAİRESEL KUYRUK

7.3. Kuyrukların Bağlı Liste Uygulaması

Kuyrukların bağlı liste uygulaması oldukça kolay bir uygulamadır. Bu uygulama, işlemler açısından tek yönlü bağlı liste yapısı ile hemen hemen aynıdır. Örneğin bağlı liste kullanarak kuyruğa eleman ekleme

işlemi tek yönlü bağlı listelerde listenin sonuna eleman ekleme işlemi ile aynıdır. Benzer şekilde bağlı liste kullanarak kuyruktan eleman çıkarma işlemi de tek yönlü bağlı listelerde listenin başından eleman silme işlemi ile aynıdır. Aşağıda bağlı liste kullanarak kuyruklar üzerinde işlem yapan C kodu verilmiştir.

```c
#include <stdio.h>
#include <stdlib.h>
struct node{
    int veri;
    struct node *ptr;
}*front=NULL,*rear=NULL,*temp,*front1;
int sayac = 0;
void elemanEkle(int sayi){
    if (rear == NULL){
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->veri = sayi;
        front = rear;
    }
    else{
        temp=(struct node *)malloc(1*sizeof(struct node));
        rear->ptr = temp;
        temp->veri = sayi;
        temp->ptr = NULL;
         rear = temp;
    }
    sayac++;
}
void elemanSil(){
    front1 = front;

    if (front1 == NULL){
        printf("\n Kuyruk Bos");
        return;
    }
    else
        if (front1->ptr != NULL){
            front1 = front1->ptr;
            printf("\n Silinen Eleman : %d", front->veri);
            free(front);
            front = front1;
        }
        else{
            printf("\n Silinen Eleman : %d", front->veri);
            free(front);
            front = NULL;
            rear = NULL;
        }
        sayac--;
        }
```

```c
void yazdir(){
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Kuyruk Bos");
        return;
    }
    while (front1 != rear){
        printf("%d ", front1->veri);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->veri);
}
void qsize(){
    printf("\n Eleman Sayisi : %d", sayac);
}
void empty(){
     if ((front == NULL) && (rear == NULL))
        printf("\n Kuyruk Bos");
    else
       printf("Kuyruk Bos Degil");
}
int bastakiEleman()
{
    if ((front != NULL) && (rear != NULL))
        return(front->veri);
    else
        return 0;
}
```

KUYRUK VERİ YAPISININ BAĞLI LİSTE UYGULAMASI

Bölüm Özeti

Kuyruk (Queue) veri yapısını öğrenecek ve kuyruk veri yapısını bağlı liste veri yapısı, dizi veri yapısı ve yığın veri yapısı ile karşılaştırabilecek,

Kuyruk (*Queue*) veri yapıları bazı özellikleriyle yığın (*stack*) veri yapılarına benzer. Örneğin kuyruk veri yapıları da yığın veri yapıları gibi sınırlı erişimli bir veri yapısıdır. Her iki veri yapısı da doğrusaldır. Kuyruk veri yapısı, eleman ekleme işleminin sondan *(rear),* eleman çıkarma işleminin baştan *(front)* yapıldığı doğrusal bir veri yapısıdır şeklinde tanımlanabilir. Başka bir ifadeyle kuyruk veri yapıları, önce gelen elemana ilk hizmetin verildiği veri yapılarıdır. Yığın veri yapısından farklı olarak (*6. Bölümde açıklandığı gibi yığın veri yapıları son giren ilk çıkar*

prensibine göre çalışır - LIFO) ilk gelen – ilk çıkar (First In First Out – FIFO) prensibine göre çalışır.

Kuyruk yapısının, bilgisayarda kullanılmasına örnek olarak işletim sistemlerinin proseslere bilgisayarın fiziksel kaynaklarını paylaştırılmasını verebiliriz. Örneğin prosesler işlemciyi ilk gelen ilk hizmet alır yaklaşımı ile kullanılır. Benzer şekilde giriş çıkış ünitelerinin kullandırılması da aynı mantıkla çalıştırılır.

Kuyruğa eleman ekleyebilecek,

Kuyruk yapısına veri eklerken başlangıçta kuyruğun dolu olup olmadığı kontrol edilir ve bu kontrolde kuyruğun dolu olduğu anlaşılırsa veri ekleme işlemi yapılmaz. Kuyruk dolu değilse ve kuyruğa ilk defa veri ekleniyorsa *front* ve *rear* değişkenlerinin değerleri 0 (sıfır) yapılır. *rear* değişkeninin gösterdiği adrese veri eklenir. Bundan sonraki adımlarda *rear* değişkeninin değeri arttırılarak, değişkenin işaret ettiği adrese veriler eklenmeye devam edilir. Kuyrukların dizi uygulamasında veri girişi rear değişkeninin değeri N-1 oluncaya kadar sürdürülebilir. Kuyruğa daha fazla sayıda veri yüklenemez. Kuyrukların bağlı liste uygulamasında kuyruğa veri ekleme işlemi tek yönlü bağlı listenin sonuna eleman ekleme işlemi ile aynıdır.

Kuyruktan eleman çıkartabilecek,

Kuyruktan eleman silinmesi için, önce kuyruğun boş olup olmadığı kontrol edilir. Eğer boş ise kullanıcıya bu doğrultuda bir mesaj verilir ve fonksiyondan çıkılır. Eğer kuyruk boş değil ve fakat kuyrukta sadece bir eleman varsa (*front == rear ise*) o zaman front değişkeninin gösterdiği adresteki eleman kuyruktan çıkarılır. *rear* ve *front* değişkenlerinin değeri -1 (eksi bir) yapılır. Bu şekilde başlangıç şartlarına dönülmüş olur. Yukarıdaki açıklanan iki karşılaştırmanın sonucu *false* ise o zaman *front* değişkenin gösterdiği adresteki eleman kuyruktan çıkartılır ve *front* değişkeni bir arttırılır. Kuyrukların bağlı liste uygulamasında yığından eleman çıkarmak tek yönlü bağlı listelerin başından eleman silme işlemi ile aynıdır.

Kaynakça

1. Robert Sedgewick, Kevin Wayne, Algoritmalar, Nobel Yayınevi, 2018

2. Muhammed Mastar, Süha Eriş, C++, KODLAB Yayın Dağıtım Yazılım ve Eğitim

Hizmetleri San. ve Tic. Ltd. Şti, 2012.

3. Nejat Yumuşak, M. Fatih Adak, C, C++ ile Veri Yapıları, Seçkin Yayıncılık, 2014.

4. Rıfat Çölkesen, Veri Yapıları ve Algoritmalar, Papatya Yayıncılık, 2002.

5. G. Murat Taşbaşı, İleri C programlama, Altaş Yayıncılık ve Elektronik Tic. Ltd. Şti, 2005.