BÖLÜM 4

KUYRUKLAR VE YIĞINLAR

ANAHTAR KAVRAMLAR
Kuyruk, Yığın, Enqueue, Dequeue, Push, Pop

ÖĞRENME HEDEFLERİ

1. Kuyruk veri yapısını tanımlayabilir ve kullanabilir.
2. Kuyruk veri yapısı üzerindeki temel işlemleri uygulayabilir.
3. Yığın veri yapısını tanımlayabilir ve kullanabilir.
4. Yığın veri yapısı üzerindeki temel işlemleri uygulayabilir.
5. Yığın veri yapısını gerçek problemlerin çözümünde kullanabilir.

DÖRDÜNCÜ BÖLÜM

GİRİŞ

Bu bölümde kuyruk ve yığın veri yapıları anlatılacaktır. Kuyruk bir listeye ilk giren liste elemanının ilk çıkması, bir başka ifade ile ilk işleme alınması kuralını uygulayan bir veri yapısıdır. Bu nedenle ilk giren ilk çıkar (First In First Out - FIFO) tipi bir yapı olarak değerlendirilir. Bir marketin kasasında bekleyen alış veriş yapan kişiler, bir bankada işlem bekleyen müşteriler günlük hayattan kuyruk veri yapısına uygun örneklerdir. Bu yapıyı uygulamak için diziler veya bağlı listeler kullanılabilir. Her iki yapıda da kuyruk veri yapısına ait temel işlemler değişmez. Sadece diziler veya bağlı listelerin karakterine göre kodlama işlemi farklılık gösterir. Kuyruk veri yapısında listenin her iki ucu da aktif olarak kullanılır. Kuyruk veri yapısında listenin sonu dışında herhangi bir noktasına ekleme yapılamaz. Aynı şekilde listenin başı dışında herhangi bir yerinden de silme işlemi gerçekleştirilemez. Kullanılan yapıya göre listenin sonu veya başı dışındaki bir elemana doğrudan erişmek, kullanılan programlama dilinin sözdizimine göre mümkün olabilir. Ancak bu kuyruk veri yapısında bu işlemin geçerli bir işlem olduğu anlamına gelmez. Eğer bu tür bir rasgele değişikliğe (ekleme veya silme) kodlamada izin verilmiş ve uygulanmışsa bu veri yapısı kuyruk olarak değerlendirilemez. Yığınlar listeye son giren elemanın ilk çıkması kuralına sahip veri yapılarıdır. Son giren ilk çıkar (Last In First Out - LIFO) tipi yapılar olarak bilinirler. Temizlenen tabakların üst üste yerleştirilmesi yığınlara örnek olarak verilebilir. Temiz bir tabağa ihtiyaç duyulduğunda en son yıkanan bir başka ifade ile tabak yığınının en üstünde yer alan tabak kullanılır. Kuyruklarda olduğu gibi yığınları uygulamak için diziler veya bağlı listeler kullanılabilir. Her iki yapının birbirlerine göre avantaj ve dezavantajları mevcuttur. Kuyruk veri yapısının aksine yığınlarda listenin sadece bir ucu kullanılır. Ekleme ve silme işlemleri aynı noktadan gerçekleştirilir. Ekleme ve silme işlemleri kesinlikle listenin farklı bir noktasına uygulanamaz. Bu işlemlerinin listede rasgele herhangi bir noktada gerçekleştiği yapılar yığın olarak isimlendirilemezler.

Bu bölümde önce kuyruk veri yapısı anlatılacak, bu yapının temel fonksiyonları gösterilecektir. Daha sonra yığın veri yapısı ve bu yapı üzerinde gerçekleştirilebilecek işlemler çalışılacaktır. Bölüm, yığın veri yapısının pratik olarak kullanıldığı bir örnek ile tamamlanacaktır.

1. KUYRUKLAR

Günlük yaşamda birçok yerde karşılaşabileceğimiz kuyruk kavramı temel veri yapılarından birisidir. Bir otobüs durağında bekleyen yolcular, markette kasada bekleyen müşteriler ve yemekhanede yemek sırasında yer alan öğrenciler kuyruk veri yapısına verilebilecek örneklerdendir. Doğrusal bir veri yapısı olup ilk giren ilk çıkar prensibi ile çalışır. Listenin bir tarafı yeni bir eleman eklemek için belirlenmiş olup diğer taraf silme veya kuyruktan çıkarma işlemi için kullanılır.

Kuyruk veri yapısını uygulamak için diziler veya bağlı listeler kullanılabilir. Şekil 4.1'de verilen dizi ile dört elemana sahip örnek bir kuyruk gösterilmiştir. Bu kuyruk belirli ekleme ve silme işlemlerinden sonra bu duruma gelmiştir.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
		14	-42	50	61				

Şekil 4.1 Dizi kullanarak örnek bir kuyruk uygulaması

Burada kuyruk veri yapısını basit bir şekilde ifade edebilmek amacıyla tamsayı kullanılmıştır. Gerçek uygulamalarda her türlü nesneye ait kuyruk oluşturmak mümkündür. Buradaki tamsayıları, bu tür nesneleri temsil eden anahtar değerler olarak da görebiliriz. Kuyruğa yeni bir eleman ekleneceği zaman, bu elemanın yerleştirileceği yer bu dizide a[6] pozisyonudur. Yeni bir elemanın başka bir indise yerleştirilmesi mümkün değildir. Aynı şekilde kuyruktan ilk alınacak/silinecek eleman da kesinlikle bellidir. Yukarıda verilen kuyruktan silinecek ilk eleman 14 değerine sahip a[2] pozisyonudur. Farklı bir eleman çıkartılamaz. Eğer kuyrukta herhangi bir eleman yok ise, kuyruk boşsa silme işleme gerçekleştirilemez. Benzer şekilde dolu olan bir kuyruğa yeni bir eleman eklenemez. Yukarıdaki kuyruk maksimum on eleman alabilir. Son pozisyon olan a[9] alanı kullanıldıktan sonra, a[0] pozisyonu ile ekleme işlemine devam edilmelidir. İlgili fonksiyonlar kodlanırken bu durumlar göz önüne alınmalıdır. Ekleme ve silme işlemleri gerçekleştirilirken kuyruk veri yapısı için iki değişken kullanılır. Bu değişkenlerden birincisi kuyruktaki son elemanı gösteren 'rear' indisidir. Diğeri ise kuyruktaki ilk elemanı gösteren 'front' indisidir.

```
1    void enqueue (int queue[], int value)
2    {
3        if((rear+1)%SIZE  != front)
4        {
5            rear=(rear+1)%SIZE;
6            queue[rear]= value;
7        }
8        else
9            cout<<"Kuyruk dolu. Yeni eleman eklenemez. \n";
10   }
```

Şekil 4.2 Bir kuyruğa yeni bir eleman ekleme fonksiyonu

DİKKAT EDELİM

enqueue ve **dequeue** fonksiyonlarında kullanılan 'front' ve 'rear' değişkenlerinin ilgili programda global değişken olarak tanımlandığını varsayalım.

Şekil 4.2 ve 4.3'te verilen fonksiyonlarda SIZE ifadesinin daha önce tanımlanmış sembolik sabit, 'rear' ve 'front' değişkenlerinin ise global değişken olduğunu kabul edelim. Şekil 4.2' de dizi ile gösterilen bir kuyruk veri yapısına yeni bir eleman ekleyen *enqueue* fonksiyonunun tanımı verilmiştir. Bu fonksiyonda önce son elemanı gösteren 'rear' değişkeni bir artırılmış ve bu değişkenin gösterdiği alana yeni değer atanmıştır. Bu fonksiyon çağrıldığında kuyrukta en az bir eleman olduğu varsayılmıştır. Şekil 4.3'te ise kuyruktan bir eleman çıkarma fonksiyonu *dequeue* tanımlanmıştır. Önce 'front' indisinin gösterdiği değer geçici bir değişkene atanmış, 'front' değeri bir artırılmıştır. Fonksiyonun sonunda geçici değişkene atanan değer döndürülmüştür. Fonksiyonlarda 'rear' ve 'front' indislerinin SIZE-1 den sonra 0 ile devam etmesi '%' işleci ile sağlanmıştır. Ayrıca dolu olan bir kuyruğa yeni eleman eklenemeyeceği, boş olan bir kuyruktan eleman silinemeyeceği uyarıları kullanıcıya aktarılmıştır.

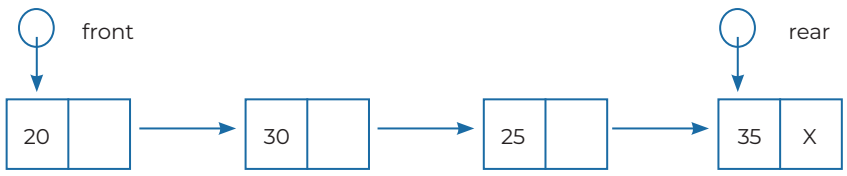
```
1    int dequeue (int queue [])
2    {
3        int temp=-1;
4        if (front==-1 && rear==-1)
5        {
6            cout<<"Kuyruk bos. Eleman silinemez.\n";
7            return temp;
8        }
9        else
10       {
11           temp=queue[front];
12           front=(front+1)%SIZE;
13           return temp;
14       }
15   }
```

Şekil 4.3 Bir kuyruktan bir eleman çıkarma fonksiyonu

ARAŞTIRALIM

Kuyruğa yeni eklenen elemanların belirli özelliklerine göre farklı noktalarda yer bulabilmesine olanak sağlayan öncelikli kuyruk veri yapısını araştıralım.

Kuyrukları uygulamak için aynı zamanda bağlı listeler de kullanılabilir. Bir bağlı listede listenin ilk ve son düğümlerini gösteren işaretçiler kuyruk veri yapısını uygulamak için gereklidir. Bağlı listenin ilk düğümü listeden alınacak/silinecek ilk elemanı gösterir. Listenin en son düğümünü gösteren işaretçi yardımıyla da listeye yeni bir eleman eklenebilir. Şekil 4.4'te dört elemana sahip bir kuyruğun tek yönlü bağlı liste ile gösterimi verilmektedir.



Şekil 4.4 Dört elemandan oluşan bir kuyruk veri yapısının bağlı liste ile gösterimi

Bağlı listeler kullanılarak bir kuyruk oluşturulduğunda *struct* yapısı kullanarak kuyruk tanımlaması yapılabilir (Şekil 4.5). *struct queue* yapısı ile kuyruk veri tipi tanımlanmıştır. Bağlı liste ile kuyruk oluşturulduğunda, listedeki ilk ve son düğümün adresi bu tanımlama için yeterlidir. Bu işaretçiler *struct node* tipinde tanımlanmış ve 'front' ile 'rear' olarak isimlendirilmişlerdir.

```
1    struct queue
2    {
3        struct node *front;
4        struct node *rear;
5    };
```

Şekil 4.5 Bir kuyruk veri yapısı için struct yapısı

Kuyruğu oluşturan her bir düğüm farklı bir *struct* ile oluşturulur (Şekil 4.6). *struct node* yapısı bağlı listede yer alan düğümleri oluşturmak için tanımlanmıştır. Her düğümün veri olarak sadece tamsayı değere sahip olduğu varsayılmıştır. Bağlı listelerdeki düğümler bir sonraki düğümü göstermek için kendi tiplerinden bir işaretçiye sahip olurlar.

```
1    struct node
2    {
3        int value;
4        struct node *ptr;
5    };
```

Şekil 4.6 Kuyruk veri yapısında kullanılan düğümler için struct yapısı

Şekil 4.7 ve 4.8'de sırasıyla bağlı liste kullanılarak kuyruk uygulamasında kuyruğa yeni bir eleman ekleme ve kuyruktan bir eleman silme için kullanılan fonksiyonlar verilmiştir.

```
1    void enqueue (struct queue *k, int number)
2    {
3        struct node *newNode=new struct node;
4        newNode>value=number;
5        newNode->ptr=NULL;
6        k->rear>ptr=newNode;
7        k->rear=newNode;
8    }
```

Şekil 4.7 Bağlı liste ile gösterilen bir kuyruğa yeni bir eleman ekleme fonksiyonu

```
1    void dequeue (struct queue *k)
2    {
3        struct node *temp;
4        temp=k->front;
5        k->front=k->front->ptr;
6        delete temp;
7    }
```

Şekil 4.8 Bağlı liste ile gösterilen bir kuyruktan bir eleman çıkarma fonksiyonu

enqueue fonksiyonunda *new* fonksiyonu ile yeni bir düğüm oluşturulmuş, bu düğümün "value" değişkenine, eklenecek tamsayı atanmıştır. Yeni düğümün bir sonraki düğümü gösteren işaretçisine ise NULL değeri atanmıştır. 'k' kuyruk değişkeninin 'rear' düğümünün işaretçisine yeni düğüm atanmış, bu şekilde yeni düğüm kuyruğun sonuna eklenmiştir. *enqueue* fonksiyonunun tanımlanmasında bellekte yeni düğüm için yer ayrılmasında sorun olmayacağı kabul edilmiştir.

ARAŞTIRALIM

Bağlı liste ile kuyruk uygulamasında **enqueue** fonksiyonunda yeni düğüm için bellekte alan ayrılmama durumunu fonksiyon içinde kontrol etmeyi araştıralım.

Ayrıca bu fonksiyon çağrıldığında kuyrukta en az bir eleman olduğu varsayılmıştır.

dequeue fonksiyonunda kuyruk yapısının ilk düğümü ('front' düğüm) geçici bir işaretçiye atanmıştır. İlk düğüm, kuyruktaki ikinci düğümü gösterecek şekilde değiştirilmiştir. Geçici bir işaretçiye atanan düğüm ise *delete* fonksiyonu ile silinmiştir. *dequeue* fonksiyonunda kuyrukta eleman olduğu kabul edilmiştir.

ARAŞTIRALIM

Bağlı liste ile kuyruk uygulamasında **dequeue** fonksiyonunda kuyrukta eleman olup olmadığını fonksiyon içinde kontrol etmeyi araştıralım.

2. YIĞINLAR

Bir listeye en son eklenen elemanın ilk kullanılan/silinen eleman olması durumunu uygulamak için yığın veri yapısı kullanılır. Bu veri yapısı son giren ilk çıkar prensibi ile çalışan bir yapı olarak değerlendirilir. Bir programın çalışması sırasında çağrılan her fonksiyonun yeni bir fonksiyon çağırma durumu yığın veri yapısı için iyi bir örnektir. Tüm açık fonksiyonlar bir yığına eklenmiş olarak düşünülebilir. En son çağrılan fonksiyon işlemini tamamladığında yığından silinir. İşlemini tamamlama sırası, en son fonksiyonu çağıran fonksiyondadır. Bu fonksiyonda işlemini tamamlayarak yığından uzaklaştırılır. Fonksiyonların tamamlanma ve yığından alınma süreci bu şekilde devam eder. Yığına en son eklenen fonksiyon, yığından ilk silinen fonksiyon olacak şekilde, işlem tamamlanır. Buradaki örnekte de görüldüğü gibi yığın veri yapısında listenin sadece bir ucu kullanılır. Ekleme ve silme işlemleri aynı noktadan gerçekleştirilir.

Kuyrukları olduğu gibi yığın veri yapısını uygulamak için diziler ve bağlı listeler kullanılabilir. Şekil 4.9'da verilen dizi ile bir yığını ifade edebiliriz.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
57	43	14	-42	50	61				

Şekil 4.9 Dizi kullanılarak örnek bir yığın uygulaması

Bir dizi ile yığını ifade ettiğimizde tamsayı bir indis değeri bize yığının en üstteki değerini gösterir. Bu değere 'top' ismi verilir. Yukarıdaki örnekte 'top' değeri 5 olarak görülmektedir. Bu yığına yeni bir tamsayı eklemek isteyelim. Bu durumda 'top' değeri 1 artırılacak ve 6 olacaktır. Yeni değer de bu alana yazılacaktır. Eğer 75 değeri yığına eklenirse Şekil 4.10'da verilen dizi oluşur.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
57	43	14	-42	50	61	75			

Şekil 4.10 Yığın veri yapısına yeni bir eleman eklenmesi (75 eklenmiştir)

Silme işlemi de aynı noktadan gerçekleştirilir. Arka arkaya üç kez silme işlemi uyguladığımızı düşünelim. Bu durumda yeni liste Şekil 4.11'deki gibi olacaktır. Yeni 'top' değeri ise 3'tür.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
57	43	14	-42						

Şekil 4.11 Yığın veri yapısından arka arkaya üç eleman silinmesi

Diziler kullanılarak yığın veri yapısı uygulandığında, yığına yeni bir eleman ekleme işlemini gerçekleştiren *push* fonksiyonu Şekil 4.12'de verilmiştir.

```
1    void push (int stack[], int value)
2    {
3        if (top+1<SIZE-1)
4        {
5            top++;
6            stack[top]=value;
7        }
8        else
9            cout<<"Yigin dolu. Yeni eleman eklenemez.\n";
10   }
```

Şekil 4.12 Yığına yeni bir eleman ekleme fonksiyonu

Şekil 4.13'te diziler kullanılarak yığın veri yapısı uygulandığında, yığından bir eleman silme işlemini gerçekleştiren *pop* fonksiyonu verilmiştir.

```
1    int pop (int stack [])
2    {
3        int temp=-1;
4        if (top!=-1)
5        {
6            temp=stack[top];
7            top--;
8            return temp;
9        }
10       else
11       {
12           cout<<"Yigin bos. Eleman silinemez.\n";
13           return temp;
14       }
15   }
```

Şekil 4.13 Yığından bir eleman çıkarma fonksiyonu

push fonksiyonunda öncelikli olarak ilgili yığının boş alana sahip olup olmadığı kontrol edilir. Yığında boş alan mevcutsa 'top' değişkeni 1 arttırılır. 'top' indisinin gösterdiği alana eklenmek istenen değer yazılır. Eğer yığında boş alan yok ise kullanıcıya yığının dolu olduğu ve yeni eleman ekleyemeyeceği bilgisi verilir.

pop fonksiyonunda yığında eleman olup olmadığı kontrol edilir. Eğer yığında eleman var ise 'top' değişkeninin gösterdiği eleman geçici bir değişkene atanarak çevrilir. 'top' değişkeni bir azaltılır. Yığında eleman yoksa bu durum kullanıcıya bilgi olarak verilir.

Bağlı listeler kullanılarak da yığınlar oluşturulabilir. Bağlı listenin sadece bir ucu kullanılır. Ekleme ve silme işlemleri aynı uçtan gerçekleştirilir. Bu uçtaki ilk düğümü gösteren işaretçi değişkeni 'top' olarak isimlendirilir. Yığınları bağlı listeler ile ifade ederken Şekil 4.14'te verilen *struct* yapısı ile oluşturulan düğümler kullanılır.

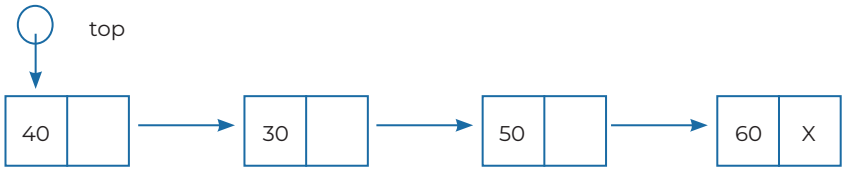
```
1    struct node
2    {
3        int value;
4        struct node *ptr;
5    };
```

Şekil 4.14 Yığın veri yapısında kullanılan düğümler için struct yapısı

Bir yığın bağlı liste kullanılarak oluşturulduğunda, bu yığına erişmek için sadece bir *struct node* işaretçisi yeterlidir. Şekil 4.15'te verilen yığında 'top' işaretçisi tamsayı değeri 40 olan düğümü göstermektedir.

DİKKAT EDELİM

Bir yığında ekleme ve silme işlemeleri listenin aynı noktasından gerçekleşmektedir.



Şekil 4.15 Dört elemandan oluşan bir yığın veri yapısının bağlı liste ile gösterimi

Bu yığına yeni bir eleman eklendiğinde 'top' işaretçisi bu yeni elemanı, yeni eleman da yığının eski 'top' düğümünü gösterecektir. Silme işlemi de aynı uçtan gerçekleşir. Eğer yukarıda verilen yığından arka arkaya 2 eleman silinirse, sırasıyla 40 ve 30 değerlerine sahip düğümler uzaklaştırılacak ve 'top' işaretçisi 50 tamsayı değerine sahip düğümü gösterecektir. Şekil 4.16 ve 4.17'de sırasıyla bağlı liste ile yığın uygulamasında kullanılan *push* ve *pop* fonksiyonları verilmektedir.

```
1    struct node *push(struct node *t, int number)
2    {
3        struct node *newNode=new struct node;
4        newNode->value=number;
5        newNode->ptr=t;
6        t=newNode;
7        return t;
8    }
```

Şekil 4.16 Bağlı liste ile gösterilen bir yığına yeni bir eleman ekleme fonksiyonu

```
1    struct node *pop(struct node *t)
2    {
3        struct node *temp;
4        temp=t;
5        t=t->ptr;
6        delete temp;
7        return t;
8    }
```

Şekil 4.17 Bağlı liste ile gösterilen bir yığından bir eleman çıkarma fonksiyonu

Bir yığını temsil etmek için sadece bir düğüm işaretçisi gerektiğinden yığın oluşturmak için ayrı bir *struct* yapısına ihtiyaç duyulmamıştır. *push* fonksiyonunda yeni bir düğüm oluşturulmuş, eklenecek tamsayı bu düğümün değeri olarak atanmıştır. Yeni düğümün işaretçi değeri eski 'top' elemanı gösterecek şekilde atanmıştır. 'top' düğüm is yeni eklenen düğüm olarak değiştirilmiştir. Bu düğümü gösteren işaretçi fonksiyondan çevrilmiştir. Ayrıca *push* fonksiyonunda yeni oluşturulan düğüm için bellekte yer ayırma işleminin başarılı olduğu varsayılmıştır. *pop* fonksiyonu eski 'top' düğümü geçici bir işaretçiye atamış ve 'top' düğümü değiştirmiştir. Geçici işaretçiye atanan düğüm ise *delete* ile silinerek, ilgili alan belleğe geri verilmiştir. *pop* fonksiyonunda, yığında eleman olduğu varsayılmıştır.

3. UYGULAMA

Yığınları kullanarak bir karakter dizisini tersten yazdırma işlemi bu bölümde uygulama olarak gösterilecektir. Şekil 4.18'de verilen programda, önce karakter dizisi kullanıcıdan okunacaktır. Birinci karakterden başlayarak bu dizinin her bir elemanı *push* fonksiyonu ile bir yığına eklenecektir. En son yığına eklenen karakter, dizinin son elemanıdır ve yığının en üstünde yer alan 'top' eleman olacaktır. Yığındaki karakterler birer birer *pop* fonksiyonu ile alınıp ekrana yazdırılır. Ekrandaki çıktı dizinin elemanlarının ters sırada yazdırılması ile oluşur. Yığına en son eklenen eleman, dizinin son elemanı, yığından ilk alınan eleman ve ekrana yazdırılan ilk karakter olacaktır. Aynı şekilde yığına sondan bir önce eklenen eleman yığından alınan ve ekrana yazdırılan ikinci karakterdir. Bu şekilde devam edildiğinde dizinin ilk elemanı ekrana en son yazdırılan eleman olacaktır ve bu istenen durumdur. Programda SIZE sembolik bir sabittir ve değeri 20 olarak belirlenmiştir. 'top' değişkeni global olarak tanımlanmıştır. 'main' fonksiyon ile *push* ve *pop* fonksiyonlarında 'top' değişkenine doğrudan erişilebilir.

```
1   #include <iostream>
2   using namespace std;
3   #define SIZE 20
4
5   int top=-1;
6
7   void push (char stack[], char value)
8   {
9       if(top+1<SIZE-1)
10      {
11          top++;
12          stack[top]=value;
13      }
14      else
15          cout<<"Yigin dolu. Yeni eleman eklenemez.\n";
16  }
```

```
17  char pop(char stack[])
18  {
19      char temp=' ';
20      if(top!=-1)
21  {
22          temp=stack[top];
23          top--;
24          return temp;
25      }
26      else
27      {
28          cout<<"Yigin bos. Eleman silinemez.\n";
29          return temp;
30      }
31  }
```

Şekil 4.18 Bir karakter dizisini tersten yazdırmak için yığın kullanımı gerçekleştiren program

```
32  int main ()
33  {
34          char a[SIZE];
35          char st[SIZE];
36          int i, count=0;
37
38          gets(a);
39          for(i=0; a[i]!='\0';i++)
40          {
41                  push(st, a[i]);
42                  count++;
43          }
44          for(i=1; i<=count; i++)
45                  cout<<pop(st);
46          cout<<endl;
47
48          return 0;
49  }
```

ARAŞTIRALIM

postfix ve **prefix** gösterimi ile verilen matematiksel ifadelerin değerlendirilmesinde yığın kullanımını araştıralım.

BÖLÜM ÖZETİ

Bu bölümde kuyruk ve yığın veri yapıları anlatılmıştır. Önce kuyruk yapısının temel özellikleri verilmiş ve bu yapının diziler ve bağlı listeler ile uygulanması gösterilmiştir. Kuyruk ilk giren ilk çıkar temel kuralını temsil eden bir veri yapısıdır. Kuyrukta listenin bir ucundan ekleme diğer ucundan silme işlemi gerçekleşir. Sonra yığın yapısının temel kuralları anlatılmış, kuyrukta olduğu gibi diziler ve bağlı listeler kullanılarak uygulanması verilmiştir. Yığın son giren ilk çıkar prensibine göre çalışan bir veri yapısıdır. Listenin aynı ucu ekleme ve silme işlemi için kullanılır. Bu bölümde son olarak bir kullanıcıdan okunan karakter dizisinin tersten yazdırılma işlemi gösterilmiştir. Bu işlem yığın kullanılarak gerçekleştirilmiştir. Yığın uygulaması diziler ile yapılmış, uygulama gereği yığının her bir elemanı için karakter kullanılmıştır.

GÖZDEN GEÇİRELİM

1. **Listenin bir ucundan ekleme diğer ucundan silme işlemi gerçekleşen doğrusal veri yapısı aşağıdakilerden hangisidir?**

   a) Kuyruk
   b) Dizi
   c) Bağlı liste
   d) Yığın
   e) Ağaç

2. **Kuyruk ile ilgili aşağıdaki ifadelerden hangisi yanlıştır?**

   a) Dizi kullanılarak uygulanabilirler.
   b) Bağlı liste kullanarak uygulanabilirler.
   c) Ekleme ve silme işlemi farklı noktalarda gerçekleşir.
   d) Dizi kullanıldığında kuyruk veri yapısının dolu olma ihtimali vardır.
   e) Ekleme ve silme işlemi listenin aynı ucundan gerçekleşebilir.

3. **Boş bir yığına sırasıyla 14, 35 ve 12 değerleri, push işlemi ile eklenmiştir. Daha sonra sırasıyla, 2 kez pop işlemi uygulanmıştır. Son olarak, 20 ile 15 değerleri push fonksiyonu ile eklenmişir. Bu durumda** yığında hangi elemanlar vardır?

   a) 14 20 15
   b) 12 20 15
   c) 35 20 15
   d) 14 12 20
   e) 14 35 20

4. **Boş bir kuyruğa sırasıyla 14, 35 ve 12 değerleri ile enqueue işlemi ile eklenmiştir. Daha sonra sırasıyla 2 kez dequeue işlemi uygulanmıştır. Son olarak, 15 ile 20 değerleri enqueue fonksiyonu ile eklenmiştir. Bu durumda kuyrukta hangi elemanlar vardır?**

   a) 14 15 20
   b) 12 15 20
   c) 20 15 35
   d) 20 14 12
   e) 35 20 14

5. **Listenin aynı ucundan ekleme ve silme işlemi gerçekleşen doğrusal veri yapısı aşağıdakilerden hangisidir?**

   a) Kuyruk
   b) Dizi
   c) Bağlı liste
   d) Yığın
   e) Ağaç

6.

- **Bir programda sırasıyla birbirlerini çağıran fonksiyonlar**
- **Bir markette kasanın önünde oluşan sıra**

Yukarıda verilen işlemler sırasıyla hangi veri yapıları ile ifade edilebilirler?

   a) Kuyruk - Yığın
   b) Ağaç - Yığın
   c) Ağaç - Kuyruk
   d) Yığın - Ağaç
   e) Yığın - Kuyruk

7. **Aşağıdakilerin hangisi bir kuyruğu bağlı liste ile ifade etmenin avantajlarından birisidir?**

   a) Boyutunun önceden belirlenme zorunluluğu olmaması

   b) Bellekte ardışık alanlarda yerleşim

   c) Sabit belirlenmiş boyutunun olması

   d) Kolay sıralama işlemine sahip olması

   e) Kolay arama işlemine sahip olması

8. **Aşağıdakilerin hangisi kuyruk veri yapısının özelliklerinden birisi değildir?**

   a) Sadece bir uçtan ekleme yapılabilir

   b) Silinen eleman kuyruğun sonuna tekrar eklenir

   c) Diziler veya bağlı listeler ile uygulanabilir

   d) Aynı uç silme ve ekleme için kullanılamaz

   e) Sadece bir uçtan silme yapılabilir

9.
```
struct node{
    int v;
    struct node *p;
}*top;
```

Yığın veri yapısını uygulamak için yukarıda verilen düğüm yapısı kullanılsın. Aşağıdakilerden hangisi pop fonksiyonu içinde yazılabileck ifadelerden birisidir?

   a) struct node *top=new struct node;

   b) top->value++;

   c) top=top->p;

   d) (top->p)++;

   e) top=top+1;

10.
```
struct node{
    int v;
    struct node *p;
}*top;
```

Yığın veri yapısını uygulamak için yukarıda verilen düğüm yapısı kullanılsın. Aşağıdakilerden hangisi push fonksiyonu içinde yazılabileck ifadelerden birisidir?

   a) struct node *newNode=new struct node;

   b) top->value++;

   c) top=top->p;

   d) (top->p)++;

   e) top=top+1;

Yanıt Anahtarı: 1-A 2-E 3-A 4-B 5-D 6-E 7-A 8-B 9-C 10-A

Kaynakça

Deitel, P., Deitel, H. (2017). C++ How to Program. 10th Edition. Pearson.

Thareja R. (2014) Data Structures Using C, 2nd Edition, Oxford University Press.

Levitin A. (2012) Introduction to The Design & Analysis of Algorithms, 3rd Edition, Pearson.

Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. (2009) Introduction to Algorithms, 3rd Edition, MIT Press.