CENG 201 Veri Yapıları 2: Yığıtlar ve Kuyruklar

Öğr.Gör. Şevket Umut ÇAKIR

Pamukkale Üniversitesi

Hafta 2

Anahat

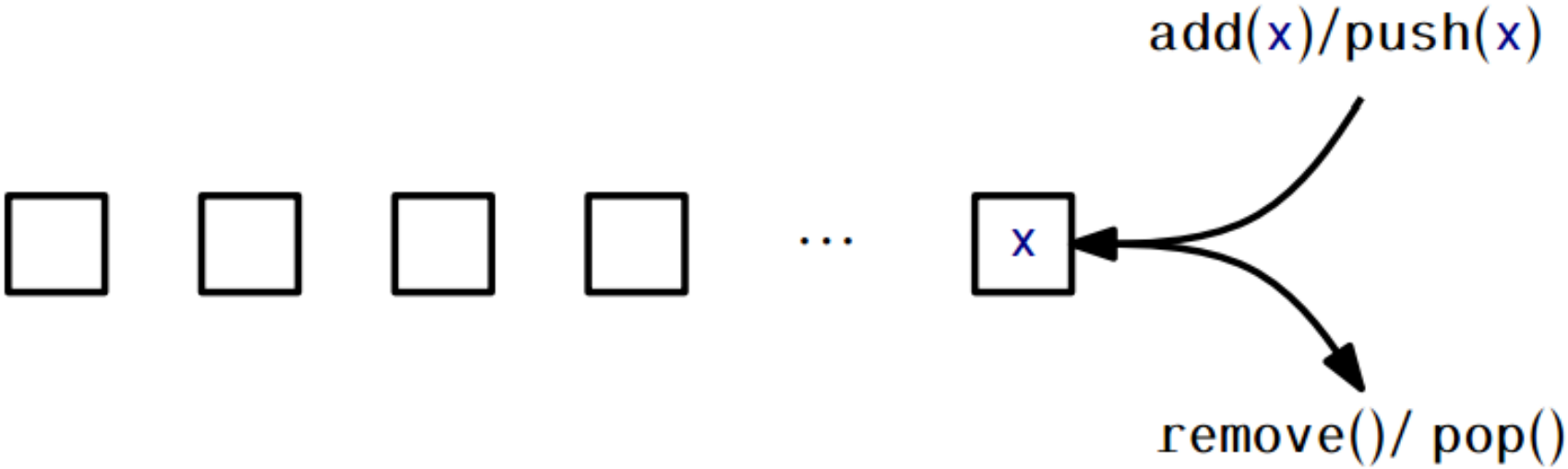
Soyut Veri Tipleri(SVT)

- Bir dizi işlem içeren nesnelerdir
- Tanımında işlemlerin **nasıl** yapıldığı belirtilmeyen matematiksel soyutlamalardır
- İşlemlerin gerçekleştirilmesinde(implementation) bir değişiklik olması durumunda son kullanıcı bundan etkilenmez
- Set(küme) SVT için *add*, *remove*, *size*, *contains*, *union* ve *find* işlemleri bulunur
- Listeler, kümeler, graflar SVT'lere örnek gösterilebilir

Yığıt Özellikleri

- Mutfak rafına üst üste dizilen tabaklar gibi düşünülebilir
- İçerisine farklı tiplerdeki verileri ekleyip çekebildiğimiz bir soyut veri yapısı
- Son giren ilk çıkar (Last In First Out, LIFO) mantığına göre çalışır
- Ekleme *push*, silme *pop* ve ilk elemanı görme *peek* işlevlerine sahiptir

Yığıt Görseli



Soru

- Aşağıdaki işlemler sonucunda yığıtın son hali ne olur?

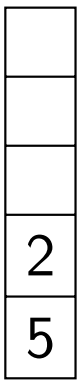
```
push 5
push 2
push 7
pop
push 6
push 9
push 4
pop
pop
push 12
```

Cevap

(a)
push
5
2
7

Cevap

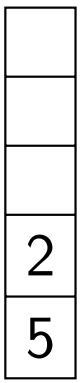
(a)
push
5
2
7



(b)
pop
$\rightarrow$
7

Cevap

(a)
push
5
2
7

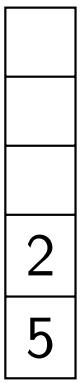


(b)
pop
$\rightarrow$
7

(c)
push
6
9
4

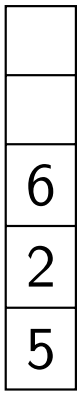
Cevap

(a)
push
5
2
7



(b)
pop
$\rightarrow$
7

(c)
push
6
9
4



(d)
pop
$\rightarrow$
4,
pop
$\rightarrow$
9

Cevap

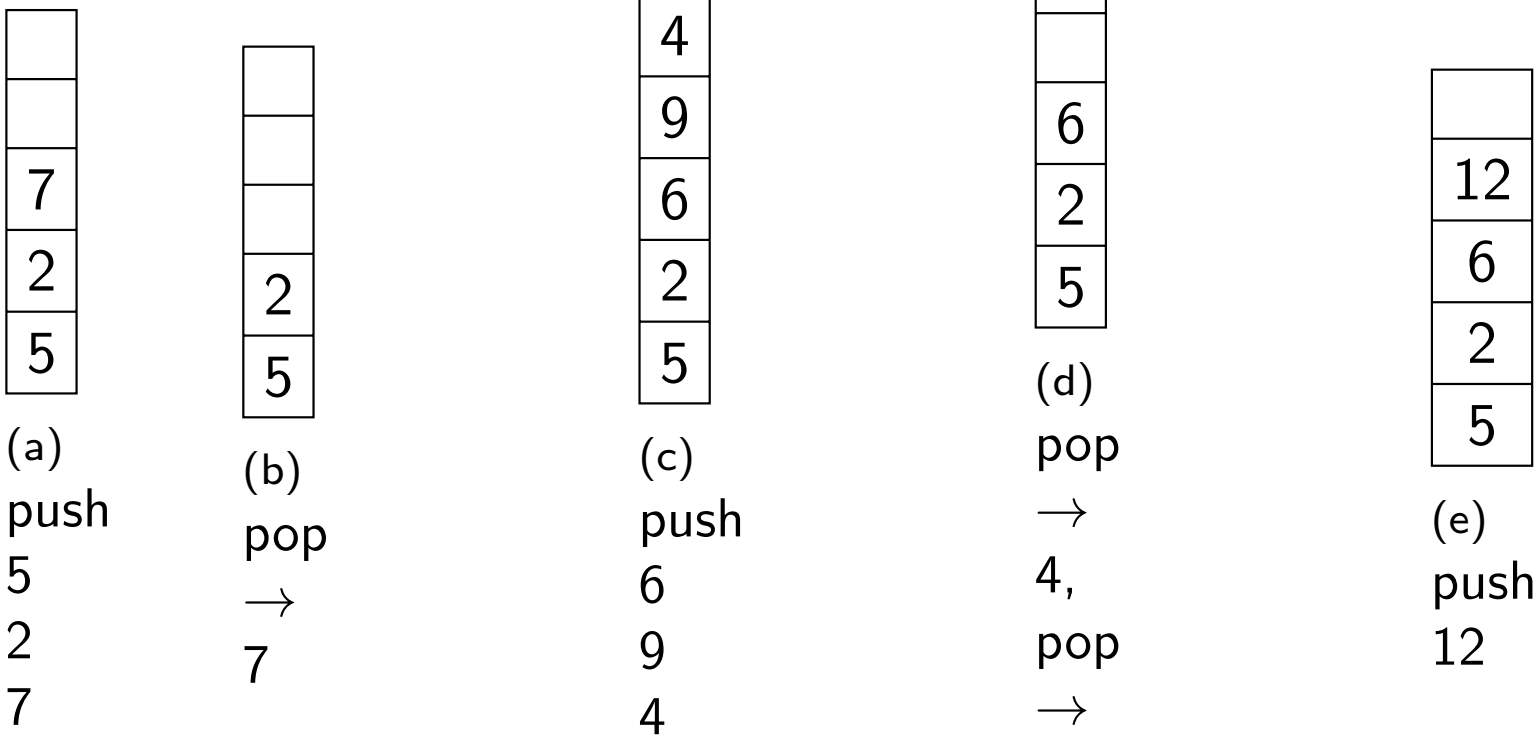


Figure: Cevap

Gerçekleştirme(Implementation)

- Yığıt soyut bir veri tipidir
- Farklı veri yapıları kullanılarak gerçekleştirilebilir(Dizi, bağlı liste vb.)
- Ekleme *push*, silme *pop* ve ilk elemanı görme *peek* işlevlerini içermesi gerekir
- Hangi tip verileri saklayacak(integer, string, double)? Generic?
- Dizi kullanılarak nasıl gerçekleştirilebilir?

MyStack.cs I

```
1  using System;
2
3  namespace mystack
4  {
5      public class MyStack<T>
6      {
7          T[] dizi;
8          int es=0;
9          public MyStack ()
10         {
11             dizi = new T[10];
12         }
13         public MyStack(int boyut)
14         {
15             dizi = new T[boyut];
16         }
17         public void push(T eleman)
18         {
19             if (es == dizi.Length)
20                 throw new Exception ("Stack overflow");
21             dizi [es++] = eleman;
22         }
```

MyStack.cs II

```
23          public T pop()
24          {
25              if (es == 0)
26                  throw new Exception ("Stack underflow");
27              return dizi [es--];
28          }
29          public T peek()
30          {
31              if (es == 0)
32                  throw new Exception ("Stack underflow");
33              return dizi [es];
34          }
35          public void print()
36          {
37              for (int i = es-1; i >= 0; i--) {
38                  Console.WriteLine (dizi[i]);
39              }
40          }
41      }
42  }
```

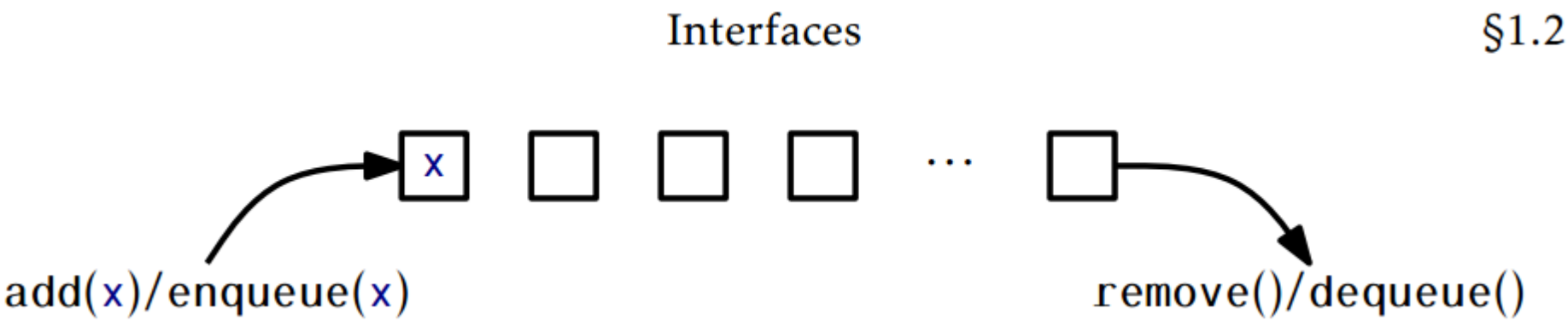
MyStack.cs Test Programı

```
1  using System;
2
3  namespace mystack
4  {
5      class MainClass
6      {
7          public static void Main (string[] args)
8          {
9              MyStack<int> stack = new MyStack<int> ();
10             stack.push (5);
11             stack.push (2);
12             stack.push (7);
13             stack.pop ();
14             stack.push (6);
15             stack.push (9);
16             stack.push (4);
17             stack.pop ();
18             stack.pop ();
19             stack.push (12);
20             stack.print ();
21         }
22     }
23 }
```

Kuyruk

- Gerçek hayattaki herhangi bir kuyruk/sıra gibidir
- İlk giren değer ilk çıkar(First In First Out, FIFO)
- Ekleme *enqueue* ve silme *dequeue* işlevlerini içermesi gerekir

Kuyruk Görseli



Kuyruk örneği

- Aşağıdaki işlemler sonucunda yığıtın son hali ne olur?

```
enqueue 5
enqueue 7
enqueue 4
dequeue
dequeue
dequeue
enqueue 6
```

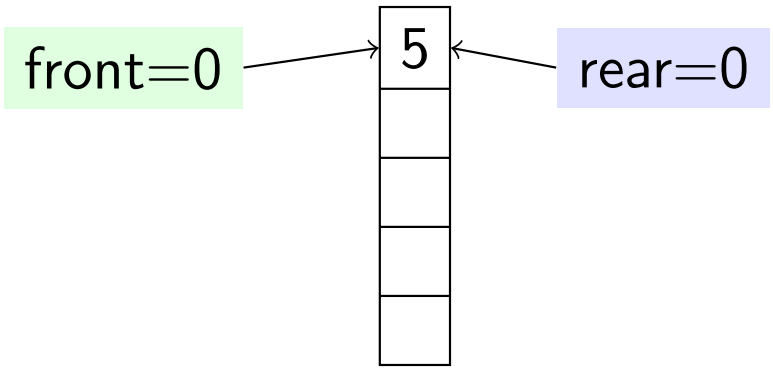


Figure: Enqueue 5

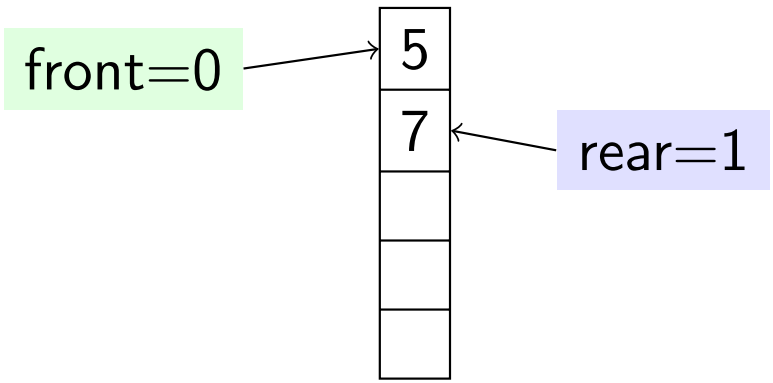


Figure: Enqueue 7

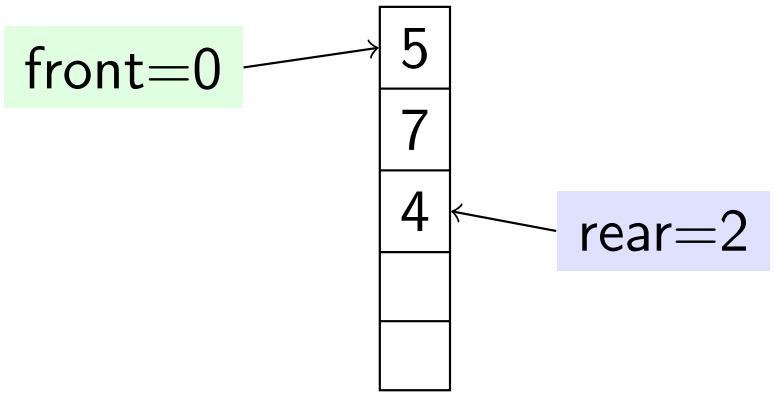


Figure: Enqueue 4

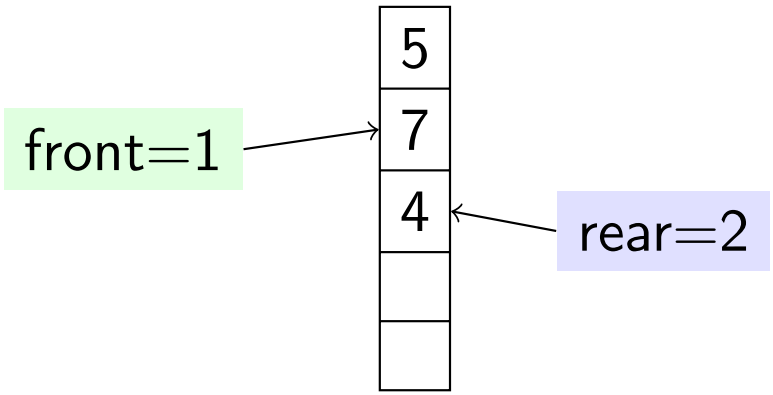


Figure: Dequeue $\rightarrow$ 5

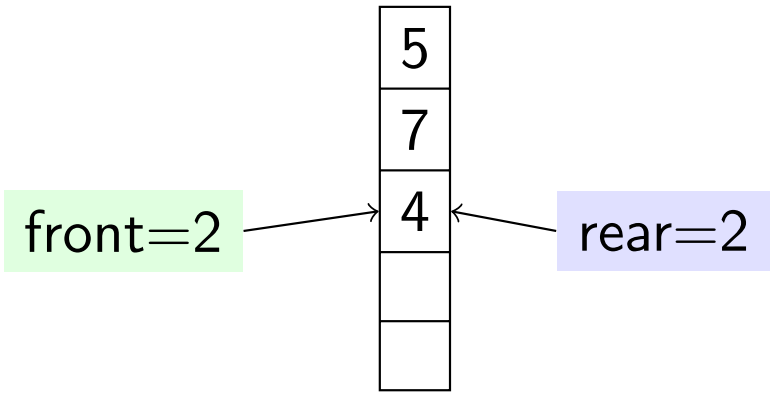


Figure: Dequeue $\rightarrow$ 7

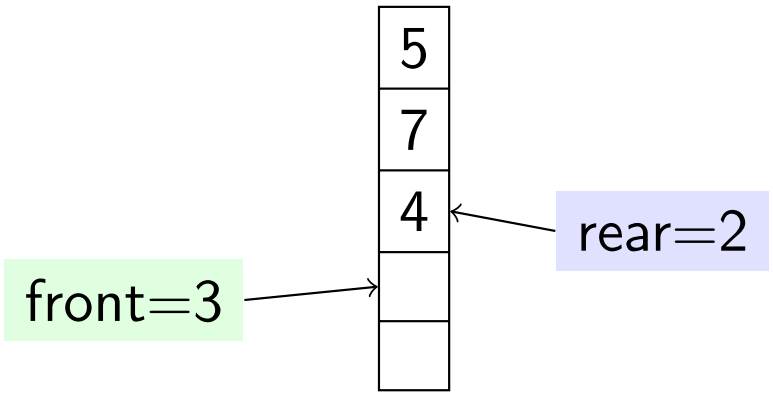


Figure: Dequeue $\rightarrow$ 4, set front=rear=-1

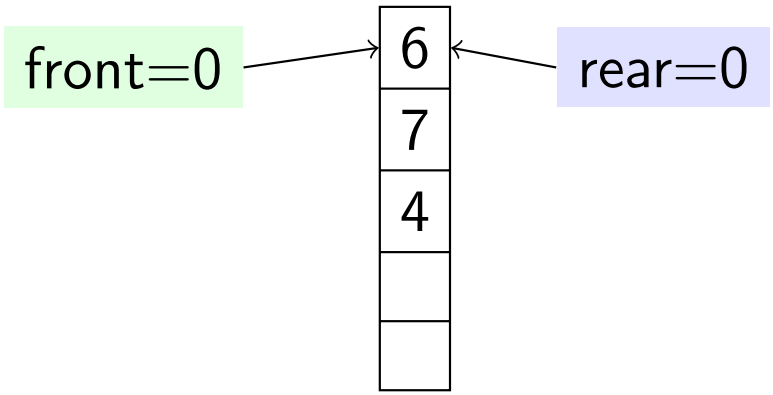


Figure: Enqueue 6

Kuyruk gerçekleştirmesi

- Dizilerle gerçekleştirilecek
- Generic olacak

MyQueue.cs I

```
1  using System;
2
3  namespace H2myqueue
4  {
5      public class MyQueue<T>
6      {
7          int front=-1;
8          int rear=-1;
9          T[] dizi;
10         public MyQueue ()
11         {
12             dizi = new T[20];
13         }
14         public MyQueue(int boyut)
15         {
16             dizi = new T[boyut];
17         }
18         public bool isEmpty()
19         {
20             return front == -1 && rear == -1;
21         }
```

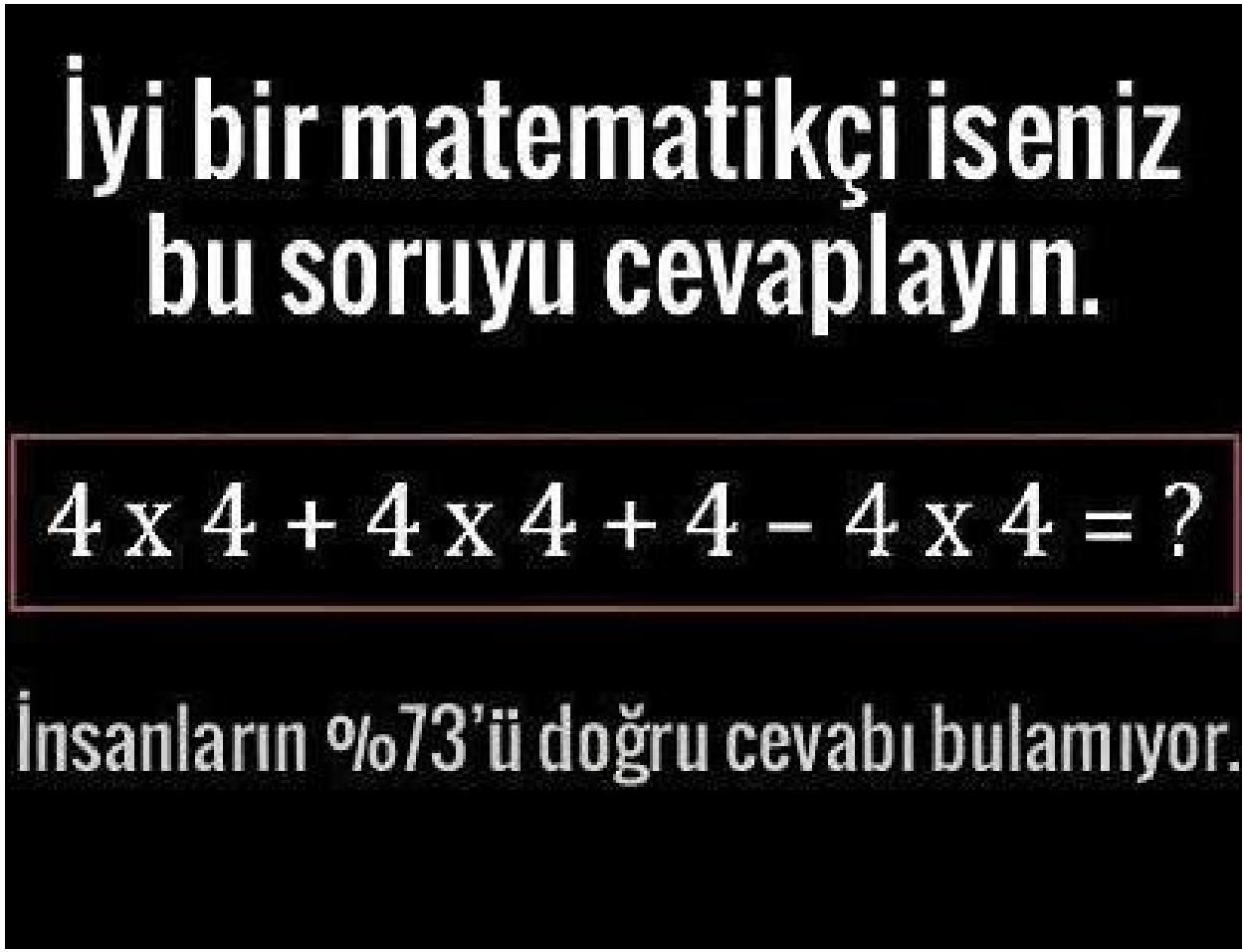
MyQueue.cs II

```
22          public bool isFull()
23          {
24              return rear == dizi.Length - 1;
25          }
26          public void enqueue(T item)
27          {
28              if (isFull ())
29                  throw new Exception ("Queue is full!");
30              if (isEmpty ()) {
31                  front = 0;
32              }
33              dizi [++rear] = item;
34          }
35          public T dequeue()
36          {
37              if (isEmpty ())
38                  throw new Exception ("Queue is empty");
39              T donen = dizi [front++];
40              if (front > rear)
41                  front = rear = -1;
42              return donen;
43          }
44          public int Count{
45              get {
46                  if (isEmpty ())
47                      return 0;
48                  return rear - front + 1;
49              }
50          }
```

MyQueue.cs III

```
51          public void print()
52          {
53              for (int i = front; i <= rear; i++) {
54                  Console.WriteLine (dizi[i]);
55              }
56          }
57      }
58  }
```

İşlem Önceliği



Notasyonlar/Gösterimler

- **Infix**: İşlem(operator) ortada, değerler(operand) sağda ve solda
- **Prefix**: İşlem önde, değerler sonda
- **Postfix**: İşlem sonda, değerler önde

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

4 x 4 + 4 x 4 + 4 - 4 x 4

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

4 x 4 + 4 x 4 + 4 - 4 x 4

Example (Postfix gösterimi)

4 4 * 4 4 * + 4 + 4 4 * -

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

3 * 2 + 4 - 7 / 5

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

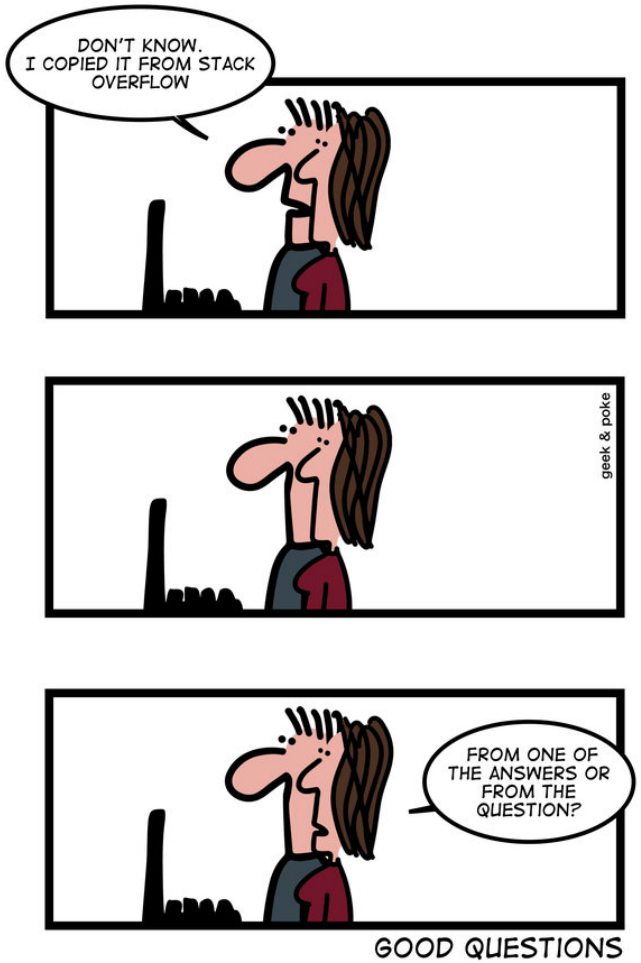
3 * 2 + 4 - 7 / 5

Example (Postfix gösterimi)

3 2 * 4 + 7 5 / -

Postfix ifadelerin yığıt ile çözümü

```
Boş bir değer yığıtı oluştur: S
Girdi metninde değer kalmayana kadar oku:O
  Eğer O bir sayı ise
    S yığıtına O değerini it(push)
  Aksi takdirde
    Yığıttan D1 değerini çek(pop)
    Yığıttan D2 değerini çek(pop)
    D2 O D1 işlemini yap ve sonucu yığıta it(push)
Yığıttan S değerini çek ve yazdır
```


DON'T KNOW.
I COPIED IT FROM STACK OVERFLOW
geek & poke
FROM ONE OF THE ANSWERS OR FROM THE QUESTION?
GOOD QUESTIONS

1

[1] geek & poke