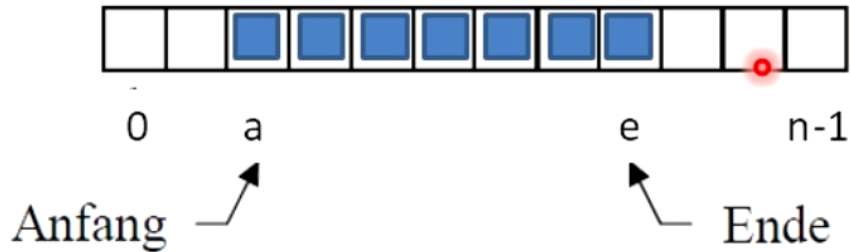


## Eigenschaften

- implementiert mit [[Array]]
- FIFO Prinzip
  - first in first out

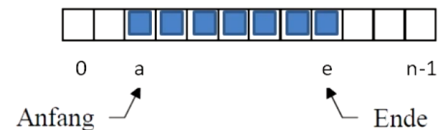


## Operationen

Queue (Schlange):

$Q[0 \dots n-1]$

Init:  $\text{anz} \leftarrow 0; e \leftarrow -1; a \leftarrow 0$



Einfügen:

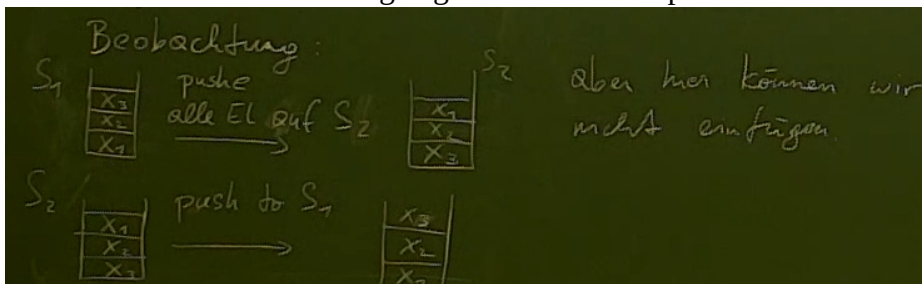
```

PUT(Q,x)
1: IF  $\text{anz} = n$  THEN „overflow“
2: ELSE  $\text{anz} \leftarrow \text{anz} + 1$ 
3:   IF  $e < n-1$  THEN  $e \leftarrow e + 1$ 
4:   ELSE  $e \leftarrow 0$ 
5:    $Q[e] \leftarrow x$ 
  
```

- $O(1)$  für alle Operationen

## Queue Implementation mittels [[Stack]]

- verwendet zwei Stacks  $S_1, S_2$ 
  - $S_1$  zum Einfügen
  - $S_2$  zum Auslesen/Löschen
- Elemente von  $S_1$  wenn Auslesen/Löschen gefordert nach  $S_2$  pushen
- Elemente von  $S_2$  wenn Einfügen gefordert nach  $S_1$  pushen



- Algorithmus/Funktionen

Wir haben zwei Stacks  $S_1, S_2$   
 Eine Hilfsvar.  $is\_queue$ : Wenn TRUE sind die  
 Elemente auf  $S_2$  in "Queue-Ordnung"

---

**init()**

```

S1 ← emptyStack()
S2 ← emptyStack()
is_queue ← FALSE
  
```

**makeQueue()**

```

WHILE NOT isEmpty(S1)
  Push(Pop(S1), S2)
is_queue ← TRUE
  
```

**makeStack()**

```

WHILE NOT isEmpty(S2)
  Push(Pop(S2), S1)
is_queue ← FALSE
  
```

**Put(x)**

```

IF is_queue
  makeStack()
  Push(x, S1)
else()
  IF NOT is_queue
    makeQueue()
  RETURN Pop(S2)
  
```