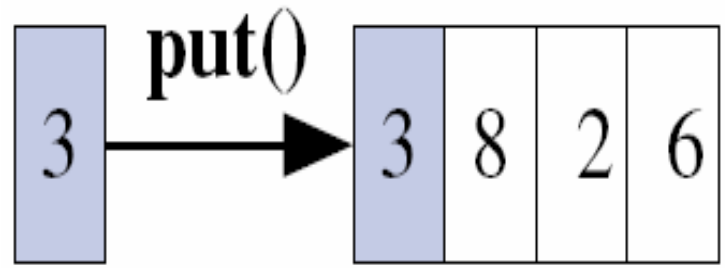
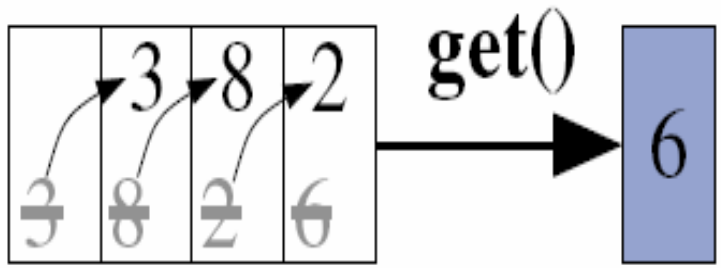
Queue (Warteschlange)



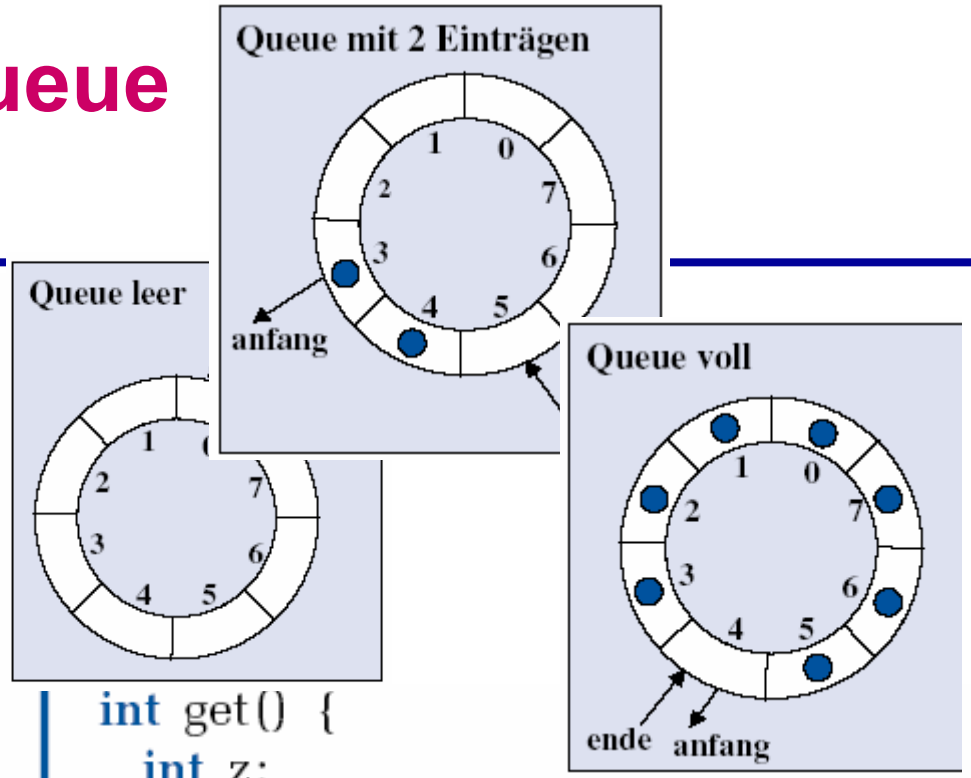


put(): fügt ein Element am Ende der Warteschlange hinzu.

get(): entnimmt ein Element am Anfang der Warteschlange und liefert es zurück.

Während Stacks **LIFO-Prinzip (last-in-first-out)** verwenden, arbeiten Queues nach dem **FIFO-Prinzip (first-in-first-out)**.

Realisierung einer Queue als Array (in C)

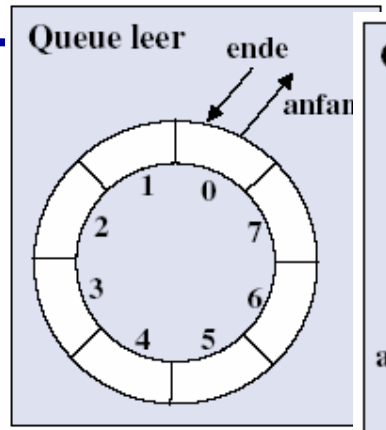


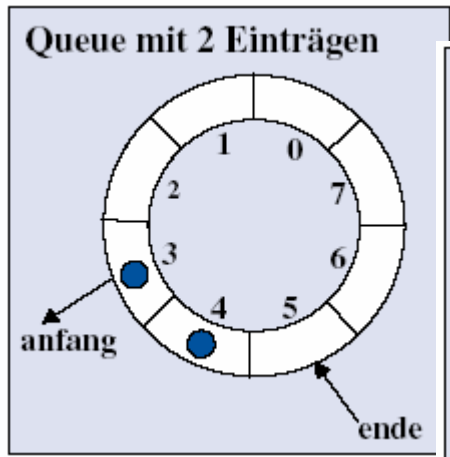

```
void queueinit(int n) {
  queue =
    (int*)malloc((n+1)*sizeof(int));
  anfang = ende = 0;
  max    = n;
}
int isEmpty() {
  return anfang==ende && !voll;
}
int isFull() { return voll;  }
void put(int wert) {
  if (queue == NULL) {
    printf("Queue fehlt\n"); return;
  }
  if (isFull()) {
    printf("Queue voll\n"); return;
  }
  queue[ende] = wert;
  ende = (ende+1) % max;
  voll = (anfang==ende);
}
```

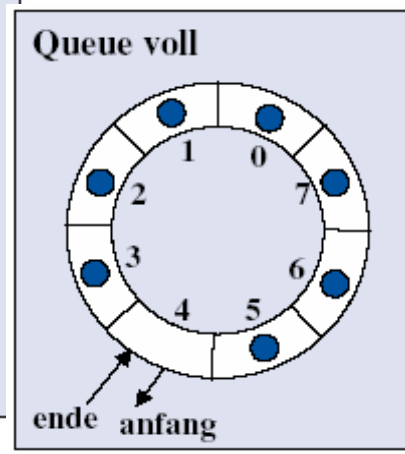
```
int get() {
  int z;
  if (queue == NULL) {
    printf("Queue fehlt\n");return -1;
  }
  if (isEmpty()) {
    printf("Queue leer\n"); return -1;
  }
  z      = queue[anfang];
  anfang = (anfang+1) % max;
  voll   = 0;
  return z;
}
```

Realisierung einer Queue als Array (in Java)

```
Queue(int n) {
    queue = new int[n+1];
    anfang = ende = 0;
    max    = n;
}
boolean isEmpty() {
    return anfang==ende && !voll;
}
boolean isFull()   { return voll;   }
void put(int wert) throws QueueFehler {
    if  (queue == null)
      throw new QueueFehler("Queue fehlt");
    if  (isFull ())
      throw new QueueFehler("Queue voll");
    queue[ende] = wert;
    ende = (ende+1) % max;
    voll  = (anfang==ende);
}
```








```
int get() throws QueueFehler {
    if  (queue == null)
      throw new QueueFehler("Queue fehlt"
    if  (isEmpty())
      throw new QueueFehler("Queue leer")
    int z   = queue[anfang];
    anfang = (anfang+1) % max;
    voll    = false;
    return z;
}
```

Realisierung einer Queue als verkettete Liste (in C)

```
struct elem {
    int           zahl;
    struct elem *next;
};
static struct elem *anfang = NULL,
                   *ende   = NULL;
static int anzahl = 0;
int isEmpty() { return anzahl == 0; }
void put(int wert) {
    struct elem *neu =
      (struct elem *)malloc(sizeof *neu);
    neu->zahl = wert;
    neu->next = NULL;
    if (isEmpty())
        anfang = neu;
    else
        ende->next = neu;
    ende = neu;
    anzahl++;
}
```

```
int get() {
    int      wert;
    struct elem *cursor = anfang;
    if (isEmpty()) {
      printf("Queue leer\n"); return -1;
    }
    anfang = anfang->next;
    if (anfang == NULL)
        ende = NULL;
    wert = cursor->zahl;
    free(cursor);
    anzahl--;
    return wert;
}
```

Realisierung einer Queue als verkettete Liste (in Java)

```
class Elem {
   int   zahl;
   Elem next = null;
   Elem(int z) {  zahl = z;  }
}
class QueueListe {
   private Elem anfang = null, ende = null;
   private int  anzahl = 0;
   boolean isEmpty() { return anzahl == 0; }
   void put(int wert) {
      Elem neu = new Elem(wert);
      if  (isEmpty())
          anfang = neu;
      else
          ende.next = neu;
      ende = neu;
      anzahl++;
   }
```

```
   int get()  throws QueueFehler {
      if  (isEmpty())
        throw new QueueFehler("Queue leer"
      Elem cursor = anfang;
      anfang = anfang.next;
      if  (anfang == null)
          ende = null;
      anzahl--;
      return cursor.zahl;
   }
}
```