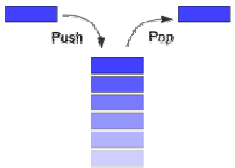
Elementare Datenstrukturen

Inhalt:

- Stack
- Queue



Folieninhalte teilweise übernommen von PD Dr. Klaus Kriegel (Informatik B, SoSe 2000)

Stack I

Ein Stack (Stapel oder Keller) verwaltet eine geordnete Menge von Objekten, wobei ein Element angefügt oder das letzte hinzugefügte wieder weggenommen werden kann.

Definition des Binomialkoeffizienten:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}.$$

Rekursive Definitionsvorschrift:

$$\binom{n}{0} = 1 \qquad \binom{n}{1} = n \qquad \binom{n}{n} = 1$$

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$$

Effiziente Vorschrift:

$$\binom{n}{k} = \prod_{i=1}^{k} \frac{n+1-i}{i}$$

$$\binom{n}{n-k} = \binom{n}{k}$$

Übungsaufgabe

Rekursive Funktion in Java:

```
public static int bino(int n, int k) {
   if (n==0)
      if (k==0) return 1;
      else      return 0;

   return bino(n-1, k) + bino(n-1, k-1);
}
```

Freie Universität Berlin

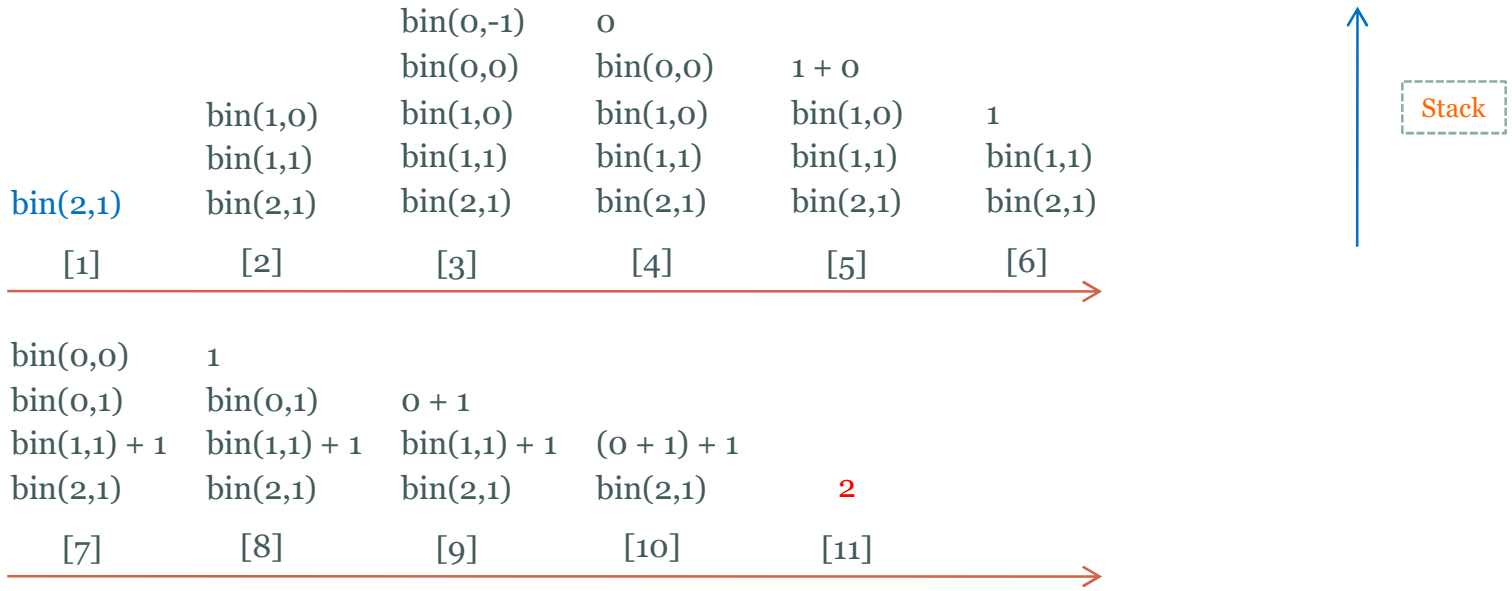
Stack II

Rekursive Funktion in Java:

```
public static int bino(int n, int k) {
   if (n==0)
      if (k==0) return 1;
      else      return 0;

   return bino(n-1, k) + bino(n-1, k-1);
}
```

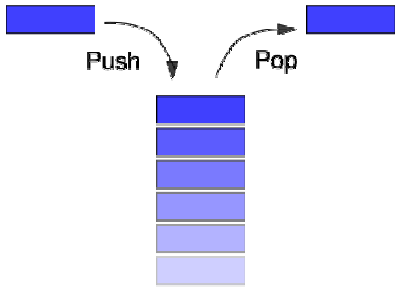
Abarbeitung der Funktion:



Stack III

Methoden eines Stacks:

push(Objekt)	fügt Objekt als oberstes Element dem Stack hinzu
Objekt pop()	gibt das oberste Element des Stacks zurück und entfernt es
boolean isEmpty()	prüft, ob der Stack leer ist
int size()	liefert die Anzahl der im Stack vorhandenen Elemente zurück



(Abbildung von Wikipedia)

Beispielanwendung:

Auswertung eines Rechenausdrucke in UPN (umgekehrt polnische Notation) oder Postfixnotation.

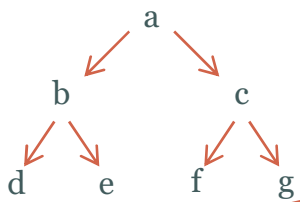
normale Schreibweise: (7*4+2) / (3*4+3)

UPN: 7 4 * 2 + 3 4 * 3 + /

```
public int evaluateUPN(String[] e) {
   for (int i=0; i<=n; i++){
      if (isNumber(e[i]))
         push(e[i]);
      else
         push(pop() e[i] pop())  // e[i] ist dabei der ermittelte Operator
   }
   return pop();
}
```

Prefix, Infix, Postfix

Verschiedene Notationsformen in binären Bäumen:

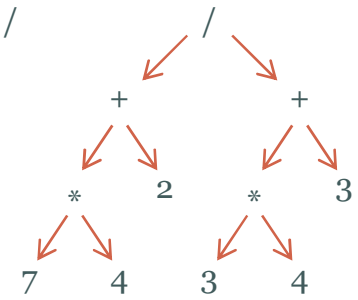


also erst den Knoten, dann links und anschließend rechts

Die Rekursionsformel für prefix lautet KLR. [a, b, d, e, c, f, g]
Die Rekursionsformel für infix lautet LKR. [d, b, e, a, f, c, g]
Die Rekursionsformel für postfix lautet LRK. [d, e, b, f, g, c, a]

Syntaxbaum für die Beispielformel

normale Schreibweise: (7*4+2) / (3*4+3)
UPN (postfix): 7 4 * 2 + 3 4 * 3 + /



Realisierung durch ein Array I

Wir speichern die Objekte in einem Array und merken uns, wie viele Elemente zur Zeit in dem Array sind. Der Arrayanfang ist das untere Ende und das Arrayende das aktuell letzte Elementindex des Stacks.

```
public class Stack {
   private int currentIndex;
   private int[] array;

   public Stack(){
      currentIndex      = 0;
      array             = new int[100];
   }

   public void push(int a){
      if (currentIndex<99) {
         currentIndex++;
         array[currentIndex] = a;
      }
   }

   public int pop() {
      if (currentIndex<=0)
         return -1;
      currentIndex--;

      return array[currentIndex+1];
   }
   ...
```

Realisierung durch ein Array II

Weiter geht's:

```
   ...
   public Boolean isEmpty(){
      return currentIndex == 0;
   }

   public int size(){
      return currentIndex;
   }

   public static void main(String[] args){
      Stack s = new Stack();
      System.out.println("size: "+s.size());
      s.push(7);
      s.push(4);
      System.out.println("size: "+s.size());
      System.out.println("pop "+s.pop());
      System.out.println("pop "+s.pop());
      System.out.println("pop "+s.pop());
      System.out.println("size: "+s.size());
   }
}
```

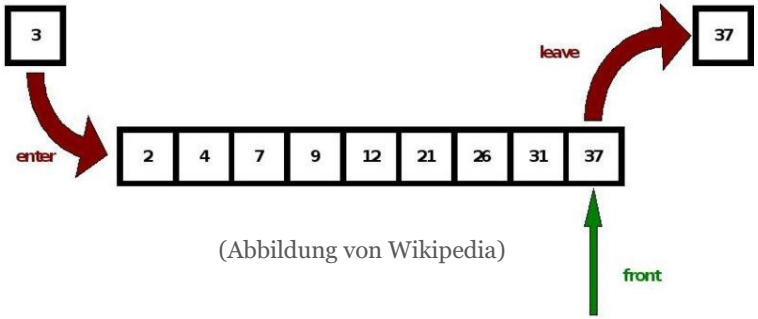
liefert:

```
size: 0
size: 2
pop 4
pop 7
pop -1
size: 0
```

Queue I

Eine Queue (Warteschlange) liefert im Gegensatz zum Stack das am längsten vorhandene Objekt zurück. Wie bei einer Warteschlange, muss gewartet werden, bis man an der Reihe ist.

Stack	LIFO (last in - first out)
Queue	FIFO (first in - first out)



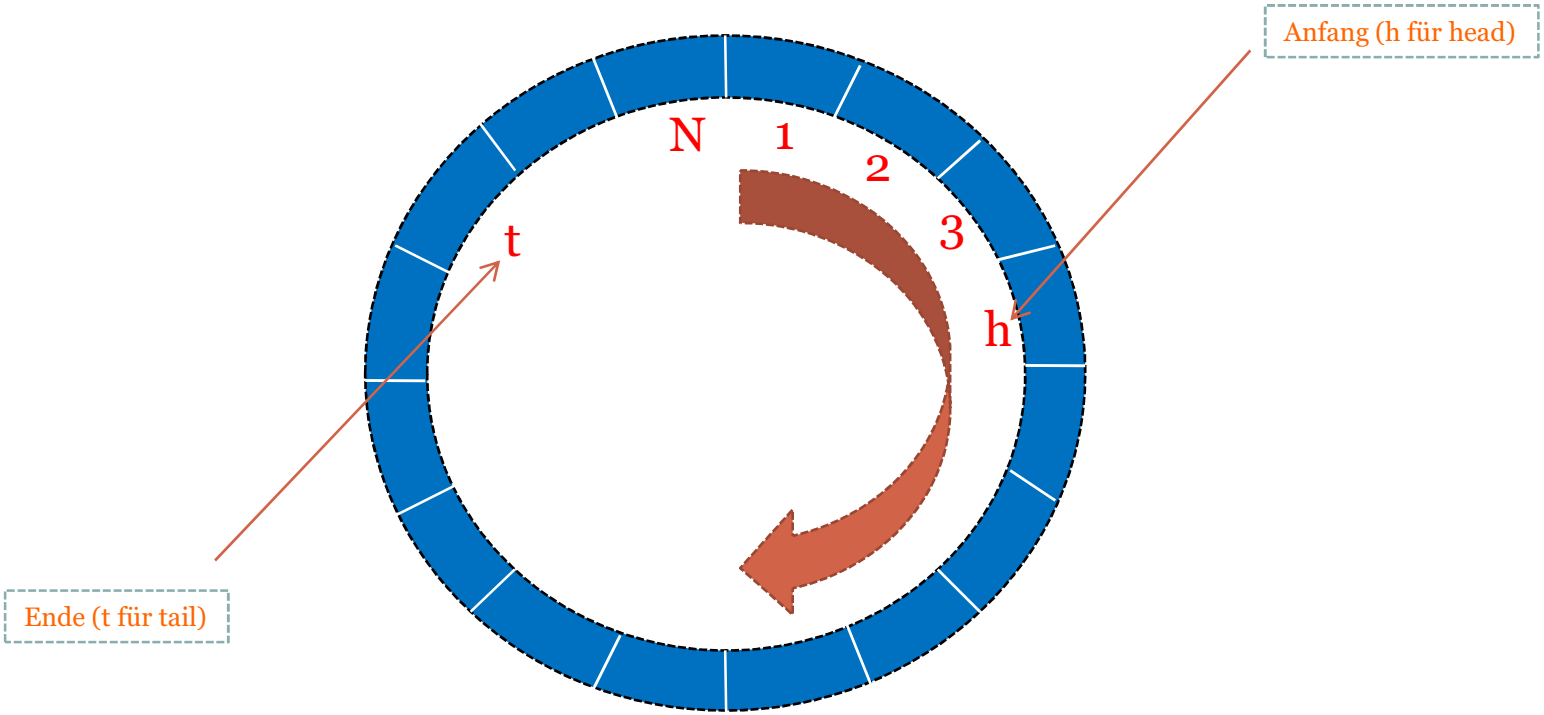
(Abbildung von Wikipedia)

Methoden einer Queue:

enqueue(Objekt)	fügt Objekt als hinterstes Element der Queue hinzu
Objekt dequeue()	gibt das vorderste Element der Queue zurück und entfernt es
boolean isEmpty()	prüft, ob die Queue leer ist
int size()	liefert die Anzahl der in der Queue vorhandenen Elemente zurück

Realisierung durch ein Array (zyklisch)

Wir stellen uns vor, dass wir die beiden Enden das Arrays zusammenkleben und somit einen Kreis erhalten.



Realisierung durch ein Array I

Wir speichern die Objekte in einem Array und merken uns, wie viele Elemente zur Zeit in dem Array sind. Im Unterschied zum Stack müssen wir sowohl den Anfangs- als auch das Endindex speichern. Damit die Daten nicht wie eine Raupe durch das Array wandern, werden wir es zyklisch verwalten.

```
public class Queue {
   private int h, t;
   private int n;                    // n = aktuelle Anzahl der Elemente
   private final int N = 100;        // maximale Anzahl zu speichernder Elemente

   private int[] queue;

   public Queue(){
      h     = 0;
      t     = 0;
      n     = 0;
      queue = new int[N];
   }

   public void enqueue(int a){
      if (n<99) {
         n++;
         if (t<N-1)
            t++;
         else
            t = 0;
         queue[t] = a;
      }
   }
   ...
```

Realisierung durch ein Array II

Weiter geht's:

```
...
public int dequeue() {
   if (n<=0)
      return -1;

      n--;
      if (h<N-1) {
         t++;
         return queue[t-1];
      } else {
         t = 0;
         return queue[N-1];
      }
}

public Boolean isEmpty(){
   return n == 0;
}

public int size(){
   return n;
}
...
```

Realisierung durch ein Array III

Weiter geht's:

```
   ...
   public static void main(String[] args){
      Queue q = new Queue();
      System.out.println("size: "+q.size());
      q.enqueue(7);
      q.enqueue(4);
      System.out.println("size: "+q.size());

      System.out.println("dequeue "+q.dequeue());
      System.out.println("dequeue "+q.dequeue());
      System.out.println("dequeue "+q.dequeue());

      System.out.println("size: "+q.size());
   }
}
```

Liefert folgende Ausgabe:

```
size: 0
size: 2
dequeue 4
dequeue 7
dequeue -1
size: 0
```