

# 1. Die rekursive Datenstruktur Liste

## 1.3 Rekursive Funktionen



Ideen zur Bestimmung der Länge einer Liste:

1. Verwalte ein globales Attribut `int laenge`.  
Fügt man ein Element zur Liste oder löscht es, wird das Attribut `laenge` aktualisiert.
2. Gehe mit einer Schleife durch die Liste, bis der Nachfolger null ist.  
In der Methode benötigt man ein lokales Attribut `int laenge`, das bei jedem Schritt um 1 erhöht wird am Ende der Methode ausgegeben wird.

3. Verwende in der Klasse Knoten eine Methode, die an den Nachfolger weitergereicht wird. Auf diese Weise wird kein zusätzliches Attribut benötigt.



```
public int laenge(){
    if(nachfolger !=null){
        return nachfolger.laenge() + 1;
    }
    else{
        return 1;
    }
}
```

Beispiel: Liste mit den Knoten k1, k2, k3 und k4.

Der Aufruf **k1.laenge()** liefert:

$$k1.laenge() = k2.laenge() + 1 = k3.laenge() + 1 + 1 = k4.laenge() + 1 + 1 + 1 = 1 + 1 + 1 + 1 = 4$$

In der Klasse Liste ruft man dann **anfang.laenge()** auf, falls anfang nicht null ist.



In der Klasse Knoten wird die Methode `laenge()` von einem Objekt derselben Klasse wieder aufgerufen. Man kann auch sagen, die Methode ruft sich selbst wieder auf. Solche Methoden und Funktionen nennt man **rekursiv**.

Beispiel:

Fakultät einer natürlichen Zahl

$$\begin{aligned} fak(5) &= 5! = \\ 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 &= \\ 5 \cdot 4! &= \\ 5 \cdot fak(4) \end{aligned}$$

Aus  $fak(5)$  lässt sich  $fak(4)$  bestimmen, aus  $fak(4)$  lässt sich  $fak(3)$  bestimmen, u.s.w.

Kennt man den Wert von  $fak(1)$  lässt sich  $fak(n)$  für alle natürlichen Zahlen ermitteln.

Eine rekursive Funktion kann man wie folgt formulieren:



$$fak(n) = \begin{cases} n \cdot fak(n - 1), & \text{wenn } n > 1 \\ 1, & \text{wenn } n = 1 \end{cases}$$

Die notwendige **Abbruchbedingung**  $fak(1) = 1$  sorgt für das Ende der Rekursion.

## Aufrufsequenz für $fak(5)$ :



$$fak(5) =$$

$$5 \cdot fak(4) =$$

$$5 \cdot 4 \cdot fak(3) =$$

$$5 \cdot 4 \cdot 3 \cdot fak(2) =$$

$$5 \cdot 4 \cdot 3 \cdot 2 \cdot fak(1) =$$

$$5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

Eine rekursive Funktionen, kann sich auch mehrmals aufrufen:



$$fib(n) = \begin{cases} fib(n-1) + fib(n-2), & \text{wenn } n \geq 2 \\ 1, & \text{wenn } n = 1 \\ 0, & \text{wenn } n = 0 \end{cases}$$

Man erhält die **Fibonacci-Folge** 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Wie lautet die Aufrufsequenz für  $fib(6)$ ?

## Weitere Beispiele für rekursive Funktionen:



Berechnung der **Potenz  $a^n$**  mit  $a > 0$  und  $n \in \mathbb{N}_0$

$$\text{pot}(a, n) = \begin{cases} a \cdot \text{pot}(a, n - 1), & \text{wenn } n \geq 1 \\ 1, & \text{wenn } n = 0 \end{cases}$$

Aufrufsequenz für  $\text{pot}(5,3)$ :

$$\begin{aligned} \text{pot}(5,3) &= 5 \cdot \text{pot}(5,2) = 5 \cdot 5 \cdot \text{pot}(5,1) = \\ &= 5 \cdot 5 \cdot 5 \cdot \text{pot}(5,0) = 5 \cdot 5 \cdot 5 \cdot 1 = 125 \end{aligned}$$

## Weitere Beispiele für rekursive Funktionen:



Berechnung des **größten gemeinsamen Teilers** mit dem **Euklidischen Algorithmus**.

$$ggT(15, 18) = 3; \quad ggt(5, 75) = 5; \quad ggT(13, 20) = 1$$

Algorithmus in **Pseudocode** formuliert:

**Wiederhole solange bis  $b = 0$ :**

**wenn  $a > b$ , dann  $a = a - b$ , sonst  $b = b - a$ ;**

**Ende wiederhole;**

**Gib den Wert von  $a$  aus.**





Durchführen des Algorithmus  
für  $a = 15$  und  $b = 18$ :

**Wiederhole solange bis  $b = 0$ :**

**wenn  $a > b$ , dann  $a = a - b$ , sonst  $b = b - a$ ;**

**Ende wiederhole;**

**Gib den Wert von  $a$  aus.**

a	b	a>b ?
15	18	nein
15	3	ja
12	3	ja
9	3	ja
6	3	ja
3	3	nein
3	0	ja

$$ggT(a, b) = \begin{cases} ggT(a - b, b), & \text{wenn } a > b \\ ggT(a, b - a), & \text{wenn } b > a \\ a, & \text{wenn } a = b \end{cases}$$



## Übung 1



Implementiere die rekursive Methode **laenge()** in der Klasse Knoten und die Methode **wortlaengeGeben()** in der Klasse Wort.

## Übung 2

Implementiere die rekursiven Methode zur Berechnung der Fakultät, des größten gemeinsamen Teilers und der Potenz einer Zahl.



## Übung 3



Eine Bakterienkultur vermehrt sich jede Stunde um 20%. Zu Beginn der Beobachtung besteht die Kultur aus 300 Bakterien.

Gib eine Rekursionsvorschrift für die Bestimmung der Bakterienanzahl nach  $n$  Stunden an und implementiere diese Funktion.



## Übung 4

a)

Beschreibe die Funktionalität der Funktion *funktion1*.

Verwende z.B. die Wörter "HANNAH", "REGAL" oder "RENTNER".

Informiere dich in der Java API oder in einem Handbuch über die verfügbaren Methoden der Klasse String.

```
public boolean funktion1(String s){
    if(s.length()==0||s.length()==1){
        return true;
    }
    else{
        return funktion1(s.substring(1,s.length()-1))
            &&(s.charAt(0)==s.charAt(s.length()-1));
    }
}
```



## Übung 4



b)

Implementiere eine weitere rekursiv definierte Methode **umdrehen(String s)**, die ein Wort rückwärts ausgibt.

Beispiel: `umdrehen("REGAL") = "LAGER"`

c)

Verwende die Methode `umdrehen` um die in Methode aus Aufgabe a) einfacher zu implementieren.



## Übung 5\*

### Umwandeln von der Dezimal- in die Binärdarstellung.



Eine Zahl wandelt man in das Binärsystem um, indem man sie als Summe von Zweierpotenzen schreibt:

$$45 = 32 + 8 + 4 + 1 = 2^5 + 2^3 + 2^2 + 2^0 = (101101)_2$$

Aufgrund folgender Eigenschaft der Zahlen im Binärsystem lässt sich dies sehr elegant rekursiv lösen:

**Fügt man an eine Binärzahl an die letzte Stelle eine 0 an, so verdoppelt sie ihren Wert. Fügt man an die letzte Stelle eine 1 an, so erhält man den doppelten Wert + 1.**

Bestätige dies am obigen Beispiel und begründe diese Eigenschaft.

Entwickle aus dieser Eigenschaft eine rekursive Funktion, die eine gegebene Zahl aus dem Dezimalsystem in das Binärsystem umwandelt.

Die Operationen  $a / b$  (ganzzahlige Division ohne Rest) und  $a \% b$  (a modulo b, d.h. Rest bei der Division  $a : b$ ) können helfen!

Schreibe auch eine rekursive Funktion, die eine gegebene Zahl aus dem Binärsystem in das Dezimalsystem umwandelt.