Rekursion

Rekursion

Was gibt das folgende Programm auf der Konsole aus?

```
void foo(int k) {
        if (k == 0) { return; }
        cout << k << endl;
        foo(k-1);
}
int main() {
    foo(42);
    return 0;
}
```

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Anwendung der Gleichungen:

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$\begin{aligned} x + 0 &= x \\ x + s(y) &= s(x + y) \end{aligned}$$

Anwendung der Gleichungen:

$$s(s(0)) + s(s(s(0)))$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Anwendung der Gleichungen:

$$s(s(0)) + s(s(s(0)))$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Anwendung der Gleichungen:

$$s(s(0)) + s(s(s(0)))$$
$$s(\qquad s(s(0)) + s(s(0)) \qquad)$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Anwendung der Gleichungen:

$$s(s(0)) + s(s(s(0)))$$
$$s(\qquad s(s(0)) + s(s(0)) \qquad)$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Anwendung der Gleichungen:

$$s(s(0)) + s(s(s(0)))$$
$$s(\qquad s(s(0)) + s(s(0)) \qquad)$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Anwendung der Gleichungen:

$$
\begin{array}{lcr}
 & s(s(0)) + s(s(s(0))) & \\
s(& s(s(0)) + s(s(0)) &) \\
s(s(& s(s(0)) + s(0) &))
\end{array}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Anwendung der Gleichungen:

$$
\begin{array}{rcl}
 & s(s(0)) + s(s(s(0))) & \\
s(& s(s(0)) + s(s(0)) &) \\
s(s(& s(s(0)) + s(0) &))
\end{array}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Anwendung der Gleichungen:

$$
\begin{array}{lcr}
 & s(s(0)) + s(s(s(0))) & \\
s(& s(s(0)) + s(s(0)) &) \\
s(s(& s(s(0)) + s(0) &))
\end{array}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Anwendung der Gleichungen:

$$\begin{array}{rcl} & s(s(0)) + s(s(s(0))) & \\ s(& s(s(0)) + s(s(0)) &) \\ s(s(& s(s(0)) + s(0) &)) \\ s(s(s(& s(s(0)) + 0 &))) \end{array}$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Anwendung der Gleichungen:

$$
\begin{array}{rcl}
 & s(s(0)) + s(s(s(0))) & \\
s(& s(s(0)) + s(s(0)) &) \\
s(s(& s(s(0)) + s(0) &)) \\
s(s(s(& s(s(0)) + 0 &)))
\end{array}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Anwendung der Gleichungen:

$$
\begin{array}{rll}
 & s(s(0)) + s(s(s(0))) & \\
s(& s(s(0)) + s(s(0)) &) \\
s(s(& s(s(0)) + s(0) &)) \\
s(s(s(& s(s(0)) + 0 &)))
\end{array}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$
\begin{aligned}
x + 0 &= x \\
x + s(y) &= s(x + y)
\end{aligned}
$$

Anwendung der Gleichungen:

$$
\begin{array}{rll}
 & s(s(0)) + s(s(s(0))) & \\
s(& s(s(0)) + s(s(0)) &) \\
s(s(& s(s(0)) + s(0) &)) \\
s(s(s(& s(s(0)) + 0 &))) \\
s(s(s(& s(s(0)) &)))
\end{array}
$$

Rekursion

Bekanntes Beispiel: Addition als Gleichungen spezifiziert

$$\begin{aligned} x + 0 &= x \\ x + s(y) &= s(x + y) \end{aligned}$$

Anwendung der Gleichungen:

$$\begin{array}{rll} & s(s(0)) + s(s(s(0))) & \\ s(& s(s(0)) + s(s(0)) &) \\ s(s(& s(s(0)) + s(0) &)) \\ s(s(s(& s(s(0)) + 0 &))) \\ s(s(s(& s(s(0)) &))) \end{array}$$

Rekursion

Rekursive Addition als C++-Programm:

```
int add(int x, int y) {
    if (y == 0) return x;          // x + 0 = x
    else return add (x,y-1) + 1;
// x + (y + 1) = (x + y) + 1
}
```

Rekursion

Wozu Rekursion?

- Manche Programme lassen sich kürzer und eleganter schreiben.

Rekursion

Wozu Rekursion?

- Manche Programme lassen sich kürzer und eleganter schreiben.
- Beispiel Fakultät:

$$fac(n) = \prod_{i=1}^{n} i \qquad \text{oder} \qquad \begin{aligned} fac(0) &= 1 \\ fac(n) &= n \cdot fac(n-1) \end{aligned}$$

Rekursion

Wozu Rekursion?

- Manche Programme lassen sich kürzer und eleganter schreiben.
- Beispiel Fakultät:

$$fac(n) = \prod_{i=1}^{n} i \qquad \text{oder} \qquad \begin{aligned} fac(0) &= 1 \\ fac(n) &= n \cdot fac(n-1) \end{aligned}$$

- Als iteratives C++-Programm:

```
int factorial_it(int n) {
    int res = 1;
    for (int i=1;i<=n;i++) {
        res *= i;
    }
    return res;
}
```

Rekursion

Wozu Rekursion?

- Manche Programme lassen sich kürzer und eleganter schreiben.
- Beispiel Fakultät:

$$fac(n) = \prod_{i=1}^{n} i \qquad \text{oder} \qquad \begin{aligned} fac(0) &= 1 \\ fac(n) &= n \cdot fac(n-1) \end{aligned}$$

- Als rekursives C++-Programm:

```
int factorial_rec1(int n) {
    if (n==0) return 1;
    return n*factorial_rec1(n-1);
}
```

Rekursion

Wozu Rekursion?

- Manche Programme lassen sich kürzer und eleganter schreiben.
- Beispiel Fakultät:

$$fac(n) = \prod_{i=1}^{n} i \qquad \text{oder} \qquad \begin{aligned} fac(0) &= 1 \\ fac(n) &= n \cdot fac(n-1) \end{aligned}$$

- Noch kürzer:

```
int factorial_rec2(int n) {
    return n==0?1:n*factorial_rec2(n-1);
}
```

Rekursion

Wozu Rekursion?

- Manche Programme lassen sich kürzer und eleganter schreiben.
- Beispiel Fakultät:

$$fac(n) = \prod_{i=1}^{n} i \qquad \text{oder} \qquad \begin{aligned} fac(0) &= 1 \\ fac(n) &= n \cdot fac(n-1) \end{aligned}$$

- Noch kürzer:

```
int factorial_rec2(int n) {
    return n==0?1:n*factorial_rec2(n-1);
}
```

- Das rekursive Programm ist sehr nah an der Definition.
- Dadurch ist leicht nachvollziehbar, ob das Programm stimmt.

Rekursion

Rekursive Definitionen folgen einem allgemeinen Schema

- Ein oder mehrere Basisfälle (Rekursionsanfang).
- Ein oder mehrere rekursive Aufrufe (Rekursionsschritt)

Rekursion

Rekursive Definitionen folgen einem allgemeinen Schema

- Ein oder mehrere Basisfälle (Rekursionsanfang).
- Ein oder mehrere rekursive Aufrufe (Rekursionsschritt)

Vergleich mit `while`-Schleifen

- Abbruchbedingung entspricht Rekursionsanfang
- Schleifenrumpf entspricht Rekursionsschritt

Rekursion

Beispiel: Fibonacci-Folge

$$
\begin{aligned}
fib(1) &= fib(2) = 1 \\
fib(n) &= fib(n-1) + fib(n-2)
\end{aligned}
$$

Rekursion

Beispiel: Fibonacci-Folge

$$\begin{aligned} fib(1) &= fib(2) = 1 \\ fib(n) &= fib(n-1) + fib(n-2) \end{aligned}$$

die ersten 10 Fibonacci-Zahlen:

n :	1	2	3	4	5	6	7	8	9	10
$fib(n)$:	1	1	2	3	5	8	13	21	34	55

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n + 1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n+1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n + 1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

$n = 2$: $2, 1, 4, 2, 1$

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n + 1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

$n = 2$: $2, 1, 4, 2, 1$

$n = 3$: $3, 10, 5, 16, 8, 4, 2, 1$

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n+1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

$n = 2$: $2, 1, 4, 2, 1$

$n = 3$: $3, 10, 5, 16, 8, 4, 2, 1$

$n = 4$: $4, 2, 1$

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n + 1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

$n = 2$: $2, 1, 4, 2, 1$

$n = 3$: $3, 10, 5, 16, 8, 4, 2, 1$

$n = 4$: $4, 2, 1$

$n = 5$: $5, 16, 8, 4, 2, 1$

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n + 1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

$n = 2$: $2, 1, 4, 2, 1$

$n = 3$: $3, 10, 5, 16, 8, 4, 2, 1$

$n = 4$: $4, 2, 1$

$n = 5$: $5, 16, 8, 4, 2, 1$

$n = 6$: $6, 3, 10, 5, 16, 8, 4, 2, 1$

Rekursion

Beispiel: Hailstone-Folge

- Beginne mit einer *natürlichen Zahl* n.
- Ist n gerade, so nimm als nächstes $n/2$.
- Ist n ungerade, so nimm als nächstes $3n + 1$.
- Wiederhole, bis der Zyklus $4, 2, 1$ erreicht ist.

Beispiele:

$n = 1$: $1, 4, 2, 1$

$n = 2$: $2, 1, 4, 2, 1$

$n = 3$: $3, 10, 5, 16, 8, 4, 2, 1$

$n = 4$: $4, 2, 1$

$n = 5$: $5, 16, 8, 4, 2, 1$

$n = 6$: $6, 3, 10, 5, 16, 8, 4, 2, 1$

$n = 7$: $7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$

Rekursion

Beispiel: Ackermann-Funktion

$$A(m,n) = \begin{cases} n+1 & \text{falls } m = 0 \\ A(m-1,1) & \text{falls } m > 0 \text{ und } n = 0 \\ A(m-1, A(m,n-1)) & \text{falls } m > 0 \text{ und } n > 0 \end{cases}$$

Die Werte dieser Funktion wachsen extrem schnell!

Rekursion

Bekannte Probleme, die mit Rekursion gelöst werden können:

- Damenproblem
  - Platziere 8 Damen auf einem Schachbrett, ohne dass sie einander schlagen können.
- Springerproblem
  - Bewege einen Springer so, dass er auf jedem Feld des Schachbretts genau einmal steht.
- Rucksackproblem
  - Gegeben: Eine Menge von Objekten mit Gewichten und Werten.
  - Aufgabe: Wähle eine Teilmenge mit maximalem Wert, deren Gesamtgewicht eine gewisse Grenze nicht überschreitet.
- Sudoku
- Türme von Hanoi

Rekursion

Schreiben Sie ein rekursives Programm, ...

1. ... das die Summe der ersten n natürlichen Zahlen berechnet.
2. ... das einen String zeichenweise ausgibt.
3. ... das eine Zahl in einer Liste sucht.
4.

Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

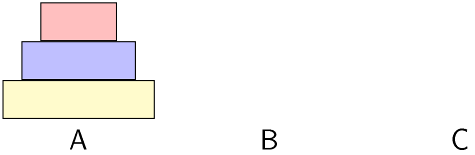
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

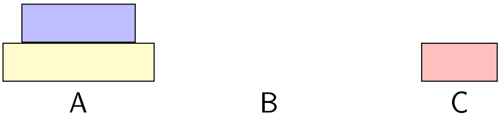
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

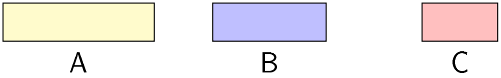
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

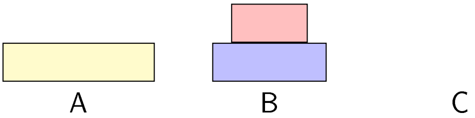
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

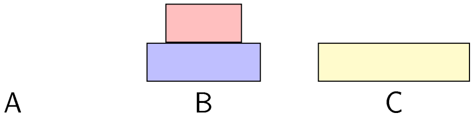
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

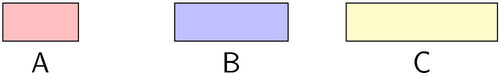
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

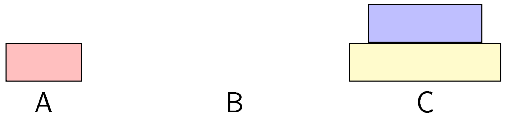
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Aufgabe: Bewege einen Turm aus Spielsteinen von A nach C

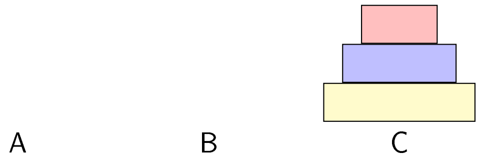
Gegeben:

- Spielsteine unterschiedlicher Größe.
- Drei Stellen A, B und C, an denen Spielsteine liegen können.

Spielregeln:

1. Die Steine werden einzeln bewegt.
2. Es wird niemals ein größerer Stein auf einen kleineren gelegt.

Beispiel mit 3 Steinen:



Die Türme von Hanoi

Frage: Wie bewegt man einen Turm der Höhe h von A nach C?

Die Türme von Hanoi

Frage: Wie bewegt man einen Turm der Höhe h von A nach C?

Naive Antwort:

1. Bewege alle bis auf die letzte Platte von A nach B
2. Bewege die letzte Platte von A nach C
3. Bewege den Turm von B nach C

Die Türme von Hanoi

Frage: Wie bewegt man einen Turm der Höhe h von A nach C?

Naive Antwort:

1. Bewege alle bis auf die letzte Platte von A nach B
2. Bewege die letzte Platte von A nach C
3. Bewege den Turm von B nach C

Überraschung: So naiv ist das gar nicht!

- Wir konstruieren einen Algorithmus auf Basis dieser Vorgehensweise.

Die Türme von Hanoi

Wir definieren stückweise eine Funktion, die das Problem löst.

Die Türme von Hanoi

Wir definieren stückweise eine Funktion, die das Problem löst.

- Bewegen einer einzelnen Platte:

```
void bewege_platte(int s, int z) {
  cout << "Bewege Platte von " << s << " nach " <<
}
```

Die Türme von Hanoi

Wir definieren stückweise eine Funktion, die das Problem löst.

- Bewegen einer einzelnen Platte:

```
void bewege_platte(int s, int z) {
  cout << "Bewege Platte von " << s << " nach " <<
}
```

- Bewegen eines Turms der Höhe 1 von `s` über `m` nach `z`:

```
void hanoi1(int s, int m, int z) {
  bewege_platte(s,z);
}
```

Die Türme von Hanoi

Wir definieren stückweise eine Funktion, die das Problem löst.

- Bewegen einer einzelnen Platte:

```
void bewege_platte(int s, int z) {
  cout << "Bewege Platte von " << s << " nach " <<
}
```

- Bewegen eines Turms der Höhe 1 von s über m nach z:

```
void hanoi1(int s, int m, int z) {
  bewege_platte(s,z);
}
```

- Bewegen eines Turms der Höhe 2 von s über m nach z:

```
void hanoi2(int s, int m, int z) {
  hanoi1(s,z,m);
  bewege_platte(s,z);
  hanoi1(m,s,z);
}
```

Die Türme von Hanoi

Konstruktion der Hanoi-Lösung (Fortsetzung)

- Bewegen eines Turms der Höhe 3 von `s` über `m` nach `z`:

```
void hanoi3(int s, int m, int z) {
  hanoi2(s,z,m);
  bewege_platte(s,z);
  hanoi2(m,s,z);
}
```

Die Türme von Hanoi

Konstruktion der Hanoi-Lösung (Fortsetzung)

- Bewegen eines Turms der Höhe 3 von `s` über `m` nach `z`:

```
void hanoi3(int s, int m, int z) {
  hanoi2(s,z,m);
  bewege_platte(s,z);
  hanoi2(m,s,z);
}
```

- Bewegen eines Turms der Höhe 4 von `s` über `m` nach `z`:

```
void hanoi4(int s, int m, int z) {
  hanoi3(s,z,m);
  bewege_platte(s,z);
  hanoi3(m,s,z);
}
```

Die Türme von Hanoi

Konstruktion der Hanoi-Lösung (Fortsetzung)

Beobachtungen:

- Die Funktionen `hanoi2`, `hanoi3`, `hanoi4`, ...sind alle gleich.
- Beim Aufruf wird nur die Zahl reduziert und dann wieder das Gleiche gemacht.
- Nur bei `hanoi1` wird kein `hanoi0` aufgerufen.

Die Türme von Hanoi

Konstruktion der Hanoi-Lösung (Fortsetzung)

Beobachtungen:

- Die Funktionen `hanoi2`, `hanoi3`, `hanoi4`, ...sind alle gleich.
- Beim Aufruf wird nur die Zahl reduziert und dann wieder das Gleiche gemacht.
- Nur bei `hanoi1` wird kein `hanoi0` aufgerufen.

Schlussfolgerung: Wenn die Höhe als Argument übergeben wird, können wir alles in eine Funktion schreiben.

Die Türme von Hanoi

Rekursive Hanoi-Lösung

```
void hanoi(int h, int s, int m, int z) {
    if (h > 1) { hanoi(h-1,s,z,m); }
    bewege_platte(s,z);
    if (h > 1) { hanoi(h-1,m,s,z); }
}
```