

EINI

LogWing/WiMa/MP

**Einführung in die Informatik für
Naturwissenschaftler und Ingenieure**

Vorlesung 2 SWS WS 18/19

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@tu-dortmund.de
<http://ls14-www.cs.tu-dortmund.de>

▶ Kapitel 4

Grundlagen imperativer Programmierung:

- ▶ Funktionen
- ▶ Rekursion

▶ Unterlagen

- ▶ Dißmann, Stefan und Ernst-Erich Doberkat: *Einführung in die objektorientierte Programmierung mit Java*, 2. Auflage. München [u.a.]: Oldenbourg, 2002, Kapitel 3.4 & 4.1. (→ ZB oder Volltext aus Uninetz)
- ▶ Echte, Klaus und Michael Goedicke: *Lehrbuch der Programmierung mit Java*. Heidelberg: dpunkt-Verl, 2000, Kapitel 4. (→ ZB)
- ▶ Gumm, Heinz-Peter und Manfred Sommer: *Einführung in die Informatik*, 10. Auflage. München: De Gruyter, 2012, Kapitel 2.7 – 2.8. (→ Volltext aus Uninetz)

- Prolog
- Funktionen
- Rekursion

Zwischenstand

- ✓ Variablen
- ✓ Zuweisungen
- ✓ (Einfache) Datentypen und Operationen
 - ✓ Zahlen
`integer, byte, short, long; float, double`
 - ✓ Wahrheitswerte (`boolean`)
 - ✓ Zeichen (`char`)
 - ✓ Zeichenketten (`String`)
 - ✓ Typkompatibilität
- ✓ Kontrollstrukturen
 - ✓ Sequentielle Komposition, Sequenz
 - ✓ Alternative, Fallunterscheidung
 - ✓ Schleife, Wiederholung, Iteration: `while, do-while, for`
- Verfeinerung
 - Unterprogramme, Prozeduren, Funktionen
 - Blockstrukturierung
- ▶ Rekursion

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- **Prolog**
- Funktionen
- Rekursion

Wiederholung

- ▶ Def. „Informatik“ (nach der Akademie Francaise):
 - ▶ von „informatique“
 - ▶ **„Behandlung von Information mit rationalen Mitteln“**



René Descartes

- ▶ „rationale Mittel“ nach Descartes (aus *Abhandlung über die Methode*, 1637):
 - ▶ Nur dasjenige gilt als wahr, was so klar ist, dass kein Zweifel bleibt.
 - ▶ **Größere Probleme sind in kleinere aufzuspalten.**
 - ▶ **Es ist immer vom Einfachen zum Zusammengesetzten hin zu argumentieren.**
 - ▶ Das Werk muss am Ende einer abschließenden Prüfung unterworfen werden.

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- **Prolog**
- Funktionen
- Rekursion

Unterprogramme - Idee

- ▶ Grundidee:
 - ▶ Probleme werden in Teilprobleme zerlegt, die durch bekannte oder neu zu entwickelnde Algorithmen gelöst werden:
 - ▶ Aus den Lösungen der Teilprobleme wird eine Lösung für das Gesamtproblem bestimmt.
 - ▶ Dieses Konzept wird in Programmiersprachen durch **Unterprogramme** unterstützt.

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Unterprogramme

- ▶ **Block** mit eigenem Bezeichner mit Eingabeparametern und Ausgabeparametern:
- ▶ → Dadurch mehrfache Verwendung im Programm möglich.
- ▶ Wiederverwendbarkeit / Nützlichkeit hängt vom Problem, aber auch vom Grad der Abstraktion ab.

- ▶ **Varianten:**
 - ▶ **Funktion:** Unterprogramm **mit** ausgezeichnetem Rückgabeparameter
 - ▶ **Prozedur:** Unterprogramm **ohne** ausgezeichneten Rückgabeparameter
 - ▶ **Methode:** Funktion/Prozedur, die für ein Objekt/einen speziellen Datentyp definiert ist.

- Prolog
- **Funktionen**
- Rekursion

Konventionen

- ▶ Wir verwenden den Begriff **Funktion**
 - ▶ für Unterprogramme in Java
 - mit Rückgabewert
 - ohne Rückgabewert
 - ▶ solange wir **imperativ programmieren**.

- ▶ Wir verwenden den Begriff **Methode**
 - ▶ für Unterprogramme in Java
 - mit Rückgabewert
 - ohne Rückgabewert
 - ▶ sobald wir **objektorientiert programmieren**.

- Prolog
- **Funktionen**
- Rekursion

Wiederholung I

Beispiel: einfache Numerik-Funktionen

- ▶ Berechnung der Quadratwurzel `sqrt` für $n > 0$
- ▶ Nützlichkeit klar,
 - ▶ in vielen Programmen unabhängig vom Kontext verwendbar
 - ▶ daher auch in Bibliotheken (Libraries) stets verfügbar
- ▶ Eine Berechnungsidee: Intervallschachtelung
 - ▶ Finde eine untere Schranke.
 - ▶ Finde eine obere Schranke.
 - ▶ Verringere obere und untere Schranke, bis der Abstand hinreichend gering geworden ist.
 - ▶ Etwas konkreter: Halbiere Intervall, fahre mit demjenigen Teilintervall fort, das das Resultat enthält.

Wiederholung II

```
double x = 2.0,  
       uG = 0, oG = x + 1, m,  
       epsilon = 0.001;
```

```
do  
{  
    m = 0.5 * (uG + oG);  
    if (m*m > x)  
        oG = m;  
    else  
        uG = m;  
}
```

```
while (oG - uG > epsilon);
```

```
System.out.println ( "Wurzel " + x  
                    + " beträgt ungefähr "  
                    + m );
```

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Beispiel `double sqrt(double x)`

```
double sqrt( double x ) {
```

```
double uG = 0, oG = x + 1,  
m, epsilon = 0.001;
```

```
do {  
    m = (uG + oG) / 2;  
    if (m*m > x)  
        oG = m;  
    else  
        uG = m;  
}  
while (oG - uG > epsilon);
```

```
return (m);  
}
```

double: Deklaration des Datentyps für den Rückgabewert

x: Eingabeparameter, Typ **double**
sqrt: Funktionsbezeichner

uG, oG, m, epsilon:
lokale Variablen

m: Rückgabewert

- Prolog
- **Funktionen**
- Rekursion

Beispiel `double sqrt(double x)` II

```
double sqrt( double x )  
{  
    ...  
}
```

▶ **Name:** aussagekräftig!

▶ **Rückgabewert:**

▶ Deklaration des Datentyps

▶ zur Rückgabe eines Ergebnisses an das Hauptprogramm

▶ `void` → kein Rückgabewert

▶ **Parameter:**

▶ optional

▶ Klammern müssen **immer** angegeben werden!

- Prolog
- **Funktionen**
- Rekursion

Anwendung von sqrt (...)

```
import java.util.Scanner;
```

```
class Wurzel {
```

```
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        double eingabe = scanner.nextDouble();
```

```
        double ergebnis = sqrt(eingabe);
```

```
        System.out.println("Die Wurzel ist" + ergebnis);
```

```
    }
```

```
    static double sqrt(double x) {
```

```
        double uG = 0, oG = x + 1, m, epsilon = 0.001;
```

```
        ...
```

```
        return (m);
```

```
    }
```

```
}
```

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Verwendung von Funktionen

Verwendung:

- ▶ Aufruf **ohne Rückgabewert**
- ▶ Aufruf **mit Rückgabewert**

```
double m = sqrt(x);  
System.out.println("Wurzel " + x + " ist  
ca. " + m);
```

- ▶ also:
 - ▶ Funktionen mit Rückgabewert: auf der rechten Seite einer Zuweisung **und** in Ausdrücken
 - ▶ Funktionen ohne Rückgabewert: nur in Ausdrücken
- ▶ Die Lösung von Teilproblemen durch Prozeduren (Funktionen) nennt man **prozedurale (funktionale) Abstraktion**.

- Prolog
- **Funktionen**
- Rekursion

Beispiel main (...)

```
public static void main (string[] args)
{
    ...
}
```

- ▶ **Name:** main
- ▶ **Rückgabewert:** kein Rückgabewert
- ▶ **Parameter:**
 - ▶ vorhanden
 - ▶ aber: Wir nutzen die Parameter zur Zeit noch nicht.
- ▶ **public** (später im Rahmen der Objektorientierung)
- ▶ **static** (später im Rahmen der Objektorientierung)

- Prolog
- **Funktionen**
- Rekursion

Top-Down-Entwurf I

- ▶ Top-Down-Strategie:
 - ▶ Zerlege Problem in Teilprobleme.
 - ▶ Löse Teilprobleme.
 - ▶ Kombiniere Lösung der Teilprobleme zur Lösung des Gesamtproblems.
- ▶ Im Entwurf:
 - ▶ Zerlege Problem in Teilprobleme.
 - ▶ Deklarriere für jedes Teilproblem eine Funktion, die im Entwurf **zunächst unausgefüllt** bleibt:
 - **Stubs & Skeleton-Prinzip (Stummel & Skelett)**
 - ▶ Löse Gesamtproblem (**Skelett**) mit Hilfe der **Stubs**.
 - ▶ Fülle die **Stubs**.

- Prolog
- **Funktionen**
- Rekursion

Top-Down-Entwurf II

- ❖ Definiere Teilprobleme möglichst so, dass sie als allgemeine und bekannte Probleme aus der Informatik erkennbar werden, für die es bekannte Lösungen in Software-Bibliotheken gibt!
- ▶ Funktionale Abstraktion = Zerlegung nach Funktionen /Aufgaben:
 - ▶ typisch für imperative Programmierung
 - ▶ Es existieren Alternativen, z.B. die Zerlegung nach Daten.
 - ▶ Sichtweise für objektorientierte Programmierung ist etwas anders.

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Beispiel: Einfaches Spiel

```
public static void main(String[] args)
{
    int spieler = 1;
    boolean fertig = false;
    init();
    while (!fertig) {
        visualisiereSpiel();
        macheZug();
        if (Spielende())
            fertig = true;
        else
            SpielerWechsel();
    }
    GratuliereSieger();
}
```

```
void init()
void macheZug()
void SpielerWechsel()
```

```
void visualisiereSpiel()
boolean Spielende()
void GratuliereSieger()
```

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Kommunikation Haupt- und Unterprogramm I

Haupt- und Unterprogramme eines Programms teilen sich bei der Bearbeitung (= als Prozess) einen gemeinsamen Adressraum.

▶ Kommunikation über globale Variable:

- ▶ Variablen, die als global deklariert sind, können auch in Unterprogrammen gelesen und verändert werden.
- ▶ Verwendung von **globalen Variablen** in Funktionen führen dazu, dass deren genaue Auswirkungen schwierig überblickt werden können(→ **Seiteneffekte**).
- ❖ Nur sehr begrenzt sinnvoll einsetzbar!

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Kommunikation Haupt- und Unterprogramm II

- ▶ **Kommunikation über Parameter:**
 - ▶ **Eingabeparameter** (mehrere): Liefern Informationen, die innerhalb des Unterprogramms nur gelesen werden.
 - ▶ **Rückgabeparameter** (einer): Liefert Wert, der von der aufrufenden Funktion / Prozedur gelesen werden kann.
 - begrenzte Möglichkeiten
 - häufig für einfache Rückgaben (z.B. boolesche Resultate) und Auftreten von Fehlern genutzt

- ▶ **Ausweg: Variablenparameter** (Aufrufparameter, die Rückgabe erlauben)
(bei Gumm/Sommer durch Bezug zur Sprache Pascal)

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Werteparameter / Variablenparameter I

- ▶ Beobachtung: **Werteparameter**
 - ▶ Bei einem Funktionsaufruf werden die Parameter mit konkreten Werten belegt.
 - ▶ Sichtweise: Parameter sind **funktionslokale** Variablen, die bei Aufruf der Funktion mit den Werten des Aufrufs initialisiert werden.
 - ▶ **double sqrt(double x)**
 - Aufruf: **sqrt(4.0)**
 - impliziert $x = 4.0$ in der Funktion **sqrt**
 - ▶ → **Call by Value**
 - ▶ Änderungen der Parameter in der Funktion **werden nicht** zurückgegeben!

- Prolog
- **Funktionen**
- Rekursion

Anwendung von sqrt (...)

```
import java.util.Scanner;
```

```
class Wurzel {
```

```
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        double eingabe = scanner.nextDouble();
```

```
        double ergebnis = sqrt(eingabe);
```

```
        System.out.println("Die Wurzel von " + eingabe + " ist  
" + ergebnis);
```

```
    }
```

```
    static double sqrt(double x) {
```

```
        double uG = 0, oG = x + 1, m, epsilon = 0.001;
```

```
        ...
```

```
        return (m) ;
```

```
    }
```

```
}
```

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Werteparameter / Variablenparameter II

- ▶ **Variablenparameter** sind Parameter, die als Referenz / Adresse eines Datentyps deklariert sind:
 - ▶ Bei einem Funktionsaufruf werden die Parameter mit konkreten Werten belegt.
 - ▶ Parameter sind **nicht funktionslokal**.
- ▶ → **Call by Reference**
- ▶ Änderungen der Parameter in der Funktion **werden** zurückgegeben!
- ❖ **Primitive Datentypen: Call by Value**
- ❖ **Objekte: Call by Reference**

- Prolog
- **Funktionen**
- Rekursion

1. Idee: Textuelle Ersetzung

- ▶ Bei der Übersetzung des Quelltextes in Maschinensprache wird an jeder Stelle des Funktionsaufrufes der Quelltext eingefügt.
- ▶ **Nachteile:**
 - ▶ keine ineinander verschachtelten Funktionen
 - ▶ Bei Prozeduren okay, bei Funktionen wegen Rückgabeparametern umständlich.
 - ▶ Erzeugt unnötig umfangreichen Code in Maschinensprache.
- ▶ → Wird daher nur in speziellen Kontexten unterstützt.

2. Idee: Unterliegende Basismaschine (Prozessor) unterstützt Funktionsaufrufe.

- ▶ Prozess besteht aus Speicherbereichen für
 - ▶ **Verwaltungsinformationen** des Betriebssystems (Prozessorstatuswort, Programmzähler, ...)
 - ▶ **Programmcode**
 - ▶ **Heap** (= Haufen)
 - Menge aller Variablen, die zur Laufzeit verwendet wurden und noch nicht freigegeben wurden.

- Prolog
- **Funktionen**
- Rekursion

2. Idee: Unterliegende Basismaschine (Prozessor) unterstützt Funktionsaufrufe. (Forts.)

- ▶ **Stack** (= Stapel)
 - Bei jedem Funktionsaufruf wird ein neues Element auf dem Stapel erzeugt, das u.a. die **Parameter** und **lokalen Variablen** der Funktion enthält.
 - Bei Terminierung einer Funktion wird das zugehörige (oberste) Element vom Stapel entfernt.

- ▶ → **Erlaubt ineinander verschachtelte Funktionen.**

Zwischenstand

- ✓ Variablen
- ✓ Zuweisungen
- ✓ (Einfache) Datentypen und Operationen
 - ✓ Zahlen
`integer, byte, short, long; float, double`
 - ✓ Wahrheitswerte (`boolean`)
 - ✓ Zeichen (`char`)
 - ✓ Zeichenketten (`String`)
 - ✓ Typkompatibilität
- ✓ Kontrollstrukturen
 - ✓ Sequentielle Komposition, Sequenz
 - ✓ Alternative, Fallunterscheidung
 - ✓ Schleife, Wiederholung, Iteration: `while`, `do-while`, `for`
- ✓ Verfeinerung
 - ✓ Unterprogramme, Prozeduren, Funktionen
 - ✓ Blockstrukturierung
- Rekursion

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- **Prolog**
- Funktionen
- Rekursion



Artikel im EINI-Wiki:

- **Methode**
 - Parameter
 - Rückgabe
- **Main**
- **Nebeneffekt**
- **Call by Value**
- **Call by Reference**
- **Stack**
- **Heap (Speicher)**

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Rekursive Funktionen I

- ▶ **Rekursion** ist ein wichtiges Hilfsmittel zur Strukturierung des Kontrollflusses von Algorithmen und zur Beschreibung von Datenstrukturen.
- ▶ Eine Funktion f ist **rekursiv**, wenn
 - ▶ der Funktionsrumpf einen Aufruf der Funktion f **selbst** enthält oder eine Funktion g , die wiederum f **aufruft**.
 - ▶ eine **Terminierungsbedingung** existiert.
 - ▶ jede Eingabe nach **endlich** vielen Schritten **terminiert**.

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**

Was ist Rekursion?

Beobachtung:

1. Der Mann ist eher einfach in der Wahl seiner Wünsche.
2. Der Mann hat keine Ahnung von Rekursion.

Ein großes Haus,
ein schnelles Auto,
eine hübsche Frau!

**Du hast 3
Wünsche frei!**



Was ist Rekursion?

Noch

Beobachtung:

1. Der Mann ist edel in der Wahl seiner Wünsche.
2. Der Mann hat keine Ahnung von Rekursion.

Heilmittel gegen alle Krankheiten,
Arbeitsplätze für Alle,
Weltfrieden!

**Du hast 3
Wünsche frei!**



Was ist Rekursion?

Ein In

Beobachtung:

1. Der Informatiker ist eher einfachen Gemüts.
2. Er kennt die Rekursion (zum Teil)!

Einen schnelleren Prozessor,
mehr Speicher,
...

... und noch so eine Fee!

**Los! 3
Wünsche, bla,
bla, bla**



Rekursive Funktionen II

Was ist also Rekursion?

- ▶ Eine Funktion f ist **rekursiv**, wenn
 - ▶ der Funktionsrumpf einen Aufruf der Funktion f **selbst** enthält oder eine Funktion g , die wiederum f **aufruft**.
 - ▶ eine **Terminierungsbedingung** existiert.
 - ▶ jede Eingabe nach **endlich** vielen Schritten **terminiert**.

```
void fee () {  
    wunsch ();  
    wunsch ();  
    fee ();  
}
```

Anmerkung: In diesem Beispiel **fehlt** die Terminierungsbedingung!

Rekursive Funktionen III

Was ist also Rekursion?

- ▶ Eine Funktion f ist **rekursiv**, wenn
 - ▶ der Funktionsrumpf einen Aufruf der Funktion f **selbst** enthält oder eine Funktion g , die wiederum f **aufruft**.
 - ▶ eine **Terminierungsbedingung** existiert.
 - ▶ jede Eingabe nach **endlich** vielen Schritten **terminiert**.

```
void fee () {  
    wunsch ();  
    wunsch ();  
    if (noch_immer_nicht_genug ())  
        fee ();  
}
```

Anmerkung: In diesem Beispiel **existiert** die Terminierungsbedingung!

Rekursive Funktionen

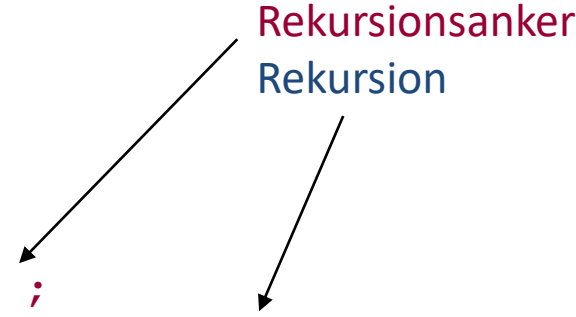
Beispiel: Fakultätsfunktion

- ▶ mathematische Definition

$$n! = \begin{cases} 1, & \text{falls } n = 0 \\ n \cdot (n-1)!, & \text{sonst} \end{cases}$$

- ▶ rekursive Funktion

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



Rekursionsanker
Rekursion

- Prolog
- Funktionen
- **Rekursion**

Aufbau der Rekursion

4! =

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

n = 4 n != 0 return (n * fakultaet(n-1))

n = 3 n != 0 return (n * fakultaet(n-1))

n = 2 n != 0 return (n * fakultaet(n-1))

n = 1 n != 0 return (n * fakultaet(n-1))

n = 0 n == 0 return (1)

EINI LogWing /
WiMa

Kapitel 4
Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**

Abbau der Rekursion

4! =

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

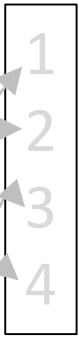
n = 4 n != 0 return(n * fakultaet(n-1))

n = 3 n != 0 return(n * fakultaet(n-1))

n = 2 n != 0 return(n * fakultaet(n-1))

n = 1 n != 0 return(n * fakultaet(n-1))

n = 0 n == 0 return(1)



EINI LogWing /
WiMa

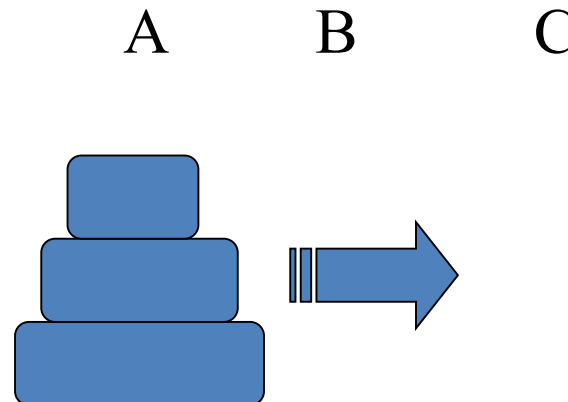
Kapitel 4
Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**

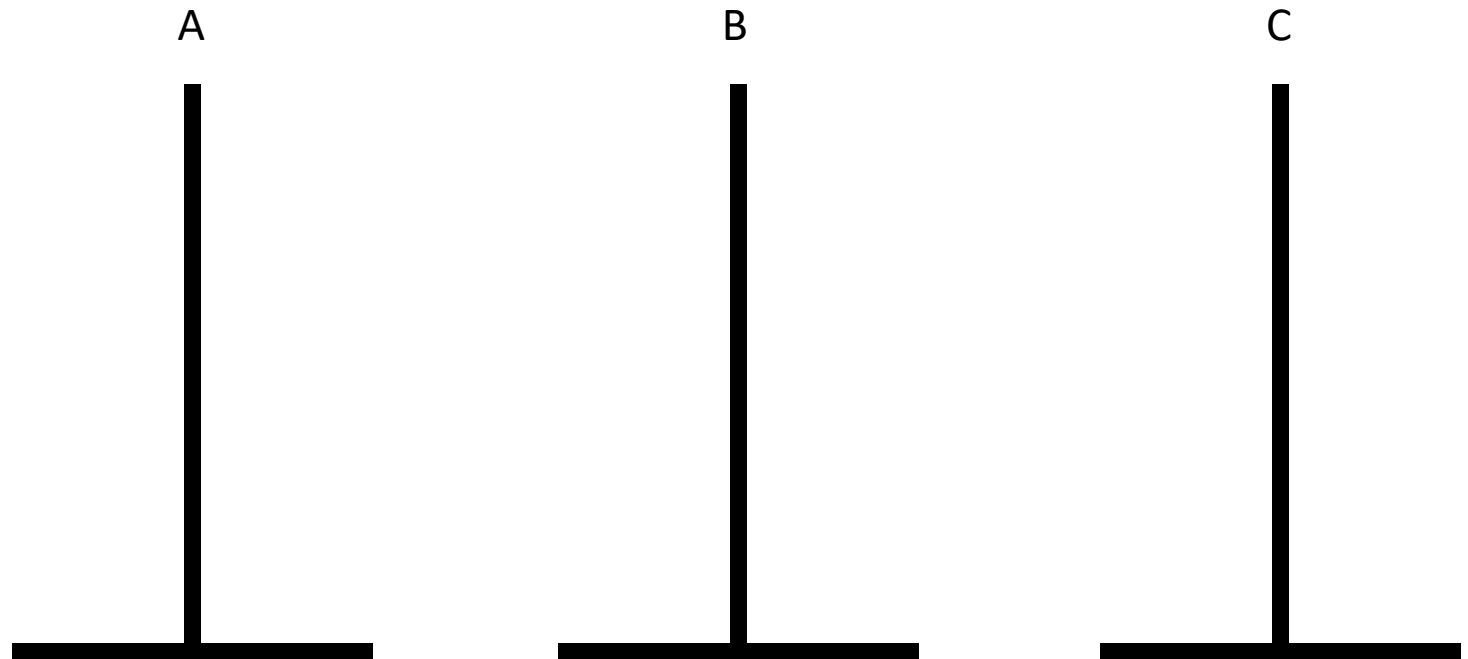
Beispiel: Türme von Hanoi I

- ▶ Ein Stapel von n Scheiben verschiedener Durchmesser sei als Turm aufgeschichtet. Der Durchmesser nimmt nach oben ab.
- ▶ Der Turm steht auf Platz A, soll nach Platz C verlagert werden, wobei Platz B als Zwischenlager benutzt werden darf.
- ▶ Randbedingungen:
 - Es darf jeweils nur 1 Scheibe bewegt werden.
 - Es darf nie eine größere auf einer kleineren Scheibe liegen.



Beispiel: Türme von Hanoi II

Einfachster Fall: 1 Scheibe von A nach C



EINI LogWing /
WiMa

Kapitel 4

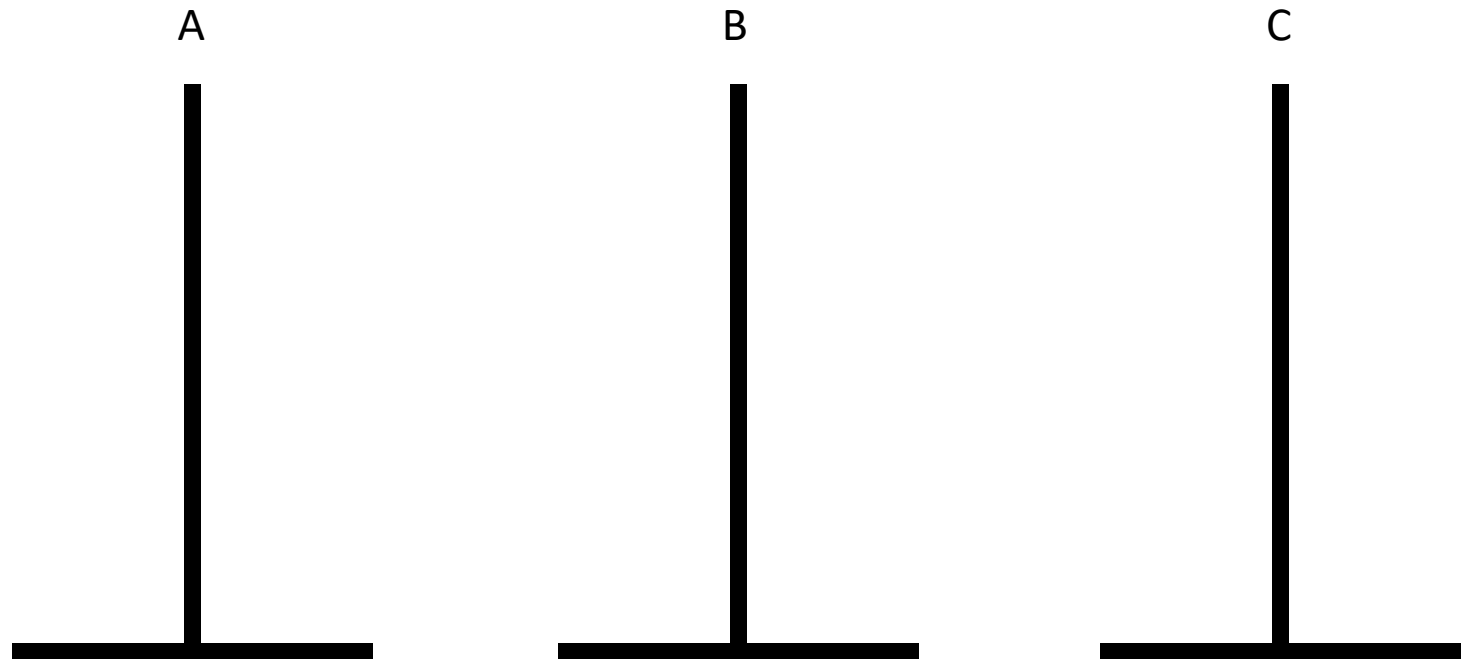
Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**

Beispiel: Türme von Hanoi III

Nächster Fall: 2 Scheiben von A nach C



EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**

Beispiel: Türme von Hanoi IV

Lösungsidee:

- ▶ Falls Turm mit $n-1$ Scheiben auf B, größte Scheibe auf A, dann kann einfach die Scheibe von A nach C verschoben werden, und äquivalentes Problem mit $n-1$ Scheiben für Start B, Ziel C und Zwischenlager A tritt auf.

```
int hanoi(int n, platz start, zwischen, ziel) {
    if (n==1) verschiebeScheibe(start,ziel) ;
    else
    {
        hanoi(n-1,start,ziel,zwischen) ;
        verschiebeScheibe(start,ziel) ;
        hanoi(n-1,zwischen,start,ziel) ;
    }
}
```

- Prolog
- Funktionen
- **Rekursion**

Beispiel: Türme von Hanoi V

Anzahl Scheiben	Benötigte Zeit*
5	31 Sekunden
10	17,1 Minute
20	12 Tage
30	34 Jahre

* Verschieben einer Scheibe dauert 1 Sekunde.

- Prolog
- Funktionen
- **Rekursion**

Beispiel: Türme von Hanoi VI

Anzahl Scheiben	Benötigte Zeit*
5	31 Sekunden
10	17,1 Minute
20	12 Tage
30	34 Jahre
40	34.800 Jahre
60	36,6 Milliarden Jahre**
64	585 Milliarden Jahre

* Verschieben einer Scheibe dauert 1 Sekunde.

** Alter des Universums: 13,7 Milliarden Jahre

EINI LogWing /
WiMa

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**



Rekursion

Artikel im EINI-Wiki:

→ **Rekursion**

Kapitel 4

Grundlagen
imperativer
Programmierung

In diesem Kapitel:

- Prolog
- Funktionen
- **Rekursion**



Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- ▶ Nächste Vorlesung – WiMa 6.12.2018, 08:15
- ▶ Nächste Vorlesung – LogWing 7.12.2018, 08:15