EINI LW/WiMa

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 14/15

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@tu-dortmund.de
http://ls1-www.cs.tu-dortmund.de

Thema

- **Kapitel 4**
  Grundlagen imperativer und objektorientierter Programmierung:
  - **Funktionen**
  - **Rekursion**

- **Unterlagen**
  - Gumm/Sommer, Kapitel 2.7 & 2.8
  - Echtle/Goedicke, Einführung in die Programmierung mit Java, dpunkt Verlag, Kapitel 4
  - Doberkat/Dißmann, Einführung in die objektorientierte Programmierung mit Java, Oldenbourg, Kapitel 3.4 & 4.1

Zwischenstand

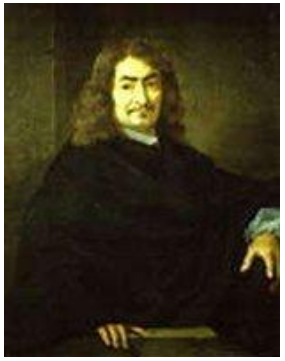
- Variablen
- Zuweisungen
- (Einfache) Datentypen und Operationen
  - Zahlen
    `integer, byte, short, long; float, double`
  - Wahrheitswerte `(boolean)`
  - Zeichen `(char)`
  - Zeichenketten `(String)`
  - Typkompatibilität
- Kontrollstrukturen
  - Sequentielle Komposition, Sequenz
  - Alternative,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

In diesem Kapitel:

- Prolog
- Funktionen
- Rekursion

Wiederholung

- Definition des Begriffs „Informatik“
  - nach der Akademie Francaise, angelehnt an „informatique“
  - „Behandlung von Information mit rationalen Mitteln“



- wobei rationale Mittel nach Descartes auszeichnet (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“

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

Unterprogramme

- Dieses Konzept wird in Programmiersprachen durch **Unterprogramme** unterstützt
  - 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.

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**

Einfaches Beispiel

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

```
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);
```


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 Variable

m: Rückgabewert

In diesem Kapitel:

- Prolog
- Funktionen
- Rekursion

Beispiel double sqrt(double x)

```
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

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) ;
  }
}
```

Verwendung von Funktionen

- Aufruf einer Funktion **ohne Rückgabewert**,
- Aufruf einer Funktion **mit Rückgabewert**,
- daher entsprechend verwendbar:

```
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 anstelle einer Zuweisung

- Die Lösung von Teilproblemen durch Prozeduren (Funktionen) nennt man **prozedurale (funktionale) Abstraktion**.

Beispiel main(…)

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

- **Name**
  - main
- **Rückgabewert**
  - kein Rückgabewert
- **Parameter**
  - vorhanden
  - wir nutzen die Parameter zur Zeit noch nicht
- **public (später im Rahmen der Objektorientierung)**
- **static (später im Rahmen der Objektorientierung)**

Top-Down Entwurf

- Top-Down Strategie
  - Zerlege Problem in Teilprobleme
  - Löse Teilprobleme
  - Kombiniere Lösung der Teilprobleme zur Lösung des Gesamtproblems

- Im Entwurf
  - Zerlege Problem in Teilprobleme
  - Deklariere für jedes Teilproblem eine Funktion, die im Entwurf **zunächst unausgefüllt** bleibt
    Stubs & Skeleton Prinzip (**Stummel** & **Skelett**)
  - Löse Gesamtproblem (**Skelett**) mit Hilfe der **Stubs**
  - fülle **Stubs**

Top-Down Entwurf

- Hinweis
  - Definiere Teilprobleme möglichst so, dass sie als bekannte allgemeine Probleme aus der Informatik erkennbar werden, für die eine bekannte Lösung genutzt werden kann
    (aus Software-Bibliotheken)

- Vorgehen liefert wg. funktionaler Abstraktion eine Zerlegung nach Funktionen, nach Aufgaben.
  - Typisch für imperative Programmierung.
  - Es existieren Alternativen: Zerlegung nach Daten.
  - Sichtweise für objektorientierte Programmierung ist etwas anders.

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()
```

Kommunikation Haupt- und Unterprogramm

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

- **Kommunikation über globale Variable**:
  - Variable, die als global deklariert sind, können in Unterprogrammen ebenfalls gelesen & verändert werden.
  - Verwendung von **globalen Variablen** in Funktionen führen zu Funktionen, deren genaue Auswirkungen schwer überblickt werden
    → so genannte **Seiteneffekte**.
  - Daher nur sehr begrenzt sinnvoll einsetzbar.

Kommunikation Haupt- und Unterprogramm

- **Kommunikation über Parameter**:
  - Die Eingabeparameter: liefern Informationen, die innerhalb des Unterprogramms nur gelesen werden
  - Der Rückgabeparameter einer Funktion: liefert Wert, der von der aufrufenden Funktion / Prozedur gelesen werden kann.
    - Begrenzte Möglichkeiten
    - Häufig für einfache Rückgaben, z.B. boolesche Resultate, Auftreten von Fehlern genutzt

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

Werteparameter / Variablenparameter

- Beobachtung: **Werteparameter / call by value**
  - bei einem Funktionsaufruf werden die Parameter mit konkreten Werten belegt
  - Sichtweise: Parameter sind **funktionslokale** Variable, 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**

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) ;
  }
}
```


In diesem Kapitel:

- Prolog
- **Funktionen**
- Rekursion

Werteparameter / Variablenparameter

- **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**

Abarbeiten von Funktionsaufrufen

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 ok, bei Funktionen wegen Rückgabeparametern umständlich
  - Erzeugt unnötig umfangreichen Code in Maschinensprache

- Daher nur in speziellen Kontexten unterstützt

Abarbeiten von Funktionsaufrufen

2. Idee: Unterliegende Basismaschine (Prozessor) unterstützt Funktionsaufrufe

- Prozess besteht aus Speicherbereichen für
  - **Verwaltungsinformation** für das Betriebssystem (Prozessorstatuswort, Programmzähler, ...)
  - **Programmcode**
  - **Heap** (= Haufen)
    - Menge aller Variablen, die zur Laufzeit verwendet wurden und noch nicht freigegeben wurden
  - **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 verschachtelten 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
- Verfeinerung
  - Unterprogramme, Prozeduren, Funktionen
  - Blockstrukturierung
- **Rekursion**

In diesem Kapitel:

- Prolog
- Funktionen
- Rekursion

Rekursive Funktionen

- **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 einer Funktion g, die wiederum f **aufruft**.
  - Eine **Terminierungsbedingung** existiert.
  - Jede Eingabe nach **endlich** vielen Schritten **terminiert**.

Was ist Rekursion

Eine k

Ein M

Beobachtung:

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

Was ist Rekursion

Noch

Beobachtung:

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

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,
…

Rekursive Funktionen

Was ist also Rekursion?

- Eine Funktion f ist **rekursiv**, wenn
  - der Funktionsrumpf einen Aufruf der Funktion f **selbst** enthält oder einer 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

Was ist also Rekursion?

- Eine Funktion f ist **rekursiv**, wenn
  - der Funktionsrumpf einen Aufruf der Funktion f **selbst** enthält oder einer 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!

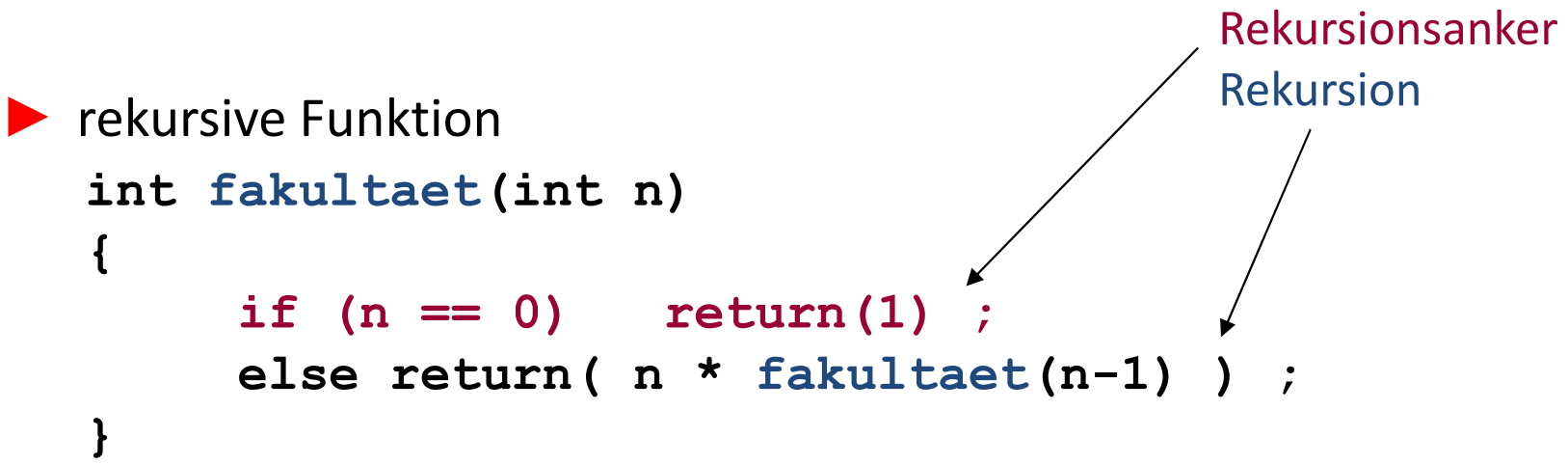
Rekursive Funktionen

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) ) ;
}
```

In diesem Kapitel:
- Prolog
- Funktionen
- Rekursion

Aufbau der Rekursion

4! =

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

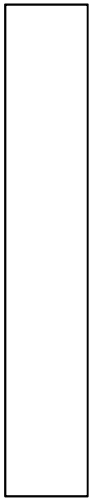
n = 4 n != 0 **return(n * fakultaet(n-1))**

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)**



Abbau der Rekursion

4! =

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

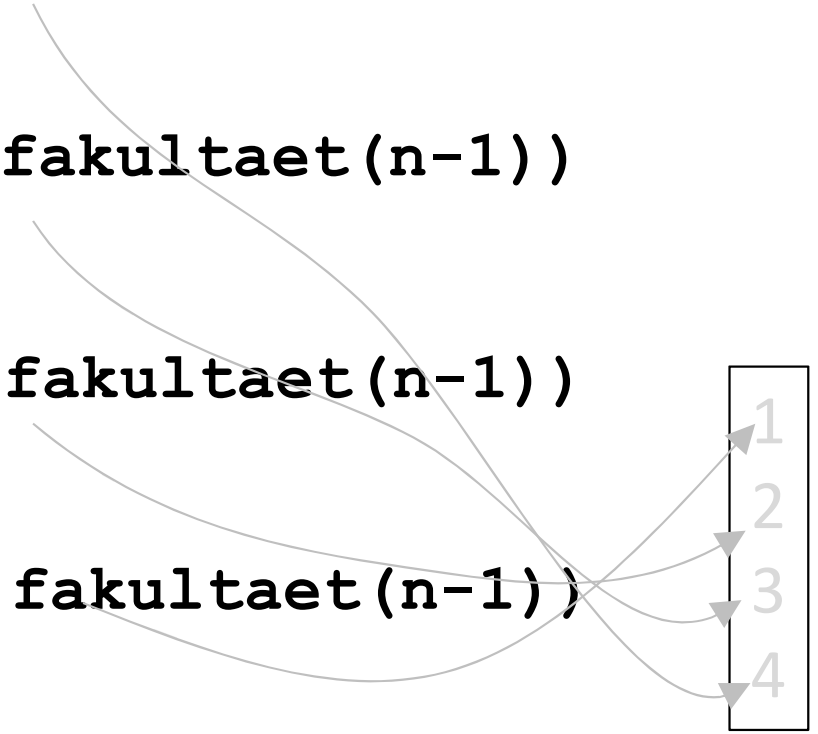
n = 4 n != 0 **return(n * fakultaet(n-1))**

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)**



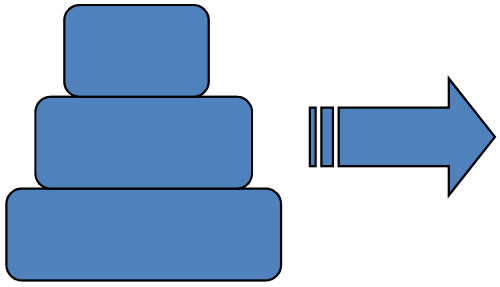
Beispiel: Türme von Hanoi

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.

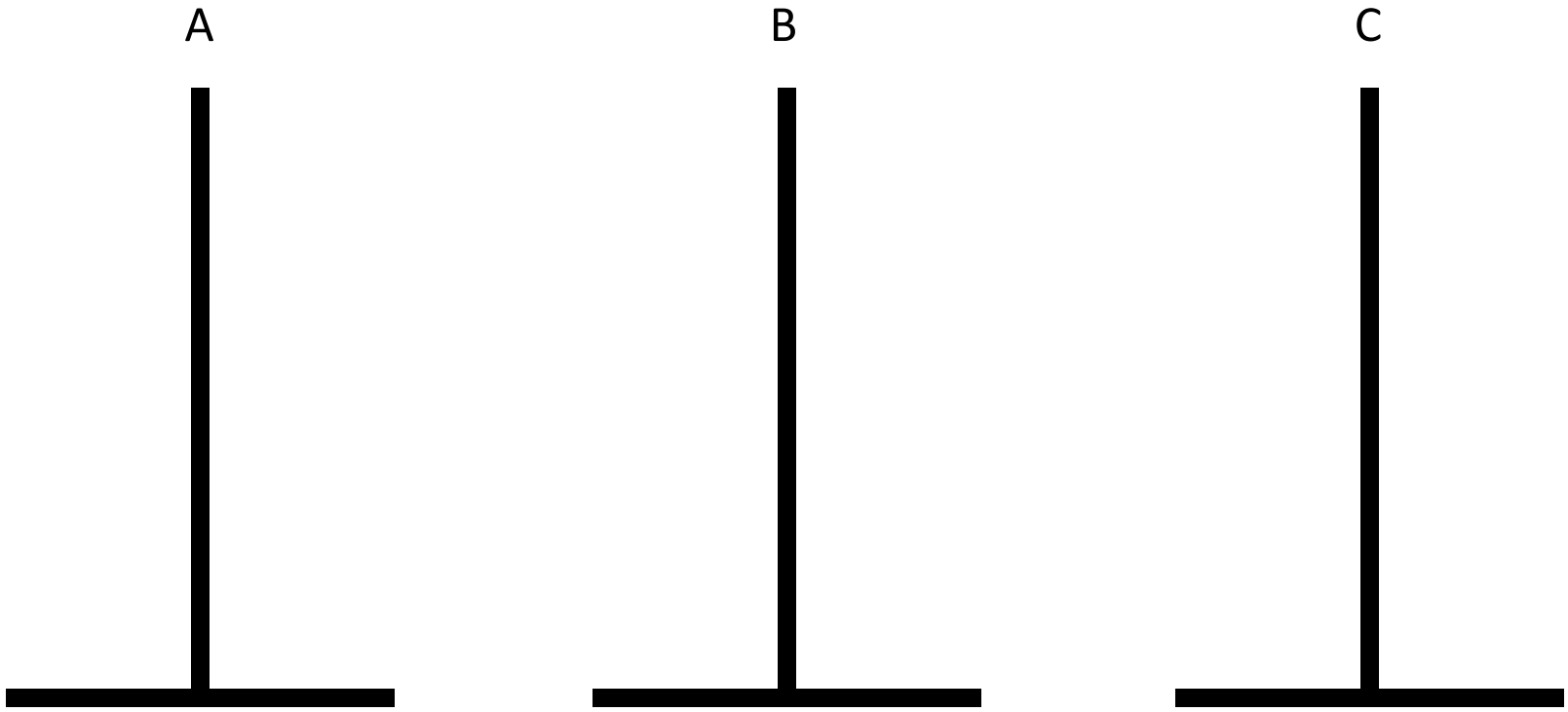
- Randbedingungen
  - jeweils nur 1 Scheibe darf bewegt werden
  - es darf nie eine größere auf einer kleineren Scheibe liegen

A B C



Beispiel: Türme von Hanoi

Einfachster Fall: 1 Scheibe von A nach C

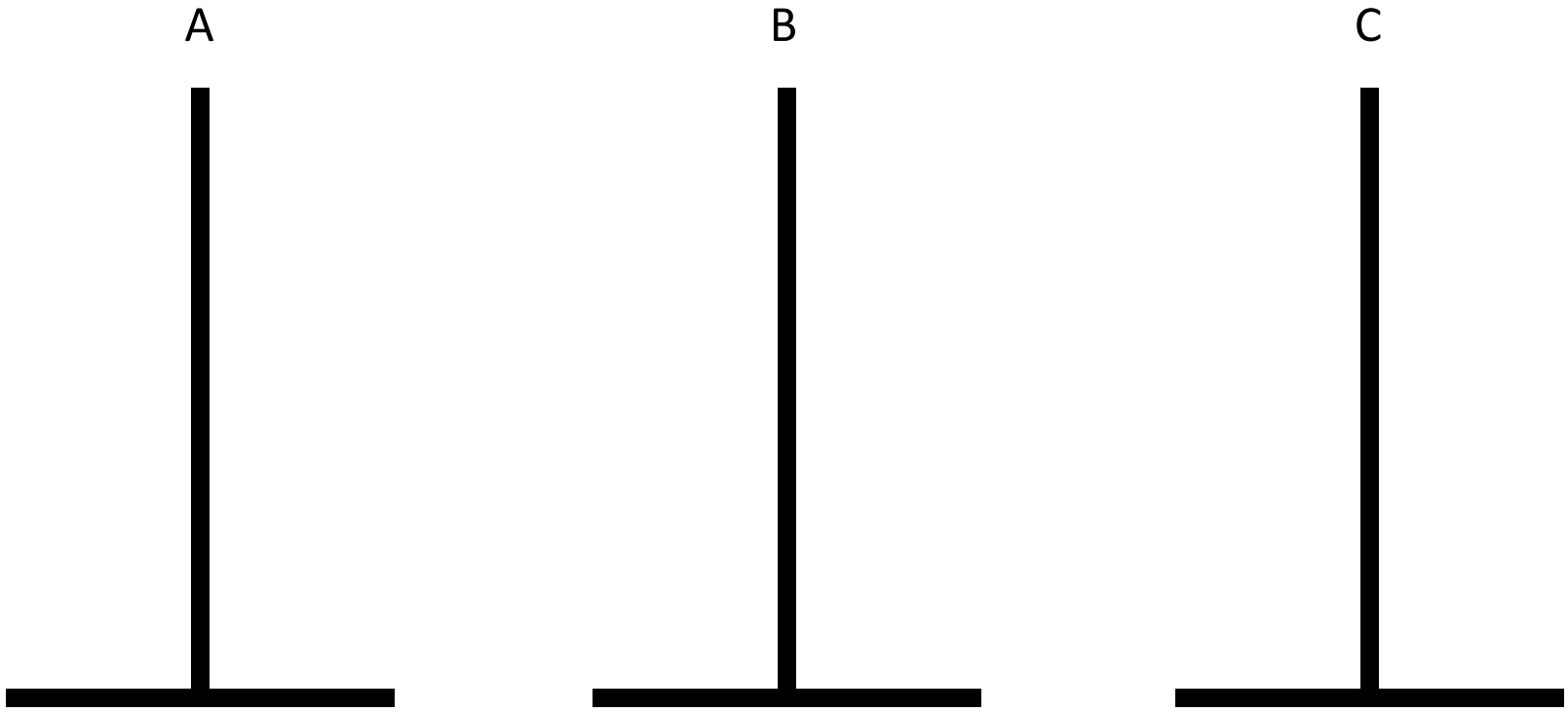


In diesem Kapitel:

- Prolog
- Funktionen
- Rekursion

Beispiel: Türme von Hanoi

Nächster Fall: 2 Scheiben von A nach C



In diesem Kapitel:

- Prolog
- Funktionen
- Rekursion

Beispiel: Türme von Hanoi

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);
   }
}
```

Beispiel: Türme von Hanoi

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

* Verschieben einer Scheibe dauert 1 Sekunde

In diesem Kapitel:

- Prolog
- Funktionen
- Rekursion

Beispiel: Türme von Hanoi

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

In diesem Kapitel:
- Prolog
- Funktionen
- Rekursion

Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- Nächste Vorlesung – WiMa 4.12.2014, 08:15
- Nächste Vorlesung – LogWIng 5.12.2014, 08:15