EINI LW

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 11/12

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

technische universität dortmund

Dr. Lars Hildebrand

- **Kapitel 4**
  Grundlagen imperativer und objektorientierter Programmierung:
  - **Funktionen**
  - **Prozeduren**
  - **Methoden**
  - **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

Dr. Lars Hildebrand

- Variablen
  - Bezeichner, Datentyp, Speicherort, Wert
- Zuweisungen
  - Zuweisungen müssen typverträglich sein
- (Einfache) Datentypen und Operationen
  - `integer (byte, short, long; float, double)`
  - `boolean`
  - `char`
  - `String`
  - Typkompatibilität
- Kontrollstrukturen
  - Sequentielle Komposition, Sequenz
  - Alternative, Fallunterscheidung
  - Schleife, Wiederholung, Iteration
- Verfeinerung
  - **Unterprogramme, Prozeduren, Funktionen**
  - Blockstrukturierung
- Rekursion

Zur Erinnerung

Dr. Lars Hildebrand

- 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

Dr. Lars Hildebrand

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

Beispiel: einfache Numerik Funktionen

Dr. Lars Hildebrand

- **Berechnung der Quadratwurzel**

  `double sqrt( double n)` für n > 0

- durch eine Funktion (Unterprogramm).

- Nützlichkeit klar,
  - in vielen Programmen unabhängig vom Kontext verwendbar
  - daher auch in Bibliotheken / Libraries stets verfügbar

- Eine Berechnungsidee: Intervallschachtelung
- vgl. diesbezügliche Ausführungen in Kap. 3

technische universität dortmund
Dr. Lars Hildebrand

```
double x = Eingabe (),
       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);
```

Beispiel double sqrt(double x)

technische universität dortmund
Dr. Lars Hildebrand

```
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

Beispiel double sqrt(double x)

technische universität dortmund

Dr. Lars Hildebrand

```
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

Beispiel main(…)

Dr. Lars Hildebrand

```
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)**

Anwendung von sqrt(…)

technische universität dortmund

Dr. Lars Hildebrand

```
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) ;
  }
}
```

Verwendung von Prozeduren, Funktionen

Dr. Lars Hildebrand

- Aufruf einer **Prozedur** gilt als **Anweisung**,
- Aufruf einer **Funktion** gilt als **Ausdruck,**
- daher entsprechend verwendbar:

  z.B.

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

- also
  - Funktionen auf der rechten Seite einer Zuweisung und in Ausdrücken
  - Prozeduren anstelle einer Zuweisung
  - Funktionen anstelle einer Zuweisung, falls kein Rückgabewert definiert ist (void) oder der Rückgabewert ignoriert werden soll

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

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 Prozedur / 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

Dr. Lars Hildebrand

```
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()
```

technische universität dortmund

Dr. Lars Hildebrand

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 **globale**n **Variablen** in Funktionen führen zu Funktionen, deren genaue Auswirkungen schwer überblickt werden
    → so genannte **Seiteneffekte**.
  - Daher nur sehr begrenzt sinnvoll einsetzbar.

technische universität dortmund

Dr. Lars Hildebrand

- **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

Dr. Lars Hildebrand

- 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(…)

technische universität dortmund

Dr. Lars Hildebrand

```
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) ;
  }
}
```

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

Dr. Lars Hildebrand

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

Dr. Lars Hildebrand

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**

Rekursive Funktionen

Dr. Lars Hildebrand

- 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 kleine Geschichte zur Rekursion:

Ein Mann geht durch den Wald und trifft eine gute Fee!

Was ist Rekursion

Eine kleine Geschichte zur Rekursion:

Ein Mann geht durch den Wald und trifft eine gute Fee!

Was ist Rekursion

rs Hildebrand

Eine

Beobachtung:

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

Ein Mann geht durch den Wald und trifft eine gute Fee.

Was ist Rekursion

Noch ein Mann geht durch den Wald und trifft eine gute Fee!

technische universität dortmund
Dr. Lars Hildebrand

Noch ein Mann geht durch den Wald und trifft eine gute Fee!

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 Informatiker geht durch den Wald und trifft eine gute Fee!

Was ist Rekursion

Dr. Lars Hildebrand

Ein Informatiker geht durch den Wald und trifft eine gute Fee!

Was ist Rekursion

Ein Informatiker geht durch den Wald und trifft eine gute Fee!

Was ist Rekursion

rs Hildebrand

Ein I !

Beobachtung:

1. Der Informatiker ist eher einfachen Gemüts.
2. Er kennt die Rekursion!

technische universität dortmund

Dr. Lars Hildebrand

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!

technische universität dortmund

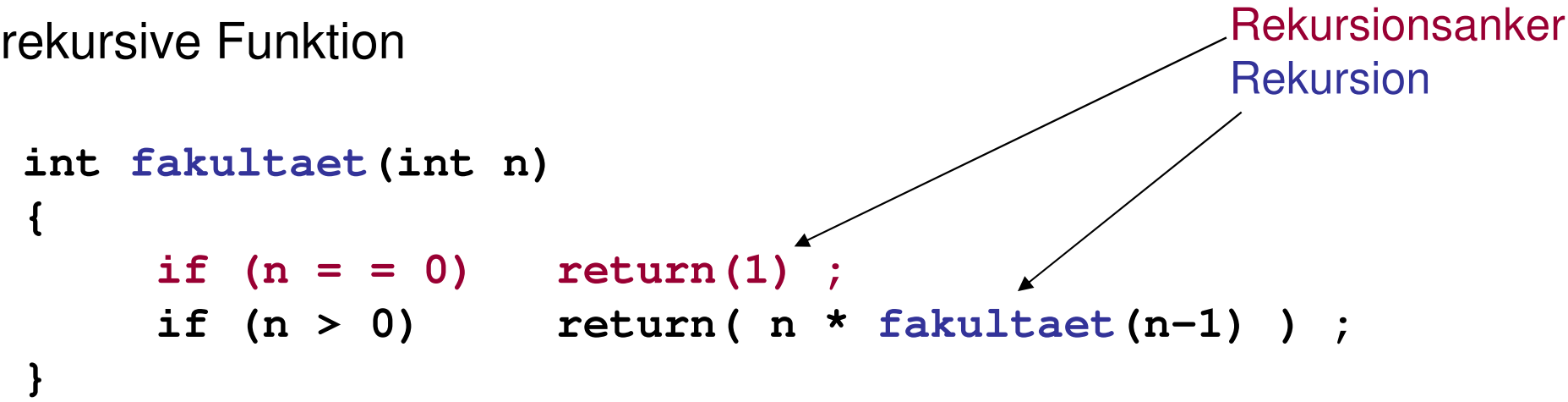
Dr. Lars Hildebrand

Beispiel: Fakultätsfunktion

- mathematische Definition

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

- rekursive Funktion

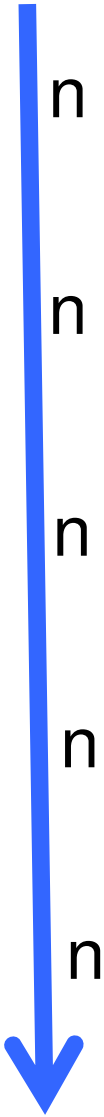


Aufbau der Rekursion

Dr. Lars Hildebrand

4! =

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



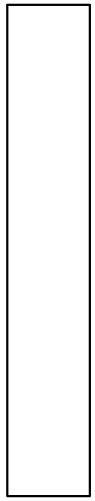
n = 4 n != 0 **return(n * fakultaet(n-1))**

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)**

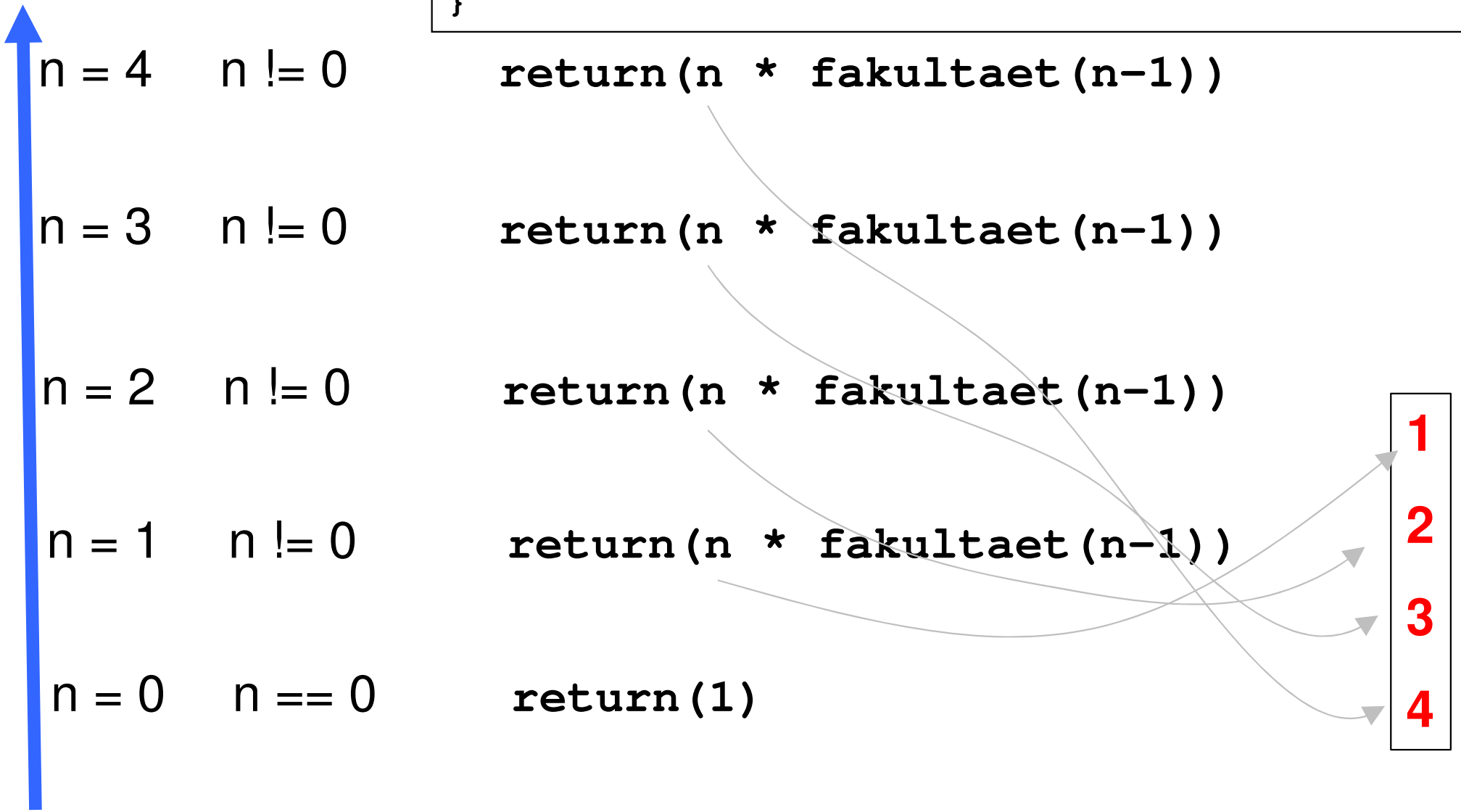


Abbau der Rekursion

Dr. Lars Hildebrand

4! =

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



Rekursion ist ein mächtiges Konzept

- Viele Probleme lassen sich mit rekursiven Funktionen elegant beschreiben und lösen

- Beispiele:
  - Türme von Hanoi
  - Backtracking für Suchalgorithmen z.B. in Spielstrategien

- **Rekursion** ist aber **nicht-trivial**:
  - z.B. für die ULAM Funktion ist Terminierung für nat. Zahlen bis heute unbewiesen

$$f(n) = \begin{cases} 1, & \text{falls } n = 1 \\ \text{f(n/2)}, & \text{falls n gerade} \\ \text{f(3n+1)}, & \text{sonst} \end{cases}$$

Rekursion ist ein mächtiges Konzept

Dr. Lars Hildebrand

- ULAM
  - benannt nach Stanislaw Marcin Ulam
  - Terminierung wird vermutet, ist zur Zeit unbewiesen

$$f(n) = \begin{cases} 1, & \text{falls } n = 1 \\ \text{f(n/2)}, & \text{falls n gerade} \\ \text{f(3n+1)}, & \text{sonst} \end{cases}$$

- ULAM von 7
  - 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
- ULAM von 27
  - 27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Rekursion ist ein mächtiges Konzept

Dr. Lars Hildebrand

- Erkennen der Berechnung

$$f(n) = \begin{cases} n - 10, & \text{falls } n > 100 \\ \text{f(f(n + 11))}, & \text{sonst} \end{cases}$$

Rekursion ist ein mächtiges Konzept

- McCarthys „91-Funktion“

$$f(n) = \begin{cases} n - 10, & \text{falls } n > 100 \\ \text{f(f(n+11))}, & \text{sonst} \end{cases}$$

f(100) = f(f(111)) = f(101) = 91

f(99) = f(f(110)) = f(100) = 91

f(98) = f(f(109)) = f(99) = 91

f(97) = f(f(108)) = f(98) = 91

....

f(91) = f(f(102)) = f(92) = 91

.....

f(88) = f(f(99)) = f (91) = 91

.....

Also: f(n) = 91 für 1≤n≤100

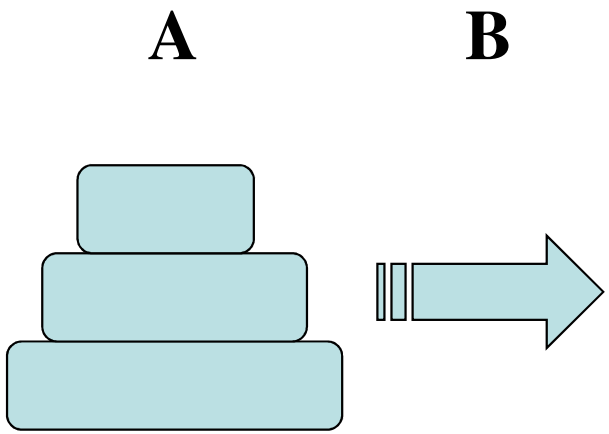
Beispiel: Türme von Hanoi

Dr. Lars Hildebrand

Ein Stapel von N Scheiben verschiedener Durchmesser sei als Turm aufgeschichtet. Der Durchmesser nimmt nach oben ab.

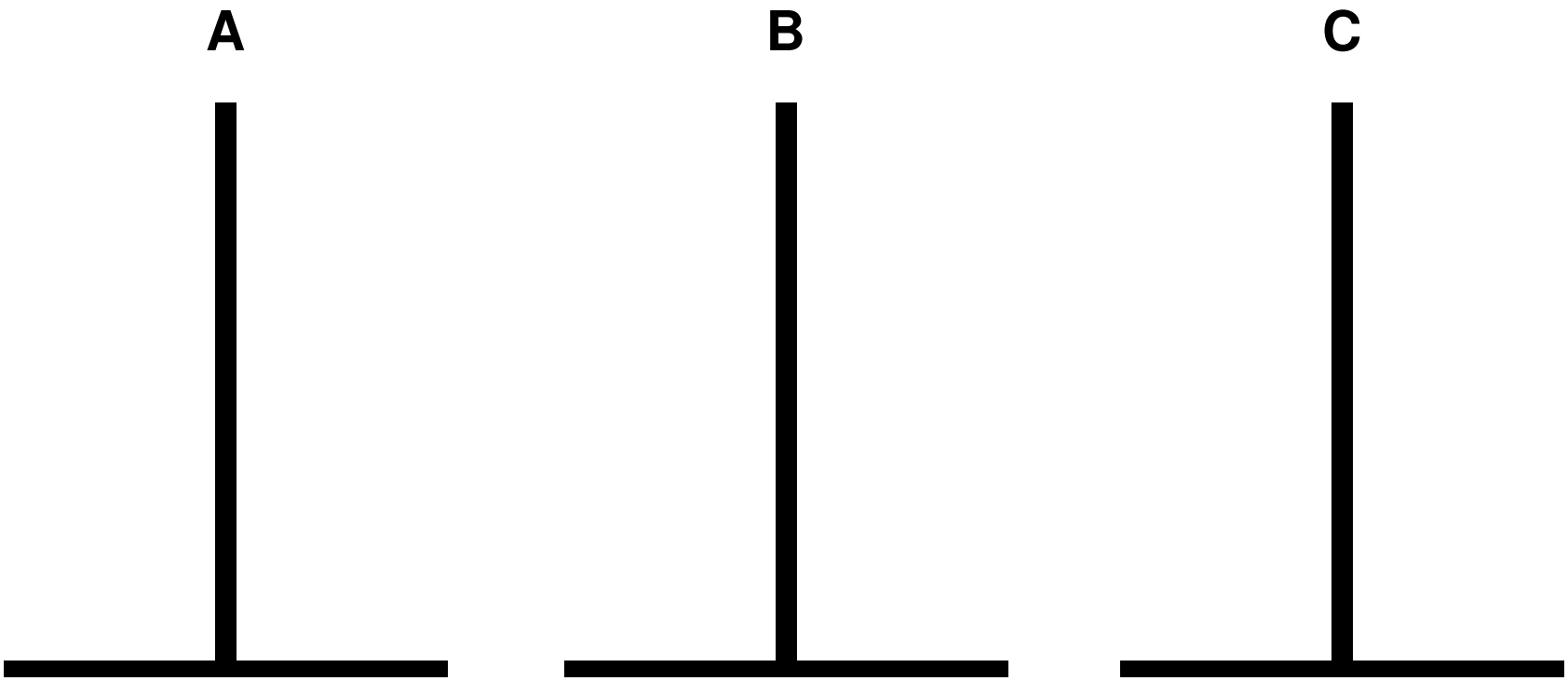
Der Turm steht auf Platz A, soll nach Platz C verlagert werden, wobei Platz B als Zwischenlager benutzt werden darf.

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



technische universität dortmund
Dr. Lars Hildebrand

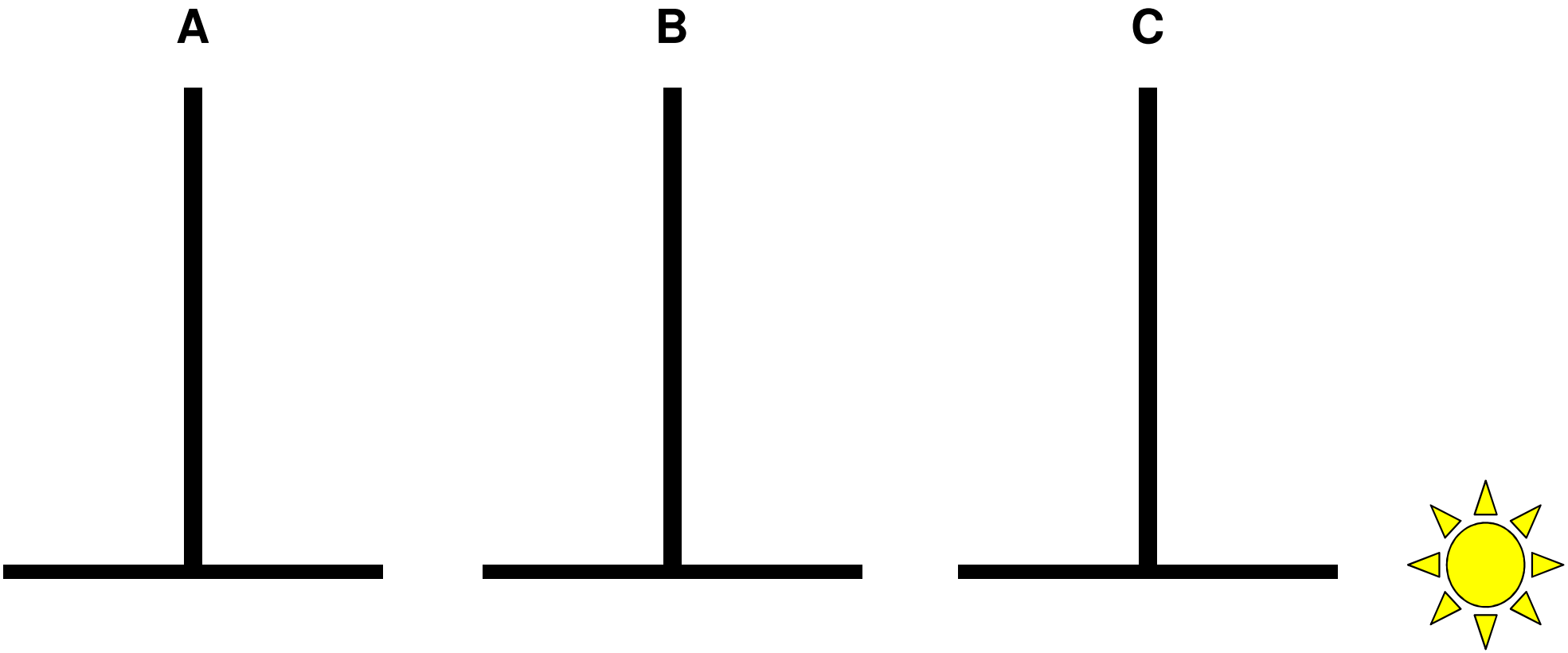
Einfachster Fall: 1 Scheibe von A nach C



technische universität dortmund

Dr. Lars Hildebrand

Nächster Fall: 2 Scheiben von A nach C



Beispiel: Türme von Hanoi

Dr. Lars Hildebrand

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
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

technische universität dortmund

Dr. Lars Hildebrand

Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- Nächste Vorlesung 2.12.2011, 8:30