# Prozedurale Programmierung

## Rekursion

**Hochschule Rosenheim - University of Applied Sciences**

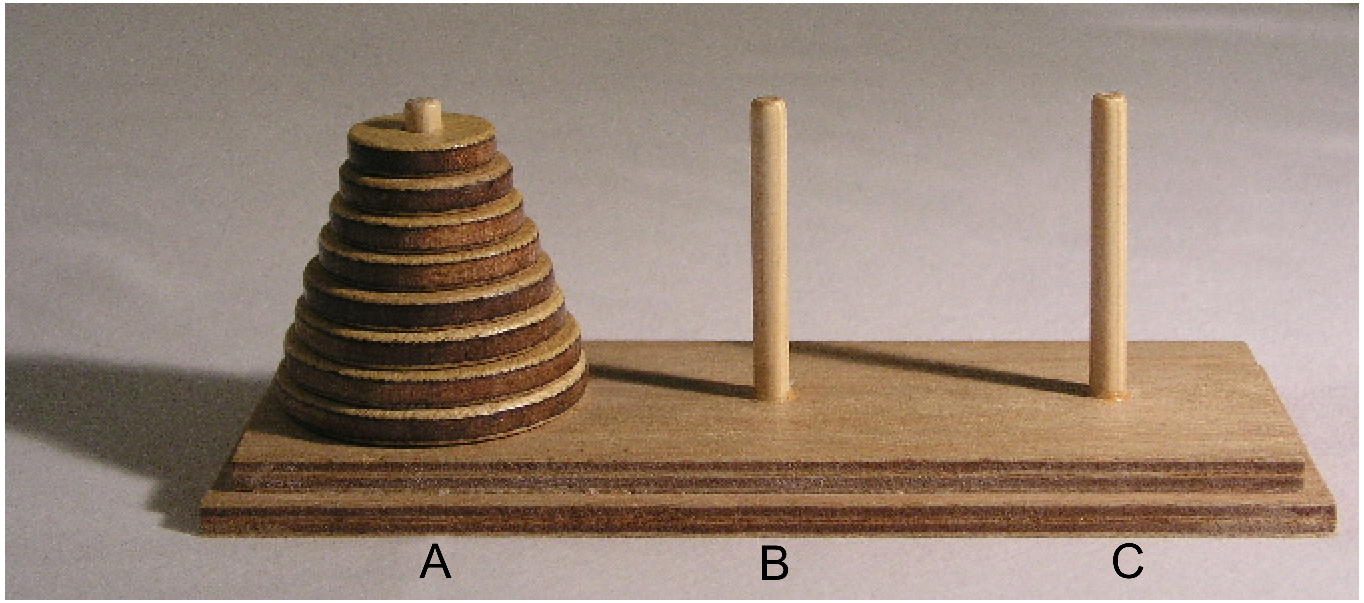
**WS 2018/19**

**Prof. Dr. F.J. Schmitt**

# Kapitel 16 Rekursion

Kapitel 16 behandelt einfache rekursive Funktionen

# Türme von Hanoi



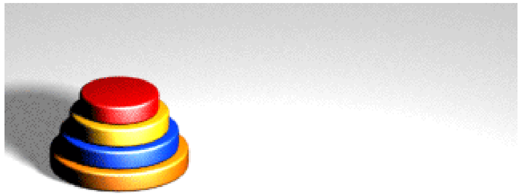

[Wikipedia, Bjarmason]

- Knobelspiel, erfunden 1883
- Ziel: Bewege den Turm von A nach C
  - ein Zug besteht aus dem Stecken einer Scheibe auf einen der Stäbe A, B, C
  - es darf immer nur eine Scheibe bewegt werden
  - es darf keine Scheibe auf eine kleinere gelegt werden (also: immer sortiert nach Größe)

# Aufgabe

- Lösen Sie das Problem für 3 Scheiben!

- Für 4 Scheiben:



[Wikipedia, Aka]

# Einführung

- jede C-Funktion hat ihren eigenen Satz lokaler Variablen
- eine Funktion kann sich selbst rekursiv aufrufen

- ermöglicht in vielen Fällen einfachere Lösungen als Iteration
  - Operationen auf Bäumen (z. B. Suche nach einem Element)
  - viele Sortieralgorithmen (Quicksort, Mergesort)

- sehr mächtiges Konzept einer Programmiersprache
  - ist nicht in jeder Sprache möglich
  - in Cobol oder Fortran geht das z.B. nicht
  - dagegen kennen einige andere Sprachen ausschließlich Rekursion, Iterationen (mit Schleifen) sind nicht möglich
    - insbesondere funktionale Sprachen, z.B. Lisp, Haskell

# Einführung

- allgemeine Form der primitiven Rekursion (für natürliche Zahlen als Parameter):

$$f(0, \boldsymbol{y}) = g(\boldsymbol{y}), \qquad \boldsymbol{y} \in \mathbb{N}_0^n$$

$$f(x+1, \boldsymbol{y}) = h\big(x, \boldsymbol{y}, f(x, \boldsymbol{y})\big), \qquad x \in \mathbb{N}_0, \boldsymbol{y} \in \mathbb{N}_0^n$$

- $f$ kommt auf linker und rechter Seite vor
- bei jedem Aufruf werden die Parameter modifiziert
- damit der Algorithmus terminiert, muss ein geeignetes Abbruchkriterium definiert sein
- Spezialfall: $f$ ruft sich direkt selbst auf, ohne zusätzliche Parameter:

$$f(0) = \text{const},$$

$$f(x+1) = h(x, f(x)), \qquad x \in \mathbb{N}_0$$

# Beispiele

- Addition von zwei Zahlen x und y
  add(x, y) = add(x – 1, y) + 1
  add(0, y) = y

- Multiplikation
  mult(x, y) = mult(x – 1, y) + y
  mult(0, y) = 0

# Fakultät

- iterative Definition:

$$n! := \prod_{i=1}^{n} i$$

- rekursive Definition:

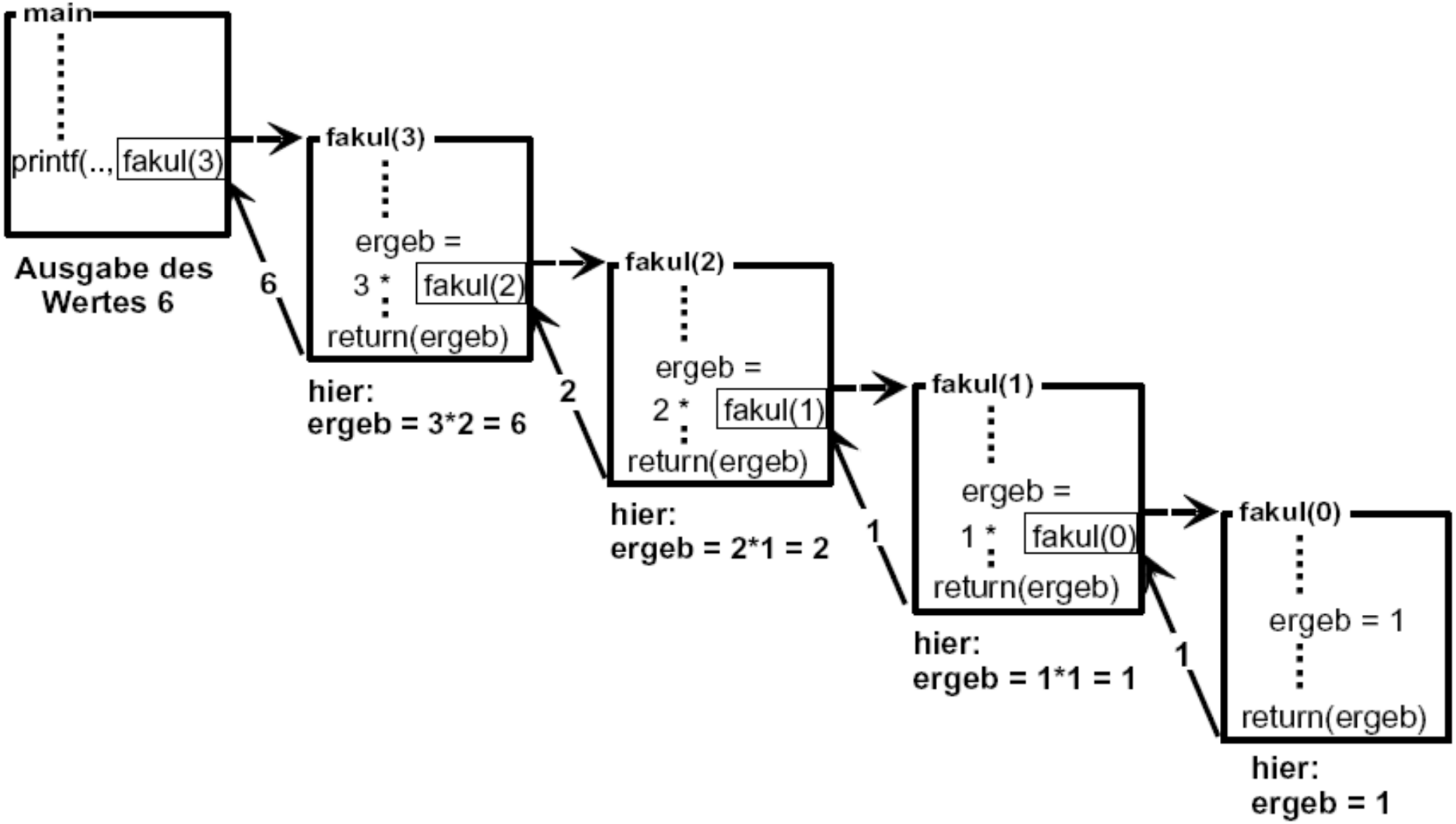
$$n! := n \cdot (n-1)!$$
$$0! := 1$$

# Fakultät – rekursiv

```c
#include <stdio.h>
/* fakul.c */
int fakul(int zahl) {
 int ergeb;
  if (zahl>0)
      ergeb = zahl * fakul(zahl- 1);
  else
      ergeb = 1;
  return(ergeb);
}
```

```c
int main(void) {
  int  bis;
  printf("Zahl: ");
  scanf("%d", &bis);
  printf("%d! = %d\n",
           bis, fakul(bis));
  return 0;
}
```

# Fakultät – Ablaufbeispiel



# Fakultät – iterativ

```
#include <stdio.h> /* fakul2.c */
int main(void) {
    int    i, bis, ergeb=1;

    printf("Zahl: ");
    scanf("%d", &bis);
    for (i=1; i<=bis; i++)
        ergeb *= i;
    printf("%d! = %d\n", bis, ergeb);
    return 0;
}
```

# Aufgabe

- Schreiben Sie eine Funktion sum(), die die Summe aller natürlichen Zahlen von 1 bis n iterativ berechnet:

$$s(n) = \sum_{i=1}^{n} i$$

- Prototyp:

```
unsigned int sum(unsigned int n);
```

# Aufgabe

- Schreiben Sie nun eine Funktion sum(), die die Summe aller natürlichen Zahlen von 1 bis n rekursiv berechnet:

$$s(n) = \sum_{i=1}^{n} i$$

- Prototyp:

```
unsigned int sum(unsigned int n);
```

# Fibonacci-Zahlen

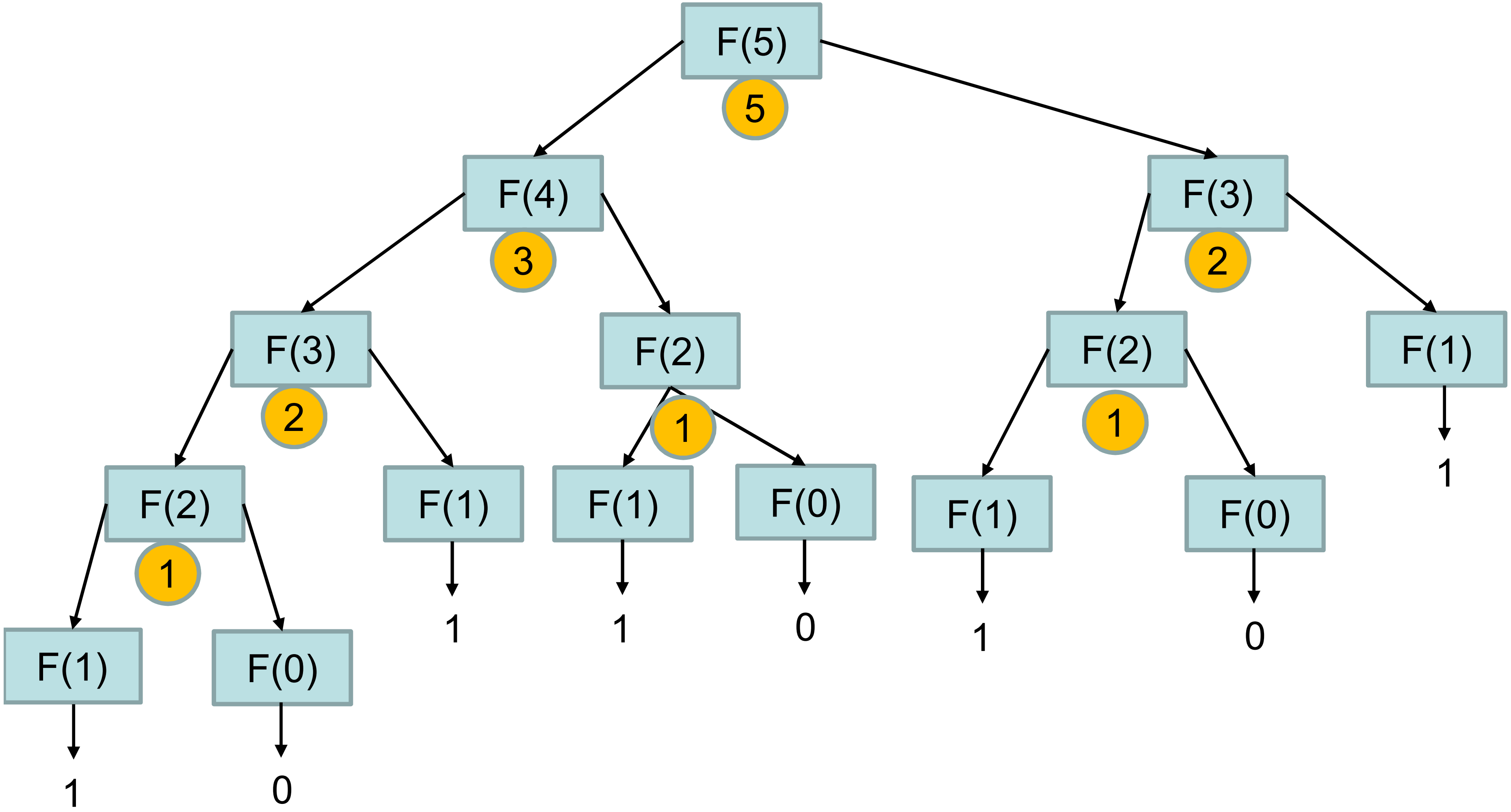
- Definition

  $$F(n) := F(n-1) + F(n-2) \qquad \text{für } n > 1$$
  $$F(1) := 1,\ F(0) = 0$$

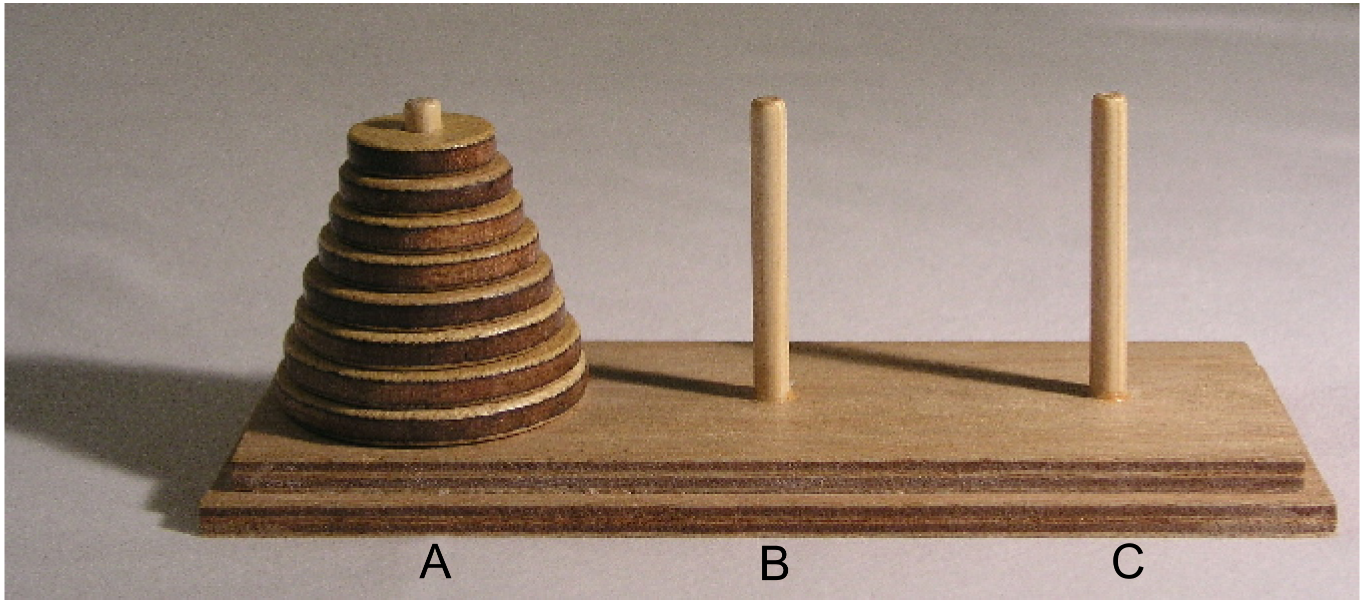
- ergibt die Fibonacci-Folge:
  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …
- rekursive Implementierung

```
int F(int n)
{
  int ret;

  if(n <= 0) ret = 0;
  else if(n == 1) ret = 1;
  else ret = F(n - 1) + F(n - 2);

  return ret;
}
```

# Fibonacci-Zahlen – Ablauf



# Türme von Hanoi




[Wikipedia, Bjarmason]

- Knobelspiel, erfunden 1883
- Ziel: Bewege den Turm von A nach C
  - ein Zug besteht aus dem Stecken einer Scheibe auf einen der Stäbe A, B, C
  - es darf immer nur eine Scheibe bewegt werden
  - es darf keine Scheibe auf eine kleinere gelegt werden (also: immer sortiert nach Größe)

# Lösung für 3 Scheiben

[Wikipedia, Aka]

# Rekursive Lösung

- Idee: gehe davon aus, dass das Problem für n – 1 Scheiben bereits gelöst ist
- Algorithmus:
  - bringe n – 1 Scheiben von A nach B (mit Hilfe von C)
  - bringe die letzte übrige Scheibe von A nach C
  - bringe die n – 1 Scheiben von B nach C (mit Hilfe von A)

```
void hanoi(int n, char von, char hilf, char nach)
{
  if(n > 0)
  {
    hanoi(n-1, von, nach, hilf);
    printf("Bewege Scheibe von %c nach %c\n", von, nach);
    hanoi(n-1, hilf, von, nach);
  }
}
```

# Aufgabe

- Schreiben Sie eine Funktion, die den ggT von zwei natürlichen Zahlen a und b berechnet

- Prototyp:

```
int ggT(int a, int b);
```

  - gehen Sie davon aus, das bei Start gilt: a >= b

# Anmerkungen

- jede Rekursion lässt sich als Iteration formulieren und umgekehrt
  - jedoch kann dazu eine while-Schleife nötig sein,
    eine reine Zählschleife genügt nicht immer
  - siehe GdI 2
- je nach Problem ist einmal die rekursive und ein andermal die iterative Lösung einfacher
- Vorsicht: Stacküberlauf möglich