WBS TRAINING

# Programmierung(2)

UNSER ♥ SCHLÄGT FÜR BILDUNG

# Agenda

- Rekursive Funktionen
  - Definition
  - Motivation
  - Beispiel
  - Gemeinsame Übung
  - Einzelübungen

- Fachpraktische Anwendungen

# Rekursive Funktionen – Definition

- Rekursion heißt wörtlich „zurücklaufen, zurückkehren“ und spielt damit bereits auf die in der Informatik übliche Bedeutung an: „**Selbstbezüglichkeit**“.
- Selbstbezüglichkeit ist ein sehr interessantes aber nicht immer unproblematisches Konzept:
  - problematisch:
    - ein Begriff X, dessen Definition selbst mit dem Begriff X arbeitet, ist offensichtlich unzulässig. (Er könnte X nur erklären, wenn X bereits erklärt wäre).
    - ein Beweis (für eine Aussage), der selbst bereits die zu beweisende Aussage voraussetzt, ist ebenfalls „zirkulär“ (dreht sich also buchstäblich „im Kreis“).
    - oder allgemein und anschaulich: Man kann sich (im Gegensatz zu Münchhausen ;-) nicht am eigenen Schopf aus dem Sumpf ziehen.
  - vorteilhaft:
    - wo Rekursion hingegen möglich ist, entstehen in der Regel ausgesprochen elegante Lösungen
    - rekursive Lösungen sind voraussetzungsarm => weniger Codierungs-Aufwand
- **Rekursive Funktionen** sind nun in sofern „selbstbezüglich“, als dass diese sich selbst aufrufen. Das heißt also: Innerhalb des Funktionsrumpfes einer Funktion X wird selbst wieder X aufgerufen.
- Das Konzept der Rekursion führte zum „Programmierparadigma“(eine besondere Form/Philosophie des Programmierens), der sogenannten „**Funktionalen Programmierung**“.
- Allerdings sei festgehalten: Jede Rekursive Lösung hätte auch in „klassischer Form“ geschehen können. Man spricht dann von einer **iterativen** Lösung (Substantiv: **Iteration**).

# Rekursive Funktionen – Motivation

- Wir haben bereits angesprochen, dass rekursive Lösungen oft **elegant** sind.
- Ebenso, dass durch eine rekursive Lösung **Code eingespart** werden kann.
- Ferner gilt: Rekursive Lösungen lassen sich einfacher „**parallelisieren**“ als dies bei iterativen Lösungen der Fall ist. (Nicht Thema dieses Kurses) : Parallelisieren=Mehrere Prozessoren arbeiten ein Problem gleichzeitig ab => Zeitersparnis
- Zudem werden wir mit dem sogenannten „**Quicksort**“ bereits morgen einen Sortier-Algorithmus kennenlernen, der (üblicherweise) in rekursiver Form genutzt wird.

- Und für alle **Ästheten** sei gesagt:

  Rekursive Lösungen sind oft sehr eindrucksvoll und zugleich von großer **Schönheit**: Eine Funktion, die zu einem Ergebnis kommt, indem sich diese Funktion „einfach“ solange selbst aufruft, bis dieses Problem quasi „verschwunden“ ist, hat durchaus etwas amüsantes und wird hoffentlich viele Teilnehmer dieses Kurses begeistern … seien Sie gespannt!

# Rekursive Funktionen – **Beispielaufgabe – Vorbemerkung(1)**

Die folgende Aufgabe verwendet das wohl berühmteste Beispiel zur Erläuterung von Rekursiven Funktionen. Es spricht dabei von einer mathematischen Operation, der sogenannten „**Fakultät**“ (Schreibweise: **!**), die wir zunächst kurz vorstellen wollen:

**0!** ist als **1** definiert

**1!** ist ebenfalls **1**

**2!** ist die Kurzschreibweise für 1*2 = **2**

**3!** ist die Kurzschreibweise für 1*2*3 = **6**

**4!** ist die Kurzschreibweise für 1*2*3*4 = **24**

**…**

allgemein:

**n!** ist die Kurzschreibweise für 1*2*3* … *(n-2)*(n-1)*n

# Rekursive Funktionen – **Beispielaufgabe – Vorbemerkung(2)**

Um den Gedanken der Rekursion am Beispiel der Fakultät deutlich werden zu lassen, bietet es sich an, das mehrfache Aufrufen der selben Funktion durch die folgende kleine Geschichte zu veranschaulichen. Dabei schlüpfen „Matheprofessoren“ in die Rolle einer Funktion: Anstelle des „Aufrufes einer Funktion“ (die etwas berechnen soll) bitten wir nun die Professoren, etwas zu berechnen:

Wir: „Herr Professor 1 (im Folgenden kurz: P1), was ist bitte das Ergebnis von **5!** ?“
P1: „**5!**, das ist das selbe wie **5 * 4!** … und das Ergebnis von **4!** soll P2 berechnen“
P2: „**4!**, das ist das selbe wie **4 * 3!** … und das Ergebnis von **3!** soll P3 berechnen“
P3: „**3!**, das ist das selbe wie **3 * 2!** … und das Ergebnis von **2!** soll P4 berechnen“
P4: „**2!**, das ist das selbe wie **2 * 1!** … und das Ergebnis von **1!** soll P5 berechnen“
P5: „Das Ergebnis von 1! Ist schlicht **1**“

**Da P5 eine konkrete Antwort geben konnte** (dies ist die Abbruchbedingung für die Rekursion, denn weitere Funktionen müssen nun nicht mehr aufgerufen werden), **können nun nacheinander auch alle anderen Professoren konkrete Antworten geben:**

P4: „Ich habe ja gesagt, dass 2!=2*1!, aber laut P5 ist 1!=1, also ist 2*1!=2*1=2, also kann ich verkünden: 2!=**2**“
P3: „Ich habe ja gesagt, dass 3!=3*2!, aber laut P4 ist 2!=2, also ist 3*2!=3*2=6, also kann ich verkünden: 3!=**6**“
P2: „Ich habe ja gesagt, dass 4!=4*3!, aber laut P3 ist 3!=6, also ist 4*3!=4*6=24, also kann ich verkünden: 4!=**24**“
P1: „Ich habe ja gesagt, dass 5!=5*4!, aber laut P2 ist 4!=24, also ist 5*4!=5*24=120, also kann ich verkünden: 5!=**120**“

**Und da wir an P1 unsere Frage gestellt haben, kennen wir nun die Antwort: 5!=120**

# Rekursive Funktionen – **Beispielaufgabe**

## Aufgabenstellung

- Das Programm fragt vom User zunächst eine positive ganze Zahl ab.
- Anschließend werden die beiden folgenden Funktionen gestartet:

  Funktionsname: …….. **fakultaet_iterativ**
  Übergabewerte: …….. 1 Integer n
  Funktionalität: …….. ermittelt per ITERATION das Ergebnis von n!
  Rückgabewert: ……. das berechnete Ergebnis

  Funktionsname: …….. **fakultaet_rekursiv**
  Übergabewerte: …….. 1 Integer n
  Funktionalität: …….. ermittelt per REKURSION das Ergebnis von n!
  Rückgabewert: ……. das berechnete Ergebnis

- Abschließend werden deren Rückgabewerte auf der Konsole ausgegeben und das Programm endet.

Da wir über Funktionen im allgemeinen schon sprachen, werden wir auch in diesem Fall auf eine Darstellung als PAP, Struktogramm oder Pseudocode verzichten und beide Funktionen nur als **Quellcode in ANSI C** vorführen.

# Rekursive Funktionen – **Beispielaufgabe** – Quellcode

```c
#include <stdio.h>

int fakultaet_iterativ(int n)
{
    int f=1,i;

    for(i=2;i<=n;i++)
    {
        f=f*i;
    }

    return f;
}

int fakultaet_rekursiv(int n)
{
    if(n<2) return 1;
    return n*fakultaet_rekursiv(n-1);
}

int main(void)
{
    int n,i,r;

    printf("Geben Sie bitte eine ganze Zahl ein: ");
    fflush(stdin);
    scanf("%d",&n);

    i=fakultaet_iterativ(n);
    r=fakultaet_rekursiv(n);

    printf("Ergebnis iterativ: %d\nErgebnis rekursiv: %d\n",i,r);

    return 0;
}
```

```
Geben Sie bitte eine ganze Zahl ein: 5
Ergebnis iterativ: 120
Ergebnis rekursiv: 120
```

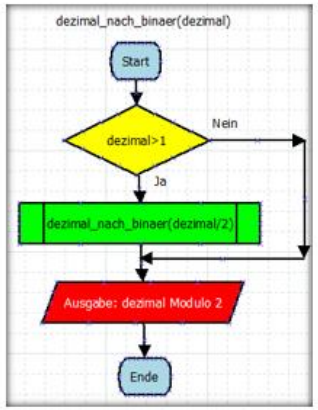
WBS TRAINING

# Rekursive Funktionen – Gemeinsame Übung A_05_02_01

WBS TRAINING

**Aufgabe_05_02_01**

Gegeben sei der folgende PAP:




**Aufgabenstellung:**
Bitte erstellen Sie dazu einen geeigneten **Quellcode** in ANSI C.

WBS TRAINING

# VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!