



# Informatik 1

Funktionen,  
Rekursive Funktionen  
Compiler,  
Compilerfehler und Laufzeitfehler

**Irene Rothe**

Zi. B 241

[irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)

Instagram: irenerothedesign



Hochschule  
Bonn-Rhein-Sieg

Vorlesung\_C\_FunktionenCompiler

# Informatik: 2 Semester für Ingenieure

**Informatik = Lösen von Problemen mit dem Rechner**

✓ Zum Lösen von Problemen mit dem Rechner braucht man **Programmierfähigkeiten** (nur mit Übung möglich): Was ist Programmieren?

✓ Was ist ein Flussdiagramm?

## **Programmiersprache C:**

- ✓ Elementare Datentypen
- ✓ Deklaration/Initialisierung
- ✓ Kontrollstrukturen: if/else, while, for

→ Funktionen

→ Felder (Strings)

→ Zeiger

→ struct

→ Speichieranforderung: malloc

→ Listen

→ Bitmanipulation

→ Wie löst der Rechner unsere Probleme? → mit **Dualdarstellung** von Zeichen und Zahlen und mit Hilfe von **Algorithmen**





→ Ein Beispiel für ein Problem: **Kryptografie**

→ Sind Rechner auch Menschen? → **Künstliche Intelligenz**

→ Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen



# Design der Folien

-  hinterlegt sind alle Übungsaufgaben. Sie sind teilweise sehr schwer, bitte absolut nicht entmutigen lassen! Wir können diese in Präsenz besprechen oder über Fragen im Forum.
-  hinterlegte Informationen und grüne Smileys sind wichtig und klausurrelevant.
- Alles hinter „**Achtung**“ unbedingt beachten!
-  verwende ich, wenn überraschende Probleme auftreten können. Wenn Sie schon programmiererfahrend sind, können das eventuell besonders große Überraschungen für Sie sein, wenn Sie eine andere Sprache als C kennen.
- „Tipp“ benutze ich, um Ihnen einen Weg zu zeigen, wie ich damit umgehen würde.
- „Bemerkung“ in Folien beziehen sich meist auf Sonderfälle, die nicht unbedingt klausurrelevant sind, aber für Sie beim Programmieren eine Bedeutung haben könnten
-  hinter diesem Symbol ist ein Link fürs Anhören bzw. Gucken weiterer Infos

Erfolg ist, wenn man von Misserfolg  
zu Misserfolg hüpfte, ohne die  
Begeisterung zu verlieren.

Churchill

# In dieser Vorlesung:

## → Funktionen

- Definition von Funktionen
- Aufruf von Funktionen
- Abarbeitung einer Funktion
- Arbeit mit Kopien der Funktionseingabewerte
- Ort der Funktionsdefinition
- Lokale Parameter

# Rekursive Funktionen Compiler Compiler- und Laufzeitfehler



## Aufbau der Folien:

- Am Anfang motiviere ich gerne mit einem Beispiel, das eventuell schwer verständlich ist. Wem das nicht zusagt, dem empfehle ich, diese Folien zu überspringen.
- Weiter arbeite ich mit vielen Beispielen, die oftmals immer wieder das Gleiche erklären nur auf unterschiedliche Arten. Hat man einen Sachverhalt einmal verstanden, braucht man eventuell diese Beispiele nicht.
- Folien, die mit **Einschub** beginnen, beinhalten Zusatzinformationen, die nicht nötig für das Verständnis des Themas sind.
- Grün hinterlegte Informationen sind das, was Sie aus der Vorlesung rausnehmen sollen, alles andere sind vertiefende Informationen und Motivation.

# Funktionen: Motivation

→ Ende mit Copy und Paste

Ausgangslage: Klausuren wurden geschrieben

Aufgabe/Problem: Was ist die schlechteste Note? Was ist die höchste Punktzahl?

# Funktionen: Motivation - Ausgangslage

Altes Programm für größte Zahl:

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i=10;
    scanf("%i",&groessteZahl);
    while ( i > 0 ) {
        scanf("%i",&neueZahl);
        if(neueZahl > groessteZahl){
            groessteZahl=neueZahl;
        }
        i=i-1;
    }
    printf("Die groesste Zahl ist %i",groessteZahl);
    return 0;
}
```





# Funktionen: Motivation - Holzhammer

Aufgabe: Schreibe ein Programm, das die schlechteste Note von 11 Noten findet und die höchste Punktzahl von 11 Klausuren.

Lösung "Holzhammer":

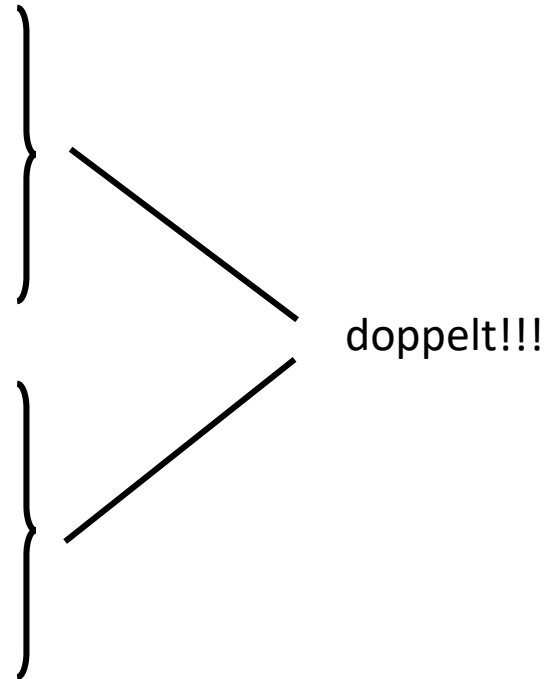
```
#include <stdio.h>
int main() {
    //Deklaration
    double note,punkte,schlechtesteNote,hoechstePunktzahl;
    int i=10;
    scanf("%lf",&schlechtesteNote);
    while ( i > 0 ) {
        scanf("%lf",&note);
        if(note > schlechtesteNote){
            schlechtesteNote=note;
        }
        i=i-1;
    }
    printf("Die schlechteste Note ist %lf", schlechtesteNote);
    i=10;
    scanf("%lf",&hoechstePunktzahl);
    while ( i > 0 ) {
        scanf("%lf",&punkte);
        if(punkte > hoechstePunktzahl){
            hoechstePunktzahl=punkte;
        }
        i=i-1;
    }
    printf("Die hoechste Punktzahl ist %lf",hoechstePunktzahl);
    return 0;
}
```

→ Ist das Programm schön? Könnte man es besser schreiben?



# Funktionen: Motivation - geht das besser?

```
#include <stdio.h>
int main() {
    //Deklaration
    double note,punkte,schlechtesteNote,hoechstePunktzahl;
    int i=10;
    scanf("%lf",&schlechtesteNote);
    while ( i > 0 ) {
        scanf("%lf",&note);
        if(note > schlechtesteNote){
            schlechtesteNote=note;
        }
        i=i-1;
    }
    printf("Die schlechteste Note ist %lf", schlechtesteNote);
    i=10;
    scanf("%lf",&hoechstePunktzahl);
    while ( i > 0 ) {
        scanf("%lf",&punkte);
        if(punkte > hoechstePunktzahl){
            hoechstePunktzahl=punkte;
        }
        i=i-1;
    }
    printf("Die hoechste Punktzahl ist %lf",hoechstePunktzahl);
    return 0;
}
```



# Funktionen: Motivation - kein doppelter Code

```
#include <stdio.h>
```

```
double findegroessteZahl(int anzahl){  
    double zahl,gz;  
    scanf("%lf",&gz);  
    while ( anzahl > 0 ) {  
        scanf("%lf",&zahl);  
        if(zahl > gz){  
            gz=zahl;  
        }  
        anzahl=anzahl-1;  
    }  
    return gz;  
}
```

Funktion

Eingabe

Rückgabe/Funktionswert

```
int main() {  
    double schlechtesteNote, hoechstePunkte;  
    int i=10;  
    printf("Bitte geben Sie nacheinander die Klausurnoten ein:\n");  
    schlechtesteNote = findegroessteZahl(i);  
    printf("Die schlechteste Note ist %lf.\n", schlechtesteNote);  
    printf("Bitte geben Sie nacheinander die Punkte ein:\n");  
    hoechstePunkte = findegroessteZahl(i);  
    printf("Die hoechste Punktzahl ist %lf.\n", hoechstePunkte);  
    return 0;  
}
```

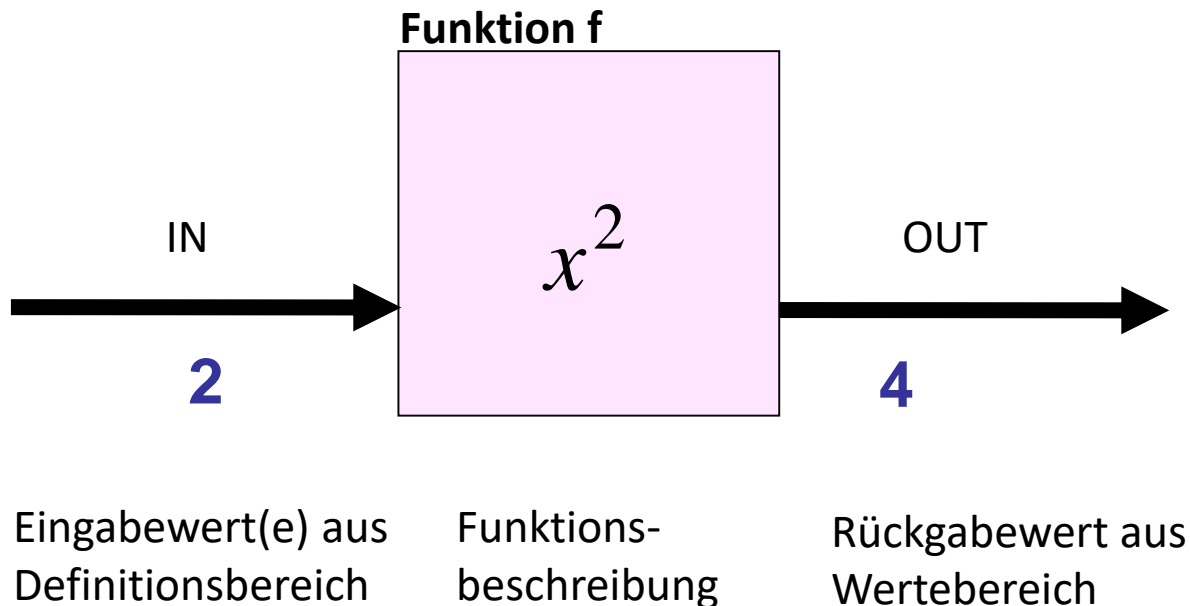


# Funktionen: nix Neues, bekannt aus der Mathematik

Funktionen sind eindeutige Abbildungen. Sie sind in C genauso definiert wie in der Mathematik.

$$\text{OUT} = f(\text{IN})$$

Beispiel:  $4 = f(2)$



# Funktionen in C: Motivationsbeispiel

```
#include <stdio.h>

double findegroessteZahl(int anzahl){
    double zahl,gz;
    scanf("%lf",&gz);
    while ( anzahl > 0 ) {
        scanf("%lf",&zahl);
        if(zahl > gz){
            gz=zahl;
        }
        anzahl=anzahl-1;
    }
    return gz;
}

int main() {
    double schlechtesteNote, hoechstePunkte;
    int i=10;
    printf("Bitte geben Sie nacheinander die Klausurnoten ein:\n");
    schlechtesteNote = findegroessteZahl(i);
    printf("Die schlechteste Note ist %lf.\n", schlechtesteNote);
    printf("Bitte geben Sie nacheinander die Punkte ein:\n");
    hoechstePunkte = findegroessteZahl(i);
    printf("Die hoechste Punktzahl ist %lf.\n", hoechstePunkte);
    return 0;
}
```

Funktion

Definitionsbereich

Funktionsinhalt

Wertebereich



# Funktionen in C: Motivationsbeispiel

```
#include <stdio.h>

double findegroessteZahl(int anzahl){
    double zahl,gz;
    scanf("%lf",&gz);
    while ( anzahl > 0 ) {
        scanf("%lf",&zahl);
        if(zahl > gz){
            gz=zahl;
        }
        anzahl=anzahl-1;
    }
    return gz;
}

int main() {
    double schlechtesteNote, hoechstePunkte;
    int i=10;
    printf("Bitte geben Sie nacheinander die Klausurnoten ein:\n");
    schlechtesteNote = findegroessteZahl(i);
    printf("Die schlechteste Note ist %lf.\n", schlechtesteNote);
    printf("Bitte geben Sie nacheinander die Punkte ein:\n");
    hoechstePunkte = findegroessteZahl(i);
    printf("Die hoechste Punktzahl ist %lf.\n", hoechstePunkte);
    return 0;
}
```

Funktion


Funktionsname

Funktionsinhalt



# Funktionen – Wozu?

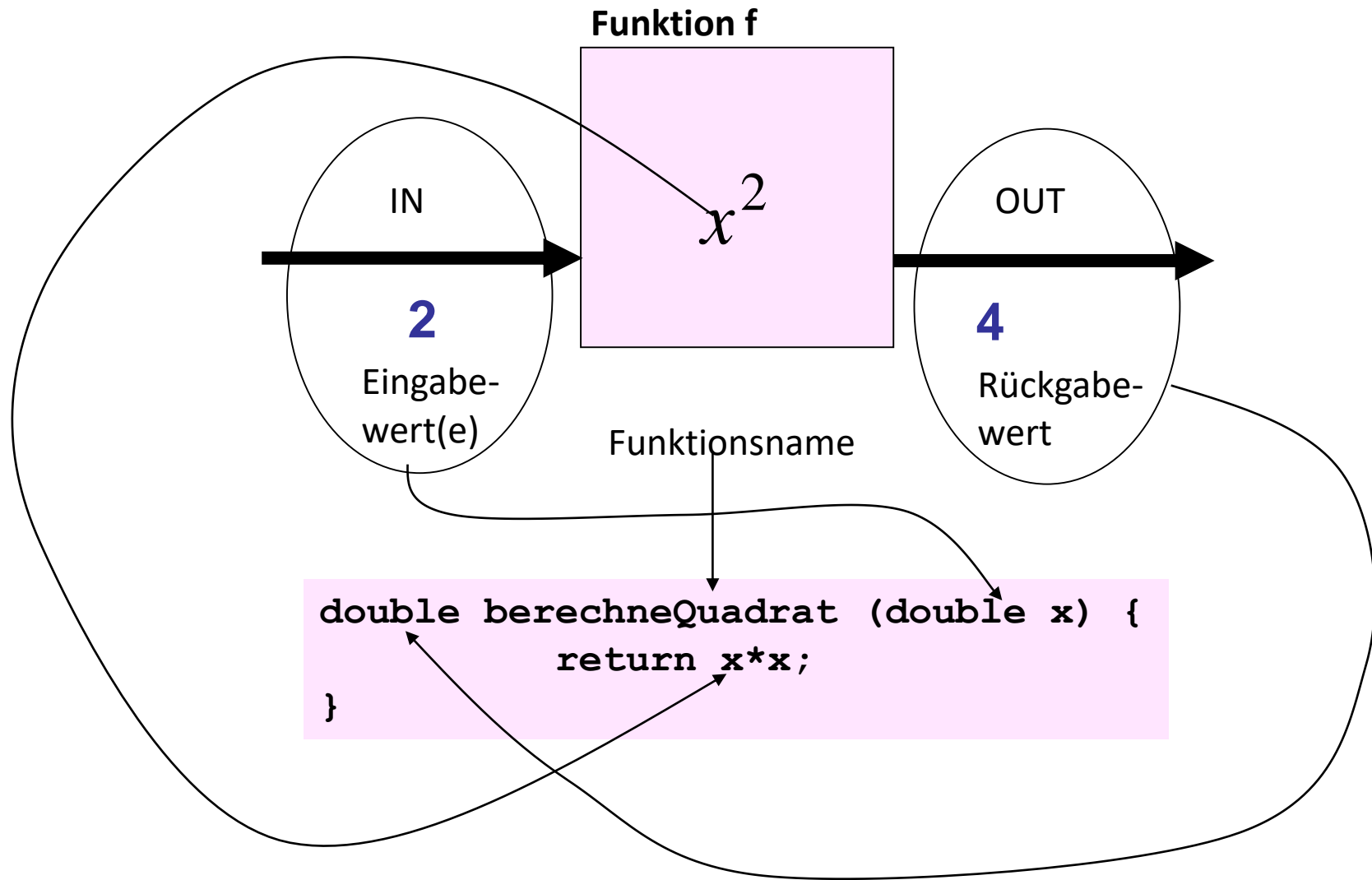
Funktionen sind nützlich, wenn man

- Programmcode mehrfach nutzen will, ohne ihn mehrmals abzutippen, also zur Vermeidung von doppeltem Code
- größere Aufgaben der Übersichtlichkeit in kleinere Teilaufgaben zerlegen möchte (der Name der Teilaufgabe kann dann z.B. der Funktionsname sein) 

Bemerkung: es sind mehrere Eingabewerte möglich.



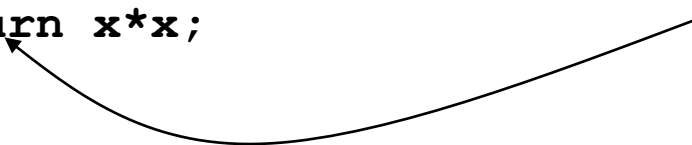
# Funktion: $OUT=f(IN)$ – erstes einfaches Beispiel





# Funktion: Aufruf

```
//Funktionsdefinition
//Ausgabe: double, Name: berechnequadrat, Eingabe: double x
//Test: 2 -> 4
#include <stdio.h>
double berechneQuadrat (double x) {
    //Funktionsinhalt
    return x*x;
}
int main(){
    double eingabe,ausgabe;
    printf("Bitte geben Sie eine Zahl ein:");
    scanf("%lf",&eingabe);
    //Aufruf der Funktion quadratfunktion
    ausgabe = berechneQuadrat(eingabe); //wie in Mathe: out=f(in)
    printf("Das Quadrat von %lf ist %lf",eingabe,ausgabe);
    return 0;
}
```



Verlassen der Funktion,  
egal, wo das return steht!

# Funktion: Aufruf

```
//Funktionsdefinition
//Ausgabe: double, Name: berechnequadrat, Eingabe: double x
//Test: 2 -> 4
#include <stdio.h>
double berechneQuadrat (double x) {
    //Funktionsinhalt
    return x*x;
}
int main() {
    double eingabe,ausgabe;
    printf("Bitte geben Sie eine Zahl ein:");
    scanf("%lf",&eingabe);
    //Aufruf der Funktion quadratfunktion
    ausgabe = berechneQuadrat(eingabe); //wie in Mathe: out=f(in)
    printf("Das Quadrat von %lf ist %lf",eingabe,ausgabe);
    return 0;
}
```

Der Wert oder das Ergebnis der Rechnung wird dann in den Speicher geschrieben, der rechts vom = beim Aufruf der Funktion benannt wird.

# Funktion: Signatur (formale Schnittstelle)

```
//Funktionsdefinition
//Ausgabe: double, Name: berechnequadrat, Eingabe: double x
//Test: 2 -> 4
#include <stdio.h>
double berechneQuadrat (double x) {
    //Funktionsinhalt
    return x*x;
}
int main(){
    double eingabe,ausgabe;
    printf("Bitte geben Sie eine Zahl ein:");
    scanf("%lf",&eingabe);
    //Aufruf der Funktion quadratfunktion
    ausgabe = berechneQuadrat(eingabe); //wie in Mathe: out=f(in)
    printf("Das Quadrat von %lf ist %lf",eingabe,ausgabe);
    return 0;
}
```



# Funktion: Definition

Eine Funktion muss deklariert werden

→ **bevor** man sie benutzt.



→ entweder als komplette Funktion (Definition) oder als Prototyp, der die Signatur einer Funktion beschreibt (Deklaration).

Warum reicht das?

Compiler (Übersetzungsprogramm) muss die Datentypen überprüfen können, der Rest ist ihm egal.

# Funktionsprototyp

```
#include <stdio.h>
//Test: 3->9
double berechneQuadrat (double);

int main() {
    double eingabe,ausgabe;
    printf("Bitte geben Sie eine Zahl ein:");
    scanf("%lf",&eingabe);
    ausgabe= berechneQuadrat (eingabe);
    printf("Das Quadrat von %lf ist %lf",eingabe,ausgabe);
    return 0;
}
double berechneQuadrat (double x) {
    return x*x;
}
```

Kann so auch nett in eine Extradatei gepackt werden (Bibliothek)

# Funktion - Abarbeitung

```
#include <stdio.h>
//Test: 3->9
double berechneQuadrat (double x) {
    double quadrat;
    quadrat=x*x;
    return quadrat;
}

int main() {
    double eingabe,ausgabe;
    printf("Eingabe:");
    scanf("%lf",&eingabe);

    ausgabe=berechneQuadrat(eingabe);

    printf("Ergebnis: %lf",ausgabe);
    return 0;
}
```

1.Start des Programms

2.Funktionsaufruf

3.Funktionsrückgabe  
=Verlassen der  
Funktion



# Funktion: Lokale Parameter

```
#include <stdio.h>
double findegroesstezahl(int anzahl){
    double zahl,gz;
    scanf("%lf",&zahl);
    gz=zahl;
    while ( anzahl > 0 ) {
        scanf("%lf",&zahl);
        if(zahl > gz){
            gz=zahl;
        }
        anzahl=anzahl-1;
    }
    return gz;
}
```

Das sind **lokale** Variablen, d.h. man kennt sie *nur* innerhalb *dieser* Funktion.

Und Tschüss! (zurückgehen, wo man herkam)

```
int main() {
    double schlechtesteNote, hoechstePunkte;
    int i=10;
    printf("Bitte geben Sie nacheinander die Klausurnoten ein:\n");
    schlechtesteNote = findegroesstezahl(i);
    printf("Die schlechteste Note ist %lf.\n", schlechtesteNote);
    printf("Bitte geben Sie nacheinander die Punkte ein:\n");
    hoechstePunkte = findegroesstezahl(i);
    printf("Die hoechste Punktzahl ist %lf.\n", hoechstePunkte);
    return 0;
}
```



# Funktion: mehrere Eingabegrößen

```
#include <stdio.h>
//Test: 2,3->5
double berechneSumme (double x, double y) {
    return x+y;
}
int main(){
    double eingabe1, eingabe2, ergebnis;
    printf("Summand1:");
    scanf("%lf",&eingabe1);
    printf("Summand2:");
    scanf("%lf",&eingabe2);
    ergebnis=berechneSumme(eingabe1,eingabe2);
    printf("Die Summe ist:%lf",ergebnis);
    return 0;
}
```



# Funktionsname...

- sollte sich lesen wie ein deutscher oder englischer Satz
- am besten beginnend mit einem Verb, sind dadurch gut unterscheidbar von Variablen, weil Funktionen was "tun"

Beispiel: `...berechnetSumme (...)`  
`...calculatesSum (...)`

# Funktionen – Beispiele

Funktionen sind allgemein wie folgt in C definiert:

**<Rückgabewert> <Funktionsname> (<Parameterliste>) { ... }**

Drei Beispiele:

Bedeutet in Englisch leer  
und hier keine Rückgabe  
bei der Funktion

```
//ohne Rueckgabe  
void ausgeben(int a) {  
    printf("*****\n");  
    printf("Zahl: %i\n", a);  
    printf("*****\n");  
}
```

```
//zwei Eingabegroessen  
float berechneProdukt(float a, float b) {  
    return a*b;  
}
```

```
//ohne Eingabegroessen  
int gib_Antwort_auf_alle_fragen() {  
    return 42;  
}
```



# Funktionen – Beispiele aus der Bibliothek

Drei Beispiele:

```
//Wurzel, definiert in math.h von C  
double sqrt(double)
```

```
//Eingabe, definiert in stdio.h in C  
//liest Zeichen in formatierter Form von der  
//Standardeingabe und speichert  
//die Werte nacheinander entsprechend des  
//Formatierungssymbols auf angegebene Adressen  
int scanf(..., ...)
```

```
//darf nur einmal existieren  
int main(){  
    return 0;  
}
```



# Funktionen – noch ein schönes Beispiel für leere Eingabe und keine Rückgabe

```
void entferneEnter() {while (getchar() != '\n' && getchar() != EOF);}
```

# Funktionen: Übung

Implementieren Sie eine Funktion, die den Absolutbetrag einer Zahl berechnet!

Eine mögliche Lösung: <https://youtu.be/l2m4iPrihgk>



# Funktionen: Übung

Implementieren Sie eine Funktion, die Celsius in Fahrenheit umwandelt und umgekehrt!

Tipp:  $\text{Celsius} = 5/9 * (\text{Fahrenheit} - 32)$ ,  $\text{Fahrenheit} = 32 + 1.8 * \text{Celsius}$



# Eventuell in einer Hörsaalübung: Spiel

Entwicklung eines Flussdiagramms in Abhängigkeit von Wünschen der Studierenden → Abstimmung

# Funktion: Übung (schwer)

Familie Guttenberg ist zu einer Party eingeladen. Leider können sich die Familienmitglieder (Anton, Berta, Claus, Doris) nicht einigen, wer hinget und wer nicht. In einer gemeinsamen Diskussion kann man sich jedoch auf folgende Grundsätze einigen:

- Mindestens einer muss gehen.
- Anton geht auf keinen Fall zusammen mit Doris.
- Wenn Berta geht, dann geht auch Claus mit.
- Wenn Anton und Claus gehen, dann bleibt Berta zu Hause.
- Wenn Anton zu Hause bleibt, dann geht entweder Doris oder Claus.

Implementiere eine Funktion, die nur 1 zurückgibt, wenn alle Bedingungen stimmen, ansonsten 0

Eine mögliche Lösung mit enthaltenem Logikfehler: <https://youtu.be/2c6hwj2AVpw>





# Funktionen – Arbeit mit Kopien (Call by Value)

```
void funktion(int a){  
    printf("a=%i\n",a);  
    a=7;  
    printf("a=%i\n",a);  
}  
int main() {  
    int a=42;  
    printf("a=%i\n",a);  
    funktion(a);  
    printf("a=%i\n",a);  
    return 0;  
}
```

Ausgabe:

42  
42  
7  
42

Warum?



# Funktionen – Arbeit mit Kopien und Rückgabe

```
int funktion(int a){  
    printf("funktion: a=%i\n",a) ;  
    a=7;  
    printf("funktion: a=%i\n",a) ;  
    return a;  
}  
int main() {  
    int a=42;  
    printf("a=%i\n",a) ;  
    a = funktion(a) ;  
    printf("a=%i\n",a) ;  
    return 0;  
}
```

Ausgabe:

42  
42  
7  
7



<https://youtu.be/moeju5Qoou4>



# Funktionen

```
int funktion(int fa) {  
    printf("funktion: fa=%i\n", fa);  
    fa=7;  
    printf("funktion: fa=%i\n", fa);  
    return fa;  
}  
int main() {  
    int a=42;  
    printf("a=%i\n", a);  
    a = funktion(a);  
    printf("a=%i\n", a);  
    return 0;  
}
```

verschiedene Namen benutzen ist weniger verwirrend. So macht man sich noch mal klar, dass das **a** unten ein konkreter Wert ist, der im Speicher **a** steht. Dieser Wert wird beim Funktionsaufruf auf die Variable **fa** kopiert.

# Funktionen

## → Rekursive Funktionen

### Compiler

### Compiler- und Laufzeitfehler



# Rekursive Funktionen: Motivation - Beispiele



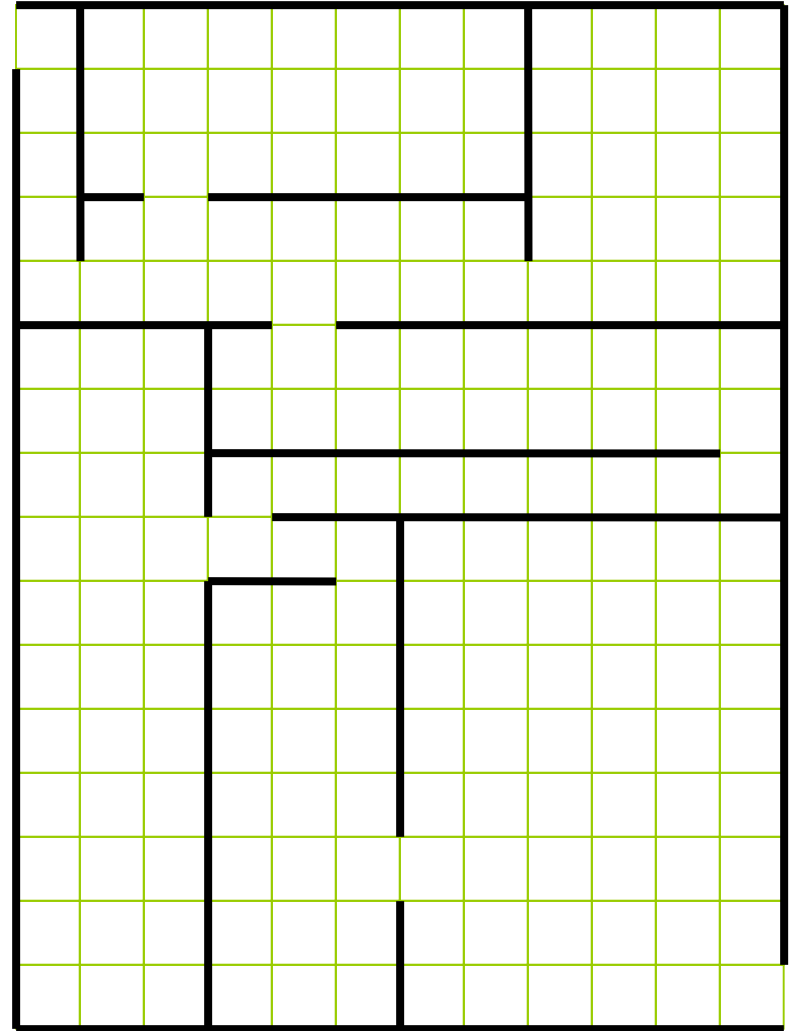
- Matroschka
- **GNU** ist **Not** **Unix**
- Minesweeper: <http://www.netzwelt.de/forum/arcade/game/687-minesweeper-online.html>
- Phytagorasbaum
- Schneeflocke
- Implementierung Minesweeper? Wie würden Sie das programmieren?



<https://youtu.be/Vy-MvJYBINU>

# Rekursion: Herstellen eines Labyrinths

1. Starte mit den Außenwänden, lass einen Eingang und einen Ausgang
2. Teile das Gebiet in zwei Teile, in dem man eine Lücke lässt
3. Wiederhole diesen Vorgang rekursiv für jedes neu entstandene Gebiet solange, bis nichts mehr zu Teilen ist (man muss immer auf den grünen Linien bleiben)



# Rekursive Funktionen: Beispiel 1

Start

Beschluss: ich mache Geld mit Häschenzüchtung

Nach 1. Monat



→Kauf eines Häschens

Nach 2. Monat



→Kauf noch eines Häschens

Nach 3. Monat



→Zusammensetzen der beiden in einen Käfig

Nach 4. Monat



Nach 5. Monat



Nach 6. Monat



Nach 7. Monat



...

Nach x-ten Monat

Anzahl(vorheriger Monat)+Anzahl(vorvorheriger Monat)

Oder anders ausgedrückt: Anzahl((x-1)-ter Monat)+Anzahl(x-2)-ter Monat)



# Rekursive Funktionen: Beispiel 2+3

Beispiel 2: Summe von 1 bis  $n = 1 + 2 + \dots + n$

$$\sum_{i=1}^n i = n + \sum_{i=1}^{n-1} i$$

mit Summe von 1 ist 1.

Beispiel 3: Fakultät von 1 bis  $n = 1 \cdot 2 \cdot \dots \cdot n$

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



# Rekursive Funktionen: Definition

- Funktion ruft sich *selbst* innerhalb der Funktion auf.
- *Jede* Funktion kann sich selbst aufrufen (Ausnahme **main()** ) .
- Eine *Abbruchbedingung* wird benötigt, sonst erfolgt ein Speicherüberlauf.
- Jede Rekursion kann auch durch Schleifen realisiert werden.
- Rekursion sollte nur verwendet werden, wenn eine Realisierung durch Schleifen zu kompliziert ist. Rekursion beansprucht viel Speicher.
- Beispiel: Listen im nächsten Semester



# Rekursive Funktion: Beispiel 3 *ohne* Funktion und *ohne* Rekursion

```
//Test: 5!=120
#include <stdio.h>
int main(){
    //Deklaration
    int i;
    double ergebnis=1;
    //Deklaration und Initialisierung
    int n=5;
    //Schleife
    for(i=1;i<=n;i=i+1){
        ergebnis = ergebnis * i;
    }
    //Ausgabe einer Gleitkommazahl ohne Nachkommastellen
    printf("%i! = %.0lf",n,ergebnis);
    return 0;
}
```

# Rekursive Funktion: Beispiel 3 *mit* Funktion und *ohne* Rekursion

```
//Definition der Funktion fakultaet
double berechneFakultaet(int n){
    double ergebnis = 1;
    int i;
    for(i=1;i<=n;i=i+1){
        ergebnis = ergebnis * i;
    }
    return ergebnis;
}

int main(){
    double wert;
    int n=5;
    printf("Bitte geben Sie eine Zahl ein:");
    scanf("%i",&n);
    //Aufruf der Funktion
    wert=berechneFakultaet(n);
    printf("%i! = %.0lf",n,wert);
    return 0;
}
```



# Rekursive Funktion: Beispiel 3 *mit* Funktion und **mit** Rekursion

//Test: 5!=120

```
double berechneFakultaet (int n){  
    //Abbruchsbedingung  
    if (n == 0) {return 1;}  
    //rekursiver Aufruf  
    return n * berechneFakultaet(n-1);  
}
```

```
int main(){  
    double ergebnis;  
    int n=5;  
    ergebnis = berechneFakultaet(n);  
    printf("%i! = %.01f",n,ergebnis);  
    return 0;  
}
```

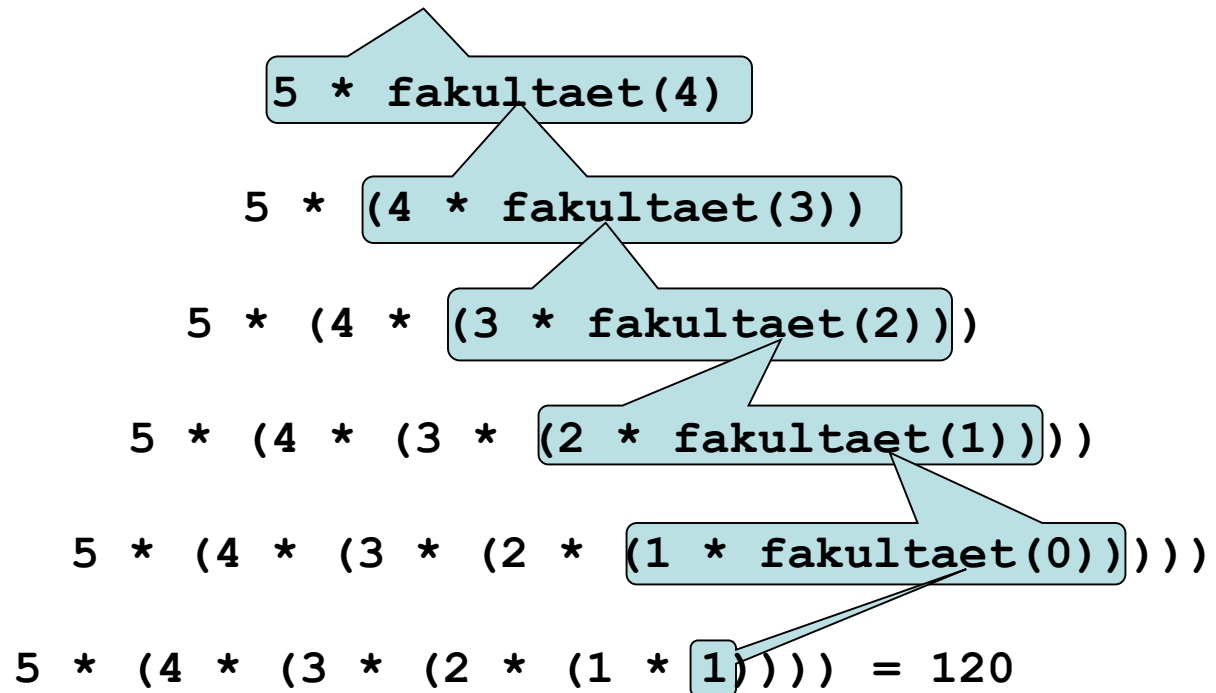
Bemerkung: Bei der Fakultätsberechnung besser **double** nehmen, weil sie eine sehr schnell wachsende Funktionen ist.



# Rekursive Funktion: Aufruf - Beispiel 3

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

Aufruf im Hauptprogramm: `fakultaet(5)`



# Rekursion: noch mehr Beispiele

Beispiele zum Ergoogeln:

- Turm von Hanoi
- Quicksort (kommt noch!)
- chaotische Folge:





$$Q(n) = Q(n - Q(n-1)) + Q(n - Q(n-2)) \text{ für } n > 2$$

$$Q(1) = Q(2) = 1$$

Anfang: 1 1 2 3 3 4 5 5 6 6 6 8 8 8 10 9 10 11 ...



# Iterativ oder rekursiv?

	iterativ	rekursiv
Vorteil	<ul style="list-style-type: none"><li>• keine hohen Systemanforderungen</li></ul> 	<ul style="list-style-type: none"><li>• einfach und kurz</li><li>• Lösung ist schnell entwickelt aus der mathematischen Formel</li></ul> 
Nachteil	<ul style="list-style-type: none"><li>• bei größeren Aufgaben kann sehr komplexer Algorithmus entstehen</li><li>• sehr langer und unübersichtlicher Code</li><li>• eventuell keine Idee, wie iterativ lösbar</li></ul> 	<ul style="list-style-type: none"><li>• sehr hohe Speicherauslastung, eventuell sogar Speichermangel → Rechner macht nichts mehr</li></ul> 

"Um Rekursion zu verstehen, muss man zunächst die Rekursion verstehen."

# Rekursive Funktion: Übung

Implementieren Sie die Berechnung der Fibonaccizahlen!





# Übung: Effektiver Sinus



# Übung: verbesserte Eingabefunktion



# Funktionen: Zusammenfassung

- Definition:  
`<Rückgabewertdatentyp><Funktionsname> (<Parameterliste>) {  
... }`
- Nutzen: Vermeidung von doppeltem Code
- Nutzen: Erhöhung der Übersichtlichkeit
- Können mehrere Eingabewerte haben, die Reihenfolge ist dabei signifikant
- Haben nur einen Rückgabewert (**int**, **double**, **char**, Zeiger (später), **structs** (später) oder keinen (**void**))
- Können sich selbst aufrufen (außer **main**)

✓ Funktionen  
✓ Rekursive Funktionen  
→ Compiler

- Wie entsteht ein Programm?
- Was macht ein Compiler?
- Schritte eines Compilers

Compiler- und Laufzeitfehler



# Wie entsteht ein Programm?

**Beispiel aus dem täglichen Leben:** ToDo-Liste für meinen Mitbewohner, wenn ich im Urlaub bin

**Liste schreiben:** Überschrift: *Aufträge während meiner Abwesenheit*

1. keine Leute in mein Zimmer lassen
2. solange es weiter regnet, keine Pflanzen auf dem Balkon gießen
3. wenn Brief von der Bang kommt, bitte öffnen und aufheben, wenn Scheck drin ist, ansonsten wegschmeißen
4. Java-Buch in der Bibliothek abgeben
5. den Abwasch nicht zu lange dreckig in der Spüle stehen lassen
6. unbedingt das Java-Buch lesen und mir beim Zurückkommen erklären

Frage: Was ist hier unlogisch oder fehlerhaft?

# Wie entsteht ein Programm?

**Beispiel aus dem täglichen Leben:** ToDo-Liste für meinen Mitbewohner, wenn ich im Urlaub bin

## **Korrekturlesen der Liste:**

- Hä? Buch soll erst abgegeben werden, aber dann doch noch gelesen werden (Reihenfolge!)
- Rechtschreibfehler Bang statt Bank

**Abarbeitung** der Liste durch meinen Mitbewohner

## **Feststellen von Mängeln:**

- alle Pflanzen vertrocknet und mein Mitbewohner behauptet frech, es hätte nur etwas von Nichtgießen bei Regen auf der Liste gestanden, aber nichts darüber, was bei Sonne geschehen soll. Es schien aber die Sonne die ganze Zeit. (also Logikfehler von mir!!!)
- es steht total viel dreckiges Geschirr rum

# Wie entsteht ein Programm?

**Beispiel aus dem täglichen Leben:** ToDo-Liste für meinen Mitbewohner, wenn ich im Urlaub bin

1. Liste schreiben
2. Liste überprüfen
3. Liste ausführen lassen
4. Fehler bei der Ausführung bemerken
5. Liste beim nächsten Mal verbessern

# Wie entsteht ein Programm?

- **Editieren:** Programm schreiben und abspeichern
- **Compilieren:** Übersetzen des Programms von Programmiersprache in Maschinsprache. Compiler sieht das Programm nach *syntaktischen Fehlern* durch und zeigt sie uns an. Die Berichtigung der Fehler müssen wir allerdings selbst vornehmen.
- **Ausführung:** Programm ausführen lassen vom Rechner
- **Laufzeitfehlersuche:** *Laufzeitfehler* entdecken und wieder beim ersten Schritt beginnen

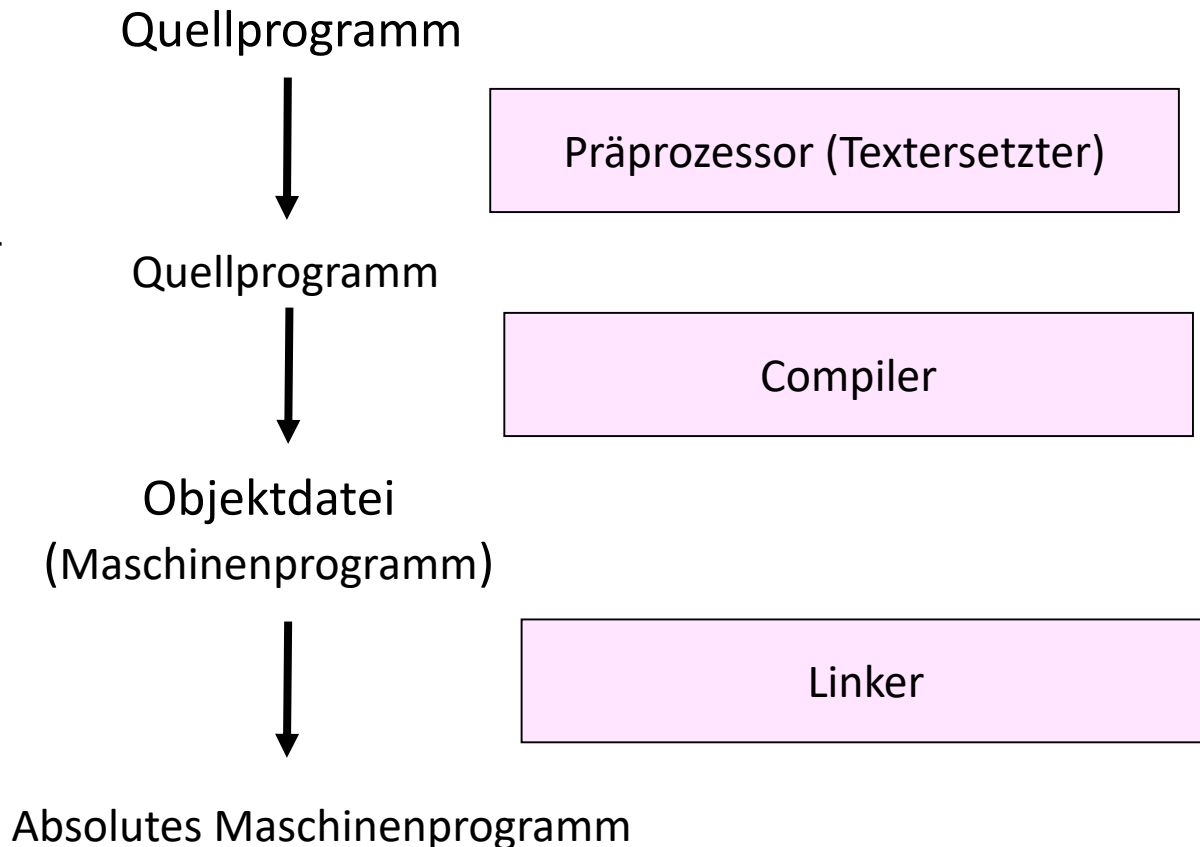
So ähnlich sieht das Berufsleben eines Softwareentwicklers aus.





# Was macht ein Compiler?

Ein Compiler übersetzt ein Programm in ein vom Rechner ausführbares Maschinenprogramm.



# Was sind die Schritte des Compilers?

Übersetzung des Programms in Maschinsprache (Compiler ist selbst ein Programm!)

- **Lexikalische Analyse:** Durchlauf Zeichen für Zeichen, alle Schlüsselwörter (**if**, **while**,...) werden neu bezeichnet durch Token
- **Syntaktische Analyse:** Überprüfen, ob die Token korrekt sind, Anlegen einer Symboltabelle mit den Namen aller Variablen, Anlegen eines Ableitungsbaumes (gibt an, wie die Schlüsselwörter abgeleitet und Operationen ausgeführt werden, siehe Wikipedia „Syntaxbaum“)
- **Semantische Analyse und Maschinencodeerzeugung:** Zuweisen von Speicherplätzen für die Eingabe, Zwischenergebnisse und Ausgabe, Adresse der Speicherplätze werden in der Symboltabelle an der entsprechenden Stelle eingetragen, erstellter Ableitungsbaum wird durchlaufen und alle Strukturen werden in eine Folge von Anweisungen in der Maschinsprache übersetzt, man erhält ein **Binärobjekt**

**Nachteil:** ausführbares Programm läuft in der Regel nur auf dem Rechner, wo das Programm übersetzt wurde



- ✓ Funktionen
  - ✓ Rekursive Funktionen
  - ✓ Compiler
- Compiler- und Laufzeitfehler



# Compilerfehler...

sind Syntaxfehler, die der Compiler findet.



# Laufzeitfehler ...

sind Logikfehler, die man durch Testen *hoffentlich* findet.

```
#include <stdio.h>
```

```
int main(){  
    int i = 10;  
    while (i > 1){  
        printf("Ich habe ganz schön zu tun!\n");  
    }  
    return 0;  
}
```



# Laufzeitfehler ...

1. Frau zum Mann: Kauf bitte ein Brot und wenn es Eier gibt, 6.
2. Der Mann kommt mit 6 Broten nach Hause.

```
brot=1;  
if(eier>0) {  
    brot=6;  
}
```

# Aber es läuft doch...



→ Ein besonders gefährlicher Satz in C

*Lebenstipp:* Wenn Ihre Kinder mal sagen, ich habe aufgeräumt und alle Hausaufgaben fertig → immer genau überprüfen....

# Wiederholung

Wozu benutzt man Funktionen?

→ um doppelten Code zu vermeiden (Wiederverwendbarkeit) und um größere Aufgaben wegen der Übersichtlichkeit in kleinere Teilaufgaben zu zerlegen (eventuell Name der Teilaufgabe = Funktionsname), daraus ergibt sich eine bessere Lesbarkeit

- Wie sehen Funktionen aus?

→ <Datentyp\_vom\_Rückgabewert ><Funktionsname>(<Liste der Eingabeparameter mit Datentyp>)

- Was macht ein Compiler?

→ Übersetzen des Programms von Programmiersprache in Maschinsprache, Suche nach syntaktischen Fehlern, ist selbst ein Programm



# Literatur

Post: „Besser coden“, Rheinwerk Verlag, 2021

