

# Informatik 1



## Elementare Datentypen, Kontrollstrukturen und Flussdiagramme

von  
**Irene Rothe**

Büro B 241

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

Instagram: irenerothedesign



# 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**

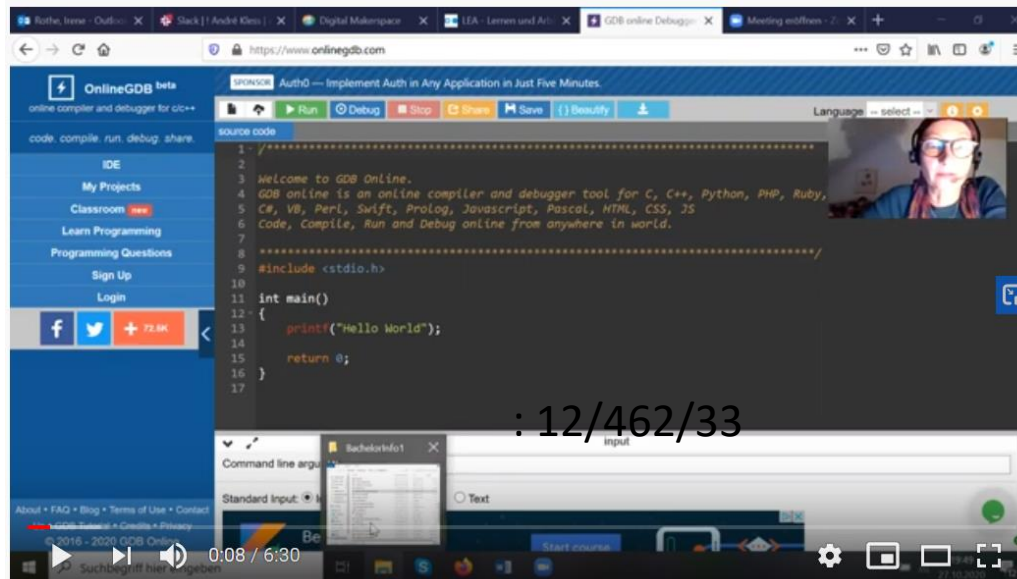
→ Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen



# Design der Folien

- **Orange** 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.
- **Grün** 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

# Einstieg: Programmierung einer Summe

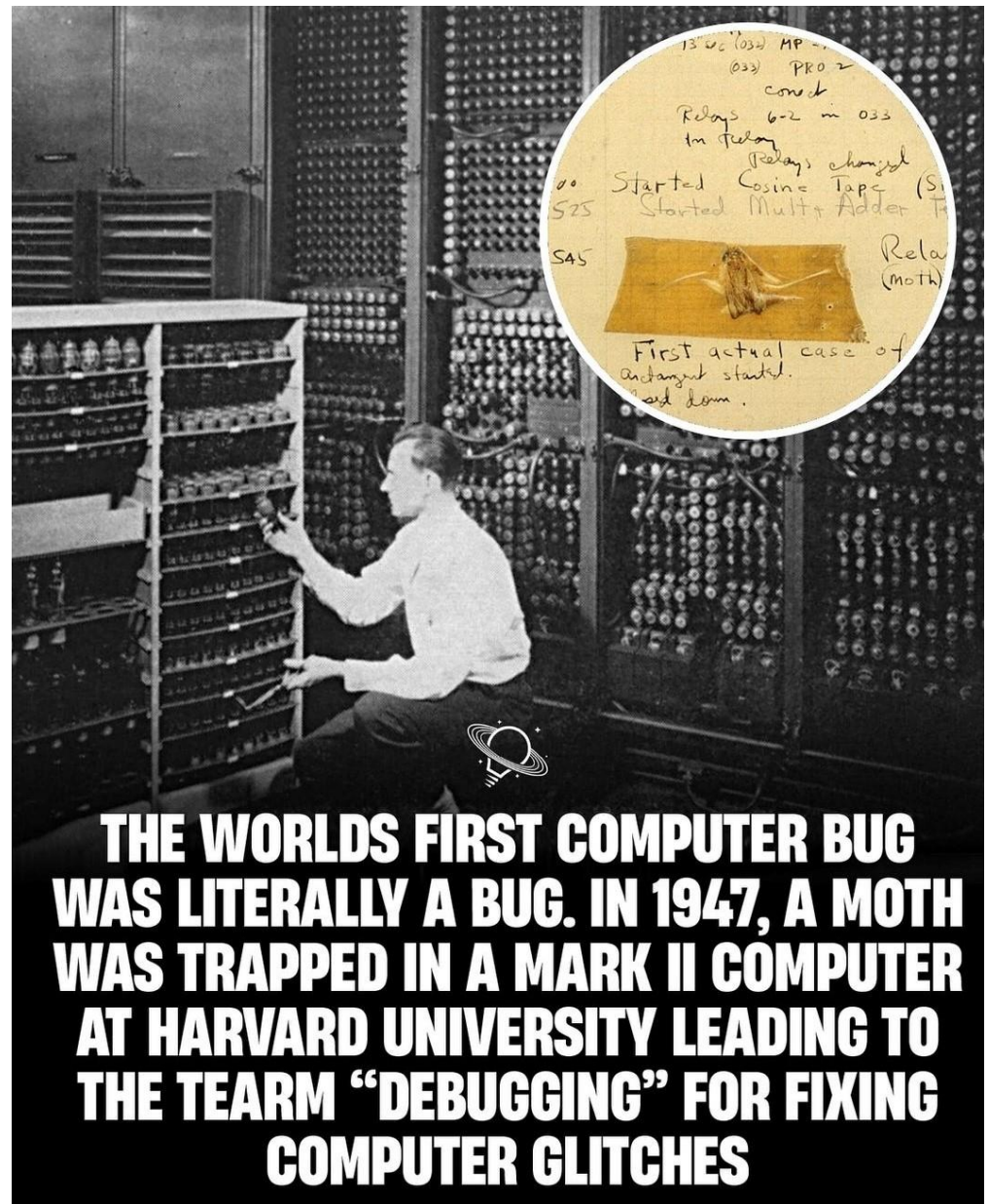


Inf1ProgrammierungSumme

<https://youtu.be/yO6hOTdTm6w>



# Was ist ein Bug?



# Wiederholung

Summe von zwei ganzen Zahlen:

```
//Irene
//Summe von zwei Zahlen
//Test:2,5->7
#include <stdio.h>
int main() {
    int summand1=42;      : 12/462/33
    int summand2=0;
    int summe;
    scanf ("%i", &summand1) ;
    scanf ("%i", &summand2) ;
    summe=summand1+summand2 ;
    printf ("%i+%i=%i", summand1, summand2, summe) ;
    return 0;
}
```

# Übung

Ehe Sie zur nächsten Folie wechseln, schreiben Sie auf, was jede Zeile bedeutet:

```
#include <stdio.h>
int main() {
    int summand1=42;
    int summand2=0;
    int summe;
    scanf ("%i", &summand1) ;
    scanf ("%i", &summand2) ;
    summe=summand1+summand2;
    printf ("%i+%i=%i", summand1, summand2, summe) ;
    return 0;
}
```

# Lösung der Übung

```
//hinter diesen Doppelstrichen stehen Kommentare
#include <stdio.h>          //Anfang des Programms (später genauer)
int main() {                //Start des Programms
    //Reservierung von Speicher für eine ganze Zahl
    //Speicherplatz heißt: summand1
    //Zuweisung der 42 auf den Speicherplatz summand1
    int summand1 = 42;
    //Deklaration von summand2 als ganze Zahl und Initialisierung mit 0
    int summand2 = 0;
    int summe;
    //Aufruf Eingabefunktion zur Speicherung einer Zahl auf summand1
    scanf("%i",&summand1);

...

    //summand1 und summand2 werden addiert und auf summe gespeichert
    summe=summand1+summand2;
    //Aufruf Ausgabefunktion für Text mit Einfügen der Inhalte
    //der Speicherplätze von summand1 und summand2 an die
    //Stellen der beiden Formatierer %i als ganze Zahlen
    printf("%i+%i=%i",summand1,summand2,summe);
    return 0;//Ende des Programms, später genauer
} //Ende des Programms
```





# In dieser Vorlesung:

## → Elementare Datentypen

- Variablen
- ganze Zahlen: int
- Kommazahlen: double
- Typumwandlung: Casting
- Zeichen: char
- Eingabe und Ausgabe
- Kommentare

Kontrollstrukturen  
Flussdiagramm



### 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.

# Variablen: Welche Werte kann ein Rechner abspeichern?

- **Zeichen** (alle auf der Tastatur)
- **Ganze Zahlen** (es gibt dabei eine größte und kleinste Zahl)
- **Kommazahlen** (es gibt dabei eine größte und kleinste Zahl und eine Genauigkeit)

Diese Werte werden im Speicher unter Adressen (also die Variablen wohnen in „Häusern“) abgespeichert.

Mit Adressen selbst will man nichts zu tun haben (101110111...), deshalb benutzt man selbst definierte **Variablennamen**, denen der Rechner dann Adressen zuordnet.

Wie lang diese Adressen sein können, hängt vom Rechner ab (32-bit, 64-bit Rechner). Umso mehr direkt erreichbare Adressen es gibt, umso schneller arbeitet der Rechner.

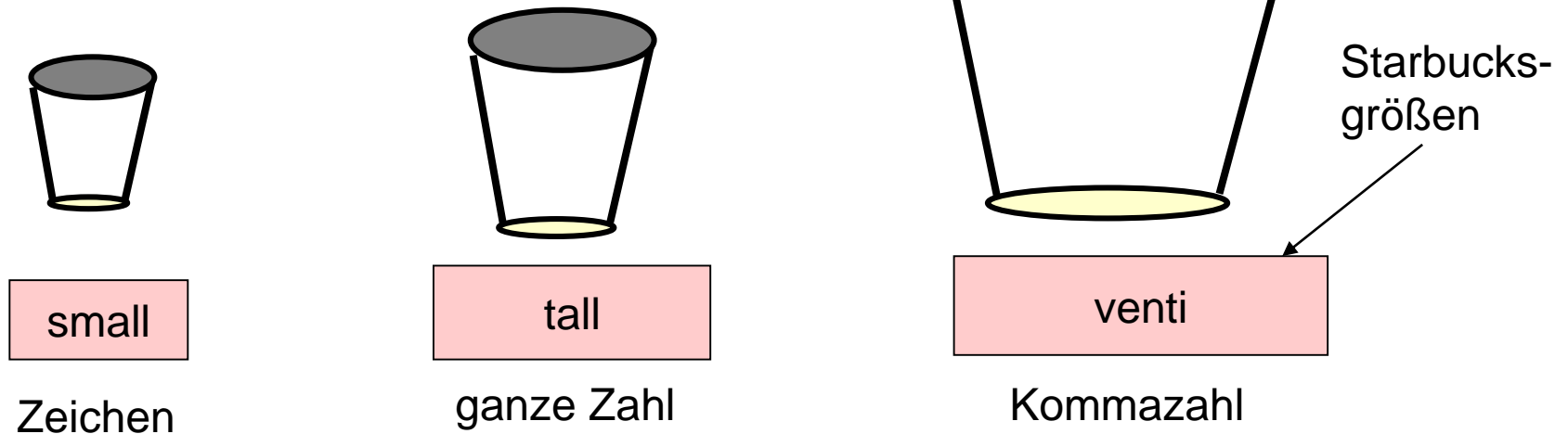
Die Speicherplatzgröße für Variablen ist abhängig von dem, was man darauf abspeichern will.



# Variablen: Motivation

Variablen = Behälter

Frage: Was passt in so einen Behälter?



# Variablen: Verwendung

Frage: Was kann man mit den Variablen jeweils tun, was ist vernünftig?

$1000010 + 1000011 = ?$  oder

$66 + 67 = ?$  oder

$B + C = ?$  oder

Kommazahl + Kommazahl = ?

Um zu wissen, was das Ergebnis bedeutet, muss man den **Typ** kennen.

Variablen (Bezeichner) werden benötigt, um Daten zu speichern.

Sie sollten am Anfang erklärt werden, damit der Compiler weiß, was sie bedeuten (wie z.B. bei einer Häkelanleitung: *LM*, *FM* muss vorher erklärt werden, sonst weiß man gar nicht, was man häkeln muss) und wie viel Platz reserviert werden soll.

Eine **Variable** hat einen **Datentyp** und einen **Namen**. Der **Datentyp** bestimmt, wie viel Speicherplatz reserviert werden muss.

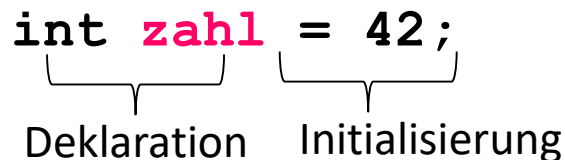


# Variablen: Deklaration und Initialisierung

**Deklaration** ist die Anforderung von Speicherplatz unter einem Namen. Das ist so ähnlich, wie das Mieten einer Wohnung einer bestimmten Größe.

**Initialisierung** ist die *erste* Wertzuweisung auf diesen Speicherplatz. Das ist so ähnlich, wie das erste Abstellen eines Gegenstandes in der gemieteten Wohnung, z.B. einer Klobürste.

```
int zahl = 42;
```



**Syntaxregeln** für die Bildung von Namen:

- nur Buchstaben (**ohne Umlaute**), Ziffern und Unterstriche (\_)
- das erste Zeichen muss ein Buchstabe oder Unterstrich sein
- In C wird Groß- und Kleinschreibung bei Namen unterschieden!
- ein Name kann beliebig lang sein (die ersten 32 Zeichen sind signifikant)
- darf kein **Schlüsselwort** wie z.B. return, main, int sein.

# Variablen: Variablennamen...

- sollten immer mit einem Kleinbuchstaben beginnen
- sollten sich lesen wie ein deutscher oder englischer Satz (oder definiert von grob nach fein)

Beispiel:

`int gibtanWievielBierimKuehlschrankSteht;`

oder

`int anzahlBierimKuehlschrank;`

oder

`int numberOfBeerinFridge;`

oder

`int number_Beer_Fridge;`

# Elementare Datentypen in C

In C gibt es drei wichtige Arten von **elementaren** Datentypen:

- Ganze Zahlen
- Kommazahlen (Gleitkommazahlen)
- Zeichen



# Elementare Datentypen in C

... sind Anforderungen von Speicher, die bestimmen, welche Operationen mit den Inhalten des Speichers durchgeführt werden können.

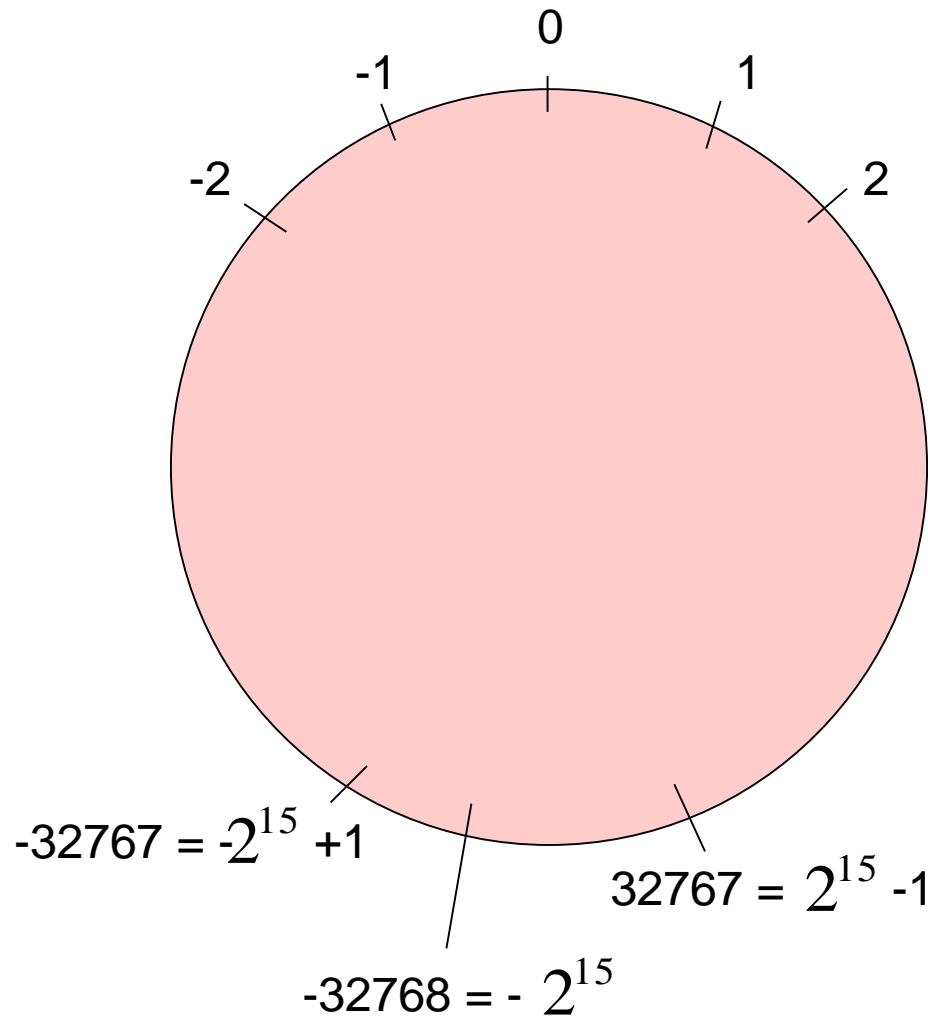
Die Auswahl eines Datentyps

- beeinflusst die Größe des reservierten Speichers,
- die Größe der darstellbaren Werte und
- was bei den zugelassenen Operationen jeweils passiert.

# Ganze Zahlen (Englisch: integer)

- Der Datentyp für ganze Zahlen heißt `int`. Er steht vor dem Namen der Variablen.
- Es gibt noch andere Datentypen, die hier nicht behandelt werden: `short`, `long`, ...
- Die Größe des `int`-Datentyps ist nicht einheitlich festgelegt, er hängt vom verwendeten Compiler ab (und vom Rechner)! In der Regel ist er heutzutage 32 Bit (kleinste Speichereinheit im Rechner) = 4 Bytes (immer 8 Bit werden auch Byte genannt) groß.
- **Die Mindestanforderung** (laut ISO) ist:  
Anzahl Bytes(`short`) <= Anzahl Bytes(`int`) <= Anzahl Bytes(`long`)
- Das bedeutet, dass mindestens ganze Zahlen im Bereich von -32767 bis 32767 ( $= 2^{15} - 1$ ) dargestellt werden können, in der Regel aber größere Zahlen.
- Ein Formatierer für `int` ist `%i`, zu verwenden z.B. in `printf` und `scanf`.

# Ganze Zahlen (int)



<https://youtu.be/F8WEwb-c-Kc>



# Ganze Zahlen (**int**)

```
#include<stdio.h>
int main(){
    int i=0;
    //Hochzaehlen bis Uebertrag erreicht
    while(i<i+1){
        i=i+1;
    }
    printf("Bereich (int) : %i ... %i\n",i+1,i);
    //int zweierpot;
    //for (i=1;i<=31;i++) {zweierpot=zweierpot*2;}
    //printf("2 hoch 31: %i\n",zweierpot);
    //printf("2 hoch 31 - 1: %i\n",zweierpot-1);
    return 0;
}
```

Bemerkung: Probieren Sie folgendes Programm aus!

<https://youtu.be/5MSZpDV65RE>



# Ganze Zahlen: Übung

Was wird hier ausgegeben?

```
int a = 13;  
int b = 2;  
int c;  
c = a/b;  
printf("%i",c) ;
```

Probieren Sie Ihre Vermutung in einem Compiler aus, um zu sehen, ob Sie richtig vermutet haben.

# Kommazahlen (double)

- Der Datentyp für Kommazahlen heißt **double**. Der Datentyp steht vor dem Namen der Variablen.
- Es gibt noch andere Datentypen, die hier nicht behandelt werden, z.B. `float`
- Die Mindestanforderung für Kommazahlen ist eine Genauigkeit von 6 Stellen für `float` und eine Genauigkeit von 15 Stellen für `double`.
- Der Formatierer z.B. in `printf` für `double` ist in C `%lf` (das heißt long float).
- Beispiel:

```
double pi = 3.141592653589793238462643383279;
```

**Achtung:** Punkt statt Komma!



kann der Rechner gar nicht verarbeiten

```
double grossezahl=123456789123456; //das sind auch 15  
//Stellen
```

# Kommazahlen: Übung

Was wird hier ausgegeben?

```
int a = 13;  
int b = 2;  
int c;  
c = a/b;  
printf("%i",c);
```

Was wird hier ausgegeben?

```
int a = 13;  
int b = 2;  
double c;  
c = a/b;  
printf("%lf",c);
```

Was wird hier ausgegeben?

```
double a = 13;  
int b = 2;  
double c;  
c = a/b;  
printf("%lf",c);
```

Formatierer für Kommazahl



Versuchen Sie zu verstehen, warum welche Ausgaben erfolgen.

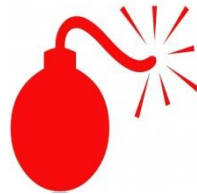


# Kommazahlen nicht auf Gleichheit prüfen!

double-Zahlen (Kommazahlen) können gleich auf dem Bildschirm aussehen, aber im Rechner trotzdem verschieden abgespeichert sein. Dazu später mehr. Bei int-Zahlen ist das nicht so.

Keine gute Idee:

```
double a;  
double b;  
...  
if (a == b) {...}
```



Besser so:

```
double a;  
double b;  
...  
if (abs(a - b) < epsilon) {...}
```



mit z.B.  $\epsilon = 0.0000005$  und abs als Funktion zur Berechnung des Absolutbetrags

<https://youtu.be/PfrygTjsEM8>



# Typumwandlung (Casting) - implizit

→ Wenn etwas nicht passt, wird es passend gemacht

Zieht man von einer kleinen Wohnung in eine große, passen alle Möbel bequem rein.

→ Implizite Typumwandlung: automatische Umwandlung eines "kleineren" Datentyps in einen "größeren":

```
int i = 42;  
double value = i;
```

Es gilt dabei: **short** → **int** → **long** → **float** → **double**

Zieht man dagegen von einer großen Wohnung in eine kleine, kann man Dinge nicht mitnehmen, die man eventuell später doch noch braucht. Diese Dinge sind dann weg.

→ Bei Typumwandlung in die **andere Richtung** droht Verlust von Information, daher wenn möglich **vermeiden!** Compiler gibt Warnung.

```
double value = 3.1415;  
int i = value; //hier überlebt nur die 3
```



<https://youtu.be/ABZAZ9bOX2Y>

# Typumwandlung (Casting) - explizit

- Explizite Typumwandlung: Gewollte Umwandlung eines Datentyps in einen anderen durch den Cast-Operator:

**(<typename>) Ausdruck;**

Beispiel: `#include<stdio.h>`

```
int main(){  
    int a = 1, b = 4;  
    double x;
```

```
    x = (double) a /b;
```

```
    printf("Ergebnis : %lf \n",x);
```

```
    return 0;
```

```
}
```

**Achtung:** Casting hat höhere  
Priorität als arithmetische  
Operation



Tipp: Folgendes ist besser lesbar: `x = ((double) a) /b;`

# Zeichen (char, Englisch: character)

- Mit dem Datentyp **char** kann Speicherplatz für ein einzelnes Zeichen (so ziemlich alle Symbole der Tastatur) reserviert werden.

- Beispiel:

```
char zeichen = 'c';
```

Hier ein  
Hochkomma



- Der Formatierer z.B. in `printf` für `char` ist in C `%c`.

Zeichen	Zahl (ASCII-Code)
'a'	97
'z'	122
'A'	65
'Z'	90
'0'	48
'9'	57

**Achtung:** In C ist ein **Zeichen** auch eine **Zahl**.  
Hier wird beides mal ein a gespeichert:

```
char zeichen1 = 'a';
```

```
char zeichen2 = 97;
```

**Achtung:** Es gibt keinen Datentyp für Zeichenketten (Strings)!



# Zeichen: Abbildung von Buchstaben auf Zahlen

→ damit arbeiten Computer: Steht in einem Speicher **zeichen** z.B. 88 und man gibt den Inhalt des Speichers mit **printf("%c", zeichen)** ; aus, erscheint ein X auf dem Bildschirm.

asctab2

ASCII-Tabelle (c) S. Steiner, www.freebasic-portal.de

0	26 →	52 4	78 N	104 h	130 é	156 f	182	208 ll	234 ñ
1 ☐	27 ←	53 5	79 O	105 i	131 â	157 ¥	183	209 T	235 ð
2 ☐	28 ↵	54 6	80 P	106 j	132 ä	158 R	184	210	236 ø
3 ♥	29 *	55 7	81 Q	107 k	133 à	159 f	185	211	237 ø
4 ♦	30 ▲	56 8	82 R	108 l	134 å	160 á	186	212	238 €
5 ♠	31 ▼	57 9	83 S	109 m	135 ç	161 í	187	213	239 n
6 ♣	32	58 :	84 T	110 n	136 ê	162 ó	188	214	240 ≡
7	33 !	59 ;	85 U	111 o	137 ë	163 ú	189	215	241 ±
8	34 "	60 <	86 V	112 p	138 è	164 ñ	190	216	242 ≥
9	35 #	61 =	87 W	113 q	139 ì	165 ñ	191	217	243 ≤
10	36 \$	62 >	88 X	114 r	140 î	166 æ	192	218	244 ∫
11 ♂	37 %	63 ?	89 Y	115 s	141 ï	167 é	193	219	245 ∫
12 ♀	38 &	64 @	90 Z	116 t	142 ð	168 ÷	194	220	246 ÷
13	39 '	65 A	91 [	117 u	143 Å	169 r	195	221	247 ≈
14 ♪	40 (	66 B	92 \	118 v	144 É	170 ∩	196	222	248 °
15 *	41 )	67 C	93 ]	119 w	145 æ	171 ½	197	223	249 ·
16 ▶	42 *	68 D	94 ^	120 x	146 ff	172 ¼	198	224	250 ·
17 ◀	43 +	69 E	95 _	121 y	147 ô	173 ÷	199	225	251 ∫
18 ‡	44 ,	70 F	96 `	122 z	148 ö	174 «	200	226	252 n
19 !!	45 -	71 G	97 a	123 {	149 ò	175 »	201	227	253 z
20 ¶	46 .	72 H	98 b	124	150 û	176	202	228	254 ■
21 §	47 /	73 I	99 c	125 }	151 ù	177	203	229	255
22 ■	48 0	74 J	100 d	126 ~	152 ü	178	204	230	
23 ±	49 1	75 K	101 e	127 Δ	153 ū	179	205	231	
24 ↑	50 2	76 L	102 f	128 Ç	154 ū	180	206	232	
25 ↓	51 3	77 M	103 g	129 ü	155 ç	181	207	233	

In Europa sehr üblich: 8-bit Code ISO Latin-1, dazu später mehr.



# Zeichen: Sonderzeichen

Zeichen	Bedeutung	ASCII-Code
'\n'	Zeilenvorschub	10
'\t'	Tabulator	9
'\\'	Backslash	92
'\"'	'	39
'\"'	"	34

# Elementare Datentypen: Formatierer für Ein- und Ausgabe

Typ	Bezeichner	Bits	Bereich	Formatierer
Ganzzahl	int	mindestens 16	-32768 ... 32767	i, d
Kommazahl (Gleitkommazahl (kurz))	float	32	10E-37 ... 10E37 6 Stellen genau	f, (g)
Kommazahl (Gleitkommazahl (lang))	double	64	10E-307 ... 10E307 15 Stellen genau	lf, (g)
Zeichen	char	8 mindestens*	-128 ... 127 oder 0 ... 255 (Systemabhängig, steht in limits.h)	c

**Achtung** beim Formatierer g im Zusammenhang mit scanf! Dort funktioniert er nicht.

\*absurdes Nerdwissen. Was ein Byte ist, ist kein Standard. Char 8 Bit ist eigentlich immer so, nur ganz ganz selten nicht so



# Variablen: Eingabe und Ausgabe

```
#include <stdio.h>
int main() {
    int    ganzzahl;
    double kommazahl;
    printf("Eingabe ganze Zahl:");
    scanf("%i", &ganzzahl);
    printf("Eingabe Kommazahl");
    scanf("%lf", &kommazahl);

    printf("Int    : %i \n",ganzzahl);
    printf("Double : %lf \n",kommazahl);

    return 0;
}
```

# Bemerkung: Eingabeproblem



Wenn man ein Zeichen eingegeben will und vorher eine Zahl eingegeben hat, bleibt oftmals ein Enter im Eingabepuffer (ein Enter ist ein Zeichen und keine Zahl), dass dann als Zeichen bei der Zeicheneingabe später genutzt wird und man gar nicht die Gelegenheit bekommt, etwas selbst über den Bildschirm einzugeben.

```
#include <stdio.h>
int main() {
    int    number1;
    char    zeichen;
    printf("Gib eine Zahl ein:");
    scanf("%i", &number1);
    printf("Gib ein Zeichen ein:");
    scanf("%c", &zeichen);

    printf("Int    : %i \n", number1);
    printf("Char   : %c \n", zeichen);
    return 0;
}
```

scanf hinterlässt  
vielfach ein  
**Enter** im  
Tastaturpuffer

Dieses scanf  
nimmt dann das  
Enter aus der  
letzten Eingabe  
und verarbeitet  
es.



# Bemerkung: Vorschlag für eine Lösung, die hoffentlich hilft

```
#include <stdio.h>
int main() {
    int    number1;
    char   zeichen,c;
    printf("Gib eine Zahl ein");
    scanf("%i", &number1);
    // "Weglesen" des Enters aus scanf und anderer
    // unerwünschter Eingaben
    while((c=getchar())!='\n' && c != EOF);
    printf("Gib ein Zeichen ein");
    scanf("%c", &zeichen);
    printf("Int    : %i \n", number1);
    printf("Char  : %c \n", zeichen);
    return 0;
}
```

# Bemerkung: Vorschlag für eine Lösung, die helfen kann

```
#include <stdio.h>

int main() {
    int    number1;
    char   zeichen;
    scanf("%i", &number1);
    fflush(stdin);
    scanf("%c", &zeichen);
    printf("Int   : %i \n", number1);
    printf("Char  : %c \n", zeichen);
    return 0;
}
```

# Kommentare

- Persönliche Notizen können in einem Programm wie folgt implementiert werden:

```
//Ich bin ein Kommentar auf einer Zeile
/*Ich bin auch ein Kommentar,
   kann aber über viele Zeilen
   gehen
*/
```

- Der Compiler überliest Kommentare.
- Kommentare sind extrem wichtig in Programmen. Sie machen es möglich, dass man nach einer Pause sein eigenes Programm wieder versteht oder sogar ein fremdes Programm.

Wann brauche ich einen Kommentar?

→ wenn ein gleich guter Programmierer Ihren Code nicht auf Anhieb versteht!

# Programmmanfang

Beginnen sollte man Programme mit seinem Namen als Autor des Programms, einer kurzen Beschreibung, was das Programm macht und einem Testbeispiel als Beispiel für eine Eingabe und die zu erwartende Ausgabe.

```
//Autor: Irene Rothe  
//Additon von 2 ganzen Zahlen  
//Testbeispiel: Eingabe: 2,3 Ausgabe: 5
```

# ✓ Elementare Datenstrukturen

## → Kontrollstrukturen

- if/else
- Lesbarkeit
- switch
- while
- do/while
- for
- do versus while
- Testen von Bedingungen
- C-Scheußlichkeiten
- Begriffe

## Flussdiagramme



# Motivation: Was ist hier das Problem?

## Flugzeugstart:

Wenn die Geschwindigkeit über 300 km/h, soll das Flugzeug abheben.

Wenn die Geschwindigkeit unter 300 km/h, dann soll das Flugzeug beschleunigen.

## Programmcode (Pseudocode):

```
if (v > 300) {Abheben}
```

```
if (v < 300) {mehr Geschwindigkeit}
```

## Diskussion:

Was ist hier das Problem?

Wie könnte man das beheben?

## Verbesserung:

```
if (v > 300) {Abheben}
```

```
else {mehr Geschwindigkeit}
```

Bemerkung: Es ist leider wirklich ein Unfall wegen dieses Programmierfehlers passiert.



# Kontrollstrukturen = Bestandteile eines Algorithmus

Ein Programm kann aus folgenden 3 Bestandteilen aufgebaut sein:

- Mach was: **Anweisung** (Sequenz) – Zeile mit Semikolon  
Beispiel: Mach jetzt sofort den Abwasch!
- Mach was wieder und wieder: **Schleife** – **while**, **for**  
Beispiel: Solange das Semester noch nicht zu Ende ist, mache mindestens einmal die Woche etwas für Informatik, danach jeden Tag, weil ich für die Klausur lernen muss 😊
- Mach was, wenn: **Verzweigung** - **if/else**  
Beispiel: Wenn Fragen für Informatik auftauchen, schicke Frau Rothe eine email, ansonsten bearbeite ein anderes Fach.

# Motivation? if/else-Konstrukt

Beispiel:

```
if ("normales Fahren == ja" ){//Lenkrad bewegt und Temperaturen verändern sich
    Abgaskontrolle aus //Scheiß auf Abgaskontrollen!
}
else {
    Abgaskontrolle an //Achtung: es findet ein Test statt,
                      //super Werte vortäuschen
}
```

Auch genannt: Volkswagening: Betrug durch Software

Auf diese Weise haben VW Vier-Zylinder-Modelle der Jahre 2009 bis 2015 erfolgreich den Abgastest bestanden, obwohl sie das festgesetzte Abgas-Limit für Dieselmotoren um das bis zu 40-fache überstiegen haben.



# if/else-Konstrukt

Verzweigungen können wie folgt programmiert werden:

```
if ( <Bedingung> ) { <Ausdruck1>;  
}  
else { <Ausdruck2>; }
```

Beispiel: Wenn ich mehr als 50 Euro habe, borge ich Klaus 20 Euro (was sich auf sein schon bei mir geborgtes Geld aufaddiert), ansonsten gebe ich ihm nur 10 Euro. Bedauerlicherweise verringert sich mein eigenes Geld jeweils um das geborgte Geld.

Der Code dafür:

```
if ( meingeld > 50 ) {  
    klausgeborgt = klausgeborgt + 20;  
    meingeld = meingeld - 20;  
}  
else {  
    klausgeborgt = klausgeborgt + 10;  
    meingeld = meingeld - 10;  
}
```



# if/else – Konstrukt: Beispiel erweitert

mit Deklaration und Initialisierung:

```
//Deklaration: ganzzahlige Eurowerte
int klausgeborgt;
int meingeld;
//Initialisierung: Startwerte fuer die Variablen
//ich habe in meiner Boerse 100 Euro
meingeld = 100;
//Klaus hat von mir schon 300 Euro geborgt
klausgeborgt = 300;

if ( meingeld > 50 ) {
    klausgeborgt = klausgeborgt + 20;
    meingeld = meingeld - 20;
}
else {
    klausgeborgt = klausgeborgt + 10;
    meingeld = meingeld - 10;
}
```



# if/else und double: Übung

Versuchen Sie zu verstehen, warum dieser Code funktioniert.

```
//Test: 13→13 ist ungerade
#include <stdio.h>
int main(){
    int zahl = 0;
    int hilf = 0;

    printf("Zahl: ");
    scanf("%i",&zahl);
    hilf = zahl/2*2;
    if(hilf == zahl){
        printf("%i ist gerade.",zahl);
    }
    else{
        printf("%i ist ungerade.",zahl);
    }
    return 0;
}
```

# if/else: Verschachtelung

Programmierkonstrukte können beliebig verschachtelt werden.

Beispiel:

```
if (a>0) {  
    printf("a positiv");  
}
```

```
else {
```

```
    if (a<0) {  
        printf("a negativ");  
    }  
    else {  
        printf("a Null");  
    }  
}
```

```
}
```

Verschachtelung im else-Fall



# Bemerkung: Bedingte Zuweisung

statt:

kann man auch folgendes Konstrukt benutzen:

```
if (a>=0) {  
    z=a;  
}  
else {  
    z=-a;  
}
```

entspricht  $z = (a \geq 0) ? a : -a;$

Anderes Beispiel:

```
printf("groessere Zahl: %i", x > y ? x : y);
```

# Lesbarkeit von Code

Dieser Code ist lauffähig, aber schlecht lesbar!

```
#include <stdio.h>
int main(){int regen=0;printf("Wenn es draussen regnet, geben
Sie bitte jetzt\n");printf("eine Eins ein ansonsten eine
Null!\n");scanf("%i", &regen);if (regen == 1 ){printf("Lies
noch mal das Vorlesungsscript!\n");printf("Nutze die Zeit und
uebe ein bisschen C!\n");}else{printf("Lass Dich nicht
ablenken von der Sonne!\n");printf("Nutze die Zeit und uebe
ein bisschen C!\n");}return 0;}
```



# Lesbarkeit

Dieser Code ist besser lesbar!

```
#include <stdio.h>
int main(){
int regen=0;
printf("Wenn es draussen regnet, geben Sie bitte jetzt\n");
printf("eine Eins ein ansonsten eine Null!\n");
scanf("%i", &regen);
if (regen == 1 ){
printf("Lies noch mal das Vorlesungsscript!\n");
printf("Nutze die Zeit und uebe eine bisschen C!\n");
}
else{
printf("Lass Dich nicht ablenken von der Sonne!\n");
printf("Nutze die Zeit und uebe ein bisschen C!\n");
}
return 0;
}
```



# Lesbarkeit

Dieser Code ist gut lesbar!



```
#include <stdio.h>
int main(){
    int regen=0;
    printf("Wenn es draussen regnet, geben Sie bitte jetzt\n");
    printf("eine Eins ein ansonsten eine Null!\n");
    scanf("%i",&regen);
    if (regen == 1){
        printf("Lies noch mal das Vorlesungsscript!\n");
        printf("Nutze die Zeit und uebe eine bisschen C!\n");
    }
    else {
        printf("Lass Dich nicht ablenken von der Sonne!\n");
        printf("Nutze die Zeit und uebe ein bisschen C!\n");
    }
    return 0;
}
```



# switch-Konstrukt

Verzweigungen abhängig vom Wert einer ganzen Zahl oder Zeichen kann man wie folgt programmieren:

```
switch (<Variable>)  
{  
    case <Konstante1>:  
        <Ausdruck 1>;  
    break;  
    case <Konstante2>:  
        <Ausdruck 2>;  
    break;  
    .  
    .  
    .  
    default:  
        <Ausdruck n>;  
}
```

# switch-Konstrukt: Beispiel

Je nachdem ob ich 10 Euro, 20 Euro oder 30 Euro habe, gebe ich Klaus entweder 5 Euro, 10 Euro oder 20 Euro. In allen anderen Fällen gebe ich Klaus nix!

Der Code dafür kommt auf der nächsten Folie.

# Beispielcode - umständlich

```
if(meingeld == 10) {
    klausgeborgt = klausgeborgt + 5;
    meingeld = meingeld - 5;
    printf ("Na gut, ich borge Dir zum allerletzten Mal 5 Euro!\n");
}
else{
    if(meingeld == 20) {
        klausgeborgt = klausgeborgt + 10;
        meingeld = meingeld - 10;
        printf ("Na gut, ich borge Dir 10 Euro!\n");
    }
    else{
        if (meingeld == 30){
            klausgeborgt = klausgeborgt + 20;
            meingeld = meingeld - 20;
            printf ("Na gut, ich borge Dir 20 Euro!");
        }
        else{
            printf ("Ich habe leider gerade kein Geld!");
        }
    }
}
}
```



# switch-Konstrukt: Beispiel

```
switch (meingeld) {  
    case 10:  
        klausgeborgt = klausgeborgt + 5;  
        meingeld = meingeld - 5;  
        break;  
    case 20:  
        klausgeborgt = klausgeborgt + 10;  
        meingeld = meingeld - 10;  
        break;  
    case 30:  
        klausgeborgt = klausgeborgt + 20;  
        meingeld = meingeld - 20;  
        break;  
    default:  
        printf ("Ich habe leider gerade kein Geld!");  
}
```

geht nur für  
char und int

**Achtung:** break nicht vergessen!



# switch: Übung

Programmieren Sie folgende Aufgabe:

**Tastenabfrage:** j für Ja, n für Nein.

- **Eingabe einer Taste**
- **Ausgabe:** Sie haben gerade Ja eingegeben  
              Sie haben gerade Nein eingegeben  
              Sie haben weder j noch n eingegeben

Lösung: <https://youtu.be/Lloz3syQy7c>



# while-Schleife (kopfgesteuert)

Schleifen können wie folgt programmiert werden:

```
while ( <Bedingung> ) { <Ausdruck>; }
```

Beispiel: Solange ich mehr als 50 Euro in meiner Geldbörse habe, borge ich Klaus (z.B. jeden Tag) 10 Euro.

Der Code dafür:

```
meingeld = 100;  
while ( meingeld > 50 ) {  
    klausgeborgt = klausgeborgt + 10;  
    meingeld = meingeld - 10;  
}
```

**Achtung:** Ändern der Bedingungsvariablen (im Beispiel: `meingeld`) innerhalb der Schleife nicht vergessen. Ansonsten erhält man eine Endlosschleife.



# while-Schleife: Beispiel

```
int zaehler = 0;
while (zaehler < 6){
    printf("%i \n",zaehler);
    zaehler=zaehler+1;
}
```

Ändern der Bedingungsvariablen nicht vergessen!  
Bedingungsvariable: `zaehler`

Ausgabe:

0  
1  
2  
3  
4  
5

# while: Übung

Geben Sie 10 Sternchen mit while aus.



# while: Übung - Abfangen falscher Eingabe

Schreiben Sie eine Schleife, die bei einer Eingabe größer 9 erneut zu einer Eingabe auffordert mit einem entsprechendem Text: Die Zahl muss kleiner oder gleich 9 sein.

# while-Schleife: Endlosschleife

```
while (1==1) {  
    printf("Ich kann einfach nicht aufhören! ");  
}
```

Big Bang Theory: <https://www.youtube.com/watch?v=k0xgjUhEG3U>

Später im 2.Semester genauer:

```
#typedef FOREVER while(1)
```



# for-Schleife: Zählschleife

Sehr praktisch ist die `for`-Schleife, wenn man weiß, wie oft eine Anweisung ausgeführt werden soll.

```
for (<Anfangszustand>; <Bedingung>; <Iterationsausdruck>){  
    <Ausdruck>;  
}
```

Beispiel: Nur 5 Mal borge ich Klaus 10 Euro.

Der Code dafür:

```
//nur 5 Mal borge ich Klaus 10 Euro  
//ich brauche keine Strichliste wegen for  
int i;  
for ( i=0; i<5; i=i+1){  
    klausgeborgt = klausgeborgt + 10;  
    meingeld = meingeld - 10;  
}
```

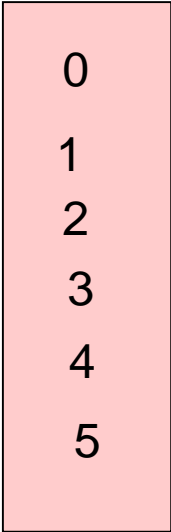
Immer dann verwenden,  
wenn die Anzahl der  
Schleifendurchläufe  
bekannt ist!



# for-Schleife: Beispiel

Ausgabe:

```
int zaehler;  
for (zaehler=0; zaehler < 6; zaehler++){  
    printf("%i \n",zaehler);  
}
```



0  
1  
2  
3  
4  
5

# for-Schleife: Endlosschleife

```
for(;;) {  
    printf("Ich kann einfach nicht aufhören!");  
}
```

# for: Übung

Geben Sie 10 Sternchen mit for aus.



# for versus while: beide können fürs Gleiche benutzt werden

for:

```
int zaehler;  
for (zaehler=0; zaehler < 10; zaehler++){  
    printf("%i \n",zaehler);  
}
```

while:

```
int zaehler=0;  
while (zaehler < 10){  
    printf("%i \n",zaehler);  
    zaehler++;  
}
```

Tipp: Wenn eine *for*-Schleife verwendet werden kann, sollte man das auch tun!



# Einschub: do-while-Schleife (fußgesteuert)

Es gibt auch eine fußgesteuerte Schleife.

```
do{  
    <Ausdruck>;  
} while (<Bedingung>) ;
```



Vergisst man schnell mal!

Beispiel: Ich bin ein großzügiger Mensch und borge, egal wie viel Geld ich habe, einmalig 10 Euro erst mal jedem, der mich fragt, z.B. auch Klaus. Will Klaus bei mir allerdings ein 2. Mal borgen, mache ich das dann doch abhängig davon, ob ich mehr als 50 Euro habe.

Der Code dafür:

```
do{  
    klausgeborgt = klausgeborgt + 10;  
    meingeld = meingeld - 10;  
} while (meingeld > 50);
```

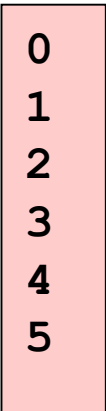
**Achtung:** Mir kann es also passieren, dass ich ins Minus rutsche. Bei einer while-Schleife ist dies nicht möglich!



# Einschub: Do-While-Schleife: Beispiel

Ausgabe:

```
int zaehler=0;
do{
    printf("%i \n",zaehler);
    zaehler++;
} while (zaehler < 6);
```



0  
1  
2  
3  
4  
5

# Testen von Bedingungen

Was kann alles in den runden Klammern von **while** und **if/else** stehen?

- Test auf Gleichheit: `==`

Beispiel: `(meingeld == 0 )`

wird mit ja beantwortet, wenn ich blank bin ansonsten mit nein

- Test auf Ungleichheit: `!=`

Beispiel: `(meingeld != 0)`

wird mit ja beantwortet, wenn ich noch Geld habe oder Schulden

- Test auf kleiner gleich: `<=`

Beispiel: `(meingeld <= 10 )`

wird mit ja beantwortet, wenn ich weniger oder genau 10 Euro habe

- Test auf größer gleich: `>=`

Beispiel: `(meingeld >= 10 )`

wird mit ja beantwortet, wenn ich mehr oder genau 10 Euro habe

# Testen von Bedingungen

- Test auf kleiner:  $<$
- Test auf größer:  $>$
- 2 Bedingungen müssen überprüft werden:  $\&\&$  (logisches UND)

Beispiel: `((meingeld > 0) && (klausgeborgt == 0))`

wird mit ja beantwortet, wenn ich Geld habe und Klaus keine Schulden bei mir hat, ansonsten mit nein

- eine von 2 Bedingungen muss überprüft werden:  $\|\|$  (logisches ODER)

Beispiel: `((meingeld > 0) || (klausgeborgt == 0))`

wird mit ja beantwortet, wenn:

- ich Geld habe und Klaus keine Schulden bei mir hat oder
- ich Geld habe, aber Klaus Schulden bei mir hat oder
- ich kein Geld habe, aber Klaus auch keine Schulden bei mir hat.

Nur wenn ich kein Geld habe und Klaus Schulden bei mir hat, lautet die Antwort nein.

# Testen von Bedingungen

Testfragen müssen so gestellt werden, dass die Antwort immer JA oder NEIN lautet.

Operatoren:	
>	größer
<	kleiner
>=	größer gleich
<=	kleiner gleich
==	gleich
!=	ungleich

Operatoren:	
&&	und (logisch)
	oder (logisch)
!	nicht

# C-Scheußlichkeiten



- Zuweisung '=' anstatt Vergleich '=='

```
if (a=3) printf("Das wollten wir nicht!");  
if (a==3) printf("Funktioniert! ");
```

- Fehlende Klammern:

```
if (a==3)  
    printf("Ausgabe in Abhängigkeit von der Bedingung!");  
printf("Ausgabe unabhängig von der Bedingung!");
```

```
if (a==3){  
    printf("So soll ");  
    printf("es sein!");  
}
```

TIPP: Immer Klammern!!!



- Kein Semikolon!

```
if (a==0) / {printf("a ist Null! ");}
```



# C-Scheußlichkeiten



In C ist eine Bedingung **erfüllt**, wenn sie ungleich Null ist. Ist die Bedingung "0" ist sie **nicht erfüllt**.

Beispiele (eventuell verwirrend)

```
i=5;  
if (i==5)
```

Bedingung erfüllt

```
i=3;  
if (i=5)
```

Bedingung erfüllt

```
i=0;  
if (i=0)
```

Bedingung nicht erfüllt

```
if (7<5<3)
```

Bedingung i.d.R. erfüllt, manchmal nicht (abhängig vom Compiler)

Begründung: Abarbeitung von links nach rechts:  
7<5 ist nicht erfüllt, also 0, 0<3 stimmt, also  
ist die Bedingung erfüllt



# Überspringen, da eine Praktikumsaufgabe

## Übung: Bedingungen

Angenommen wir wollen das Spiel HANGMAN programmieren. Wie muss die Abbruchbedingung formuliert werden, wenn das Spiel wie folgt enden soll:

- entweder das Wort wurde erraten oder
- es wurde 10x falsch geraten, also man hängt am Galgen.

Testen Sie Ihre Abbruchbedingung mit Hilfe einer Logiktablelle.

# Bemerkung: Übung - Ampelschaltung mit ODER

//gefällt mir gar nicht, aber scheint üblich für Microcontroller

```
if(cnt == 0 || cnt == 1 || cnt == 2) {  
    LED1 = 1; LED2 = 0; LED3 = 0; // "halten"  
}  
if(cnt == 3) {  
    LED1 = 1; LED2 = 1; LED3 = 0; // "anfahren"  
}  
if(cnt == 4 || cnt == 5 || cnt == 6 || cnt == 7) {  
    LED1 = 0; LED2 = 0; LED3 = 1; // "fahren"  
}  
if(cnt == 8 || cnt == 9) {  
    LED1 = 0; LED2 = 1; LED3 = 0; // "bremsen"  
}
```



# Bemerkung: Übung - switch und fall-through

```
switch (cnt) { // Ampel-Phase dekodieren
    case 0:
    case 1:
    case 2: LED1 = 1; LED2 = 0; LED3 = 0; break; // "halten"
    case 3: LED1 = 1; LED2 = 1; LED3 = 0; break; // "anfahren"
    case 4:
    case 5:
    case 6:
    case 7: LED1 = 0; LED2 = 0; LED3 = 1; break; // "fahren"
    case 8:
    case 9: LED1 = 0; LED2 = 1; LED3 = 0; // "bremsen"
}
```

# Einschub: Boolesche Algebra

Das Testen von Bedingungen verhält sich entsprechend der Booleschen Algebra.  
Hier ein paar Ausschnitte, ansonsten googeln.

1. Kommutativgesetz:  $a \overset{\text{UND}}{\wedge} b = b \wedge a$        $a \overset{\text{ODER}}{\vee} b = b \vee a$
2. Komplemente:  $a \wedge \neg a = 0$        $\overset{\text{NICHT}}{\neg}(\neg a) = a$
3. Distributivgesetz:  $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
4. Neutrale Elemente:  $\neg(0) = 1$        $\neg(1) = 0$

# Begriffe

Da Programmierung ein Fach für sich ist, hat es auch seine eigenen Begriffe:

- **Hauptprogramm**: `int main () {return 0;}`

Achtung: main darf es nur genau einmal pro Programm geben!

- **Anweisung**: alles, wonach ein Semikolon steht
- **Zuweisung**: Zeile mit nur *einem* Gleichheitszeichen und am Ende einem Semikolon

- **Variablendeklaration**: Variablenname und ihr Typ

Achtung: Variablen immer am Anfang deklarieren wegen der Übersichtlichkeit

- **Initialisierung**: *erste* Wertzuweisung
- **Operatoren**: alle Operatoren, wie man sie so aus der Schule kennt, z.B. +, -, \*, /, % ...
- **Schlüsselwörter**: `if, for, int, char, while, else, double...`
- **Kommentare**: Text nach `//` und zwischen `/* */`
- **Funktionsaufruf**: Aufruf von z.B. `printf()`, `scanf()` und anderen mitgelieferten C-Funktionen oder selbstgeschriebenen Funktionen (Funktionen werden später genauer erklärt)

# Programmmanfang

```
//Autor: Irene Rothe  
//Inhalt: Addition von zwei Zahlen  
//Testfall: Eingabe: 2,3 Ausgabe: 5
```

# Kontrollstrukturen - Zusammenfassung

- Verzweigung:

- `if (...) {...} else {...}`

- `switch (...) { case ...: ... break; default: ... }`

- (nur mit `int` und `char` auf Gleichheit)

- Schleifen:

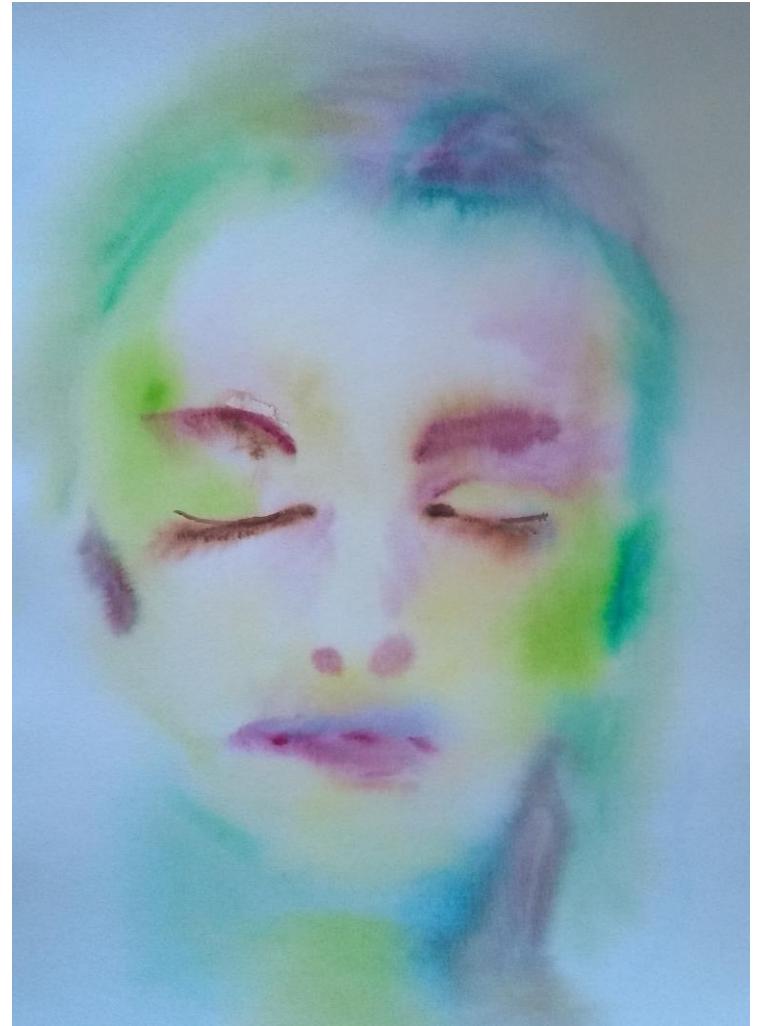
- `while (...) {...}`

- `for (...; ...; ...) {...}`

# Beispiel: Nullstellenberechnung für Polynome 3. Grades

Viel Spaß beim Gucken:

<https://youtu.be/2D0gAns0qbl>



✓ **Elementare Datenstrukturen**

✓ **Kontrollstrukturen**

→ **Flussdiagramme**

- Verzweigung
- Schleife



# Flussdiagramm: Motivation

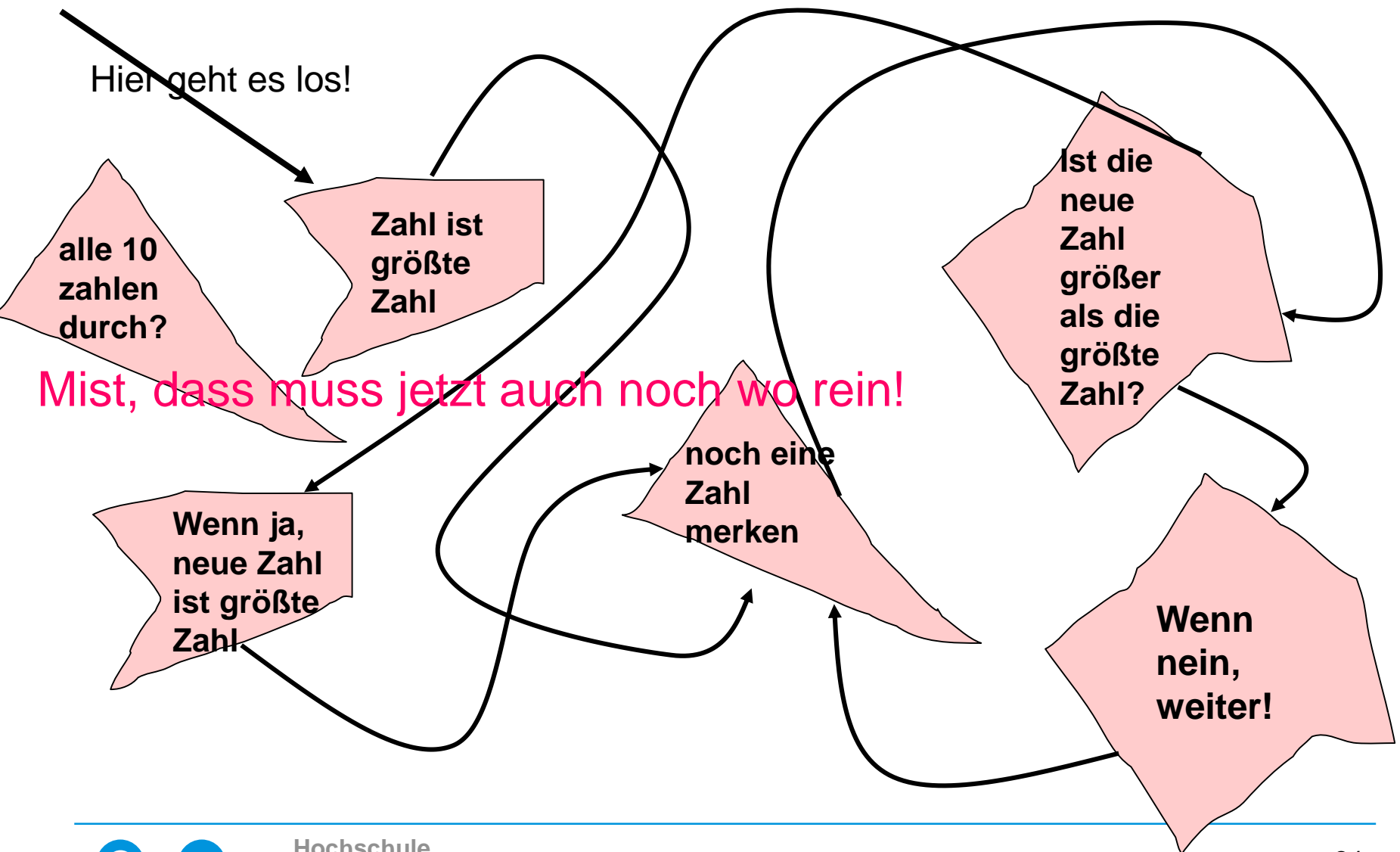
Wie geht man an ein Problem ran?

1. Problem verstehen
2. Testfall überlegen
3. Eingaben schreiben
4. Ausgaben schreiben  
→ in der Regel ergeben sich dann die ersten Ideen für den Rest
5. Ideen kann man bildlich darstellen, z.B. durch ein **Flussdiagramm (FD)**
6. Flussdiagramme verstehen alle Programmierer (egal für welche Programmiersprache) weltweit, FDs sind sehr geeignet, um über Programme zu reden und zu diskutieren.



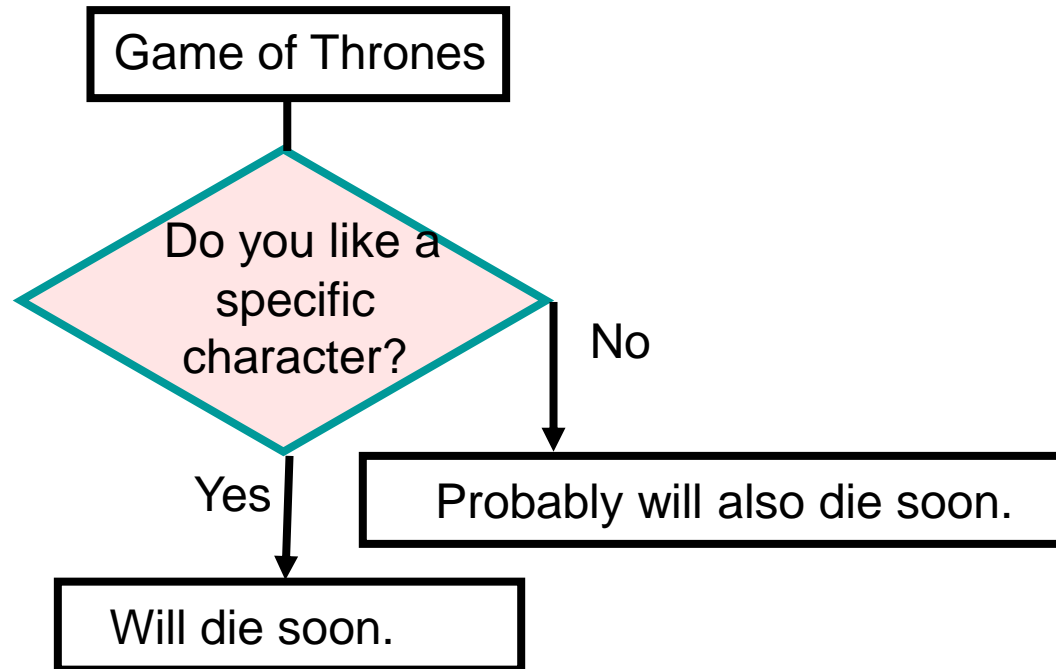


# Flussdiagramm: Motivation

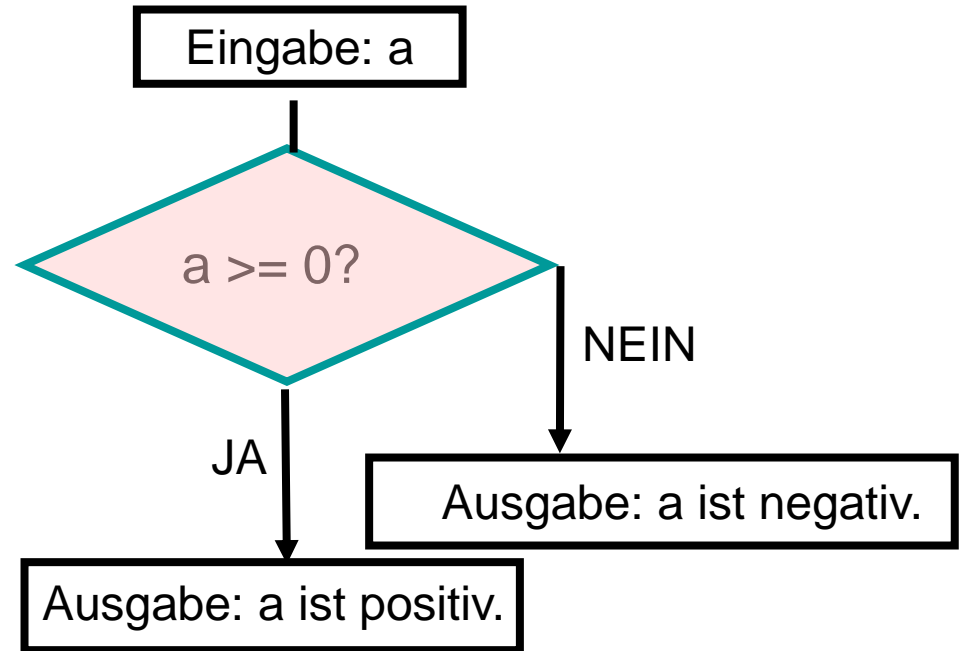


# Flussdiagramm: Verzweigung

Ein Flussdiagramm ist ein grafisches Hilfsmittel fürs Programmieren. Es wird weltweit verstanden, unabhängig von der Programmiersprache.



# Flussdiagramm: if/else- Beispiel

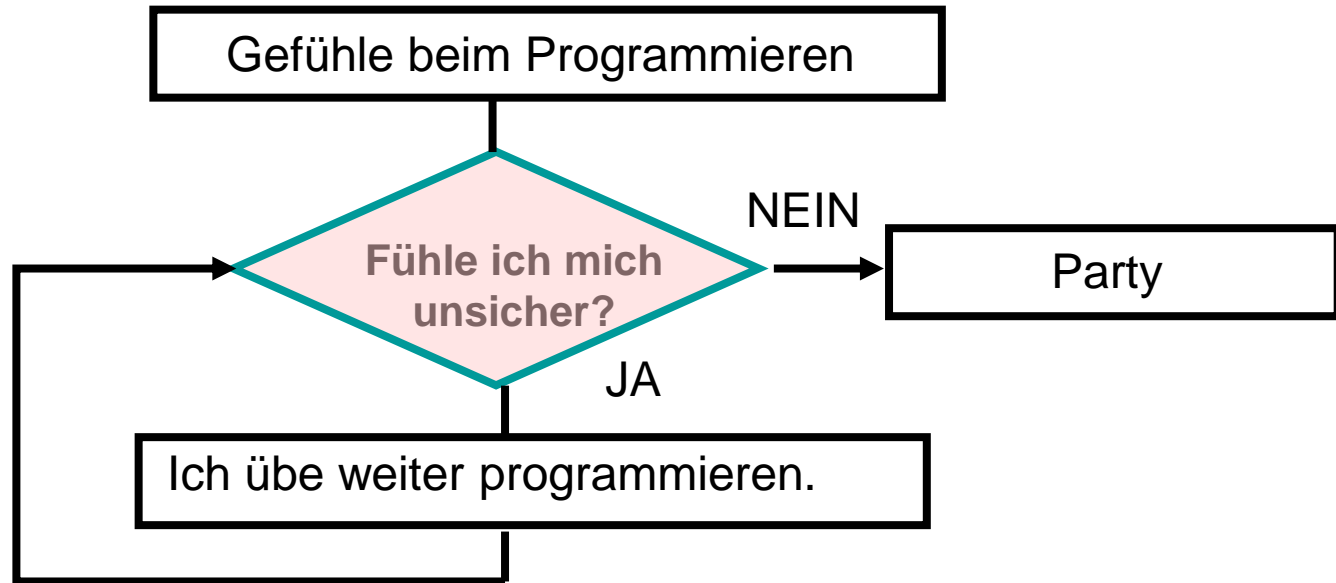


Der Code dafür:

```
printf("Eingabe:");  
scanf("%i",&a);  
if(a>=0){  
    printf("a ist positiv. ");  
}  
else{  
    printf("a ist negativ. ");  
}
```

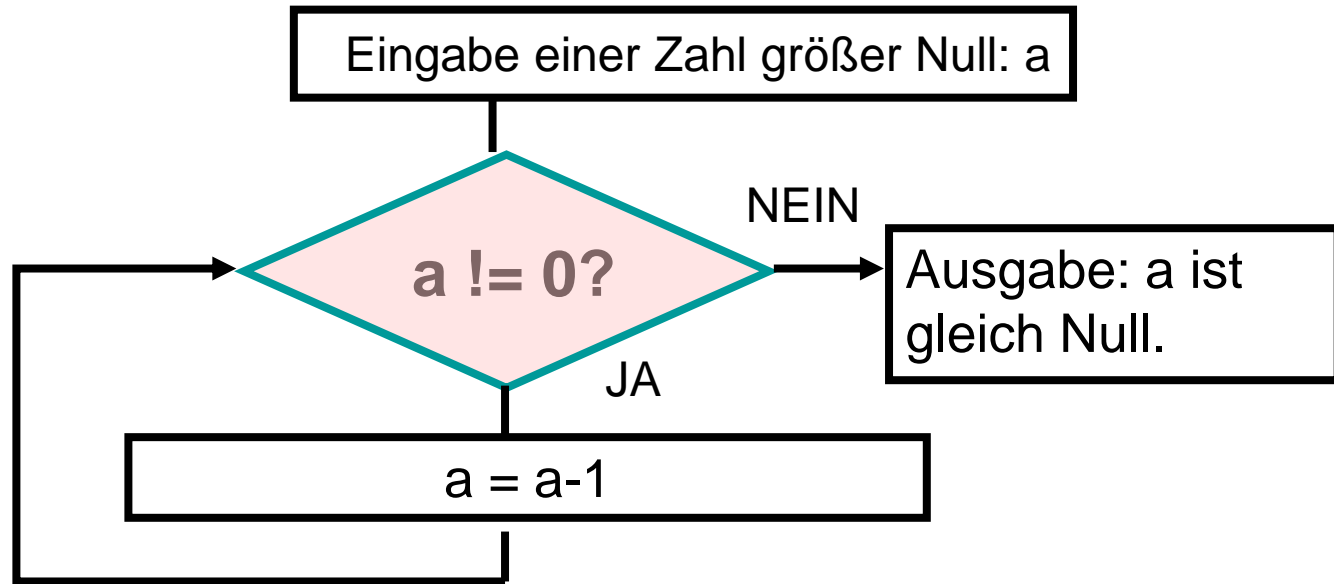


# Flussdiagramm: Schleife - Beispiel



**Achtung:** Alle Programmierkonstrukte funktionieren so, dass man IN der Schleife bleibt im Ja-Fall.

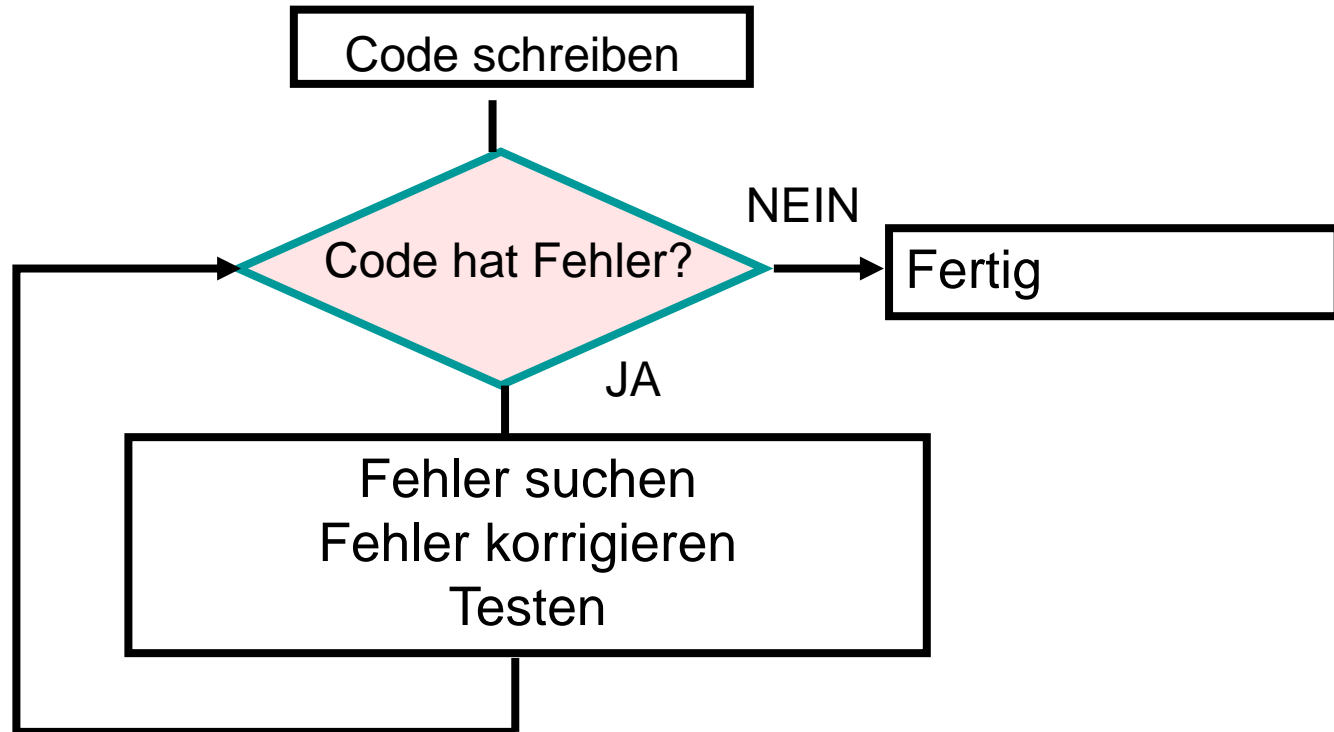
# Flussdiagramm: Schleife - Beispiel



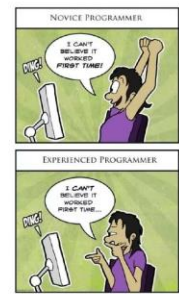
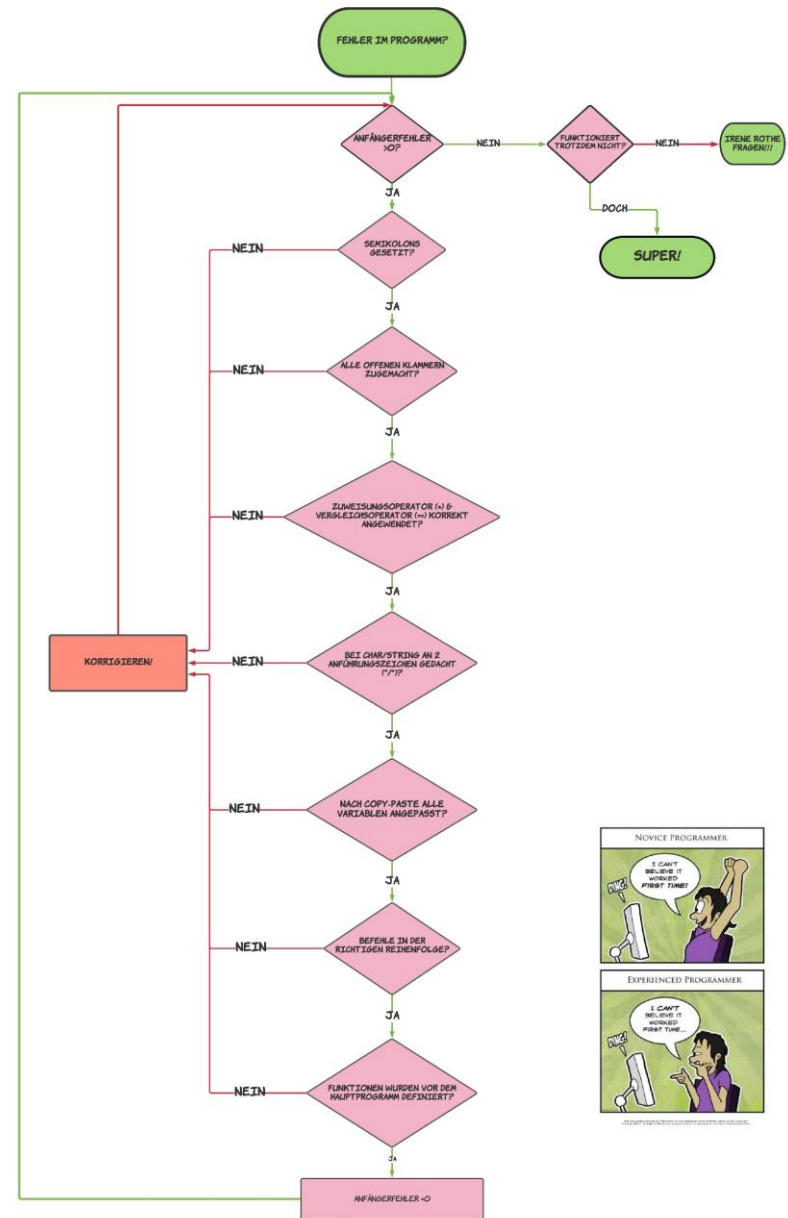
Der Code dafür:

```
printf("Eingabe einer Zahl größer Null:");  
scanf("%i",&a);  
while(a != 0){  
    a = a-1;//Runterzaehlen von a  
}  
printf("a ist gleich Null.");
```

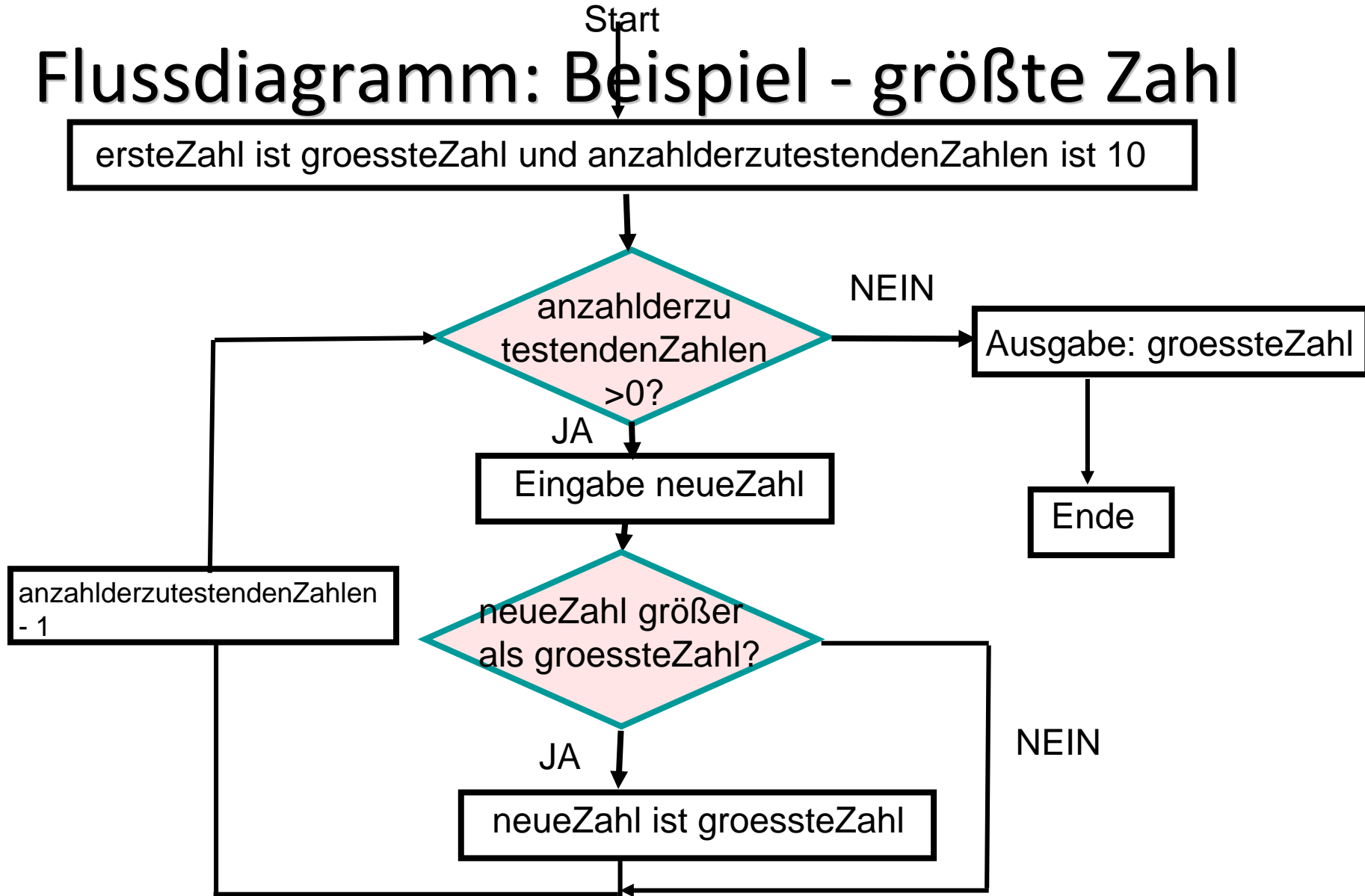
# Flussdiagramm: Beispiel



# Flussdiagramm: Fehlersuche



# Flussdiagramm: Beispiel - größte Zahl





# Beispiel: Größte Zahl - Anfang

```
//Autor: Irene Rothe  
//Programm zum Finden der größten ganzen Zahl  
//Test: Eingabe: 3,5,1,9,5,7,8,3,10,4  
//Ausgabe: 10
```

# Beispiel: Größte Zahl - Rahmen

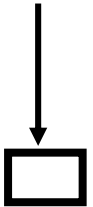
Start



```
#include <stdio.h>
int main() {
```

```
    return 0;
```

```
}
```



# Größte Zahl - Deklaration

Start  
↓

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;

    return 0;
}
```

↓  
Ende



# Größte Zahl - Zuweisung

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i;
    //zehn Zahlen sind gegeben
    i=10;

    return 0;
}
```

Start



Anzahl der zu testenden Zahlen i ist 10



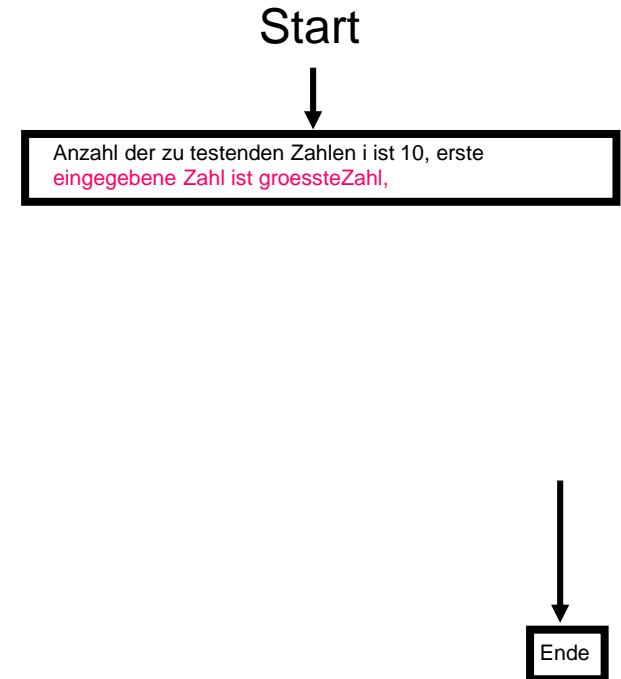
Ende



# Größte Zahl - Eingabe

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i;
    //zehn Zahlen sind gegeben
    i=10;
    //Eingabe
    scanf("%i",&groessteZahl);

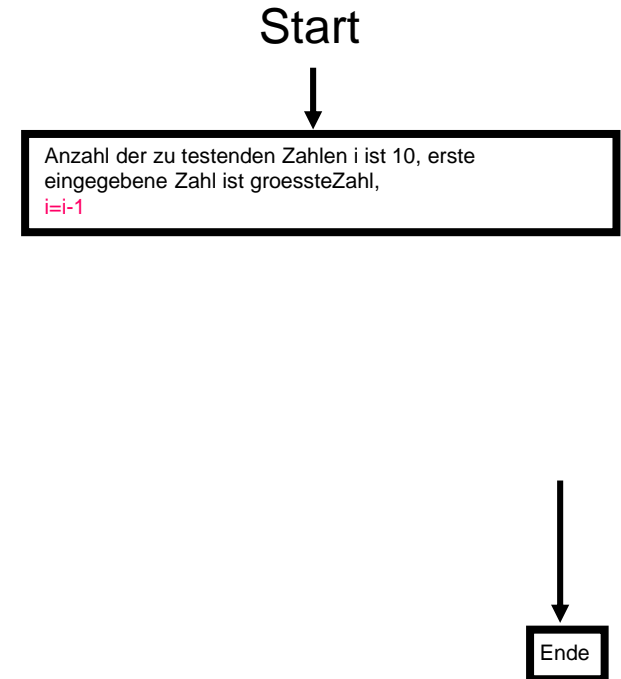
    return 0;
}
```



# Größte Zahl - Eingabe

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i;
    //zehn Zahlen sind gegeben
    i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;

    return 0;
}
```

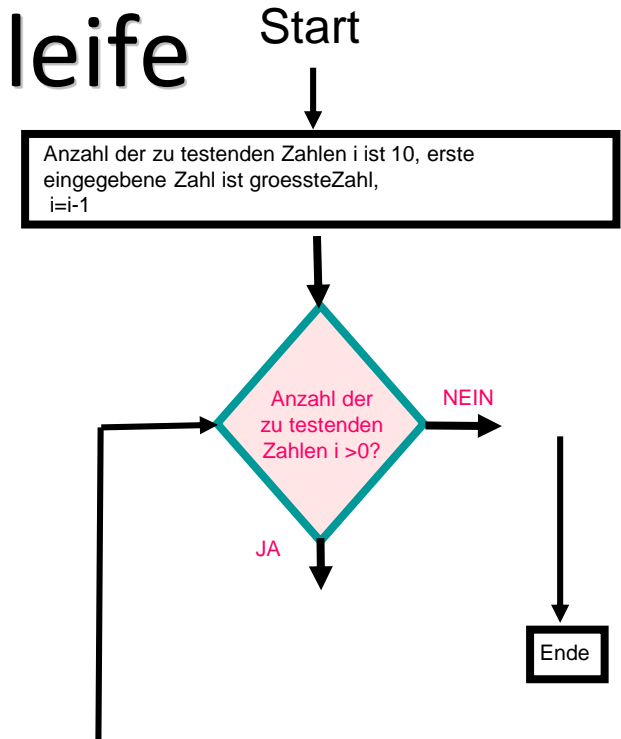


# Größten Zahl - while-Schleife

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i;
    //zehn Zahlen sind gegeben
    i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;
    //Schleife
    while ( i > 0 ) {

    }

    return 0;
}
```

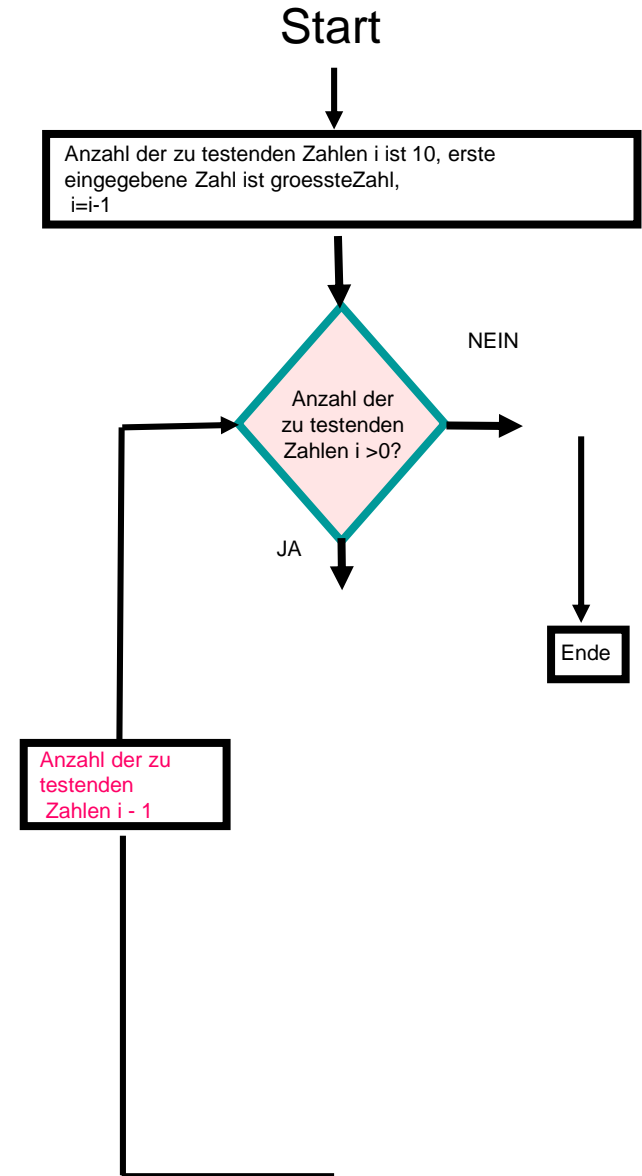


# Größte Zahl - Abbruch

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;
    //Schleife
    while ( i > 0 ) {

        i=i-1;

    }
    return 0;
}
```

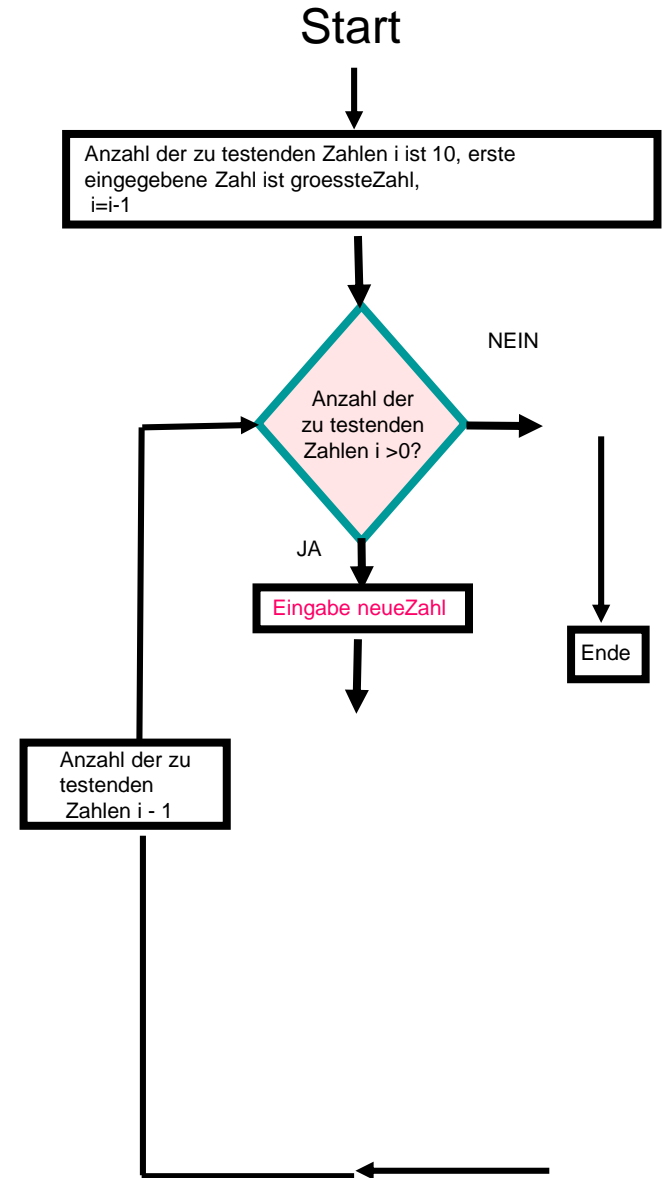




# Größte Zahl

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;
    //Schleife
    while ( i > 0 ) {
        //Eingabe
        scanf("%i",&neueZahl);

        i=i-1;
    }
    return 0;
}
```

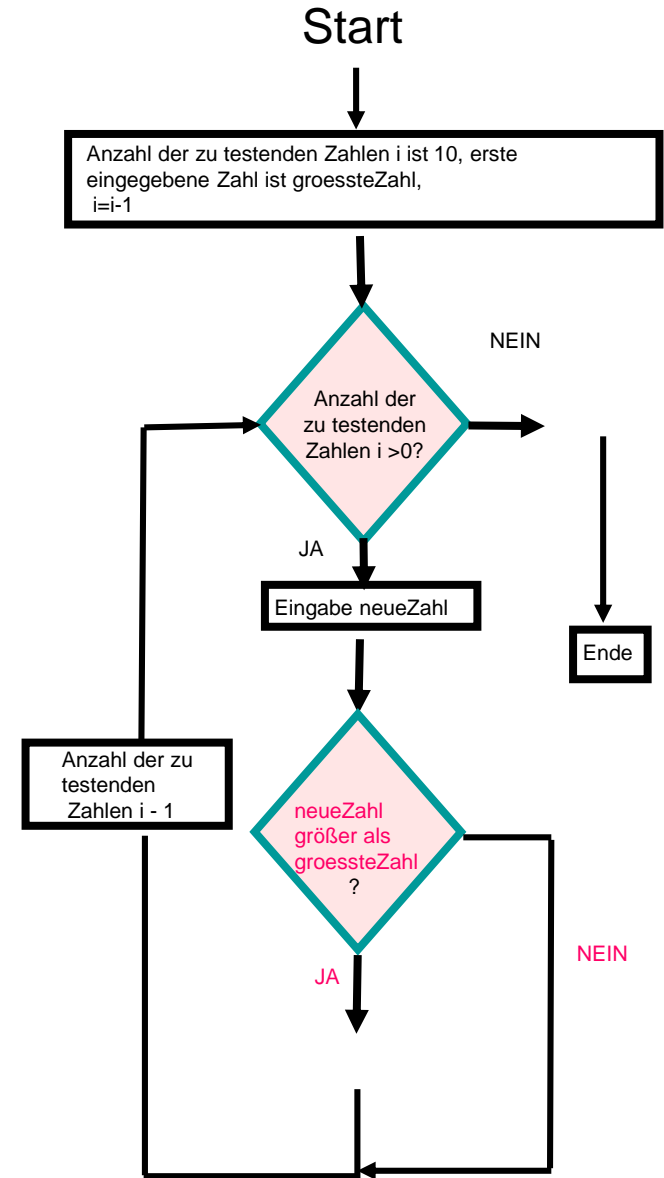


# Größte Zahl - if/else

```
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;
    while ( i > 0 ) {
        //Eingabe
        scanf("%i",&neueZahl);
        if(neueZahl > groessteZahl){

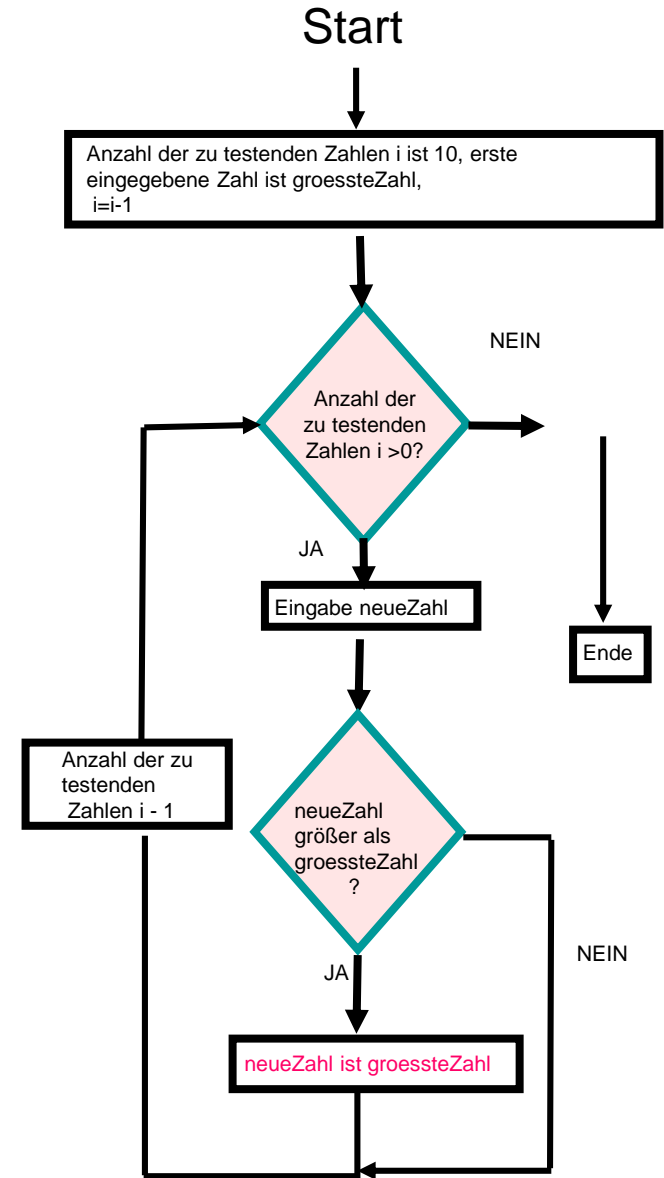
        }
        else{

        }
        i=i-1;
    }
    return 0;
}
```



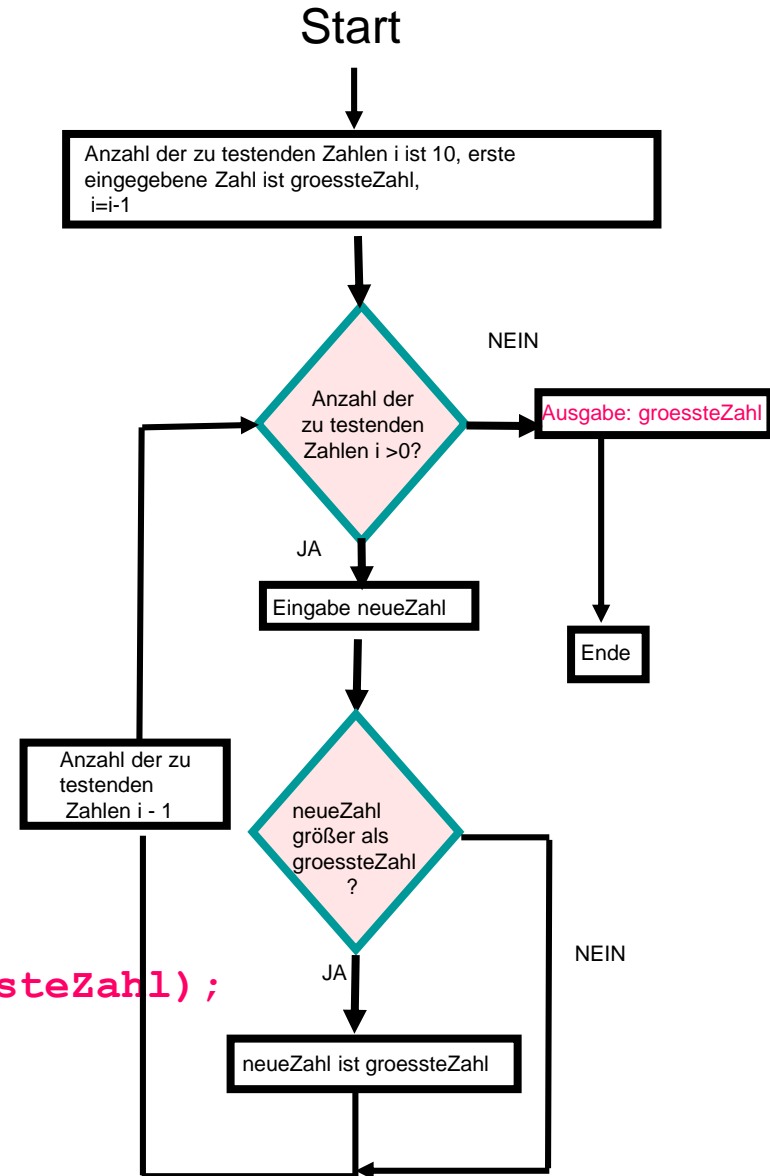
# Größte Zahl

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



# Größte Zahl - Ausgabe

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



# Noch schöner mit do/while

```
//Autor: Irene Rothe
//Programm zum Finden der größten ganzen Zahl
//Test: Eingabe: 3,5,1,9,5,7,8,3,10,4
//Ausgabe: 10
#include <stdio.h>
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl=0;
    int i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;
    do {
        scanf("%i",&neueZahl);
        if(neueZahl > groessteZahl){
            groessteZahl=neueZahl;
        }
        i=i-1;
    } while ( i > 0 ) ;
    printf("Die groesste Zahl ist %i",groessteZahl);
    return 0;
}
```

# Flussdiagramm: Zusammenfassung

- ... ist grafische Darstellung eines Algorithmus
- Es gibt zwei Symbole (Rhombus und Rechteck) für Fragen (Rhombus: Verzweigungen und Schleifen) und Anweisungen (Rechteck)
- Man bleibt nur **in** einer Schleife, wenn die Frage mit **Ja** beantwortet wird
- Schleife erkennt man daran, dass **alle** Spitzen des Rhombus besetzt sind
- Der Rhombus steht immer **am Anfang** oder **am Ende** einer Schleife
- Bei Rechtecken geht ein Pfeil dran und einer weg, nicht mehr
- Ja-Fall immer nach Möglichkeit nach unten, Nein-Fall wenn geht nach rechts
- Alle Schritte eines Algorithmus sollten im Flussdiagramm dargestellt werden

# Noch ein Beispiel für Flussdiagramme

Lernvideo für Flussdiagramme: [https://www.youtube.com/watch?v=OR1l\\_xerHsk](https://www.youtube.com/watch?v=OR1l_xerHsk)



Coco startet mit der C Programmierung Teil 3 - Das Flussdiagramm



# Flussdiagramm: Übung

1. Programm compilieren (in Maschinsprache übersetzen)
2. Syntaxfehler beseitigen bis der Compiler nicht mehr meckert und man eine ausführbare Datei (in Maschinsprache) erhält
3. Programm starten
4. Testen
5. Laufzeitfehler beseitigen





# Flussdiagramm: Übung

Malen Sie ein **Flussdiagramm** und schreiben Sie dann den **C-Code** dazu auf:

1. Sei A die größere der beiden Zahlen A und B (entsprechend vertauschen, falls B größer A).
2. Setze A auf den Wert  $A - B$ .
3. Wenn A und B ungleich sind, dann fahre fort mit Schritt 1, wenn sie gleich sind, ist der Algorithmus beendet und A oder B wird ausgegeben.

Was berechnet der Algorithmus?

C-Programm dafür: <https://youtu.be/tngTs2cTI-M>



# Collatz-Vermutung programmieren

Beginne mit einer beliebigen natürlichen Zahl  $n$ .

- Ist die Zahl gerade, teile sie durch 2,
- ist sie ungerade, multipliziere sie mit 3 und addiere 1.
- Wiederhole dies mit der so neu erhaltenen Zahl solange bis man Eins erhält.

Beispiel:  $n = 22$ : 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Test: 27, ...

Schreibe ein Programm, das dies für eine Zahl testet.

Frage: Endet dieser Algorithmus für jede natürliche Zahl?

→ Mehr zur Collatz-Vermutung, <https://www.youtube.com/watch?v=094y1Z2wpJg>

# Zusammenfassung

## Variablen:

- deklariert durch eindeutigen Namen (Groß+Kleinschreibung wird unterscheiden)
- haben Typ, der elementar sein kann (mit Liste von Fähigkeiten, was mit dem Typ bzgl. Operationen möglich ist)

## Zeichen **char**:

- speichert selbst keine Zeichen, sondern Zahlen anhand einer Tabelle (von 0-127, siehe ASCII-Tabelle)
- Zuweisung in einzelnen Anführungszeichen ('')
- 1 Byte groß
- Formatierer in C für Ein- und Ausgabe: %c

## Ganze Zahlen **int**:

- sind genau
- nicht allzu groß (immer daran denken, dass die größte ganze Zahl des Rechners erreicht werden könnte)
- mindestens 2 Byte, in der Regel 4 Byte groß
- Formatierer in C für Ein- und Ausgabe: %i, %d

## Gleitkommazahlen **double**:

- sind ungenau, aber sehr groß
- sollten niemals auf == oder != geprüft werden (Rundungsfehler!)
- Geldbeträge immer mit Ganzzahlen abbilden (99.99 → 9999)
- 8 Byte groß
- Formatierer in C für Ein- und Ausgabe: %lf

# Zusammenfassung

- Was ist ein Hauptprogramm?  
➔ die main-Funktion, der Beginn der Ausführung des Programms
- Was ist eine Deklaration und eine Initialisierung?  
➔ Definition einer Variable durch Typ und Name und erste Wertzuweisung
- Was sind die 2 wichtigsten Programmierkonstrukte?  
➔ if/else, while
- Wozu ist ein Flussdiagramm nützlich?  
➔ um den Algorithmus grafisch darzustellen.

# Das Marie Kondo Prinzip

If it doesn't spark joy, get rid of it.

- Wenn etwas keinen Spaß macht, ändere es!
- If a function or a line doesn't spark joy, get rid of it.
- Weniger ist immer mehr
- Joy: jede Zeile Code macht das, was sie soll (ohne 5 Bedingungen abzufragen oder 4 Fälle gleichzeitig abhandelt)

Quelle: <https://www.karim-geiger.de/blog/das-marie-kondo-software-design-principle>

# Literatur

- Jürgen Dankert: „Praxis der C-Programmierung“
- Siegfried Selberherr: „Programmieren in C“
- Heinz und Ulla Prinz: „C für PCs“
- P. Forbrig, O. Kerner: „Softwareentwicklung“, Fachbuchverlag Leipzig