

# Informatik

## Programmierung

von  
**Irene Rothe**

Zi. B 241

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

Instagram: irenerothedesign



# Informatik: ein Semester für TJs und VTs

Informatik = Lösen von Problemen mit dem Rechner

- Zum Lösen von Problemen mit dem Rechner braucht man **Programmierfähigkeiten (nur mit Übung möglich)**: Was ist Programmieren? Kleine Beispiele mit Code und Flussdiagramm → Vorbereitung auf die Projektwoche (Java)
- in Javascript und Webprogrammierung
- Wie löst der Rechner unsere Probleme? → mit **Dualdarstellung** von Zeichen und Zahlen und mit Hilfe von **Algorithmen**
- Was ist ein Algorithmus? Beispiele von Algorithmen: **Sortieren** und **Suche**
- Ein Beispiel für ein Problem: **Kryptografie**
- Noch ein Beispiel für ein Problem: **Bildverarbeitung**
- Sind Rechner auch Menschen? → **Künstliche Intelligenz**
- Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen
- ...



# Motivation für die Informatik

- Top 10 der Programmiersprachen (IEEE):  
<https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019>
- Linus Torvalds: Ich weiß nicht, wie ich es erklären soll, was mich am Programmieren so fasziniert, aber ich werde es versuchen. Für jemanden, der programmiert, ist es das Interessanteste auf der Welt. Es ist ein Spiel, bei dem du deine eigenen Regeln aufstellen kannst, und bei dem am Ende das herauskommt, was du daraus machst. Der Reiz besteht dann, dass der Computer das tut, was du ihm sagst. Unbeirrbar. Für immer. Ohne ein Wort der Klage. Du kannst den Computer dazu bringen, das er tut, was du willst, aber du musst herausfinden, wie. Programmieren ist eine Übung der Kreativität.
- Alan Turing (Gründer der Informatik): if thoughts (that is, information) can be broken up into simple constructs and algorithmic steps, then machines can add, subtract or rearrange them as our brains do.

# Wozu muss ich Programmieren lernen?

Damit ich weiß, was das ist, wenn ich mal darüber schreibe.



# Einschub: Boeing 737 Max

Aktuelles Beispiel: Absturz in Indonesien 2018, Absturz in Äthiopien 2019

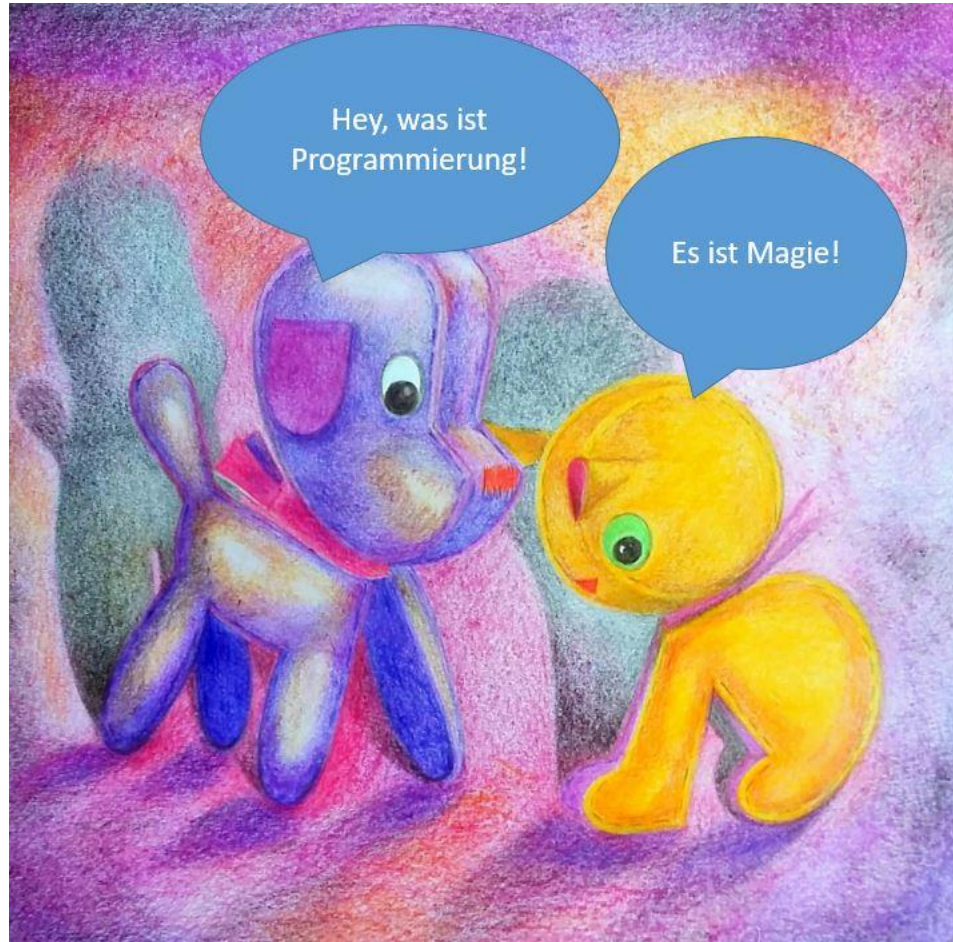
```
immer=1;
while(immer==1){
    nasenhoehe=messenNasenhoehe();
    if (nasenhoehe>...){
        anschaltenMCAS();//senkt Nase nach unten
    }
    else {
        //alles super
    }
}
```

Übersetzung in Deutsch: Immer steht für eine Variable (entspricht einem Speicherplatz im Rechner, wenn das Programm ausgeführt wird), wo der Wert 1 gespeichert ist. Solange dieser Wert gleich 1 ist, wird die Nasenhöhe des Flugzeuges gemessen und in der Variable nasenhoehe (=Speicherplatz) gespeichert. Wenn die Nasenhöhe größer als ein bestimmter Wert ist (...), wird ein System (genannt MCAS) angeschaltet, das die Nase des Flugzeuges mit Gewalt nach unten drückt. Ansonsten wird nix gemacht.

Problem: Die Piloten wussten nichts von diesem eingebauten Programm, wunderten sich über das Verhalten des Flugzeuges und steuerten dagegen (Nase nach oben und Geschwindigkeit), wodurch die Nase ständig auf und ab hüpfte. (<https://www.welt.de/wirtschaft/article190768827/Boeing-737-Max-Blowback-Problem-liefert-neue-Erklaerung-fuer-Sturzflug.html>)



# Was ist Programmieren?





# Einschub: Was ist Programmieren?



Aus Futurama (Matt Groening): Wohnungssuche in Neu-New York“

# Motivationsbeispiel in C

Auch wenn es so scheint, als würden Sie nichts verstehen, versuchen Sie für ein paar Minuten doch Sinn in diesem Text zu sehen:

```
#include <stdio.h>

int main () {
    int bierflaschenanzahl = 50;
    while (bierflaschenanzahl > 0 ) {
        printf("Kuehlschrank auf.\n");
        printf("Flasche rausnehmen und trinken.\n");
        printf("Kuehlschrank zu.\n");
        bierflaschenanzahl = bierflaschenanzahl-1;
        printf("Es stehen noch %i leckere Bierchen im\n",bierflaschenanzahl);
    }
    printf("Bier ist alle!\n");
    return 0;
}
```





# Motivationsbeispiel in Java

Auch wenn es so scheint, als würden Sie nichts verstehen, versuchen Sie für ein paar Minuten doch Sinn in diesem Text zu sehen:

```
class Biertrinken {
    public static void main(String [] args){
        int bierflaschenanzahl = 100;
        while (bierflaschenanzahl > 0 ) {
            System.out.println("Kuehlschrank auf.");
            System.out.println("Flasche rausnehmen und
                               trinken.");
            System.out.println("Kuehlschrank zu.");
            bierflaschenanzahl = bierflaschenanzahl-1;
            System.out.println("Es stehen noch
                               "+bierflaschenanzahl+" leckere Bierchen
                               im Kuehlschrank!");
        }
        System.out.println("Bier ist alle!");
    }
}
```



# Motivationsbeispiel in Java...Verbesserung

Auch wenn es so scheint, als würden Sie nichts verstehen, versuchen Sie für ein paar Minuten doch Sinn in diesem Text zu sehen:

```
class Biertrinken {
    public static void main(String [] args){
        int bierflaschenanzahl = 100;
        while (bierflaschenanzahl > 0 ) {
            System.out.println("Kuehlschrank auf.");
            System.out.println("Flasche rausnehmen und
                               trinken.");
            System.out.println("Kuehlschrank zu.");
            //Wie viel Flaschen nimmst Du gerade raus
            input...anzahlRausgenommen
            bierflaschenanzahl = bierflaschenanzahl-anzahlRausgenommen;
            System.out.println("Es stehen noch
                               "+bierflaschenanzahl+" leckere Bierchen
                               im Kuehlschrank!");
        }
        System.out.println("Bier ist alle!");
    }
}
```



# Programmiersprache

- übernimmt die Rolle eines Vermittlers zwischen Mensch und Maschine.
- bietet Möglichkeit zur Eingabe und Ausgabe von Daten
- Problem: endlicher Speicher

# Programmiersprache: Java IDE installieren

JRE (Java Runtime Environment) muss installiert sein, was aber eigentlich immer automatisch durch das Betriebssystem so ist: führt (interpretiert) plattformunabhängigen Bytecode (\*.class) durch JVM (Java Virtual Machine: Teil der JRE) plattformabhängig aus (Javaprogramm redet nur mit der JVM)

→ Zu installieren: JDK (Java Development Kit): Java-Compiler **javac**

Wozu IDEs?

→ Syntaxhervorhebung, Autovervollständiger, Unterstützung bei komplexen Codeänderungen (Refactoring) und mehr

Empfehlung: <http://javaeditor.org/doku.php?id=de:installation> → Portable Software



# Programmiersprache: Alternativ - C

Empfehlung: <https://sourceforge.net/projects/orwelldvcpp/>



# Oder Wegprogrammierung: Javascript

Extravorlesung



# Programmieren übers Internet:

Zum Beispiel: <https://www.onlinegdb.com>

→ Dann einfach eine Programmiersprache auswählen





# Woraus besteht ein Rechner/Computer?

Ein Computer besteht aus

- Teil/Teile zum Ausgeben
- Teil/Teile zum Eingeben
- Chips (zum Rechnen, Speichern/Lesen, logisch Auswerten):  
CPU
- Zusatzteile (Festplatte, ...)

# Was kann ein Rechner/Computer?

Ein Rechner kann:

1. in Speicher Werte schreiben
2. aus Speicher Werte lesen
3. Rechnen:  $+$ ,  $-$ ,  $*$ ,  $/$  und noch ein bisschen mehr
4. logische Ausdrücke auswerten (Gleichheit, Ungleichheit,  $>$ ,  $<$ , ODER, UND, ...)
4. (steuern)

# Wie bringt man einen Rechner dazu, für uns ein Problem zu lösen?

- Der Rechner kann nur Probleme lösen, wofür ein Mensch eine **Problemlösungsanleitung** (= Algorithmus) gefunden hat.
- Dieser **Algorithmus** (Problemlösungsanleitung) sollte in vernünftiger Zeit abarbeitbar sein.
- Mit dem Finden von Algorithmen kann man viel Geld verdienen, besonders wenn die Algorithmen schnell sind.

# Wie bekommt man den Algorithmus in den Rechner?

1. Man formuliert den Algorithmus in einer Programmiersprache (egal welcher) → **Programm**
2. Für jede existierende Programmiersprache gibt es Übersetzungsprogramme, die das Programm (= in einer Programmiersprache-formulierte-Problemlösungsanleitung) in Maschinensprache (01101101110...) übersetzt → **Maschinenprogramm**
3. Das übersetzte Programm (in Windows/DOS **executable**) kann man dann ausführen, und unser Problem wird vom Rechner gelöst, in dem der Rechner die Lösung anzeigt (auf Bildschirm, ...)

# Wie wird das Programm abgearbeitet?

1. Auf jedem Rechner ist ein Betriebssystem installiert.
2. Das Betriebssystem kümmert sich um die Eingabe und Ausgabe und noch ein paar andere Dinge. Mit Hilfe des Betriebssystems können wir überhaupt mit dem Rechner arbeiten.
3. Dieses Betriebssystem sucht in unserem Programm nach einer **main**-Funktion
4. Bei der **main**-Funktion beginnt die Abarbeitung.

Frage: Wie viele **main**-Funktionen darf so ein Programm haben?



# Was ist Programmieren

Was ist Programmieren?

→ Schreiben von Programmen

Was ist ein Programm?

→ etwas Virtuelles, das nützliche Dinge für uns erledigt auf einem Rechner

Was muss man tun und für Charakterzüge haben, um ein Programm schreiben zu können?

→ eine Sprache (Programmiersprache) erlernen

→ Problem verstehen

→ algorithmisches Denken

→ Hartnäckigkeit und Geduld



# Algorithmisches Denken

Es gibt 3 Möglichkeiten, aus denen ein Algorithmus (Begriff wird später ganz genau definiert; stellen Sie sich jetzt einfach eine Anleitung vor) aufgebaut ist (plus einige Dinge ringsum):

- Mach was - Dinge
- Mach was wieder und wieder - Dinge
- Mach was, falls – Dinge

Um algorithmisch zu denken, muss man:

- in einzelnen Schritten (unbedingt endlich viele Schritte!) denken
- einzelne Schritte präzise und eindeutig erklären (so dass keine andere Auslegung möglich ist)

Am besten nur noch wie folgt denken:

- tue solange wie ...
- wenn ..., dann ... ansonsten ...
- tue von ... bis ... und allem dazwischen

Dafür gibt es Strukturen: Sprach- und Datenstrukturen

ACHTUNG: eine wirkliche Methode, wie man einen Algorithmus findet, gibt es nicht. **Leider!**



# Übung

Geben Sie jeweils ein Beispiel aus Ihrem Leben an:

1. Was machen Sie mehr als einmal? Wann hören Sie auf damit?
2. Was tun Sie abhängig von einer bestimmten Bedingung? Was tun Sie, wenn diese Bedingung nicht eintritt?

# Übung

1. Stellen Sie sich vor, Ihnen werden nacheinander 20 Zahlen (also nicht alle gleichzeitig) gezeigt.
2. Denken Sie, dass Sie am Ende die größte Zahl benennen könnten?
3. Wie haben Sie das gemacht?
4. Schreiben Sie als Liste auf, wie Sie das gemacht haben. Versuchen Sie jeden einzelnen Schritt aufzuschreiben so präzise wie möglich und immer auf eine neue Zeile.
5. Sie haben Ihren ersten eigenen **Algorithmus** erstellt. Übrigens ist dies kein so einfacher Algorithmus. Es ist nicht klar, warum das eigentlich jeder kann. Wer hat Ihnen das beigebracht? Man lernt im Laufe seines Lebens eine Menge Dinge, deren man sich gar nicht bewusst ist. In der Informatik kann man einige dieser versteckten Fähigkeiten nutzen. Zwei weitere Dinge, die eigentlich jeder kann sind: Sortieren von Büchern und Suchen nach einem Buch in der Bibliothek.
6. Den Algorithmus für die **Suche nach der größten Zahl unter 20** zu programmieren, ist die viel einfachere Aufgabe, als die Idee für den Algorithmus selbst zu haben.
7. Hier lernen wir, wie man einen Algorithmus von Deutsch in eine Programmiersprache umschreibt. Dies ist am Anfang nicht so einfach, aber jede neue Sprache ist am Anfang ungewohnt und neu.
8. Der Algorithmus, formuliert in einer Programmiersprache (in unserem Fall wird das C sein), wird dann von einem Programm, genannt **Compiler** oder Übersetzungsprogramm, in Maschinensprache umgewandelt. Wenn Sie Fehler in der Sprache gemacht haben (eine Art Rechtschreibfehler), kann der Compiler Ihr Programm nicht übersetzen.
9. Das Programm in Maschinensprache kann dann Ihr Rechner ausführen.
10. Man muss dann noch testen, ob das Programm so abläuft, wie man das wollte. Man sollte das Programm mit verschiedenen Eingaben ausprobieren.



# Einschub: Testen

- Flugzeugunglück in Barcelona
- Bug Bounty

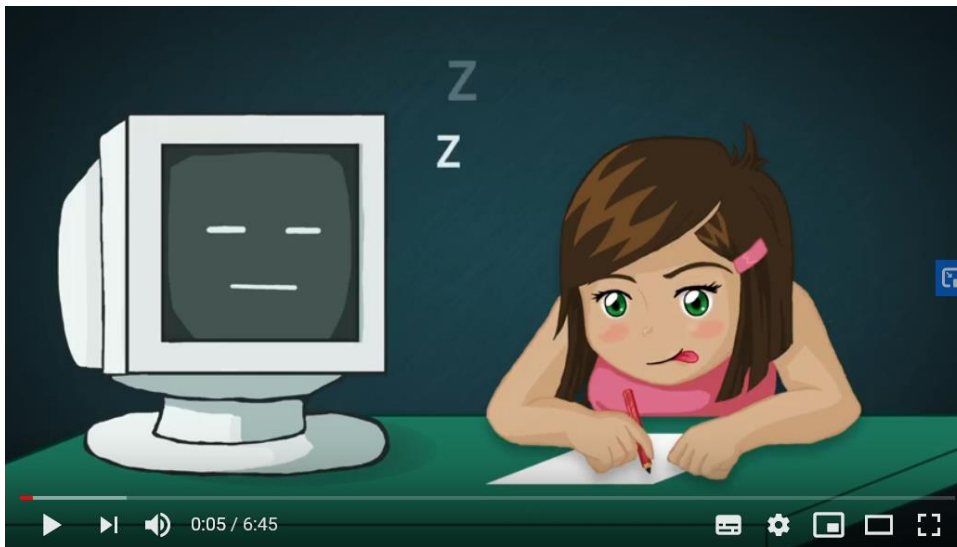


[https://youtu.be/ss0\\_IrtnoCk](https://youtu.be/ss0_IrtnoCk)

# Übung

Programmieren Sie folgendes Spiel:

- Das Programm würfelt eine Zahl zwischen 1 und 100
- Der Spieler soll die Zahl raten, indem er eine Zahl eingibt
- Das Programm gibt eine Ausgabe, ob die zu ratende Zahl größer oder kleiner als die eingegebene Zahl ist oder
- Ob die Zahl geraten wurde, dann hat man gewonnen.



[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



Hochschule  
Bonn-Rhein-Sieg

Vorlesung\_B\_Programmierung

# „Rate eine Zahl“ als Android-App

1. <http://appinventor.mit.edu/explore/#>
2. Klicke links auf „**Create App**“
3. Zugriff auf Google-Konto: Klick auf "**Zulassen**".
4. Klick auf "Never take survey,, dann auf "Continue"
5. App-Inventor: Klick auf "**New Project**,, ...

**Achtung:** Google Account nötig!

# NXC-Beispiele

- Robot solves the magic cube:  
<http://www.youtube.com/watch?v=eaRcWB3jwMo>
- Flexpicker:  
[http://www.youtube.com/watch?v=yxsgu\\_uzxQQ&feature=related](http://www.youtube.com/watch?v=yxsgu_uzxQQ&feature=related)
- Flexpicker with Lego:  
[http://www.youtube.com/watch?v=YoXCn4Gh\\_HA&feature=Playlist&p=0DF1677B61FCD3F8&playnext\\_from=PL&playnext=1&index=2](http://www.youtube.com/watch?v=YoXCn4Gh_HA&feature=Playlist&p=0DF1677B61FCD3F8&playnext_from=PL&playnext=1&index=2)

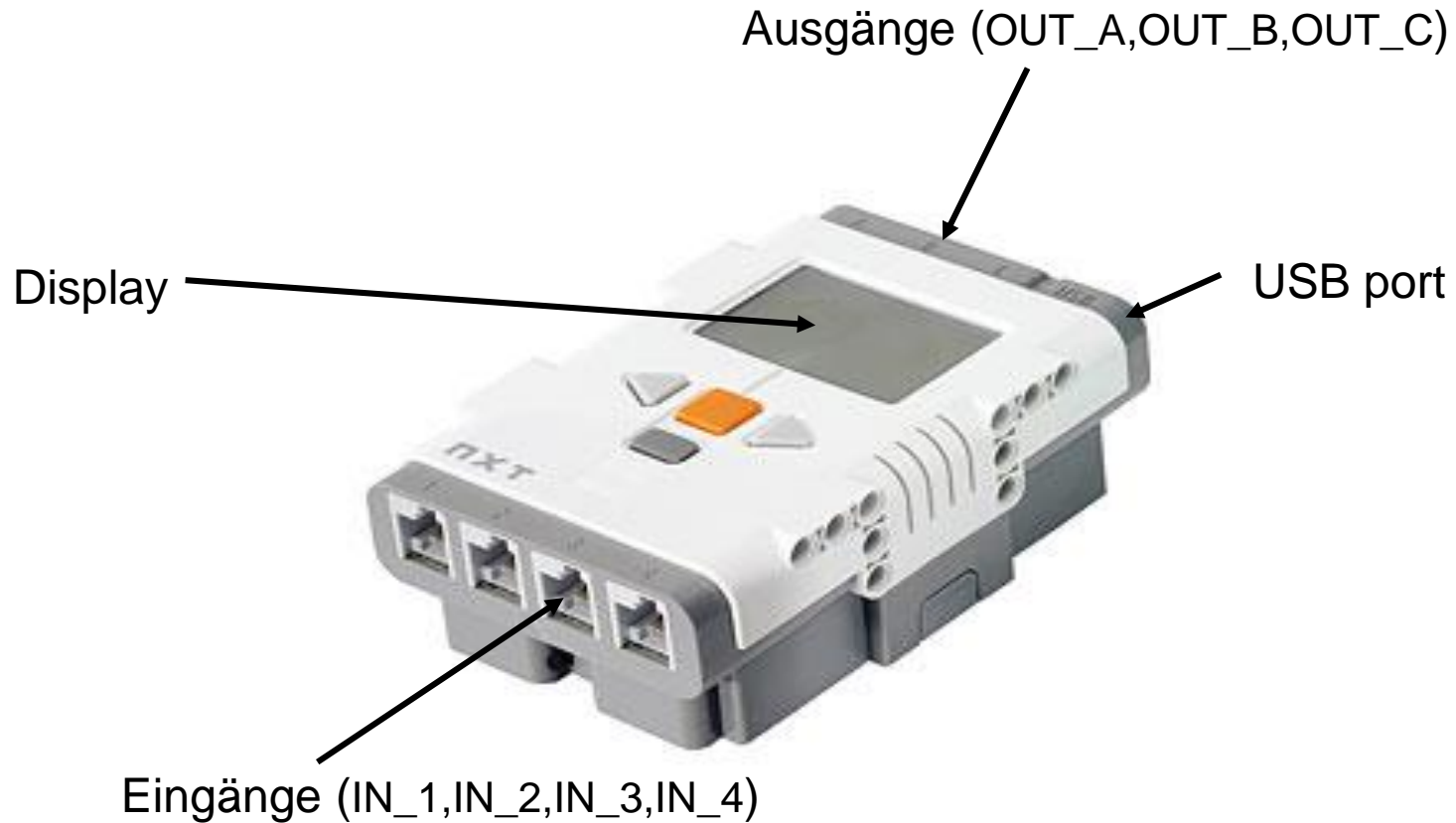
# NXT Hardware

- Lego NXT computer with Firmware installed
- 3 motors (for the 3 output ports)
- Touch sensor
- Light sensor
- Rotation sensor (inside the motors)
- Ultrasonic range sensor
- Sound sensor
- Bluetooth connection
- Lots of Lego bricks





# NXT-Microcontroller



# Programmieren

- ...ist das Erlernen einer Sprache mit **Syntax** (Wörtern, Punkt, Komma, ...) und **Semantik** (Bedeutung)
- Kleiner Unterschied: in der Regel sprechen wir diese Sprache nicht mit einem Menschen
- Programm besteht aus
  - einem Anfang,
  - einem Ende und
  - dem Algorithmus dazwischen.
- Alles hat einen Anfang: In der Programmiersprache Java sieht der Anfang wie folgt aus:

```
class meinproblem{  
    public static void main (String[] args){
```

- Und das Ende wie folgt:  
 }}



# Das erste Java-Programm

```
class meinproblem{  
    public static void main (String[] args){  
        System.out.println("Hallo Welt!");  
    }  
}
```

Anfang des  
Programms

Ende des Programms

Hier fängt die Ausführung des Programms an. Das ist sozusagen die Seite 1 eines Buches, wo man mit dem Lesen beginnt (da lässt man auch das Inhaltsverzeichnis und Vorwort meist weg).

Ausgabe auf dem Bildschirm

1. Geben Sie diesen Text in Ihr Compilerprogramm und nennen es meinproblem.java.
2. Übersetzen Sie den Text (Programm) in eine andere Sprache (damit nicht nur Sie das Programm verstehen, sondern auch der Rechner) durch Klicken auf **Compile**.
3. Lassen Sie Ihr Programm durch Ihren Rechner ausführen, z.B. durch Klicken auf **Run**
4. Es sollte folgende Ausgabe auf dem Bildschirm erscheinen: Hallo Welt!



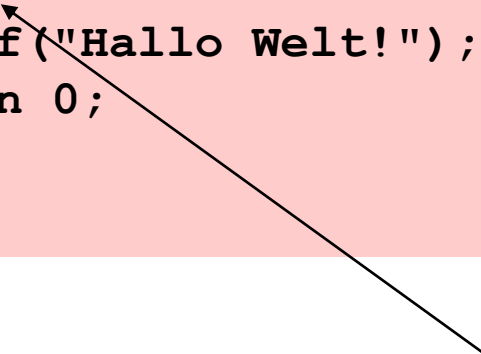
# Wozu ist dieses erste Programm nützlich?

Dieses Programm ist gut, um zu wissen,

- dass ein Compiler auf unserem Rechner installiert ist,
- dass der Compiler funktioniert,
- dass wir mindestens ein Programm schon mal hinbekommen haben.

# Das erste C-Programm: mit Anfang, Ende und Inhalt

```
#include<stdio.h>
int main(){
    printf("Hallo Welt!");
    return 0;
}
```



Vorgefertigte C-Funktion:  
Ausgabe von Text auf dem Bildschirm

# Das erste NXC-Programm: mit Anfang, Ende und Inhalt

```
task main () {  
    TextOut(0,50,"Hello World !");  
}
```

Vorgefertigte NXC-Funktion:  
Ausgabe von Text auf dem NXT-Bildschirm

# Das zweite NXC-Programm

```
task main () {  
    OnFwd(OUT_B, 60); //Losfahren mit Motor B  
    OnFwd(OUT_C, 60); //Losfahren mit Motor C  
    Wait(2500); //eine Zeit lang fahren  
    Off(OUT_B); //Stoppen von Motor B  
    Off(OUT_C); //Stoppen von Motor C  
    TextOut(0, 50, "Programmieren macht Spass!");  
}
```

Was passiert hier?

Vorgefertigte NXC-Funktionen





# 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 **Variablen**, 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 rechnet der Rechner

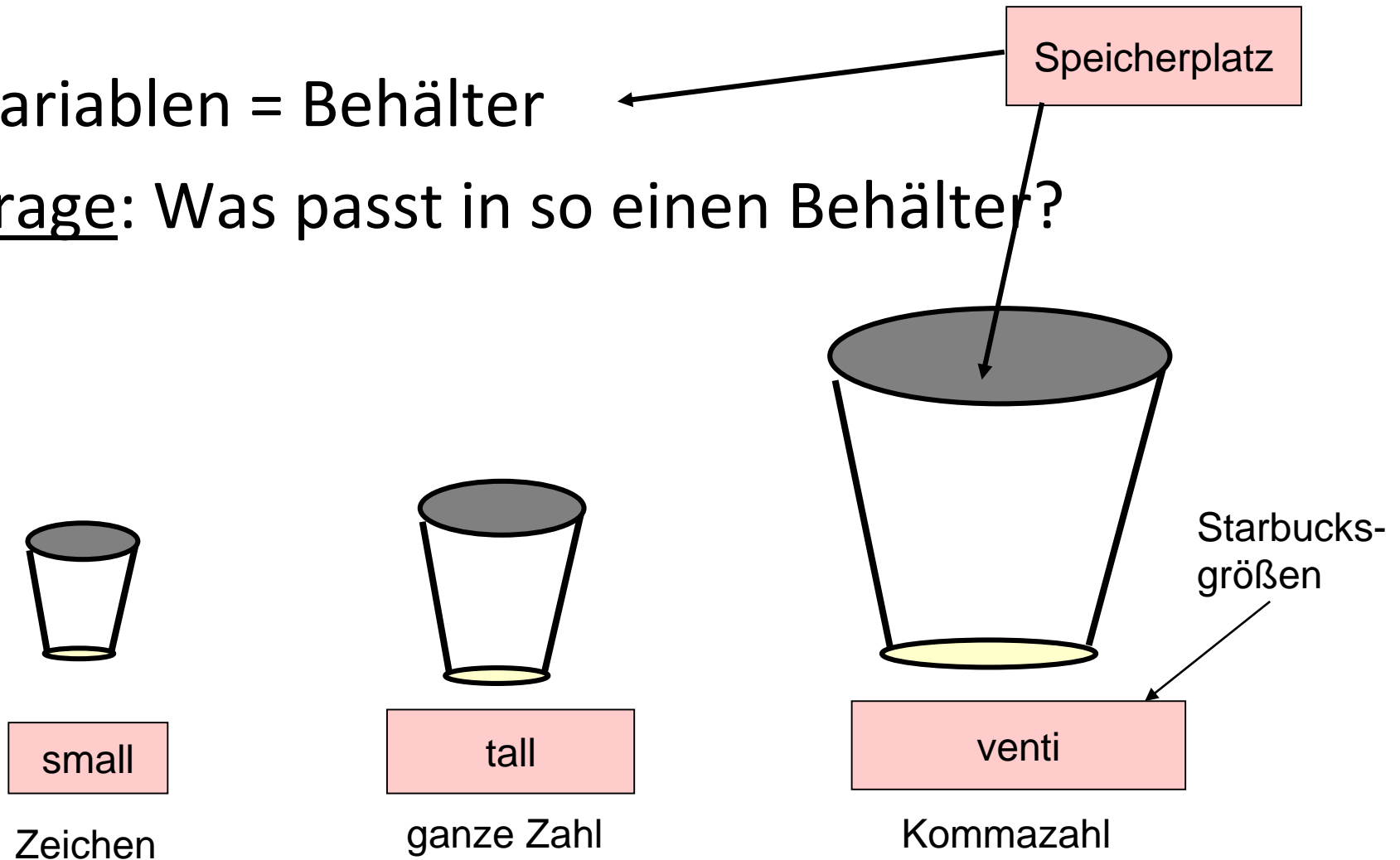
Variablen haben eine Größe, abhängig von dem, was man darauf abspeichern will.



# Variablen: Motivation

Variablen = Behälter

Frage: Was passt in so einen Behälter?



# Variablen: Welche Werte kann ein Rechner abspeichern?

1000001 kann verschiedene Bedeutungen haben:

- A (Zeichen)
- 65 (Zahl)
- Befehl in Maschinensprache

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;
```

Deklaration      Initialisierung

**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 sich lesen wie ein deutscher oder englischer Satz

Beispiel:

```
int gibtanWievielBierimKuehlschrankSteht;
```

oder

```
int anzahlBierimKuehlschrank;
```

oder

```
int numberOfBeerinFridge;
```

# Programmiersprache

- besteht aus Syntax
- Mit Hilfe der Syntax kann man eine bestimmte Semantik (Bedeutungslehre) umsetzen (Syntax/Zeichen kann aus Wörtern, Phrasen oder Symbolen bestehen. Die Semantik beschäftigt sich mit den Beziehungen zwischen den Zeichen und den Bedeutungen dieser Zeichen)
- Englische Wörter haben sich in Programmiersprachen durchgesetzt.
- Deshalb:
  - **char** (Zeichen),
  - **int** (ganze Zahlen),
  - **double** (Kommazahlen)

# Besonderheiten beim Programmieren

- = steht für Zuweisung auf eine Variable (Speicherung auf einen Speicherplatz)

Beispiel: **a=5**

- Semikolon: Abschluss einer Anweisung

Beispiel: **a=5 ;**

- Abarbeitung einer Zeile:

1. Zuerst ausführen aller Operationen, die rechts stehen
2. Dann Ergebnis auf Variable schreiben, die links steht

Beispiel: **a=a+2 ;** oder **a=b\*c+7 ;**

**Achtung:** = ist also nicht benutzbar, wenn man zwei Variablen vergleichen will.

- Vergleich von zwei Variablen: ==

Beispiel: **a==b**

**Achtung:** logische Ausdrücke immer in runde Klammern schreiben

Beispiel: **(a==b)**



# Elementare Datentypen:

Für uns in der Projektwoche sind folgende elementare Datentypen wichtig:

Ganze Zahlen: `int zahl = 42;`

Zeichen: `char zeichen = 'c';`

Mehrere einzelne Zeichen hintereinander (Java): `char[] zk = new char [10];`

Strings (Java): `String name = "Irene";`





# Kleines Beispiel

```
int a=2;  
int b=6;  
a = a + 3 * b;  
System.out.println(a) ;
```

Frage: Was wird hier ausgegeben?

**Achtung:** Der Rechner hält sich an die Rechenregeln, z.B. Punktrechnung geht vor Strichrechnung!

Frage: Was ist noch nicht erlaubt in der Mathematik?

# 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.



# 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.
- 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;
}
```

# NXC if/else

```
int licht;  
int speed = 100;           //Geschwindigkeit 100  
SetSensorLight(IN_1);      //Festlegung IR-Sensor auf Eingang 1  
licht = Sensor(IN_1);  
if(licht < 45){            //nach rechts fahren  
    Off(OUT_A);  
    OnFwd(OUT_B, speed);  
}  
else{                      //nach links fahren  
    Off(OUT_B);  
    OnFwd(OUT_A, speed);  
}
```



# 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 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 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;
}
```

# 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.



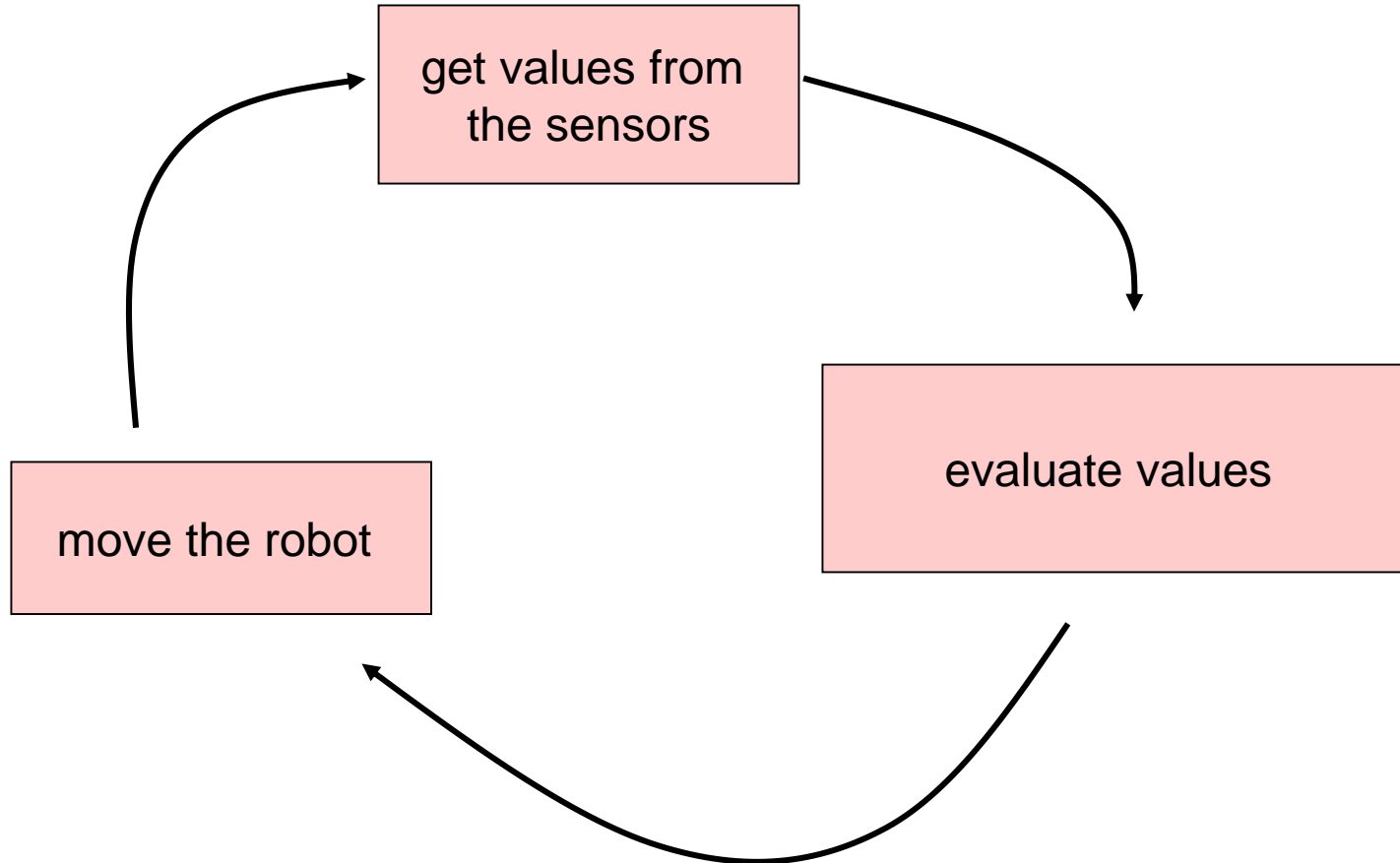
# Übung: Was wird hier gemacht?

```
class GroessteGanzeZahl {  
    public static void main (String [] args){  
        int i=0;  
  
        while (i<i+1) {  
            i=i+1;  
        }  
        System.out.println("Bereich von int: "+(i+1)+" bis "+i);  
    }  
}
```

Vorführen!



# How to program a robot?



# NXC: While-Schleife

```
while (TRUE) {  
  
}
```

# 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!





# NXC: For-Schleife

```
for(int i=0; i<4;i++){  
    OnFwd(OUT_AB, 50);  
    Wait(1100);  
    Off(OUT_AB);  
    OnRev(OUT_A, 40);  
    OnFwd(OUT_B, 40);  
    Wait(700);  
    OnFwd(OUT_AB, 50);  
    Wait(1100);  
    Off(OUT_AB);  
  
}
```

Was passiert hier?



# Bestandteile eines Programmes grob

- Eingabe
- Ausgabe
- Reservierung von Speicherplatz
- Zuweisungen und Rechnen
- Verzweigungen
- Schleifen

# 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

- 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.

# Übung

Erklären Sie das logische UND und ODER an einem eigenen Beispiel.



# 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

# Flussdiagramm: Motivation

Wie geht man an ein Problem ran?

1. Problem verstehen

2. Testfall überlegen

3. Eingaben schreiben

4. Ausgaben schreiben

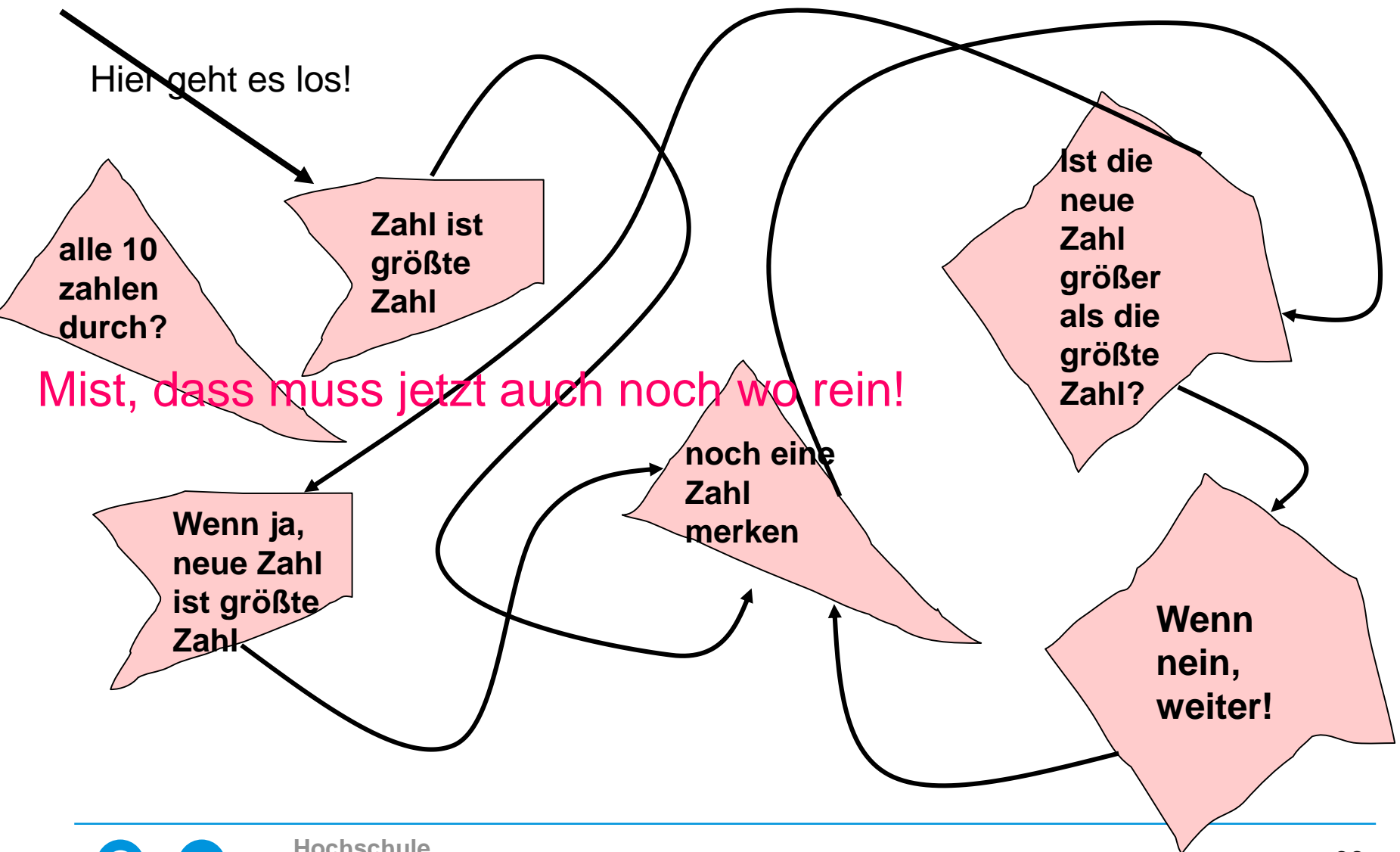
→ in der Regel ergeben sich dann die ersten Ideen für den Rest

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.

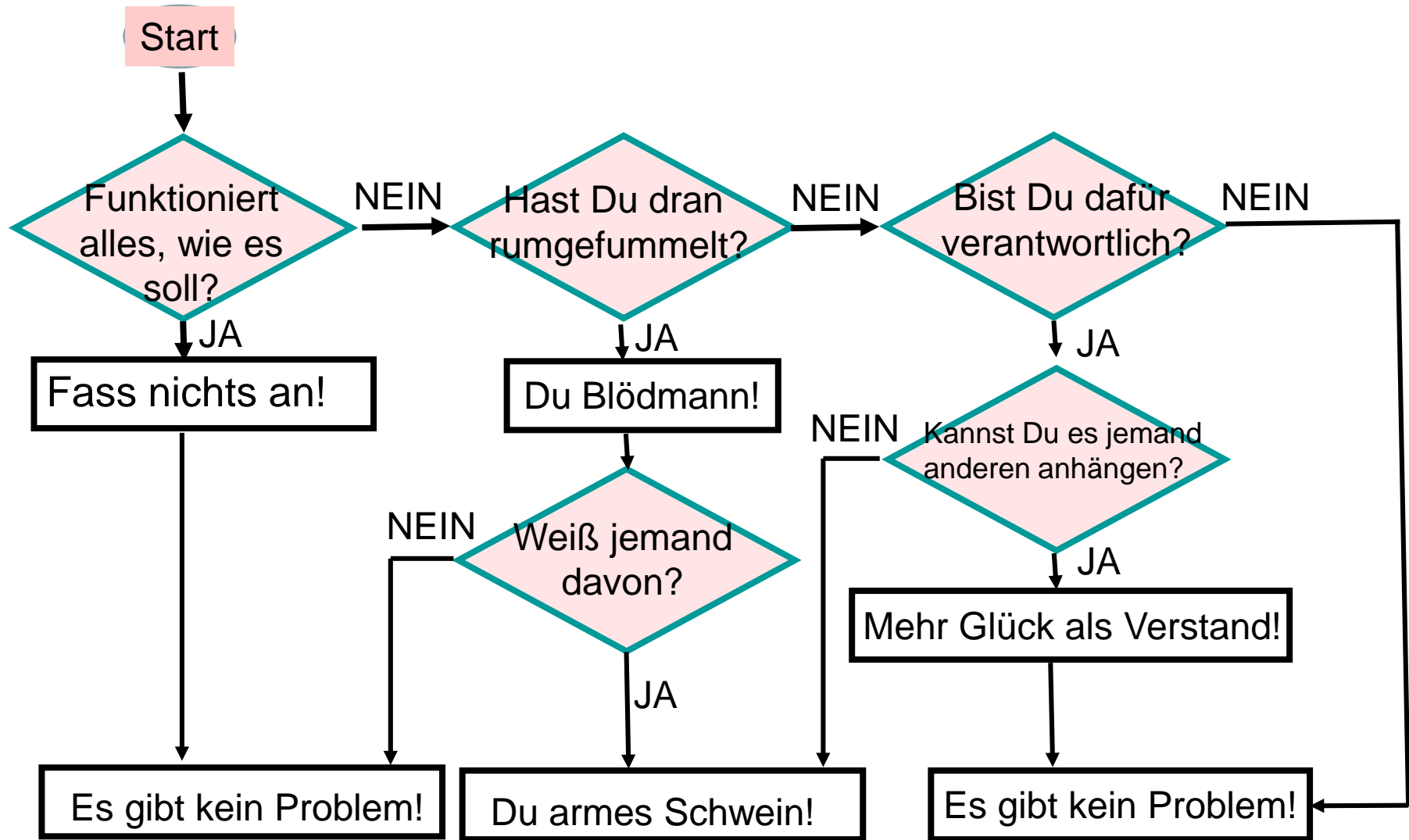


# Flussdiagramm: Motivation



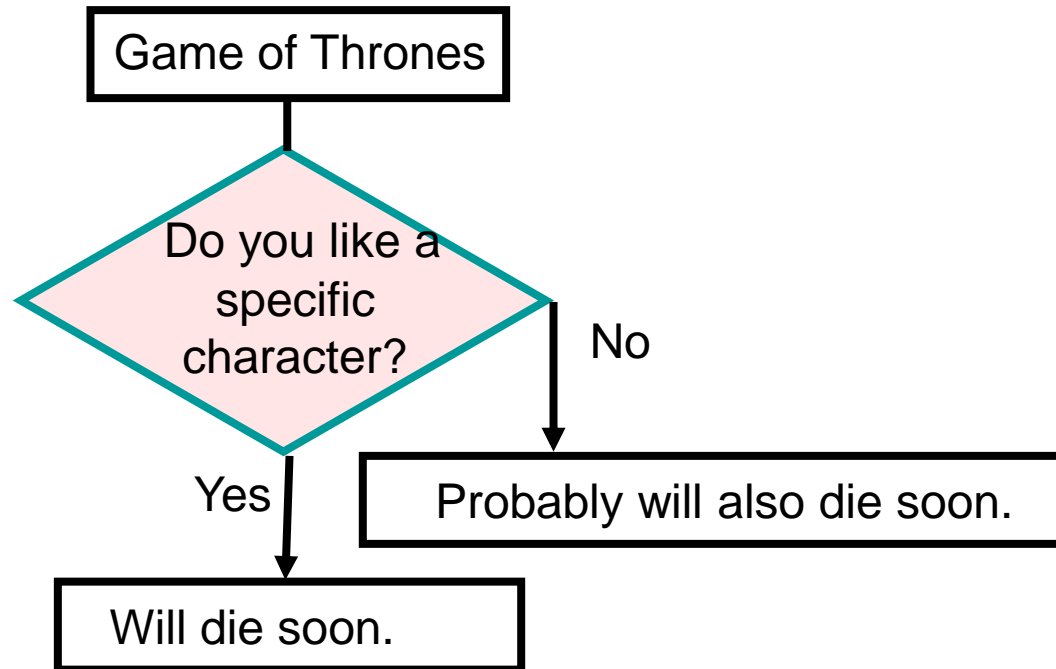


# Flussdiagramm für Ingenieure

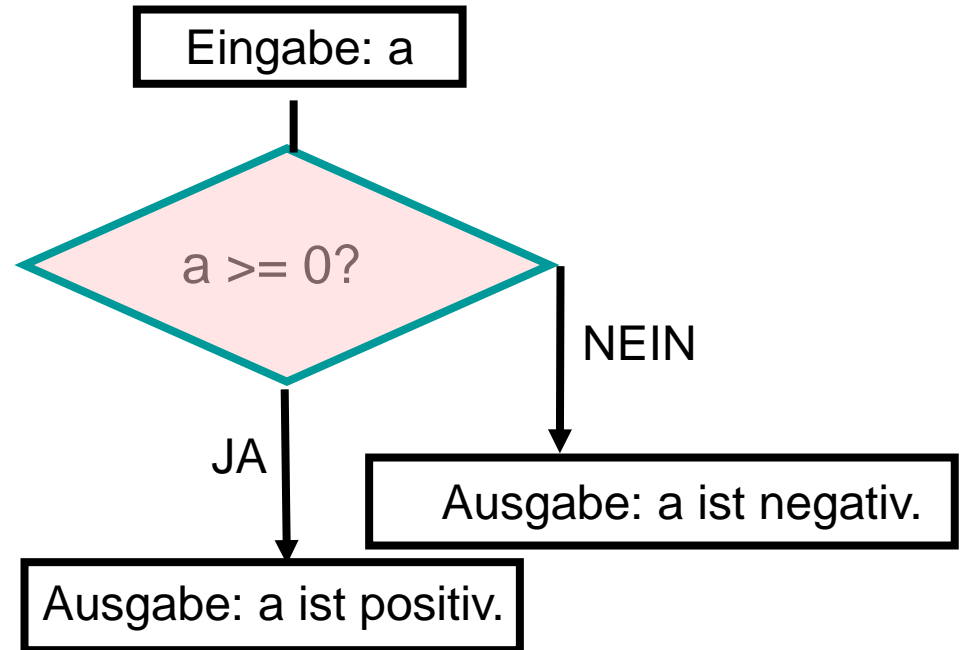


# Flussdiagramm: Verzweigung

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



# Flussdiagramm: if/else- Beispiel

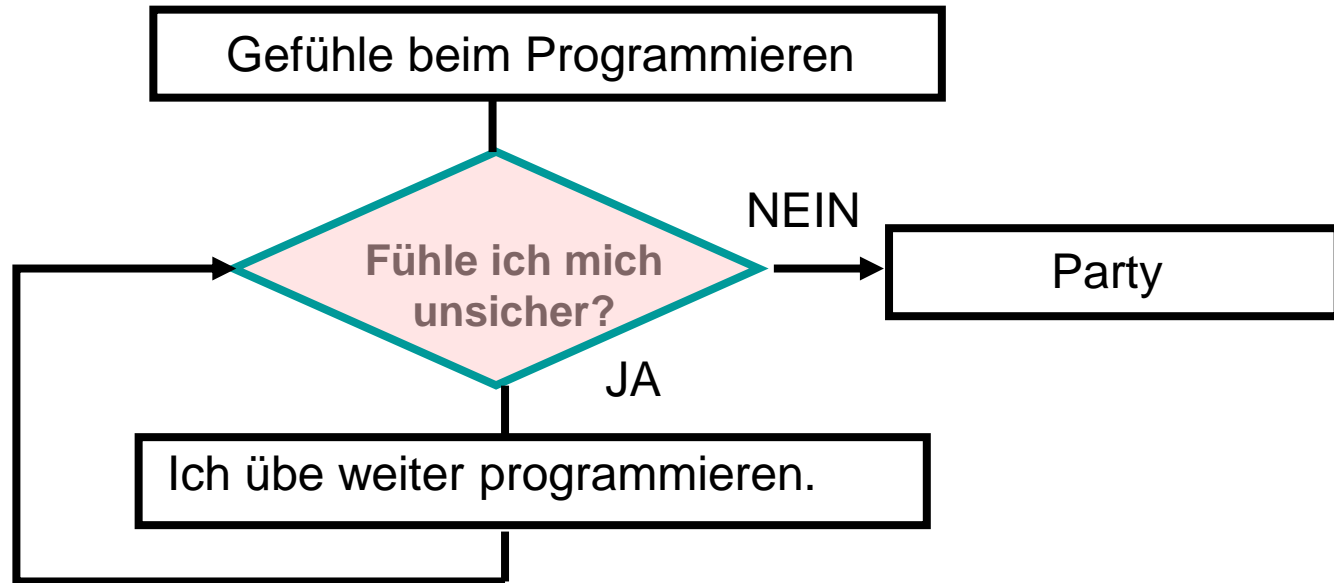


Der Code dafür:

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

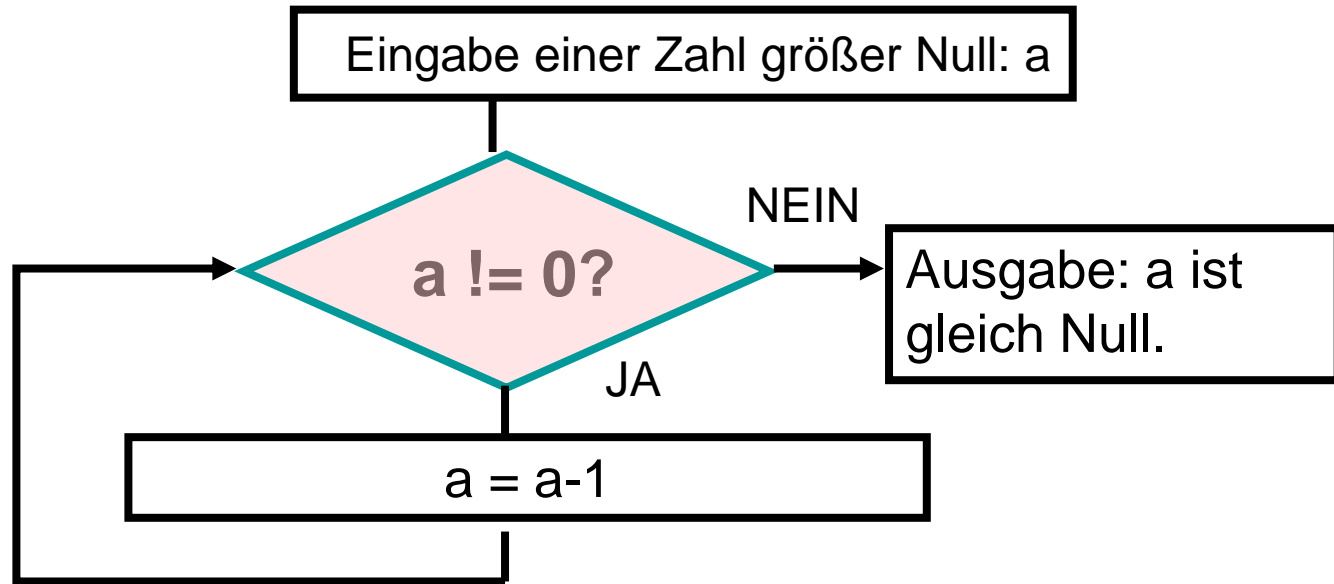


# Flussdiagramm: Schleife - Beispiel



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

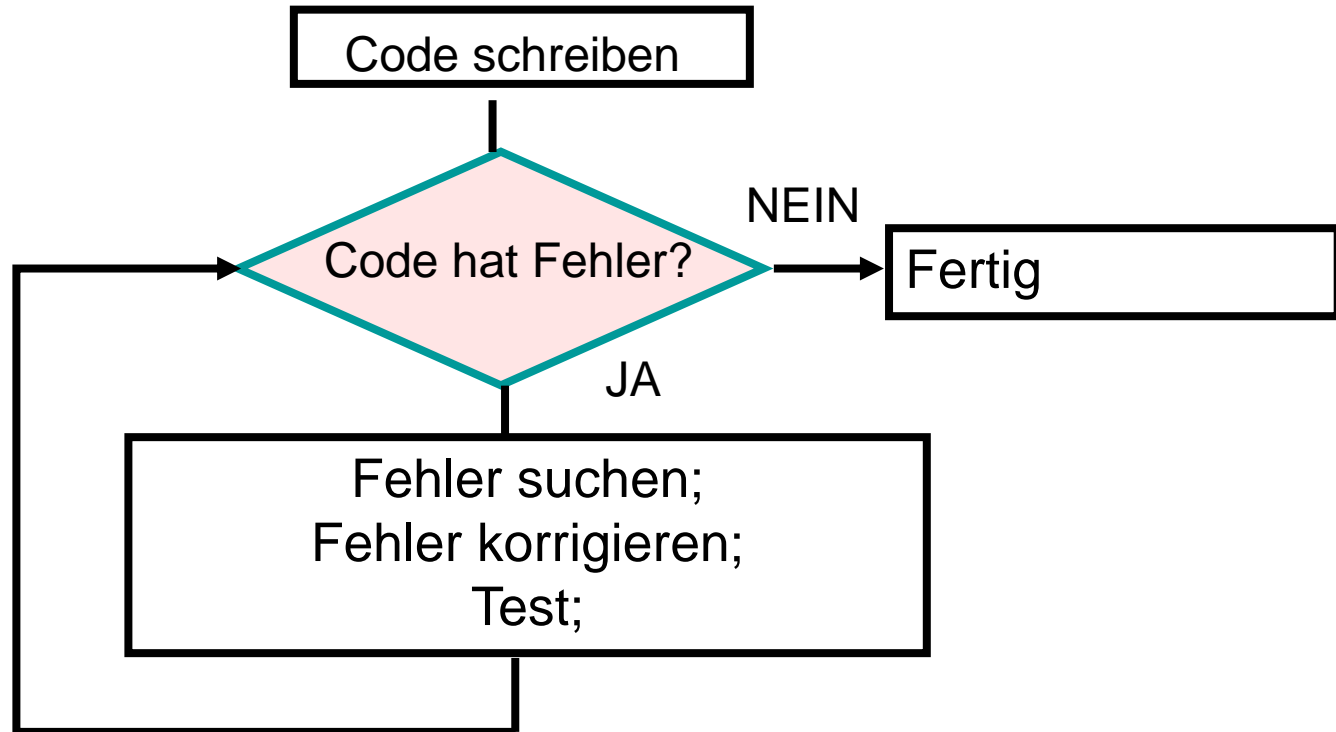
# Flussdiagramm: Schleife - Beispiel



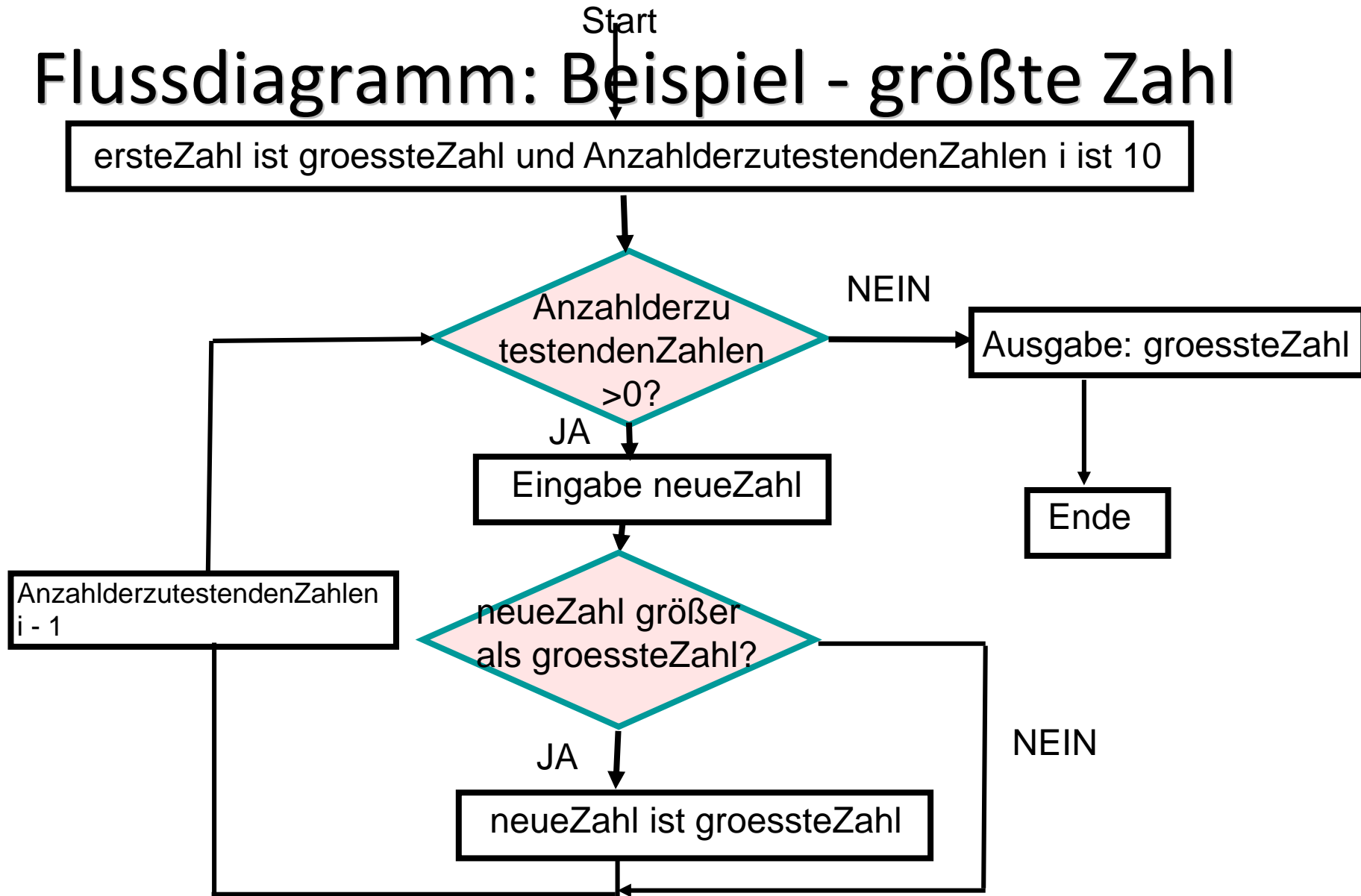
Der Code dafür:

```
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: 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

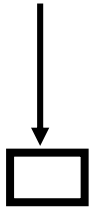
Start



```
#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;
}
```

Start



Anzahl der zu testenden Zahlen i ist 10, erste  
eingegebene Zahl ist groessteZahl,



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);
    i=i-1;

    return 0;
}
```

Start



Anzahl der zu testenden Zahlen i ist 10, erste  
eingegebene Zahl ist groessteZahl,  
i=i-1



Ende

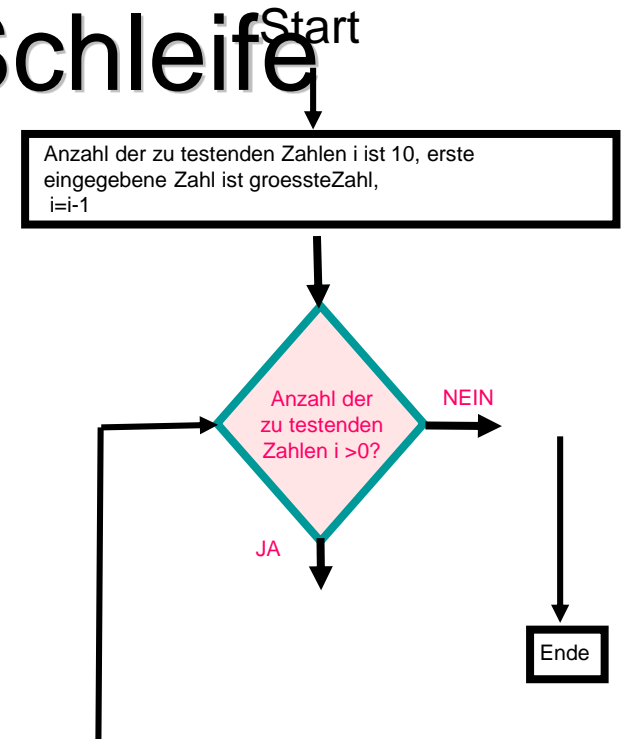


# Größten Zahl - while-Schleife

```
#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;
}
```

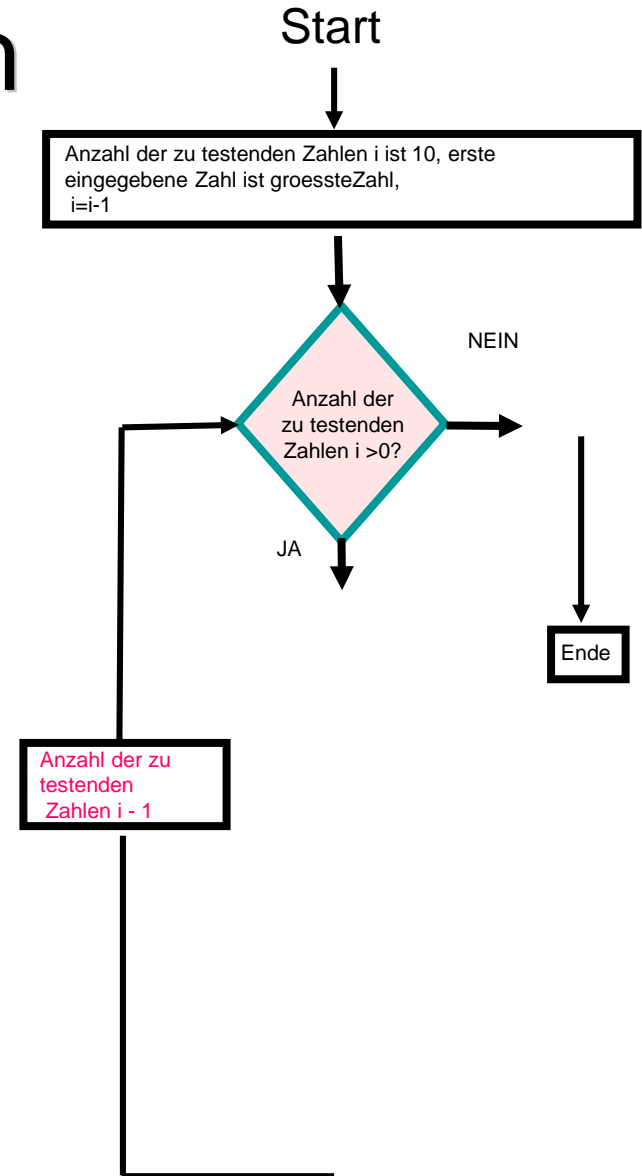


# Größte Zahl - Abbruch

```
#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;

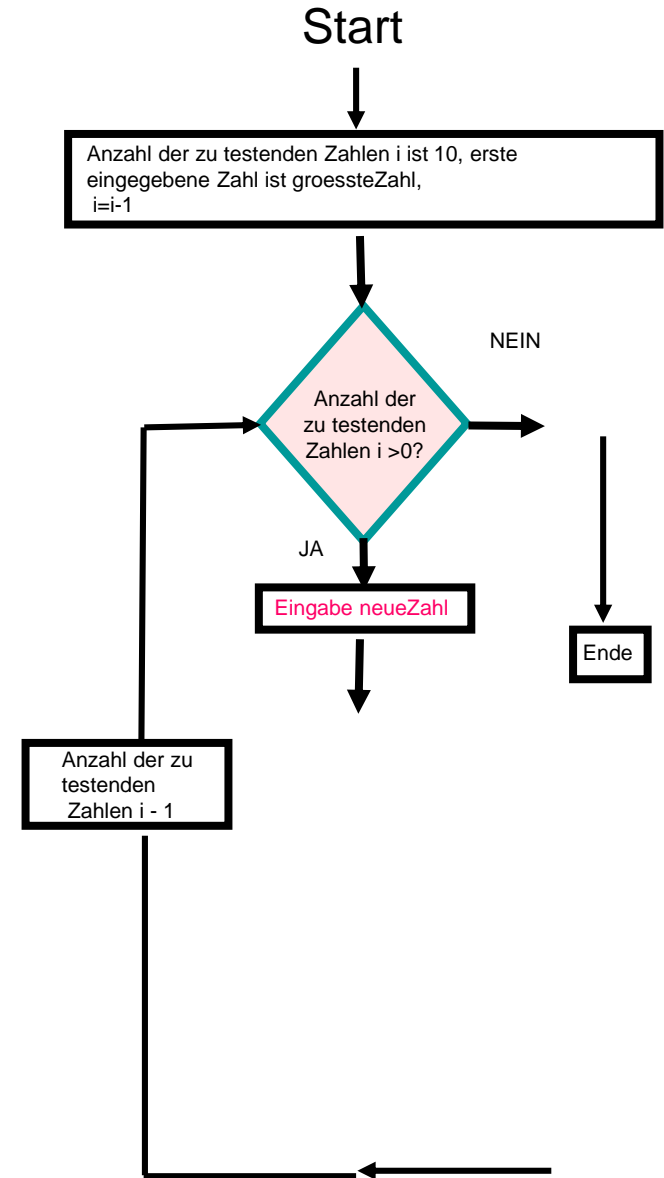
    }
    return 0;
}
```



# 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;
}
```



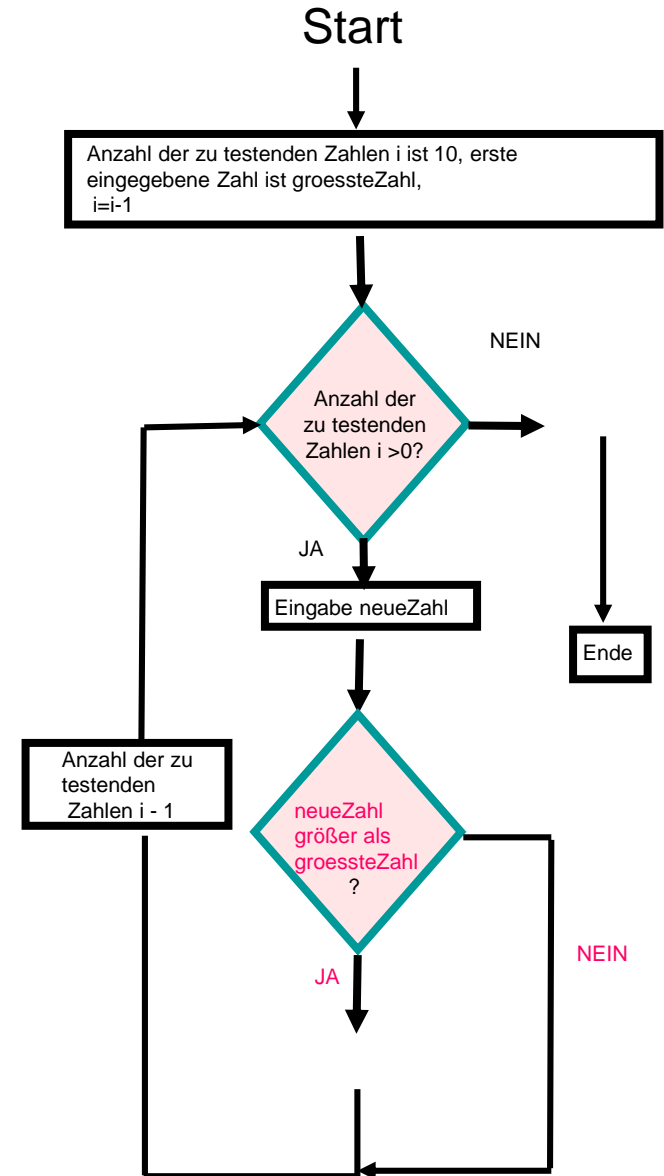


# Größte Zahl - if/else

```
#include <stdio.h>

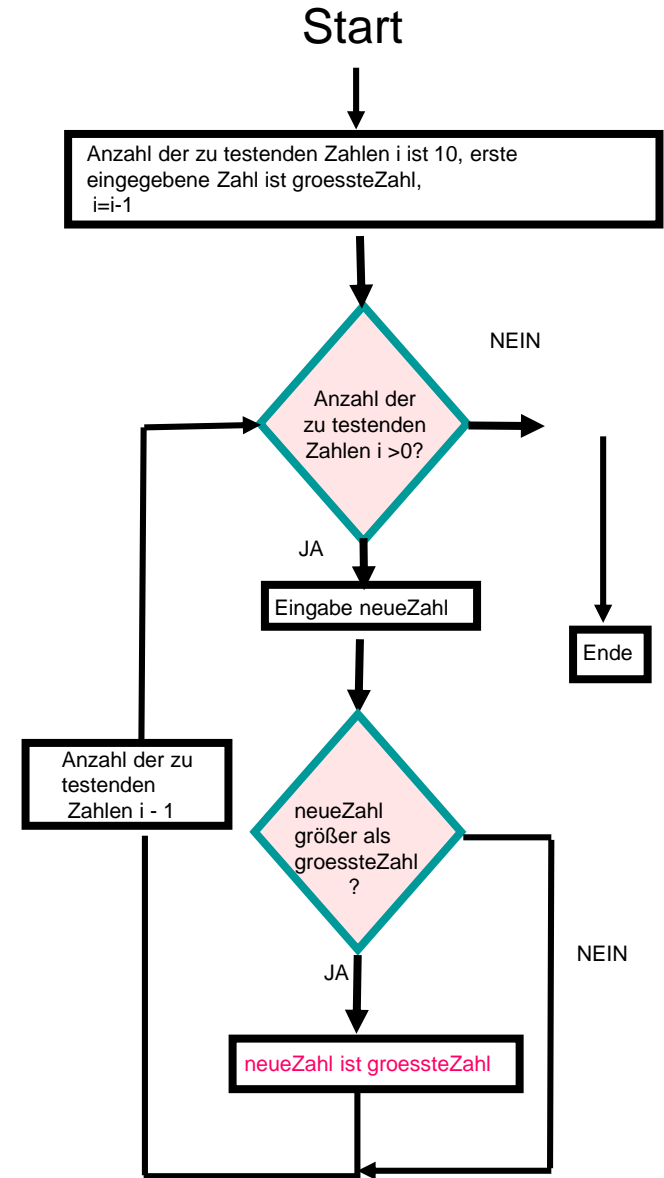
int main() {
    //Deklaration
    int neueZahl;
    int groessteZahl;
    int i=10;
    //Eingabe
    scanf("%i",&groessteZahl);
    i=i-1;
    while ( i > 0 ) {
        //Eingabe
        scanf("%i",&neueZahl);
        if(neueZahl > groessteZahl){

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



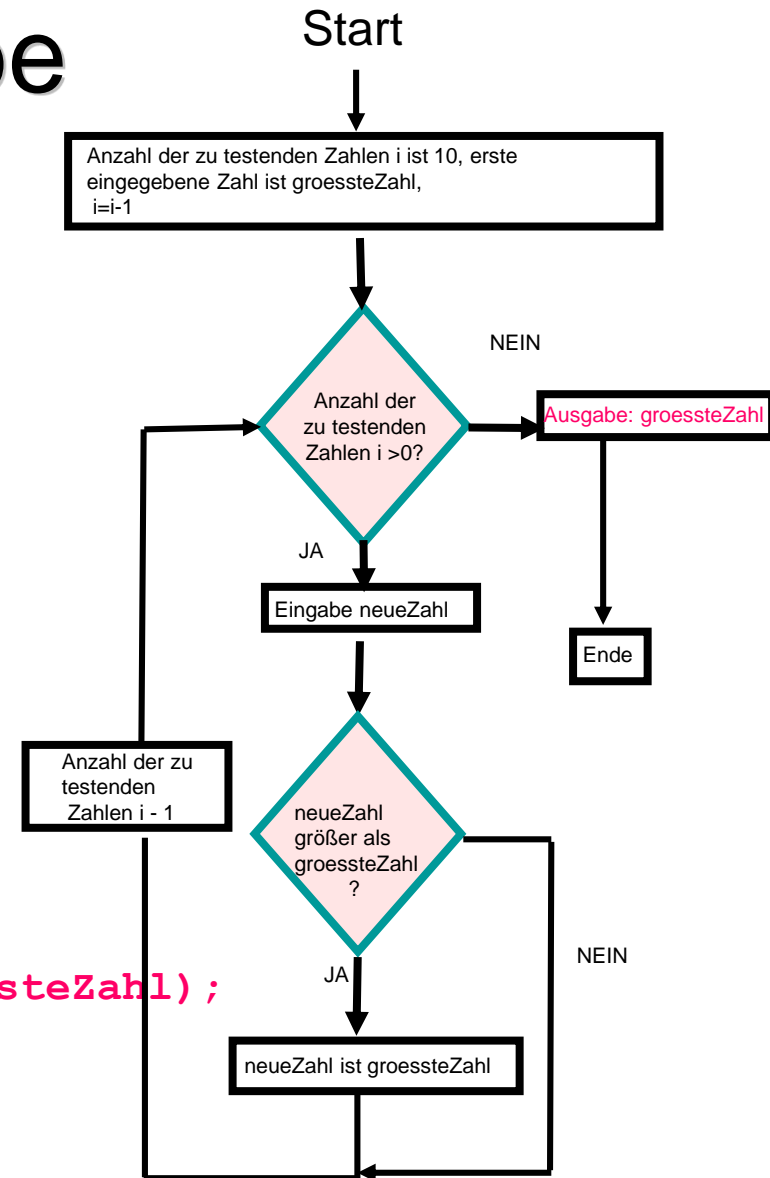
# Größte Zahl

```
#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;
}
```



# Übung

Malen Sie ein Flussdiagramm zur Lösung eines selbstausgedachten Problems aus dem Alltag.

# Flussdiagramm: Zusammenfassung

- ... ist grafische Darstellung eines Algorithmus
- Es gibt zwei Symbole (Rhombus und Kasten) für Fragen (Verzweigungen und Schleifen) und Anweisungen
- Man bleibt nur in einer Schleife, wenn die Frage mit Ja beantwortet wird
- Schleife erkennt man daran, dass alle Spitzen des Rhombus besetzt sind
- Bei Kästen geht ein Pfeil ran 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=OR1I\\_xerHsk](https://www.youtube.com/watch?v=OR1I_xerHsk)



Coco startet mit der C Programmierung Teil 3 - Das Flussdiagramm



# Flussdiagramm: Übung

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





# 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>



# 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.

# Übung: Was wird hier ausgegeben?

```
class Meinproblem{
    public static void main (String[] args) {
        int a;
        a=1;
        int ergebnis;
        int b=6;
        if (b<5) {ergebnis = a+42;}
        else {ergebnis = b-3;}
        System.out.println("Ergebnis: "+ergebnis);
    }
}
```

# Übung: Was ist der Unterschied zwischen Java und Deutsch?



# 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.

# 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;  
}
```



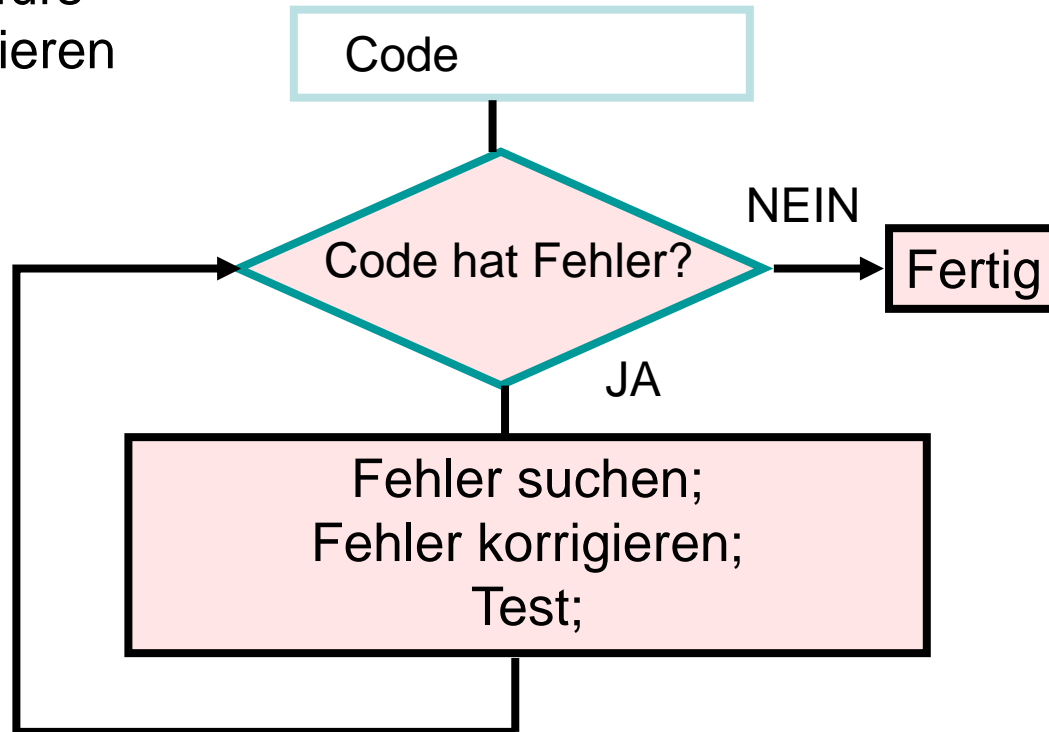
# Laufzeitfehler ...

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

```
brrot=1;  
if(eier>0) brrot=6;
```

# Flussdiagramm: Beispiel

Hilfsmittel fürs  
Programmieren



# Schreiben eines Java-Programms

```
class meinproblem {  
    public static void main(String [] args){  
        System.out.println("Ich habe heute gute Laune.");  
    }  
}
```

1. Text in irgendeinen Editor schreiben und abspeichern als:  
meinproblem.java
2. Compilieren, z.B. von der Kommandozeile aus:  
javac meinproblem.java
3. Ausführung auch z.B. von der Kommandozeile aus:  
java meinproblem
4. Ausgabe auf der Kommandozeile:  
Ich habe heute gute Laune.

# Schreiben eines NXC-Programms

```
task main () {  
    OnFwd(OUT_A,60); //Losfahren mit Motor A  
    Wait(25); //eine Zeit lang fahren  
    Off(OUT_A); //Stoppen von Motor A  
    TextOut(0,50,"Hello World !");  
}
```

1. Text in speziellem Editor schreiben und abspeichern als:  
**meinprogramm.nxc**
2. Compilieren durch Klicken auf das Zahnradsymbol
3. Rüberladen auf den NXT-Computer via USB durch Klicken auf das blaue Dreieckssymbol
4. Test durch Start des Programms auf dem NXT

# Literatur

- Film gesponsert von der Vize-Präsidentin für Lehre, Studium und Weiterbildung Iris Groß, gemacht von Elisa Metze in meinem Auftrag:  
<https://www.youtube.com/watch?v=-7Xxokla4UU>
- Kathy Sierra und Bert Bates: „Java von Kopf bis Fuß“, O'Reilly
- Judi Bishop: „Java“