



# Informatik 1

## Einführung

von  
**Irene Rothe**

Büro B 241

[irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de) (Antwort in 24 Stunden, wenn keine Out of Office email kommt)

Instagram: irenerothedesign



<https://youtu.be/d3xIkpQpdoY>

# Ihre Ziele, Ihre Motivation

- Welche Erwartungen haben Sie an das Modul?
  - Welches Vorwissen haben Sie? Auf welche Fragen möchten Sie Antworten finden?
  - Können Sie Erfahrungen einbringen? Welche?
  - Was haben Sie vor zu tun, damit Sie von dem Lehrangebot profitieren?
- Bitte beantworten Sie diese Fragen *schriftlich* anonym oder mit Namen (10 Minuten)

# Motivation für die Informatik

- Top 10 der Programmiersprachen (IEEE):  
<https://spectrum.ieee.org/top-programming-languages-2022/ieee-spectrums-top-programming-languages-2022>
- 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.

# Was ist Informatik? Teil 1



Was ist Informatik?

<https://www.youtube.com/watch?v=y80yQEQENZ0>



# Was ist Informatik? Teil 2



<https://youtu.be/a-sx2FnyUVs>

Informatik =  
Lösen von Problemen mit dem Rechner

Was braucht man dafür?  
→ Algorithmus

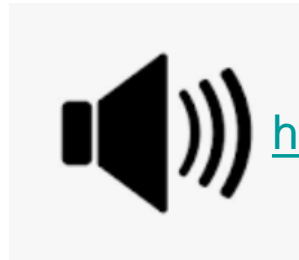
# Hauptziel der Veranstaltung

Sie schreiben Code, der

- niemanden umbringt,
- niemanden verletzt und
- niemanden in den Wahnsinn treibt, also gut lesbar (verständlich) ist.
- und Sie bestehen die Klausur!

Beispiele für schlechten Code:

- Flugzeugabsturz
- Zu viel Röntgenstrahlung
- Server reagiert nicht
- VW ruiniert?



<https://youtu.be/62hHeeahGCM>

# Mein Teaching Style 1

„Ich möchte nicht zu den Studierenden, sondern mit ihnen sprechen.“

(ALEXANDER VON HUMBOLDT)



# Mein Teaching Style 2

- Ich möchte Sie unterstützen und Ihnen helfen, wo ich kann. Sprechen Sie mich an!
- Ein Lehrender will die Menschen weiterbringen. Das will ich auch.
- Ich wäre gern: Ihr Coach, Ihr Motivator, Ihr Unterstützer
- Meine Methode: learning by doing, Lernen aus Fehlern
- Ich würde gerne Ihre Namen lernen!
- Wer keinen Laptop oder Zugang zu einem Computer hat, mir bitte eine email schreiben an [irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)



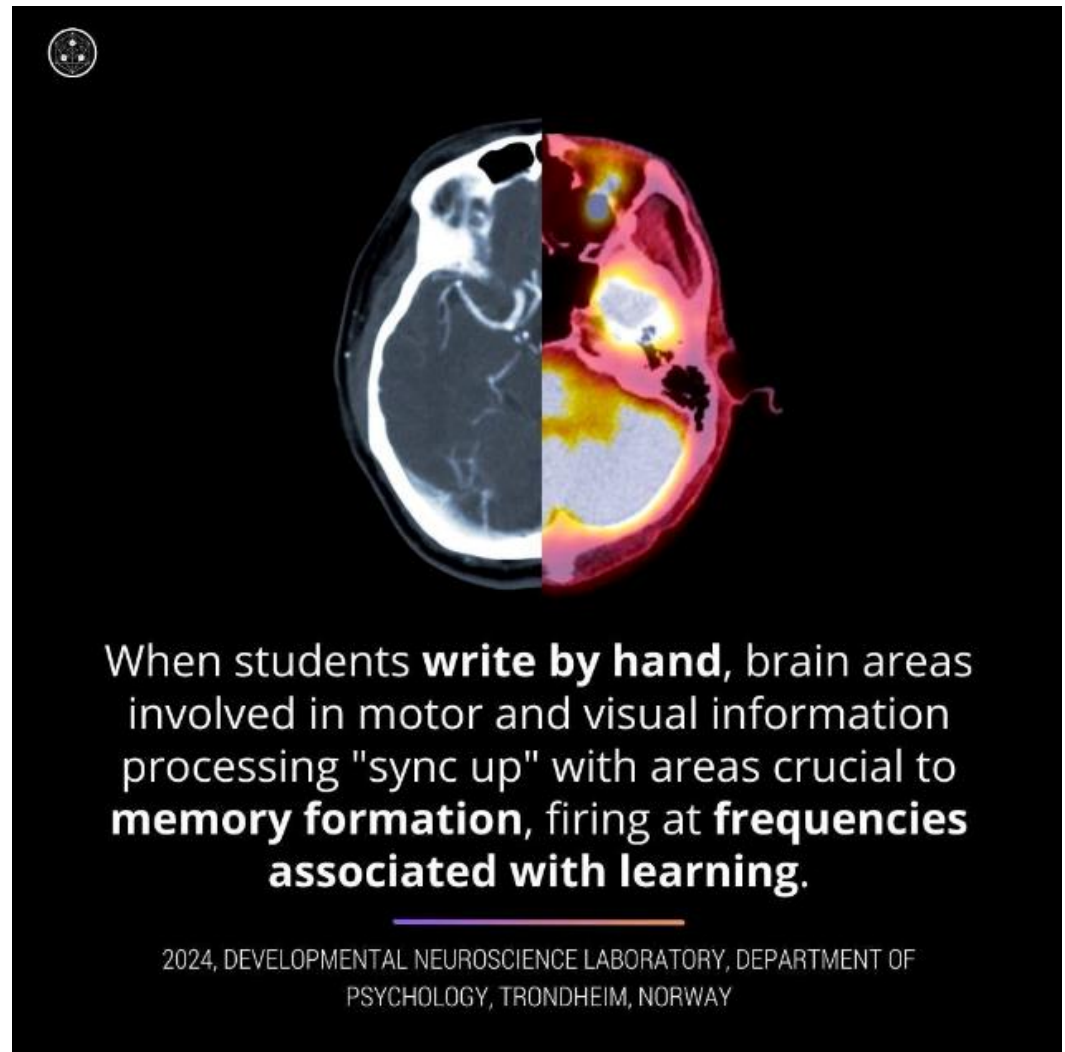
# Mein Teaching Style 3

- **Per Hand** mitschreiben ist gut für die Gehirnaktivität (<https://www.welt.de/wissenschaft/article252589950/Per-Hand-oder-Tastatur-So-wirkt-sich-Schreiben-auf-das-Gehirn-aus.html>)
- **Bildschirme** nur benutzen im Zusammenhang mit der Veranstaltung, ansonsten bitte vor die Hörsaaltür gehen, um andere **nicht** abzulenken
- **Handy nicht in der Nähe** haben ist gut für bessere kognitive Leistung → Handybenutzung nur im Zusammenhang mit Pingo benutzen, alles andere bitte vor oder nach der Vorlesung erledigen (Handy am besten in die Tasche tief unten verstecken, damit die Verführung nicht so stark ist, siehe SkoSeiLin 2023: Skowronek, Seifert, Lindberg: Die bloße Anwesenheit eines Smartphones verringert die basale Aufmerksamkeitsleistung. <https://www.nature.com/articles/s41598-023-36256-4>, zuletzt geprüft am 23.09.2023)







# Lernen für Klausuren

- Per Hand schreiben
- Handy nicht in der Nähe haben
- Auf Rechner nur Compiler und Vorlesungsfolien benutzen



# Design der Folien

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

# Hilfen: Foren

Bekanntestes Forum: **Stackoverflow**

Beim Suchen:

- Einfach Ihr Problem als Frage in Deutsch oder Englisch formuliert in Google (findet garantiert Stackoverflow als erstes) oder direkt bei Stackoverflow eingeben.
- Immer Freunde, Kommilitonen oder Dozenten wie mich fragen, alles andere wäre Zeitverschwendung
- Etwas hartnäckig sein, bei ungewöhnlichen Problemen kann das auch mal Tage dauern

# Hilfen: Foren

Bekanntestes Forum: **Stackoverflow**

Beim Suchen:

- Einfach Ihr Problem als Frage in Deutsch oder Englisch formuliert in Google (findet garantiert Stackoverflow als erstes) oder direkt bei Stackoverflow eingeben.
- Immer Freunde, Kommilitonen oder Dozenten wie mich fragen, alles andere wäre Zeitverschwendung
- Etwas hartnäckig sein, bei ungewöhnlichen Problemen kann das auch mal Tage dauern

# Weitere Hilfen

- Nachdenken über Ordnung auf Rechner oder Stick, z.B. können wenige Unterordner besser sein als viele, Nachdenken über sinnvolle Dateinamen (nicht *Vorlesung1* – diese Info nutzt einem i.R. später wenig, besser *VorlesungMitOrdnungstipps*)
- Backups regelmäßig machen
- *readme.txt* Dateien anlegen mit Infos, die man immer wieder braucht und/oder vergisst
- Emails, bei denen man auf eine Antwort wartet, an sich selbst zur Erinnerung schicken
- Arbeitsplan anlegen
- Auf Rothes Informatik1 Webseite unter Hilfreiches:
  - ProgrammierenAlleinzuHause: Schritt für Schritt Anleitung
  - Lerntipps
- Benennen von Bottlenecks ([www.decodingthedisциплиnes.org](http://www.decodingthedisциплиnes.org)), Formulieren von Fragen



# Feedback

Bitte melden Sie sich in meinem Büro oder per email ([irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)),

- wenn etwas Falsches in meinen Folien steht oder ich etwas Falsches drauf gesprochen habe
- wenn Sie Probleme haben
- wenn Sie eine Idee haben, wie ich etwas besser machen kann

→ Für fast alles gebe ich Bonuspunkte für die Klausur Informatik 2!!!



# Ganz Wichtig

Ich antworte auf emails an

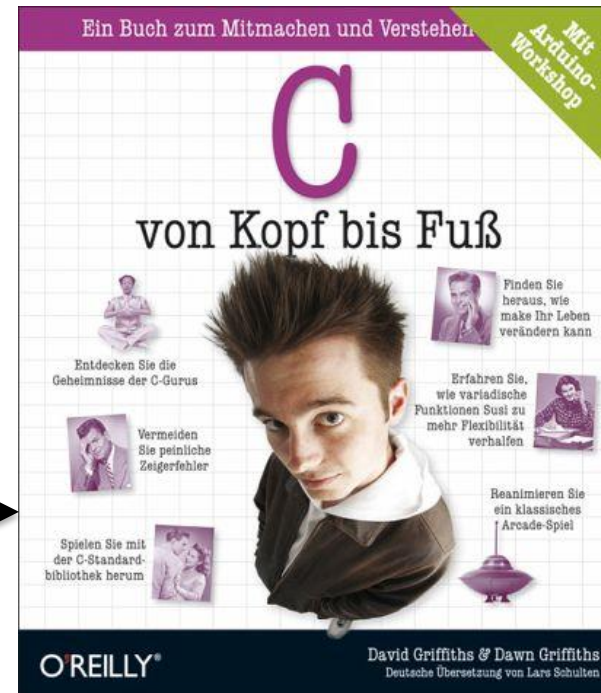
[irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)

innerhalb von 24 Stunden. Wenn ich das nicht tue und Sie keine „Out of office“-email erhalten,

schicken Sie Ihre email noch einmal, dann habe ich sie übersehen.

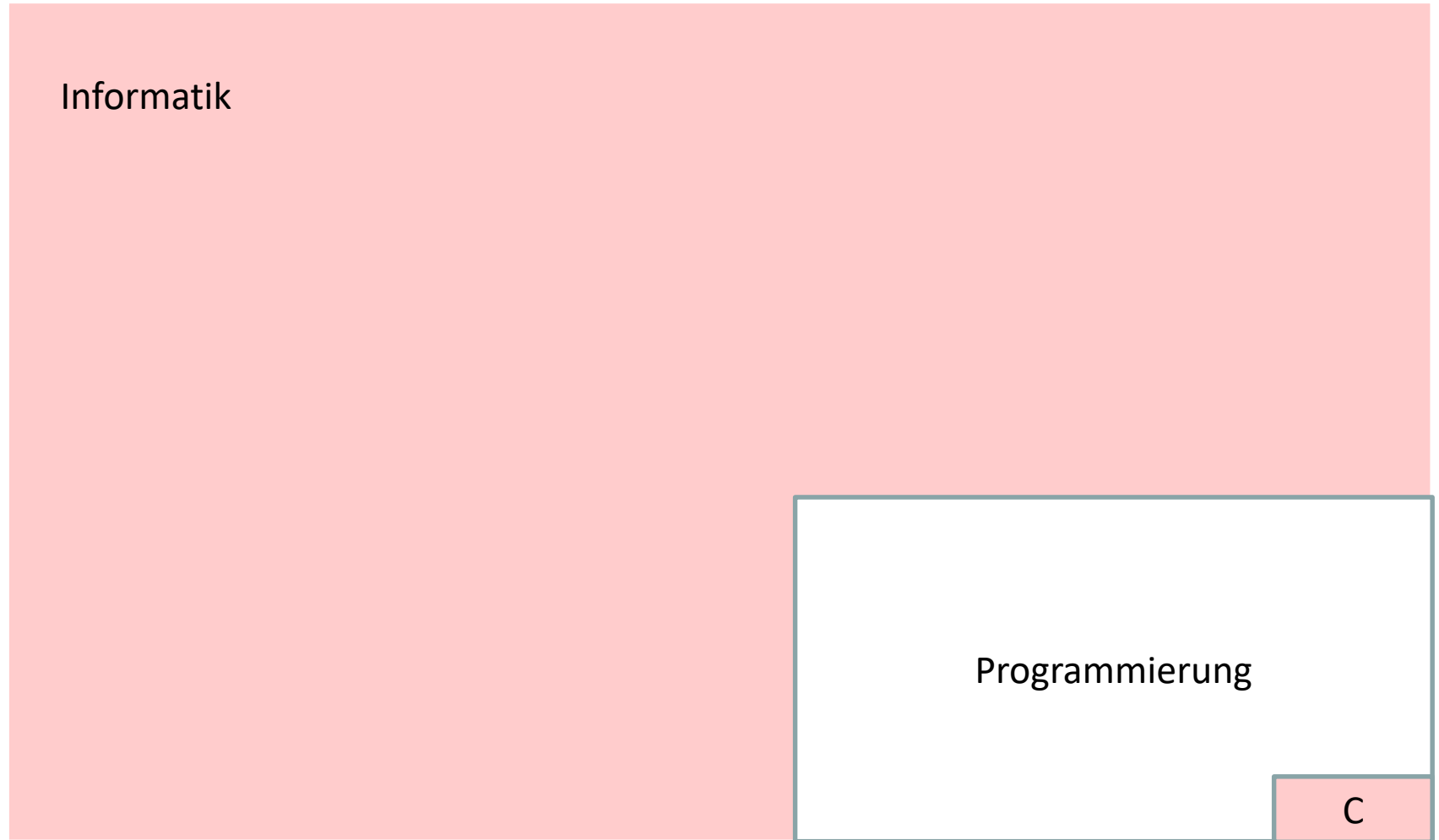
# Literatur

- Der Klassiker:  
Kernighan/Ritchie:  
**Programmieren in C**,  
Hanser
- Griffiths&Griffiths:  
**C von Kopf bis Fuß**,  
O'Reilly



- Leider nicht mehr verfügbar: Kurz und knapp, liegt gut vor einem auf dem Tisch: "Programmieren in C" RRZN-Handbuch → erhältlich in der Bibliothek für 3,70 €
- "C-Howto" Elias Fischer

# Themengebiete der Informatik



# Themengebiete der Informatik

Informatik			
Technische	Praktische	Theoretische	Angewandte
Hardware-Komponenten	Algorithmen Datenstrukturen	Automatentheorie	Computergrafik
Schaltnetze, Prozessoren	Programmiersprachen	Komplexitätstheorie	Datenbanken
Mikroprogrammierung	Betriebssysteme	Berechenbarkeit	Künstliche Intelligenz
Rechnerorganisation/-architektur	Mensch-Maschine-Kommunikation	Formale Sprachen	Digitale Signalverarbeitung
Rechnernetze	Verteilte Systeme	Formale Semantik	Simulation & Modellierung



# Themengebiete der Informatik 1 und 2

Informatik			
Technische	Praktische	Theoretische	Angewandte
Hardware-Komponenten	Algorithmen Datenstrukturen	Automatentheorie	Computergrafik
Schaltnetze, Prozessoren	Programmiersprachen	Komplexitätstheorie	Datenbanken
Mikroprogrammierung	Betriebssysteme	Berechenbarkeit	Künstliche Intelligenz
Rechnerorganisation/-architektur	Mensch-Maschine-Kommunikation	Formale Sprachen	Digitale Signalverarbeitung
Rechnernetze	Verteilte Systeme	Formale Semantik	Simulation & Modellierung

Bemerkung: zu allen rosa Felder, werden Sie etwas in dieser Vorlesung erfahren



# Informatik 1+2

**Vorlesung:** wöchentlich inklusive Übungen,  
Fragenbeantwortung

**Praktikum:** wöchentlich, alle Aufgaben müssen gelöst  
sein bis zum Ende des Semesters um Testat zu erhalten  
als Zulassungsvoraussetzung für die Klausur

**Projektwoche:** 3 halbe Tage als Zusatzangebot: Arbeiten  
im Team an einer selbstgewählten Programmieraufgabe



# Übung: Brief an Euch selbst schreiben

Schreibt an Euch selbst einen Brief mit folgendem Inhalt:

- welche Maßnahmen nehmt Ihr Euch vor für die Veranstaltung?
- Welche Aktivitäten möchtet Ihr durchführen?
- Was wollt Ihr unternehmen, wenn Ihr Schwierigkeiten feststellt?

→lest den Brief einmal im Monat durch



# 5 Beispiele für Anwendungen

- Skype
- Bildverarbeitung
- Einkaufen im Internet
- Kryptografie
- Linux



# Compiler für C89 (mehr oder weniger)

Um zu programmieren, brauchen Sie einen Compiler, welches ein Programm ist, das Programmcode in Maschinencode übersetzt, den Ihr Rechner versteht und ausführen kann.

Kostenlose Compiler:

- **Meine Empfehlung: Dev-C++** (ist total veraltet, wird nicht mehr gepflegt, existiert portable – kann also auf Stick kopiert werden, und man kann überall auch ohne Internet programmieren): <http://sourceforge.net/projects/orwelldvcpp/>
- Code::Blocks: siehe <https://www.youtube.com/watch?v=clx8HthehKs>



Coco startet mit der C Programmierung Teil 1 - Der Compiler

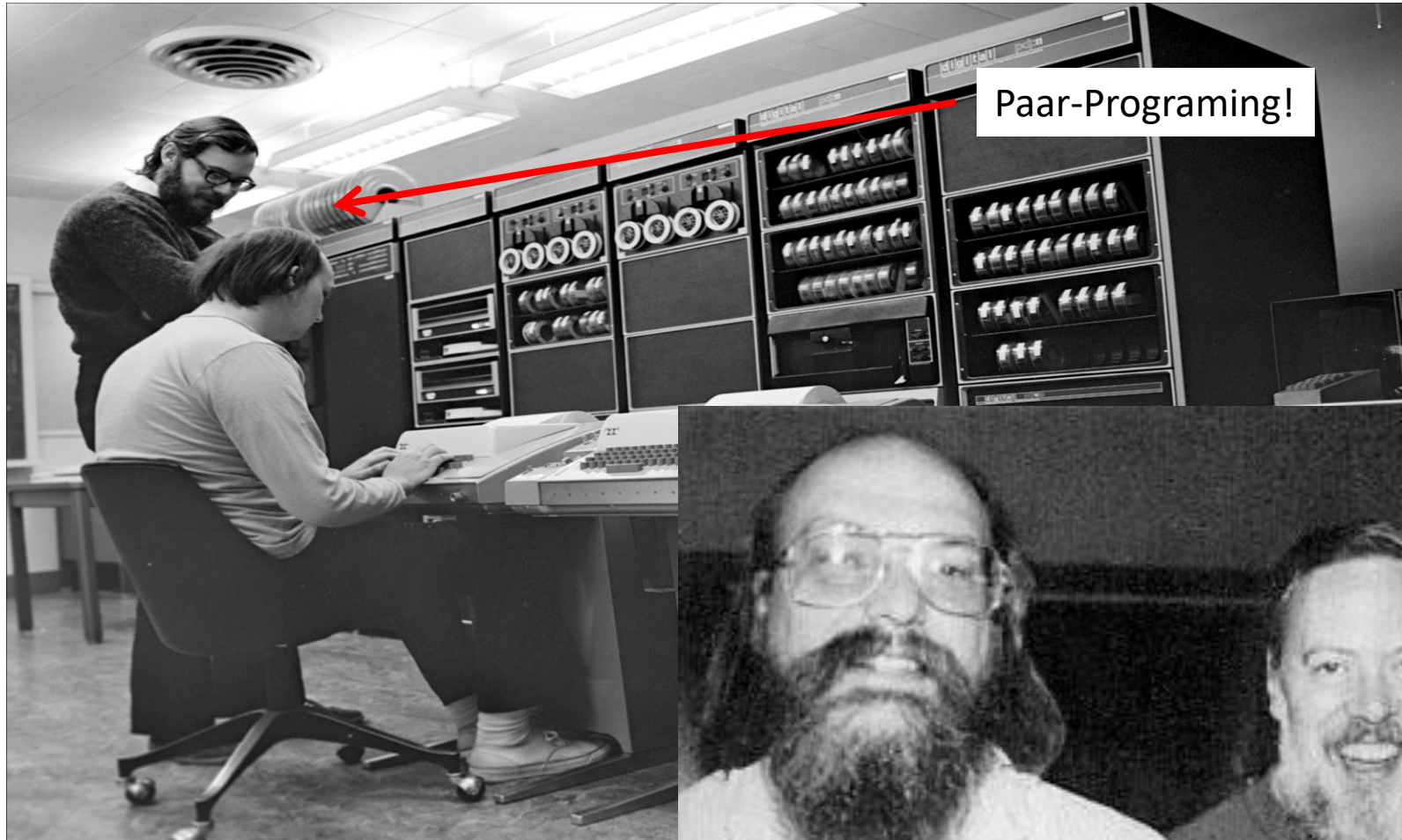
- Visual Code: sehr modern, sehr chic, aber man muss C selbst installieren
- gcc + Eclipse
- Für kleine schnelle Tests: <https://www.onlinegdb.com>
- jeder beliebige andere C-Compiler

**Achtung:** Egal womit Sie arbeiten, legen Sie keine Projekte an, sondern wählen Sie immer:

File→Neu→SourceFile oder Quelldatei



# Etwas Geschichte: Ken Thompson and Dennis Ritchie



([http://commons.wikimedia.org/wiki/File:Ken\\_Thompson\\_%28sitting%29\\_and\\_Dennis\\_Ritchie](http://commons.wikimedia.org/wiki/File:Ken_Thompson_%28sitting%29_and_Dennis_Ritchie))  
Urheber: Peter Hamer; Lizenz: CC BY-SA 2.0)

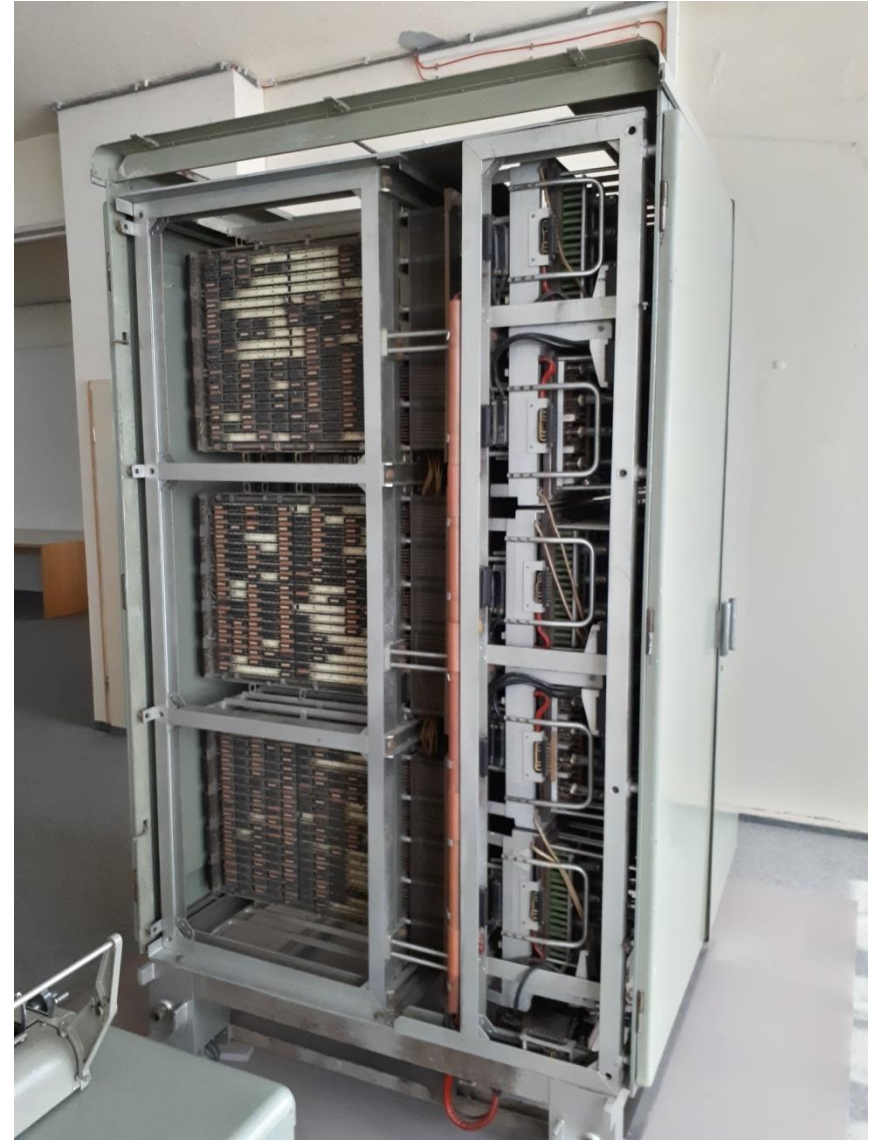


Hochschule  
Bonn-Rhein-Sieg

Vorlesung

# Etwas Geschichte: Erster Rechner der DDR: ZRA

- von VEB Carl Zeiss
- Halbleiter-Dioden und Elektronenröhren
- Speicher: 4096 48-Bit-Worte
- Eingabe: Lochkartenleser
- Ausgabe: Druckwerk
- Platz:  $6 \times 8 \text{ m}^2$
- Strukturell: von-Neumann-Architektur (gemeinsamer Programm- und Datenspeicher)
- acht Schnellspeicher
- Befehle der CPU: wie heute (Holen und Abspeichern von Daten im Hauptspeicher und den Prozessorregistern, Grundrechenarten, Fest- und Gleitkommazahlen, logische Operationen: Konjunktion und Disjunktion sowie Schiebeoperationen)
- 120 FLOPS (2005 3 Milliarden Flops)



Die nächste Folie zeigt alle Themen, die ich in zwei Semester Informatik behandeln möchte.

- Ich gehe nicht chronologisch vor, sondern hüpfte zwischen den Themen hin und her.
- Dabei sind immer die Themen umrandet, die ich in diesem geöffneten Foliensatz behandelt werde.
- Haken sind vor Themen, die bereits in vergangenen Foliensätzen behandelt wurden, deren Inhalte also vorausgesetzt werden.

# Informatik: 2 Semester für Ingenieure

→ 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



### Aufbau der einzelnen Themen:

- 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.

# Wozu muss ich Programmieren lernen?

Ingenieure sind immer an allem Schuld.

# 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>)





# Einschub: Wozu muss ich Programmieren lernen?

## → Zum Gedichte schreiben: Perl Poetry

```
#!/usr/bin/perl                                write me if-you-please;

APPEAL:                                         sort your feelings, reset goals, seek (friends, family,
                                              anyone);

listen (please, please);

open yourself, wide;
    join (you, me),
connect (us,together),

tell me.

do something if distressed;

    @dawn, dance;
    @evening, sing;
    read (books,$poems,stories) until
peaceful;
    study if able;

                                              accept (yourself, changes),
                                              bind (grief, despair);

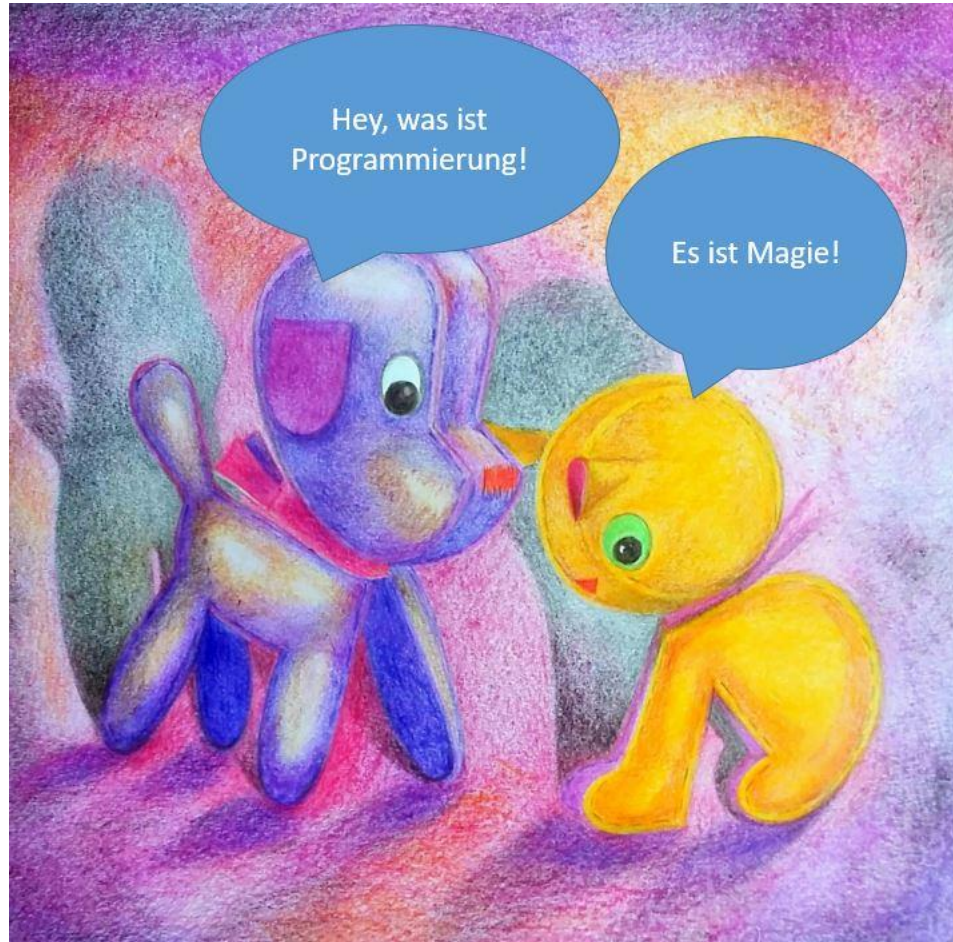
require truth, goodness if-you-will, each moment;

select (always), length(of-days)

# listen (a perl poem)
# Sharon Hopkins
# rev. June 19, 1995
```



# Was ist Programmieren?



# Einschub: Was ist Programmieren?



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

# Motivationsbeispiel

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



# 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 Programm 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? Wann hören Sie nicht 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)

# Wie finden wir einen Algorithmus?

- Suche nach der HOLZHAMMERLÖSUNG (einfachste Lösung)
- Suche nach einem besseren = effizienteren Algorithmus (hierbei kann der Holzhammeralgorithmus gut zum Testen dienen)
  - effizient in der Zeit
  - Algorithmus arbeitet gut mit tatsächlichen Daten, zufälligen Daten und extremen Daten

# Programmiersprache

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

# Einschub: Programmiersprache C

- darin wurde UNIX programmiert, der Linux/Android Kernel
- designed for experienced programmers with the power to do whatever needs to be done
- this includes the power to hang yourself by invalid pointer references and array-out-of-bound violations
- am Ende von 2 Semestern wissen Sie, was die 2 obigen Sätze bedeuten.

# Programmieren

- ...ist das Erlernen einer Sprache mit **Syntax** (Wörtern, Punkt, Komma, ...) und **Semantik** (Bedeutung)
- Programm besteht aus
  - einem Anfang,
  - einem Ende und
  - dem Algorithmus dazwischen.
- Alles hat einen Anfang: In der Programmiersprache C sieht der Anfang wie folgt aus (2 Zeilen, die später noch genau erklärt werden):

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

- Und das Ende wie folgt (2 Zeilen, die später noch genau erklärt werden):

```
    return 0;
}
```

# Ein Programm...

1. Kann **Eingabe/n** bekommen
2. Macht irgendwas (**Algorithmus**), z.B.  
irgendwas mit den Eingabe/n
3. Und reagiert durch **Ausgabe/n**

# Programme...

- Werden in einem Schreibprogramm (Editor) geschrieben
- Sollten am Anfang folgende Informationen als Kommentare (//) haben
  1. Name der Programmierer
  2. Was es macht
  3. Testfall/Was soll passieren (wenn sinnvoll)

# Das erste C-Programm - Ausgabe

```
//Irene
//Ausgabe
//Was soll passieren: Hallo Welt!
#include <stdio.h>
int main() {
    printf("Hallo Welt!");
    return 0;
}
```

} Anfang des Programms (wird später genau erklärt)

← Ausgabe auf dem Bildschirm

} 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).

1. Geben Sie diesen Text in Ihr Compilerprogramm, z.B. dev-cpp, ein.
2. Übersetzen Sie den Text (Programm) in Maschinensprache (damit nicht nur Sie das Programm verstehen, sondern auch der Rechner) durch Klicken auf z.B. ein kleines gelbes Zahnrad oder Compile oder...
3. Lassen Sie Ihr Programm durch Ihren Rechner ausführen, z.B. über cmd in Windows oder durch Klicken auf einen grünen ausgefüllten Pfeil oder **Run** oder....
4. Es sollte folgende Ausgabe auf dem Bildschirm erscheinen: Hallo Welt!





# Das erste C-Programm - Erklärung

- Programme werden von oben nach unten abgearbeitet. Das ist ähnlich wie bei einem Buch, das man auch von Anfang bis Ende liest.
- Dabei beginnt die Abarbeitung bei dem Schlüsselwort **main** (Hauptprogramm), was vergleichbar zur Seite 1 eines Buches ist, wo man in der Regel mit Lesen beginnt.
- Hinter **//** stehen Kommentare, die der Rechner ignoriert.
- Zeilen, wo der Rechner etwas macht, enden mit einem Semikolon

```
#include <stdio.h>           //Anfang
int main() {                 //Anfang, bei main der Abarbeitung
    printf("Hallo Welt!\n"); //Ausgabefunktion, \n - neue Zeile
    printf("Wie geht es?");
    return 0;                //Ende
}                             //Ende
```

# Wozu ist dieses erste Programm nützlich?

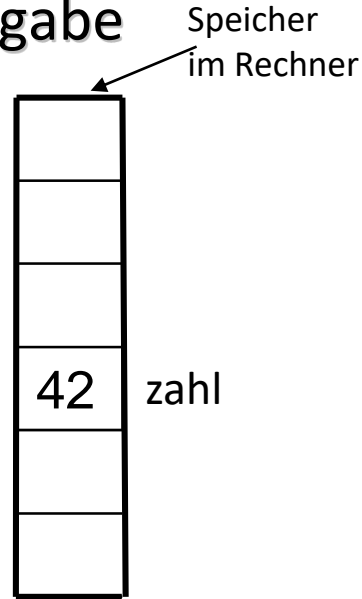
Dieses Programm ist gut, um zu wissen,

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

# Zweites Programm: Speichern einer ganzen Zahl und Ausgabe

```
//Test:Zahl=42
#include <stdio.h>
int main() {
    int zahl=42;
    printf("Zahl=%i", zahl);
    return 0;
}
```

Das hier ist eine Bibliothek, wo die Funktion printf definiert ist.



- Das `int` in der 4. Zeile bedeutet, dass es sich um eine ganze Zahl handelt, die abgespeichert werden soll. So weiß der Rechner (eigentlich das Betriebssystem), wie viel Speicher reserviert werden soll. Dazu später mehr.
- Mit `zahl` wird ein Speicherplatz bezeichnet. Dieser Speicherplatz gehört dem Programm, so lange es läuft. Der Rechner spricht seine Speicherplätze mit Adressen an, aber für den Programmierer ist es einfacher, diese Speicherplätze mit Namen anzusprechen.
- Mit dem Gleichheitszeichen `=` (hat nicht die Bedeutung wie in Mathe) weist man Speicher Werte zu.
- `%i` ist ein Formatierungszeichen, das angibt, in welcher Art der Inhalt des Speichers nach dem Komma ausgegeben wird. Man kann sich hier merken: `i` wie `int` (ganze Zahl)
- Die Ausgabe sieht wie folgt aus: `Zahl=42`

# Drittes Programm - Eingabe einer ganzen Zahl

//Test: 234->Zahl=234

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

```
int main() {
```

```
    int zahl=42;
```

```
    printf("Eingabe einer ganzen Zahl");
```

```
    scanf("%i",&zahl);
```

```
    printf("Zahl=%i",zahl);
```

```
    return 0;
```

```
}
```

& bedeutet Speicheradresse, wo die Zahl hingeschrieben werden soll. Zum &-Operator später mehr.

Funktion aus der Bibliothek `stdio.h` zur Eingabe: `scanf()`



# Programmieren – Wie geht man vor?

1. Testfall
2. Was sind die Eingaben?
3. Was sind die Ausgaben?
4. Algorithmus zwischen Eingaben und Ausgaben implementieren, z.B. Berechnung (rechts vom =)

# Übung: Summe von zwei Zahlen



# Übung: Fehlerprogramm

Finden Sie Syntaxfehler (die Zahlen am Anfang bitte ignorieren):

```
1: #insert <stdio.h>
2: int program(){
3:   int zahl1 = 15;
4:   int zahl1 = 0;
5:   print("Zahl    = %i \n",zahl1);
6:   scanf("%h",zahl2);
7:   printf("Zahl    = %i \n",zahl2)
8:   printf("Summme = %I \n",{zahl1+zahl2});
9:   return 0;
}
```

# Typische Anfangsfehler

Hilfe bei der Fehlersuche: <https://www.youtube.com/watch?v=Ji6bMBDuqKU>



Coco startet mit der C Programmierung Teil 2 - Fehler beim Programmieren



# Ingenieure sind

Informatiker, die keine Angst  
vor Strom haben.

# Begriffe

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

Achtung: **main** darf es nur genau einmal pro Programm geben! Hier fängt die Ausführung des Programmes an.

- **Anweisung:** alles, wonach ein Semikolon steht
- **Schlüsselwörter** (Wörter, die zur C-Sprache gehören): `int`, `return`, `include`

