Rechtlicher Hinweis

Legal notice

Informatik I (B.Sc. Physik)

Zahlendarstellungen

Dr. Paul Bodesheim
(Paul.Bodesheim@uni-jena.de)

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Fakultät für Mathematik und Informatik
Lehrstuhl für Digitale Bildverarbeitung

SoSe 2020

Inhalt

Inhalt

Grundlage des Datenspeichers

- Speicherzellen im Computer speichern einen von zwei Zuständen
- Ein/Aus, Strom/kein Strom, 1/0
- Größe dieses Speicherplatzes: 1 Bit (binary digit)

  $\Rightarrow$ Basiseinheit im binären Zahlensystem
- Werte/Zahlen belegen mehrere Zellen
- 1 Byte = 8 Bit

Speichereinheiten

Byte:	1 Byte	=	8 Bit	
Kilobyte:	1 kB	=	1024 Byte	
Megabyte:	1 MB	=	1024 kB	$(= 1024 \cdot 1024$ Byte$)$
Gigabyte:	1 GB	=	1024 MB	
Terabyte:	1 TB	=	1024 GB	
(Petabyte:	1 PB	=	1024 TB)	
(Exabyte:	1 EB	=	1024 PB)	

Schnelle Umrechnung mittels Approximation: $2^{10} = 1024 \approx 1000 = 10^3$

Inhalt

Stellenwertsystem am Beispiel der Dezimalzahlen

- Üblich im Alltag: Dezimalzahlen mit Ziffern $0 - 9$
- **Stellenwertsystem**:
  Position (Stelle) einer Ziffer ist entscheidend für ihren Wert
- Beispiel:

$$\begin{aligned} \mathbf{128} &= \mathbf{1} \cdot 100 + \mathbf{2} \cdot 10 + \mathbf{8} \cdot 1 \\ &= \mathbf{1} \cdot 10^2 + \mathbf{2} \cdot 10^1 + \mathbf{8} \cdot 10^0 \end{aligned}$$

- Dekadisches Zahlensystem: Stellenwertsystem zur Basis 10
- Darstellung einer n-stelligen Zahl z aus Ziffern $a_i \in \{0, \ldots, 9\}$:

$$z = \sum_{i=0}^{n-1} a_i \cdot 10^i$$

- Schreibweise: $z = a_{n-1} a_{n-2} \ldots a_2 a_1 a_0$

Andere Systeme, z.B. römisches Zahlensystem

- **Zahlschrift** bestehend aus:
  - Einzelzeichen mit Grundzahlwert:
    I=1, V=5, X=10, L=50, C=100, D=500, M=1000
  - Regeln zur Schreibweise hinsichtlich Wiederholung, Kombination und Positionierung der Zeichen

I	1	XI	11	XXI	21	XXXI	31	XLI	41
II	2	XII	12	XXII	22	XXXII	32	XLII	42
III	3	XIII	13	XXII	23	XXXIII	33	XLIII	43
IV	4	XIV	14	XXIV	24	XXXIV	34	XLIV	44
V	5	XV	15	XXV	25	XXXV	35	XLV	45
VI	6	XVI	16	XXVI	26	XXXVI	36	XLVI	46
VII	7	XVII	17	XXVII	27	XXXVII	37	XLVII	47
VIII	8	XVIII	18	XXVIII	28	XXXVIII	38	XLVIII	48
IX	9	XIX	19	XXIX	29	XXXIX	39	XLIX	49
X	10	XX	20	XXX	30	XL	40	L	50

- Beispiel: Jahreszahlen / Baujahr von Gebäuden
  - MDLVIII = 1558 (Gründung der FSU)
  - MCMLXXII = 1972 (Eröffnung JenTower)

Allgemeines Stellenwertsystem

- Darstellung einer n-stelligen Zahl im Stellenwertsystem zur Basis b:

$$z = \sum_{i=0}^{n-1} a_i \cdot b^i$$

- z dargestellt mit n Ziffern $a_i \in \{0, \ldots, b-1\}$
- Schreibweise: $z = (a_{n-1} a_{n-2} \ldots a_2 a_1 a_0)_b$
- Dekadisches System, Dezimal-System: $b = 10$

Binäres Zahlensystem

- Nur zwei mögliche Ziffern 0 und 1 für ein **Bit**
  (1 Bit ist eine einstellige Zahl im binären Zahlensystem)
- Darstellung einer n-stelligen Zahl im Stellenwertsystem zur Basis $b = 2$:

$$z = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

- z dargestellt mit n Ziffern $a_i \in \{0, 1\}$
- Schreibweise: $z = (a_{n-1} a_{n-2} \ldots a_2 a_1 a_0)_2$
- Jede natürliche Zahl lässt sich dadurch darstellen
- Binäres, dyadisches oder duales Zahlensystem

Umwandlung in das Dezimal-System

- Von einer binären Zahlendarstellung in das dekadische System
- Direkte Anwendung der Formel:

$$z = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

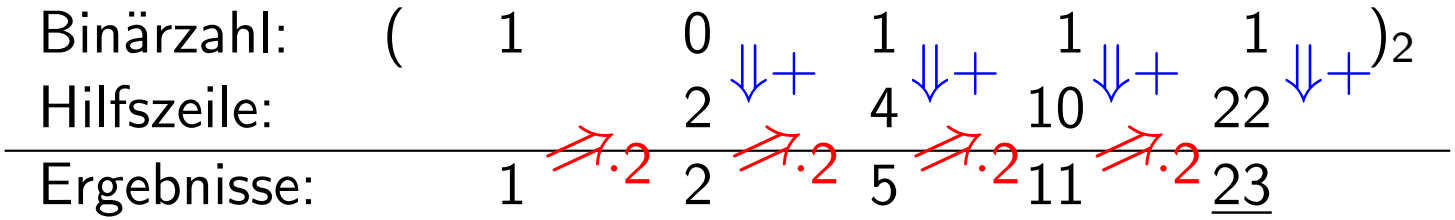
- Beispiel:

$$\begin{aligned}(10111)_2 &= \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0 \\ &= 16 + 4 + 2 + 1 \\ &= \underline{23}\end{aligned}$$

Binärzahl:	(	1		0		1		1		1)$_2$
Hilfszeile:		2^4	$\Downarrow\cdot$	2^3	$\Downarrow\cdot$	2^2	$\Downarrow\cdot$	2^1	$\Downarrow\cdot$	2^0 $\Downarrow\cdot$
Ergebnis:	$\mathbf{1} \cdot 2^4$	$+$	$\mathbf{0} \cdot 2^3$	$+$	$\mathbf{1} \cdot 2^2$	$+$	$\mathbf{1} \cdot 2^1$	$+$	$\mathbf{1} \cdot 2^0$	$= \underline{23}$

Umwandlung in das Dezimal-System (2)

- Horner-Schema:



$$
\begin{aligned}
23 &= \mathbf{1}\cdot 2^4 + \mathbf{0}\cdot 2^3 + \mathbf{1}\cdot 2^2 + \mathbf{1}\cdot 2^1 + \mathbf{1}\cdot 2^0 \\
&= (\mathbf{1}\cdot 2^3 + \mathbf{0}\cdot 2^2 + \mathbf{1}\cdot 2^1 + \mathbf{1})\cdot 2 + \mathbf{1} \\
&= ((\mathbf{1}\cdot 2^2 + \mathbf{0}\cdot 2^1 + \mathbf{1})\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1} \\
&= (((\mathbf{1}\cdot 2 + \mathbf{0})\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1}
\end{aligned}
$$

Umwandlung in das Binärsystem

- Abspaltung der höchsten Zweierpotenz
- Zweierpotenzen:

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1

- Beispiel:

$$23 = 16 + 7$$
$$23 = 16 + 4 + 3$$
$$23 = 16 + 4 + 2 + 1$$
$$23 = \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0$$
$$23 = (10111)_2$$

Umwandlung in das Binärsystem (2)

- Divisionsrestverfahren
- Ganzzahlige Division durch die Basis liefert nacheinander die Ziffern

$$
\begin{aligned}
23/2 &= 11 \text{ Rest } \mathbf{1} \\
11/2 &= 5 \text{ Rest } \mathbf{1} \\
5/2 &= 2 \text{ Rest } \mathbf{1} \\
2/2 &= 1 \text{ Rest } \mathbf{0} \\
1/2 &= 0 \text{ Rest } \mathbf{1}
\end{aligned}
$$

- Binärzahl kann von **unten nach oben** abgelesen werden:

$$23 = (10111)_2$$

Zahlenbereiche

- Anzahl der möglichen Ziffern (verfügbarer Speicher) bestimmt Zahlenbereiche
- Beispiel mit 4 Binärziffern:
  - Kleinste Zahl: $(0000)_2 = 0$
  - Größte Zahl: $(1111)_2 = 2^3 + 2^2 + 2^1 + 2^0 = 2^4 - 1 = 15$
- n Binärziffern:
  - Kleinste Zahl: 0
  - Größte Zahl: $2^n - 1$

Bit	min	max		
8	0	$2^8 - 1$	$=$	255
16	0	$2^{16} - 1$	$=$	65535
32	0	$2^{32} - 1$	$\approx$	$4.3 \cdot 10^9$
64	0	$2^{64} - 1$	$\approx$	$1.85 \cdot 10^{19}$

Wie weit kann man mit den Fingern zählen?

Addition von Binärzahlen

- Addition von Binärziffern:

$$
\begin{aligned}
0+0 &= 0 \\
0+1 &= 1 \\
1+0 &= 1 \\
1+1 &= 10 \quad (0 + \text{Übertrag } 1)
\end{aligned}
$$

- Beispiel: 14 + 9

$$
\begin{array}{rccccccc}
 & & 1 & 1 & 1 & 0 & = & 14 \\
+ & & 1 & 0 & 0 & 1 & = & 9 \\
\hline
 & 1 & 0 & 0 & 0 & & & \\
= & 1 & 0 & 1 & 1 & 1 & = & 23
\end{array}
$$

Multiplikation von Binärzahlen

- Multiplikation von Binärwerten

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

- Beispiel: 5 * 6 = 5*(4+2+0) = 5*(1*4 + 1*2 + 0*1)

	1	0	1	·	1	1	0	
			1	0	1			= 5*4
+				1	0	1		= 5*2
+							0	= 5*0
=			1	1	1	1	0	= 16+8+4+2 = 30

Erläuterungen zur Multiplikation

- Multiplikation mit 2 ($= 2^{\mathbf{1}}$) entspricht Verschiebung um **1** Stelle nach links
- Multiplikation mit 4 ($= 2^{\mathbf{2}}$) entspricht Verschiebung um **2** Stellen nach links
- Multiplikation mit 8 ($= 2^{\mathbf{3}}$) entspricht Verschiebung um **3** Stellen nach links
- ...
- Rechts mit Nullen auffüllen
- (Vergleiche Dezimalsystem: Multiplikation mit 10 = Null anhängen)

- Zerlege einen Faktor in Summe aus Zweierpotenzen
- Verschiebe zweiten Faktor um den jeweiligen Exponenten nach links und fülle rechts mit Nullen auf)
- Addiere die aus der Verschiebung erhaltenen Zahlen

- Hinweis: Verschiebungen ergeben sich aus den Stellen, an denen in der Binärdarstellung des zerlegten Faktors eine 1 steht

Inhalt

Negative Zahlen

- Wie lassen sich negative Zahlen darstellen?
- Vorzeichen wird in einem Bit gespeichert, Betrag in restlichen Bits

Einerkomplement

- Ausgehend von fester Anzahl Binärstellen, z.B.: 8 Stellen
- Das erste Bit (mit dem höchsten Wert, ganz links) wird das Vorzeichen
- Positive Zahlen werden direkt dargestellt
- Negative Zahlen durch eine bitweise Negation des Betrages
  (Umkehrung von jedem Bit: aus 0 wird 1 und umgekehrt)

Positiver Wertebereich	Negativer Wertebereich
“**0**0000000“ = 0	“**1**0000000“ = −127
“**0**0000001“ = 1	“**1**0000001“ = −126
.....	.....
“**0**1111111“ = 127	“**1**1111111“ = −0

- Vorteil: Schnelle Entscheidung positiv/negativ möglich
- Nachteil: Es gibt zwei Nullen: +0 und -0

Zweierkomplement-Darstellung

- Ausgehend von fester Anzahl Binärstellen, z.B.: 8 Stellen
- Positive Zahlen: Umwandlung in Dualzahl, führende Null[en] ergänzen
- $+7 = (111)_2$, 8-Bit-Zweierkomplement: 0000 0111
- Negative Zahlen:
  - Umwandlung des Betrags in Dualzahl, führende Null[en] ergänzen
  - Umkehr aller Bit (Negation)
  - Addition von 1
- $-7 \rightarrow 7 = (111)_2 \rightarrow 00000111 \rightarrow 11111000 \rightarrow 11111001$

- Vorteile:
  - Erstes Bit entspricht Vorzeichen: 0 - positiv, 1 - negativ
  - Es gibt genau eine Null
  - Rechnen ist einfach

Zweierkomplement-Darstellung (2)

3-Bit Zahlen:

$z =$	a_2	a_1	a_0	z^*	
$0 =$	0	0	0	0	
$1 =$	0	0	1	1	
$2 =$	0	1	0	2	
$3 =$	0	1	1	3	
$4 =$	1	0	0	-4	$= 4 - 2^3$
$5 =$	1	0	1	-3	$= 5 - 2^3$
$6 =$	1	1	0	-2	$= 6 - 2^3$
$7 =$	1	1	1	-1	$= 7 - 2^3$
$0 =$	0	0	0	0	

8-Bit Zahlen:

Positiver Wertebereich	Negativer Wertebereich
“00000000“ $= 0$	“10000000“ $= -128$
“00000001“ $= 1$	“10000001“ $= -127$
.....	.....
“01111111“ $= 127$	“11111111“ $= -1$

Addition im Zweierkomplement

- Addition wie bei Dualzahlen ohne Vorzeichen
- Streichen des entstehenden **Übertrages** am Anfang

00000100	4
11111101	− 3
100000001	1

11111100	− 4
11111101	− 3
111111001	− 7

Zahlenbereiche beim Zweierkomplement

Bit	min	max	min	max
8	-2^7	2^7-1	-128	127
16	-2^{15}	$2^{15}-1$	-32768	32767
32	-2^{31}	$2^{31}-1$	$\approx -2.1 \cdot 10^9$	$\approx 2.1 \cdot 10^9$
64	-2^{63}	$2^{63}-1$	$\approx -9.1 \cdot 10^{18}$	$\approx 9.1 \cdot 10^{18}$

Datentyp int in C++: 32 Bit

Darstellung mit verwandten Zahlensystemen

1. Hexadezimalsystem mit $b = 16$
   - $a_i \in \{0, \ldots, 15\}$ oft dargestellt durch $0, 1, \ldots, 9, A, B, C, D, E, F$
   - Codierung eines halben Bytes (4 Bit) mit einem Zeichen
   - 8-Bit-Zahlen lassen sich durch 2 hexadezimale Ziffern darstellen
   - Hexadezimalzahlen erlauben eine kompakte Darstellung von Binärzahlen
   - b ist 2er-Potenz $\rightarrow$ direkte Umwandlung ins Binärsystem möglich
2. Oktalsystem mit $b = 8$ (3 Bit)

Hexadezimalzahlen und Oktalzahlen in C++

- Hexadezimal- oder Oktalzahlen als Literale verwendbar
- Führende 0 für Oktalzahlen
- Führendes "0x" für Hexadezimalzahlen
- "Ziffern" 10 bis 15 der Hexadezimalzahlen sowohl als Großbuchstaben (A,B,C,D,E,F) als auch als Kleinbuchstaben (a,b,c,d,e,f) verwendbar

```
int i = 012;
int j = 0x12;
int m = 0xab;
int n = -0xCD;

cout << i << endl; // 10
cout << j << endl; // 18
cout << m << endl; // 171
cout << n << endl; // -205
```

- Führende Null kann nicht ignoriert werden!

Inhalt

Festkomma-Darstellung (fixed point number)

- Darstellung im Stellenwertsystem:

$$z = \sum_{i=-m}^{n} a_i \cdot 2^i$$

- Leicht aufspaltbar in ganze Zahl (links vom Komma) und gebrochenen Anteil (rechts vom Komma):

$$z = \sum_{i=0}^{n} a_i \cdot 2^i + \sum_{i=-m}^{-1} a_i \cdot 2^i$$

- Ganzzahliger Anteil wie bereits behandelt

Umwandlung des gebrochenen Anteils ins Dualsystem

- Beispiel 1:

$$0.75 \cdot 2 = \mathbf{1}.5$$
$$0.5 \cdot 2 = \mathbf{1}.\underline{0}$$

- $0.75 = (0.11)_2 = 2^{-1} + 2^{-2} = 0.5 + 0.25$

Umwandlung des gebrochenen Anteils ins Dualsystem (2)

- Beispiel 2:

$$
\begin{aligned}
0.45 \cdot 2 &= \mathbf{0}.9 \\
0.9 \cdot 2 &= \mathbf{1}.8 \\
0.8 \cdot 2 &= \mathbf{1}.6 \\
0.6 \cdot 2 &= \mathbf{1}.2 \\
0.2 \cdot 2 &= \mathbf{0}.4 \\
0.4 \cdot 2 &= \mathbf{0}.8 \\
0.8 \cdot 2 &= \mathbf{1}.6 \\
0.6 \cdot 2 &= \mathbf{1}.2 \ldots \text{(Periode!)}
\end{aligned}
$$

- $(0.01110011\ldots)_2 = (0.01\overline{1100})_2 = 2^{-2} + 2^{-3} + 2^{-4} + \ldots = 0.25 + 0.125 + 0.0625 + \ldots = 0.4375 + \ldots$

Umwandlung des Bruchteils ins Dezimalsystem

- Verschieben nach links ist Multiplikation mit 2er-Potenz:

$$\begin{aligned} (0.11)_2 &= z && \mid \cdot 2^2 \\ (11)_2 = 3 &= 4z && \mid -z \\ z &= \frac{3}{4} = 0.75 \end{aligned}$$

- mit Periode:

$$\begin{aligned} (0.\overline{101})_2 &= z && \mid \cdot 2^3 \\ (101.\overline{101})_2 &= 8z && \mid -z \\ (101)_2 &= 8z - z \\ 7z &= 5 \\ z &= \frac{5}{7} = 0.714\ldots \end{aligned}$$

Umwandlung des Bruchteils ins Dezimalsystem (2)

- Beispiel 2, zunächst Periode eliminieren:

$$\begin{aligned} (0.01\overline{1100})_2 &= z && \mid \cdot 2^4 \\ (111.00\overline{1100})_2 &= 16z && \mid -z \\ (110.11)_2 &= 15z \end{aligned}$$

- ... dann in ganze Zahl umwandeln:

$$\begin{aligned} (110.11)_2 &= 15z && \mid \cdot 2^2 \\ (11011)_2 = 27 &= 60z && \mid /60 \\ z &= \frac{9}{20} = 0.45 \end{aligned}$$

Festkomma-Darstellung im Computer

- Speicherplatz-Festlegung für Ziffern des Vorkomma-Teiles und des Nachkomma-Teiles
- Komma (bzw. Punkt) wird nicht gespeichert
- Nachkommateil muss auf gegebene Länge beschnitten werden
- $6.45 = (110.01\overline{1100})_2 \approx (110.01110011)_2$

0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1

Inhalt

Gleitkomma-Darstellung (floating point number)

- Nachteil der Festkomma-Darstellung: sehr große und sehr kleine Zahlen erfordern eine große Stellenanzahl
- Viele Stellen sind unbenutzt: kleine Zahlen belegen nur die hinteren Stellen, große Zahlen nur die vorderen
- Idee der Gleitkommazahlen: die Ziffern bzw. das Komma sind verschiebbar
- Deutschland hat $8.267 \cdot 10^7$ Einwohner
- Ein Floh wiegt $2 \cdot 10^{-3}$ Gramm

Gleitkommadarstellung (2)

- Darstellung als $z = m \cdot b^e$
- m . . . Mantisse
- b . . . Basis
- e . . . Exponent

- Dezimal: $1.2 \cdot 10^{-3}$
- Literal in C++: $1.2e-3$
- Dualzahlen: $(11 \cdot 10^{-10})_2 = (0.11)_2$ bzw. $(3 \cdot 2^{-2})_{10} = (0.75)_{10}$

Gleitkommadarstellung (3)

- Binärdarstellung von z durch

Vorzeichen	Exponent	Mantisse

- Computer: Wahl der binären Basis $b = 2$

- Vorzeichen: 1 Bit
- Mantisse (z.B. 52 Bit):
  - Enthält die Ziffern der Zahl
  - Je mehr Stellen gespeichert werden können desto genauer ist die Zahlendarstellung
- Exponent (z.B. 11 Bit):
  - Speichert die Position des Kommas
  - Zahlenbereich des Exponenten bestimmt den Zahlenbereich der Gleitkommazahl

Normalisierung bei Gleitkommazahlen

- Für ein gegebenes z gibt es mehrere Darstellungen:
  $0.2 \cdot 10^1 = 2 \cdot 10^0 = 20000 \cdot 10^{-4}$
- Normalisierung sinnvoll, z.B. : die erste von Null verschiedene Stelle der Mantisse ist die Stelle vor dem Komma
- $0.2 \cdot 10^1 \rightarrow 2 \cdot 10^0$
- $0.125 \rightarrow 1.25 \cdot 10^{-1}$
- Formal: $1 \leq m < b$
- Im Dualsystem ist damit die erste Stelle vor dem Komma immer 1. Diese muss deshalb nicht gespeichert werden.

IEEE Format

- Standard bei Gleitkommazahlen: IEEE 754
- Normalisierung mittels $1 \leq$ Mantisse ≤ 2, d.h. $(1, \ldots)_2$ in Binärdarstellung
- Single precision (32 Bit, float in C++):
  - Exponent: 8 Bit
  - Mantisse: 23 Bit
- Double precision (64 Bit, double in C++):
  - Exponent: 11 Bit
  - Mantisse: 52 Bit
- Achtung: Angaben sind Binärstellen!

Sonderwerte bei Gleitkommazahlen

- Der Wert Null besitzt ein Vorzeichen
- Sonderwert `nan` (**n**ot **a** **n**umber) dient zur Darstellung undefinierter Ergebnisse, zum Beispiel bei $\frac{0}{0}$
- Sonderwert `inf` (**inf**inity) steht für ∞ bzw. Bereichsüberschreitungen

```
double a = 0.0;
double b = 1.0;
double c = -1.0;
cout << a * b << endl; // 0.0
cout << a * c << endl; // -0.0
cout << 1e200 / 1e-200 << endl; //  inf
cout << -1e200 / 1e-200 << endl; // -inf
```

Anmerkungen zu Gleitkommazahlen

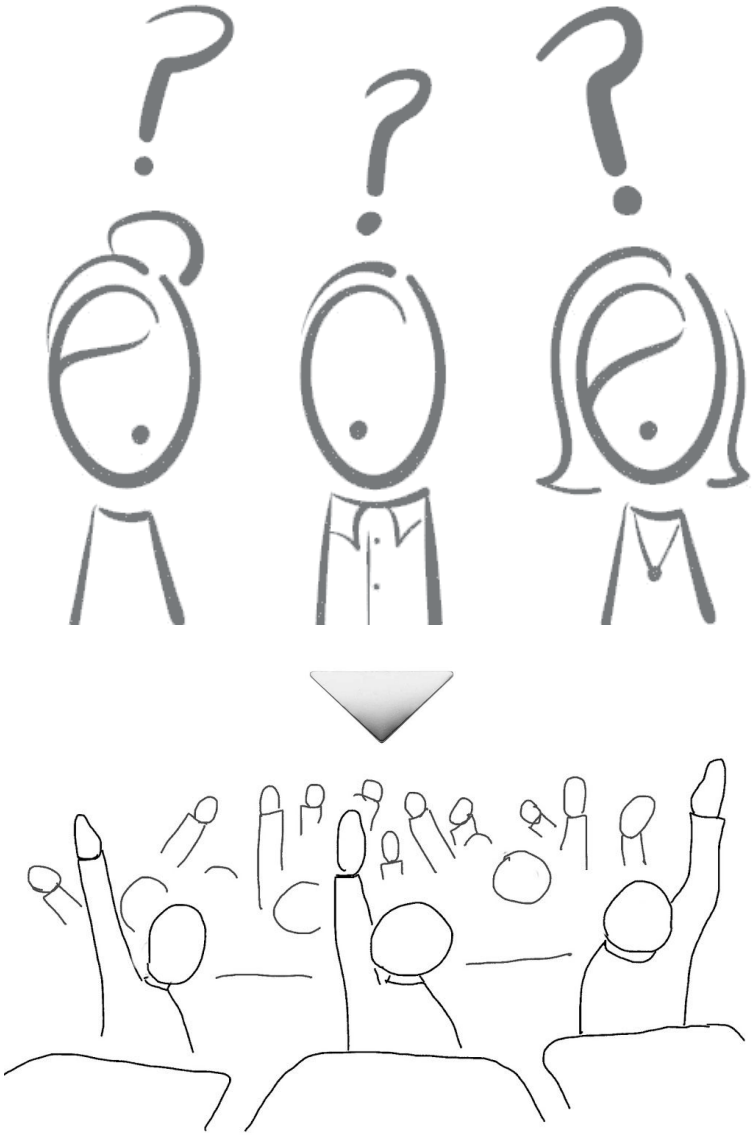
- **Viele Zahlen lassen sich nicht exakt am Rechner darstellen!**
- Daher kommt es zu offensichtlichen Berechnungsfehlern

```
if (0.362 * 100 == 36.2)
  // this statement will be never reached
  cout << "Condition is true" << endl;

if (0.362 * 100 / 100 == 0.362)
  // this statement will be never reached
  cout << "Condition is true" << endl;
```

- Numerik: Entwicklung nicht nur mathematisch exakter Verfahren, sondern auch numerisch stabiler Algorithmen

Gibt es Fragen? (Es gibt keine dummen Fragen!)



"Excuse me, is this the Society for Asking Stupid Questions?"