# Algorithmen und Datenstrukturen (für ET/IT)

Wintersemester 2012/13

Dr. Tobias Lasser

Computer Aided Medical Procedures
Technische Universität München


## Organisation (Wiederholung)

**Vorlesungswebseite**

`http://campar.in.tum.de/Chair/TeachingWs12AlgoDs`

- aktuelle Nachrichten
- Vorlesungs-Folien
- Übungsblätter
- Code-Beispiele

**Moodle**

`https://www.moodle.tum.de/course/view.php?idnumber=950071893`

- Diskussions-Forum



## Kontakt

Bei Fragen:

- Sprechstunde Vorlesung
  - Mittwoch, 8:30 - 9:30, Raum 1200
- Sprechpunkt Zentralübung
  - Montag, 13:30, Raum 0999
- Email
  - `algods@mailnavab.in.tum.de`
- Diskussions-Forum
  - Moodle: `https://www.moodle.tum.de/course/view.php?idnumber=950071893`
- Tutor-Fragestunden
  - Freitag, 9:30 - 11:30, Raum 1977
  - Freitag, 14:00 - 16:00, Raum 0999



## Programm heute

## Primitive Datentypen (präzisiert!)

### Primitive Datentypen

In Computern verwendet man Binärdarstellung mit einer fixen Anzahl Ziffern (genannt Bits).

Die primitiven Datentypen sind

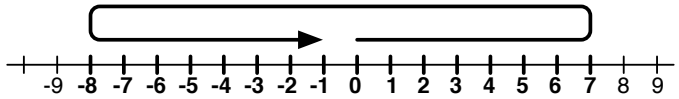
- 8 Bits (auch genannt Byte), darstellbare Zahlen: $\{0, \ldots, 255\}$
  in C, C++: `unsigned char`
- 16 Bits, darstellbare Zahlen: $\{0, \ldots, 65535\}$
  in C, C++: `unsigned short`
- 32 Bits, darstellbare Zahlen: $\{0, \ldots, 4294967295\}$
  in C, C++: `unsigned long`
- 64 Bits, darstellbare Zahlen: $\{0, \ldots, 2^{64} - 1\}$
  in C, C++: `unsigned long long`



## Negative Zahlen

Darstellung durch 2-Komplement

Beispiel für 4 Bits (darstellbare Zahlen: $2^4 = 16$):




Damit erhält man:

| | | | |
|---|---|---|---|
| 0000 = +0 | 0100 = +4 | 1000 = -8 | 1100 = -4 |
| 0001 = +1 | 0101 = +5 | 1001 = -7 | 1101 = -3 |
| 0010 = +2 | 0110 = +6 | 1010 = -6 | 1110 = -2 |
| 0011 = +3 | 0111 = +7 | 1011 = -5 | 1111 = -1 |

Das erste Bit ist also das Vorzeichen!



## 2-Komplement Darstellung I

### 2-Komplement Darstellung

Sei $x \in \mathbb{N}$, $x > 0$. Die 2-Komplement Darstellung $-x_z$ von $-x$ mittels $n$ Bits ist gegeben durch

$$-x_z = 2^n - x.$$

Vorheriges Beispiel war: $-5 = 1011$, also $x = 5$ und $n = 4$.

Nun:

$$-5_z = 2^4 - 5 = 16 - 5 = 11 = 1011_2$$



## 2-Komplement Darstellung II

Sei $b_n b_{n-1} \ldots b_1$ eine Bitfolge.

- $(b_n b_{n-1} \ldots b_1)_z$ sei der Zahlwert in 2-Komplement Darstellung
- für positive Zahlen von 0 bis $2^{n-1} - 1$ entspricht $(b_n b_{n-1} \ldots b_1)_z$ der Binärdarstellung:

$$(0 b_{n-1} \ldots b_1)_z = (0 b_{n-1} \ldots b_1)_2$$

- für negative Zahlen von $-2^{n-1}$ bis $-1$ gilt

$$(1 b_{n-1} \ldots b_1)_z = -2^{n-1} + (0 b_{n-1} \ldots b_1)_2$$

- allgemein:

$$(b_n b_{n-1} \ldots b_1)_z = b_n \cdot (-2^{n-1}) + (b_{n-1} \ldots b_1)_2$$

## Eigenschaften 2-Komplement

- Für $n \in \mathbb{N}$ gilt

$$\begin{aligned}(111\ldots11)_z &= (-2^{n-1}) + 2^{n-2} + \ldots + 2^1 + 2^0 \\ &= -2^{n-1} + (2^{n-1} - 1) \\ &= -1\end{aligned}$$

- Um $-x$ aus $x$ in 2-Komplement Darstellung zu erhalten:
  Bilde bitweises Komplement und addiere 1.
  - Beispiel: Negatives von $6 = (0110)_2$ mit $n = 4$

$$-6 = (\bar{0}\bar{1}\bar{1}\bar{0})_z + 1 = (1001)_z + 1 = (1010)_z$$

  - und zurück:

$$6 = (\bar{1}\bar{0}\bar{1}\bar{0})_z + 1 = (0101)_z + 1 = (0110)_z$$



## Ganze Zahlen in C, C++

### Ganze Zahlen in C, C++

- 8 Bits: `unsigned char` $\{0, \ldots, 255\}$
  `signed char` $\{-128, \ldots, 127\}$
- 16 Bits: `unsigned short` $\{0, \ldots, 65535\}$
  `signed short` $\{-32768, \ldots, 32767\}$
- 32 Bits: `unsigned long` $\{0, \ldots, 2^{32} - 1\}$
  `signed long` $\{-2^{31}, \ldots, 2^{31} - 1\}$
- 64 Bits: `unsigned long long` $\{0, \ldots, 2^{64} - 1\}$
  `signed long long` $\{-2^{63}, \ldots, 2^{63} - 1\}$

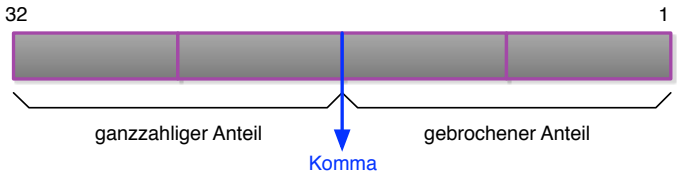
- `signed` kann weggelassen werden (ausser bei `char`!)
- `unsigned int` und `signed int` sind je nach System 16, 32 oder 64 Bit



## Rationale Zahlen I

Festkomma Darstellung:

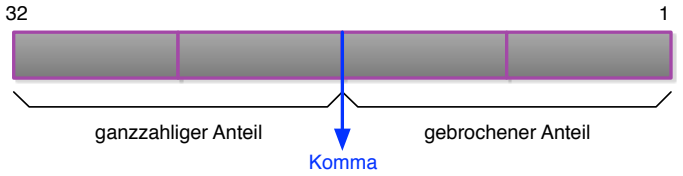
- Komma an fester Stelle in Zahl
- Beispiel mit $n = 32$:




- Nachteile:
  - weniger große Zahlen darstellbar
  - feste Genauigkeit der Nachkommastellen



## Rationale Zahlen II




- Interpretation für $r \in \mathbb{Q}$:

$$r = c_n \cdot 2^n + \ldots + c_0 \cdot 2^0 + c_{-1}2^{-1} + \ldots + c_{-m} \cdot 2^{-m}$$

  mit $n$ Vorkomma- und $m$ Nachkomma-Ziffern
- Beispiel:

$$\begin{aligned}11.01_2 &= 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} \\ &= 2 + 1 + 0 + \tfrac{1}{4} = 3.25_{10}\end{aligned}$$

# Floating Point Zahlen I

## Wissenschaftliche Notation:

- $x = a \cdot 10^b$ für $x \in \mathbb{R}$, wobei:
  - $a \in \mathbb{R}$ mit $1 \leq |a| < 10$
  - $b \in \mathbb{Z}$

- Beispiele:
  - $-2.7315 \cdot 10^2$ °C absoluter Nullpunkt
  - $1.0 \cdot 10^9$ Hz Taktfrequenz A5X Prozessor

- Drei Bestandteile:
  - Vorzeichen
  - Mantisse $|a|$
  - Exponent $b$
- Problem: bei fester Länge der Mantisse (z.B. 3 Ziffern)
  - zwischen $1.23 \cdot 10^4 = 12300$ und $1.24 \cdot 10^4 = 12400$ keine Zahl darstellbar!



# Floating Point Zahlen II


- wissenschaftliche Darstellung mit Basis 2

$$f = (-1)^V \cdot (1 + M) \cdot 2^{E - bias}$$

- Vorzeichen Bit $V$
- Mantisse $M$ hat immer die Form $1.abc$, also wird erste Stelle weggelassen („hidden bit“)
- Exponent $E$ wird vorzeichenlos abgespeichert, verschoben um $bias$
  - bei 32 bit float: $bias = 127$, bei 64 bit double: $bias = 1023$



# Floating Point Zahlen III

## Übliche Floating Point Formate:

| Bit | Vorz. | Exponent | Mantisse | gültige Dezimalst. | darstellbarer Bereich |
|---|---|---|---|---|---|
| 32 | 1 Bit | 8 Bit | 23 Bit | $\sim 7$ | $\pm 2 \cdot 10^{-38}$ bis $\pm 2 \cdot 10^{38}$ |
| 64 | 1 Bit | 11 Bit | 52 Bit | $\sim 15$ | $\pm 2 \cdot 10^{-308}$ bis $\pm 2 \cdot 10^{308}$ |
| 80 | 1 Bit | 15 Bit | 64 Bit | $\sim 19$ | $\pm 1 \cdot 10^{-4932}$ bis $\pm 1 \cdot 10^{4932}$ |

In C, C++:

`float` (32 Bit), `double` (64 Bit), `long double` (80 Bit)



# Vorsicht mit Floating Point!

Floating Point Zahlen sind bequem, aber Vorsicht!

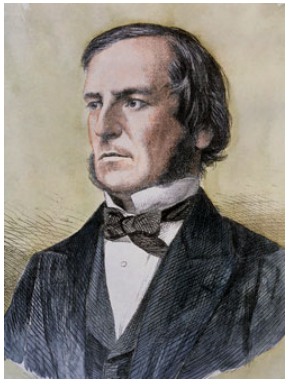
- Viele Dezimalzahlen haben keine Floating Point Darstellung
  - Beispiel: $0.1_{10} = 0.0001100110011\ldots_2$ (periodisch)
- Kritisch sind Vergleiche von Floating Point Zahlen
  - Beispiel: `(0.1 + 0.2 == 0.3)` ist meist FALSE!
- Zins-Berechnungen und dergleichen NIE mit Floating Point Zahlen!
  - Stattdessen: spezielle Bibliotheken wie GMP

## Programm heute





## George Boole



Englischer Mathematiker (1815-1864)



## Logische Werte

Boolesche Logik: Logik mit zwei Werten

- Repräsentationen:
  - 1 und 0
  - W und F (in Englisch: T und F)
  - L und O
- Mengensymbol $\mathbb{B}$
  - $\mathbb{B} = \{0, 1\} = \{F, W\} = \{O, L\}$



## Logische Werte und Verknüpfungen

„Grundrechenarten“ mit logischen Werten:

- Konjunktion: $\wedge : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - ähnlich zu Multiplikation bei Zahlen
  - auch bezeichnet als UND bzw. AND
- Disjunktion: $\vee : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - ähnlich zu Addition bei Zahlen
  - auch bezeichnet als ODER bzw. OR
- Negation: $\neg : \mathbb{B} \to \mathbb{B}$
  - auch bezeichnet als NICHT bzw. NOT

Wahrheitstabelle:

| $a$ | $b$ | $a \wedge b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $a$ | $b$ | $a \vee b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $a$ | $\neg a$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Weitere Verknüpfungen I

- NAND: $\uparrow$: $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - $a \uparrow b = \neg(a \wedge b)$
  - mit NAND lassen sich NOT, OR, AND erzeugen
- NOR: $\downarrow$: $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - $a \downarrow b = \neg(a \vee b)$
  - mit NOR lassen sich ebenso NOT, OR, AND erzeugen
- XOR: $\oplus$ : $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - auch exklusiv oder genannt
  - erzeugbar aus $\neg(a \wedge b) \wedge (a \vee b)$ (siehe Übung)

Wahrheitstabelle:

| $a$ | $b$ | $a \uparrow b$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| $a$ | $b$ | $a \downarrow b$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

| $a$ | $b$ | $a \oplus b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



# Weitere Verknüpfungen II

- Implikation: $\Rightarrow$: $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - oft verwendet für mathematische Sätze: „$a$ impliziert $b$“, „aus $a$ folgt $b$“
  - Beispiel: „aus $n < 3$ folgt $n < 5$“
  - erzeugbar aus $\neg a \vee b$
- Äquivalenz: $\Leftrightarrow$: $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - oft verwendet für mathematische Sätze: „$a$ gilt genau dann, wenn $b$ gilt“, „$a$ und $b$ sind äquivalent“
  - Beispiel: „$f$ ist bijektiv genau dann, wenn $f$ injektiv und surjektiv ist“
  - erzeugbar aus $(a \wedge b) \vee (\neg a \wedge \neg b)$

Wahrheitstabelle:

| $a$ | $b$ | $a \Rightarrow b$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $a$ | $b$ | $a \Leftrightarrow b$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



# Rechenbeispiel

Zeige:

$$a \Leftrightarrow b \text{ entspricht } (a \wedge b) \vee (\neg a \wedge \neg b)$$

| $a$ | $b$ | $a \wedge b$ | $\neg a$ | $\neg b$ | $\neg a \wedge \neg b$ | $(a \wedge b) \vee (\neg a \wedge \neg b)$ | $a \Leftrightarrow b$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | | | | | | |
| 0 | 1 | | | | | | |
| 1 | 0 | | | | | | |
| 1 | 1 | | | | | | |



# Rangfolge und Rechenregeln

Rangfolge:

- NICHT vor UND
- UND vor ODER

Beispiel:

$$\neg 0 \vee 1 \wedge 0 = (\neg 0) \vee (1 \wedge 0) = 1 \vee 0 = 1$$

De Morgan-Gesetze:

- $\neg(a \wedge b) = \neg a \vee \neg b$
- $\neg(a \vee b) = \neg a \wedge \neg b$

## Ausblick

(Ausblick nicht klausur-relevant)

- Umformung von Booleschen Ausdrücken in Normalformen
  - Disjunktive Normalform (DNF)
  - Konjunktive Normalform (KNF)
- K-V Diagramme
- Entwurf von Schaltungen

$\longrightarrow$ Schaltungstechnik



## Logische Ausdrücke in C, C++

- logische Variablen: `bool a,b;`
- logische Werte: `true` und `false`
- NOT Operator: `!a`
- AND Operator: `a && b`
- OR Operator: `a || b`

Beispiele:

- `( (2 == 2) && (3 < 1) )`
  ergibt `(true && false)`, also `false`
- `( !(2 == 2) || (3 > 1) )`
  ergibt `(false || true)`, also `true`
- Kurzform für `!(2 == 2)` ist `(2 != 2)`



## Zusammenfassung