Algorithmen und Datenstrukturen (für ET/IT)

Sommersemester 2018

Dr. Stefanie Demirci

Computer Aided Medical Procedures
Technische Universität München

Tutorübungen starten heute

Tutorübung	Tutorübung	Tutorübung	Raum
1	Freitag 9:45-11:15	Vincent von Büren	"0999"
2	Freitag 9:45-11:15	Izlen Erenoglu	"0406"
3	Donnerstag 8:00-9:30	Lea Straumann	"0406"
4	Freitag 11:30-13:00	Benedikt Böck	"0999"
5	Dienstag 15:00-16:30	Artem Gazizov	"0999"
6	Dienstag 15:00-16:30	Andreas Finkenzeller	"N5325"
7	Montag 15:00-16:30	Leon Garidis	"0999"
8	Montag 15:00-16:30	Rojda Hicsanmaz	"0406"
9	Donnerstag 8:00-9:30	Verena Keßler-Weißenbrucker	"N5325"
10	Dienstag 15:00-16:30	Adrian Schiechel	"0406"

Programm heute

1. Einführung
2. Grundlagen von Algorithmen
3. Grundlagen von Datenstrukturen
   - Primitive Datentypen und Zahldarstellung
   - Felder als sequentielle Liste
   - Zeichen und Zeichenfolgen

Was sind primitive Datentypen?

Primitive Datentypen

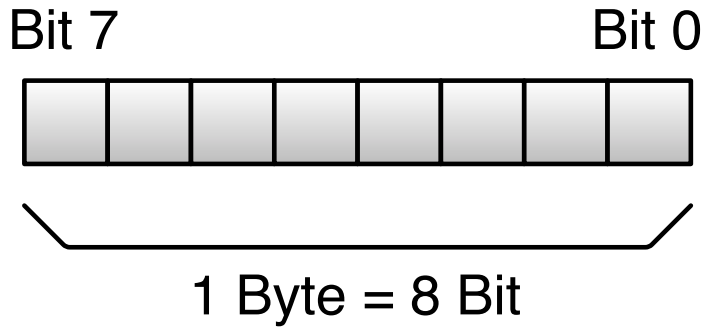
Wir bezeichnen grundlegende, in Programmiersprachen eingebaute Datentypen als primitive Datentypen.

Durch Kombination von primitiven Datentypen lassen sich zusammengesetzte Datentypen bilden.

Beispiele für primitive Datentypen in C:

- `int` für ganze Zahlen
- `float` für floating point Zahlen
- `bool` für logische Werte

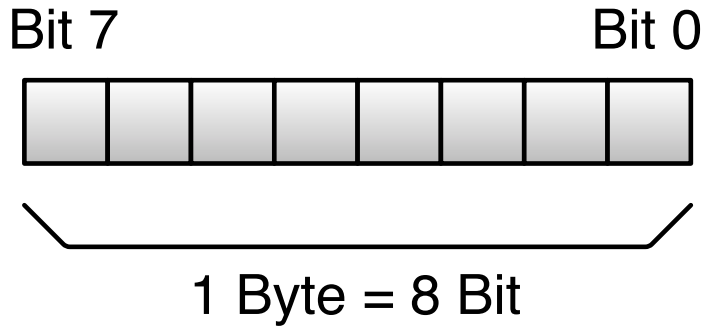
Bits und Bytes



Bytes als Maßeinheit für Speichergrössen (nach IEC, traditionell):

- 2^{10} Bytes = 1024 Bytes = 1 KiB, ein Kilo Byte (Kibi Byte)
- 2^{20} Bytes = 1 MiB, ein Mega Byte (bzw. MebiByte)
- 2^{30} Bytes = 1 GiB, ein Giga Byte (bzw. GibiByte)
- 2^{40} Bytes = 1 TiB, ein Tera Byte (bzw. TebiByte)
- 2^{50} Bytes = 1 PiB, ein Peta Byte (bzw. PebiByte)
- 2^{60} Bytes = 1 EiB, ein Exa Byte (bzw. ExbiByte)

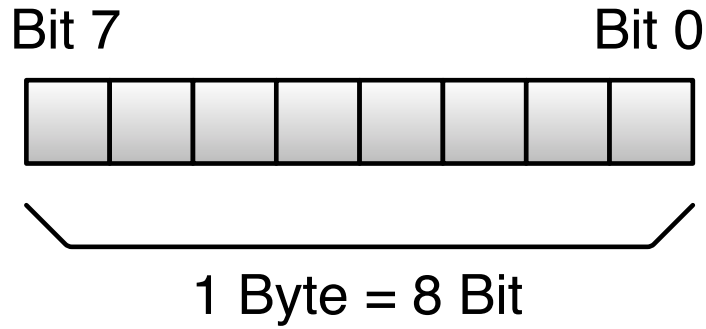
Bits und Bytes



Bytes als Maßeinheit für Speichergrössen (nach IEC, metrisch):

- 10^3 Bytes = 1000 Bytes = 1 kB, ein kilo Byte (großes B)
- 10^6 Bytes = 1 MB, ein Mega Byte
- 10^9 Bytes = 1 GB, ein Giga Byte
- 10^{12} Bytes = 1 TB, ein Tera Byte
- 10^{15} Bytes = 1 PB, ein Peta Byte
- 10^{18} Bytes = 1 EB, ein Exa Byte

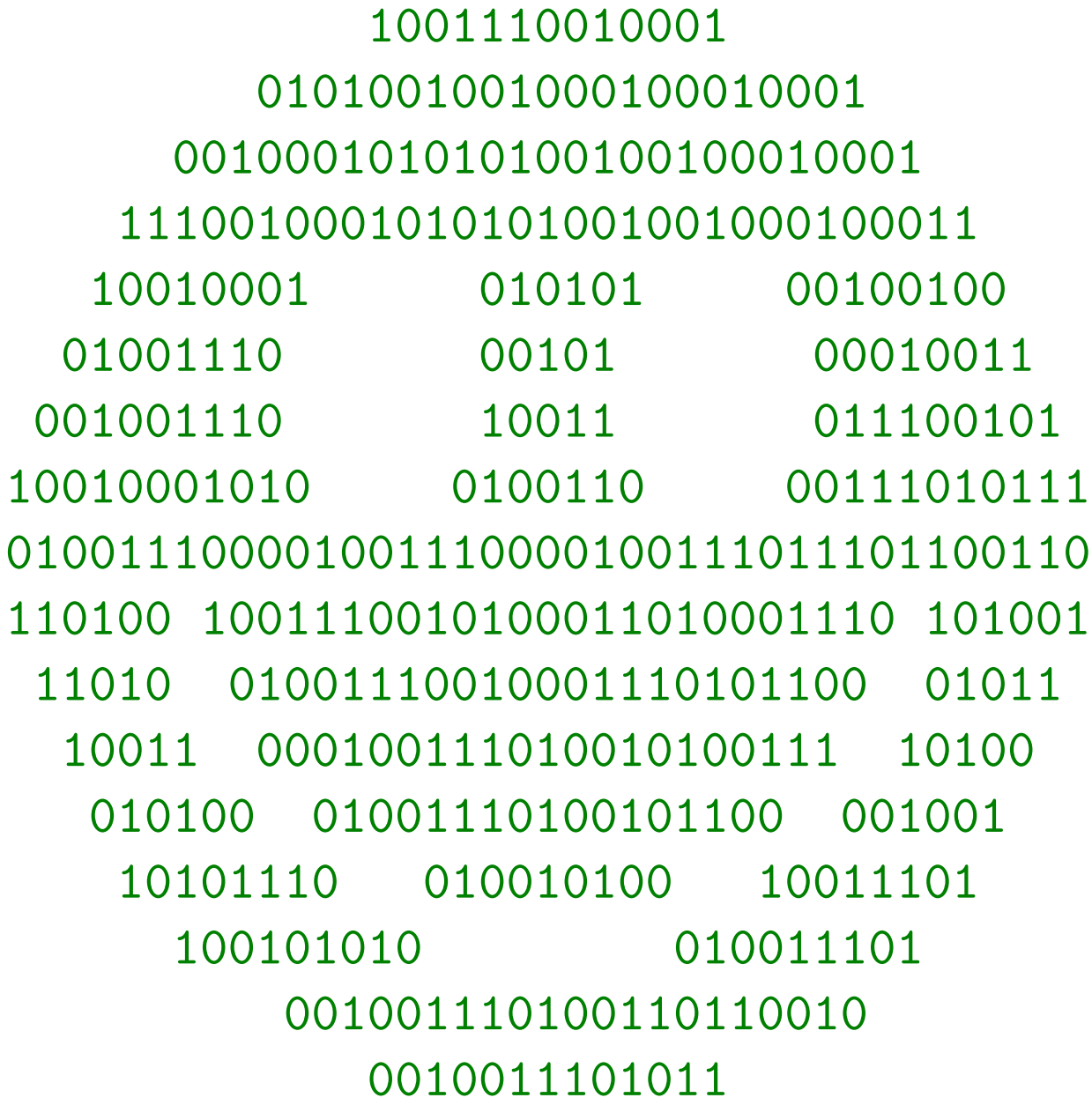
Bits und Bytes



Bytes als Maßeinheit für Speichergrössen (nach IEC, metrisch):

- 10^3 Bytes = 1000 Bytes = 1 kB, ein kilo Byte (großes B)
- 10^6 Bytes = 1 MB, ein Mega Byte
- 10^9 Bytes = 1 GB, ein Giga Byte
- 10^{12} Bytes = 1 TB, ein Tera Byte
- 10^{15} Bytes = 1 PB, ein Peta Byte
- 10^{18} Bytes = 1 EB, ein Exa Byte

Hinweis: auch Bits werden als Maßangabe verwendet, z.B. 16 Mbit oder 16 Mb (kleines b).


1001110010001
01010010010001000010001
00100010101010010010001000 10001
11100100010101010010010001000 10011
10010001 010101 00100100
01001110 00101 00010011
001001110 10011 011100101
10010001010 0100110 00111010111
0100111000010011100001001110111011001 10
110100 10011100101000110100011 10 101001
11010 0100111001000111010110 0 01011
10011 0001001110100101001 11 10100
010100 0100111010010110 0 001001
10101110 010010100 10011101
100101010 010011101
0010011101001101100 10
0010011101011

Primitive Datentypen in C-ähnlichen Sprachen

Wir betrachten im Detail primitive Datentypen für:

1. natürliche Zahlen *(unsigned integers)*
2. ganze Zahlen *(signed integers)*
3. floating point Zahlen *(floats)*

Zahldarstellung

- Dezimalsystem:
  - Basis $x = 10$
  - Koeffizienten $c_n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - Beispiel: $123_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$

Zahldarstellung

- Dezimalsystem:
  - Basis $x = 10$
  - Koeffizienten $c_n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - Beispiel: $123_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$
- Binärsystem:
  - Basis $x = 2$
  - Koeffizienten $c_n \in \{0, 1\}$
  - Beispiel: $1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13_{10}$

Zahldarstellung

- Oktalsystem:
  - Basis $x = 8$ $(= 2^3)$
  - Koeffizienten $c_n \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
  - Beispiel: $173_8 = 1 \cdot 8^2 + 7 \cdot 8^1 + 3 \cdot 8^0 = 123_{10}$

Zahldarstellung

- Oktalsystem:
  - Basis $x = 8\ (= 2^3)$
  - Koeffizienten $c_n \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
  - Beispiel: $173_8 = 1 \cdot 8^2 + 7 \cdot 8^1 + 3 \cdot 8^0 = 123_{10}$
- Hexadezimalsystem:
  - Basis $x = 16\ (= 2^4)$
  - Koeffizienten $c_n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
  - Beispiel: $7B_{16} = 7 \cdot 16^1 + B \cdot 16^0 = 123_{10}$

Wie viele Ziffern pro Zahl?

Problem

Gegeben Zahl $z \in \mathbb{N}$, wie viele Ziffern m werden bezüglich Basis x benötigt?

Lösung

$$m = \lfloor \log_x(z) \rfloor + 1$$

Erläuterung: $(a \in \mathbb{R})$

- $\lfloor a \rfloor = \text{floor}(a) =$ größte ganze Zahl kleiner gleich a
- $\lceil a \rceil = \text{ceil}(a) =$ kleinste ganze Zahl größer gleich a

$$a - 1 < \lfloor a \rfloor \leq a \leq \lceil a \rceil < a + 1$$

- $\log_x(z) = \frac{\ln(z)}{\ln(x)}$, wobei „ln“ der natürliche Logarithmus ist

Wie viele Ziffern pro Zahl?

Lösung

$$m = \lfloor \log_x(z) \rfloor + 1$$

Beispiele: $z = 123$

- Basis $x = 10$:

$$m = \lfloor \log_{10}(123) \rfloor + 1 = \lfloor 2.0899\ldots \rfloor + 1 = 3$$

- Basis $x = 2$:

$$m = \lfloor \log_2(123) \rfloor + 1 = \lfloor 6.9425\ldots \rfloor + 1 = 7$$

- Basis $x = 8$:

$$m = \lfloor \log_8(123) \rfloor + 1 = \lfloor 2.3141\ldots \rfloor + 1 = 3$$

- Basis $x = 16$:

$$m = \lfloor \log_{16}(123) \rfloor + 1 = \lfloor 1.7356\ldots \rfloor + 1 = 2$$

Größte Zahl pro Anzahl Ziffern?

Problem

Gegeben Basis x und m Ziffern, was ist die größte darstellbare Zahl?

Lösung

$$z_{max} = x^m - 1$$

Beispiele:

- $x = 2,\ m = 4$:

$$z_{max} = 2^4 - 1 = 15 = 1111_2$$

- $x = 2,\ m = 8$:

$$z_{max} = 2^8 - 1 = 255 = 11111111_2$$

- $x = 16,\ m = 2$:

$$z_{max} = 16^2 - 1 = 255 = FF_{16}$$

Natürliche Zahlen in C-ähnlichen Sprachen

Natürliche Zahlen

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

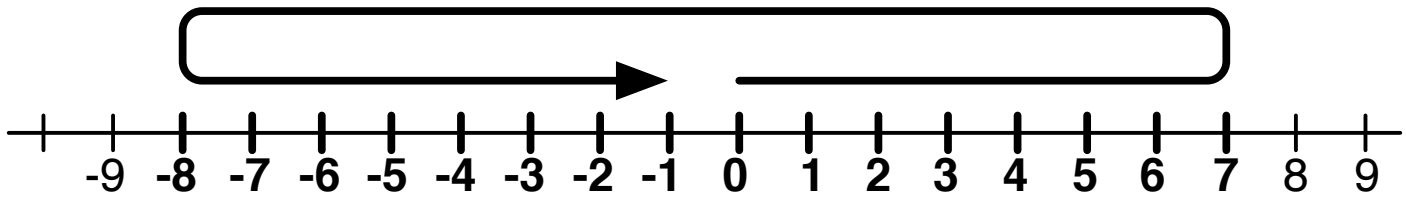
Die primitiven Datentypen für natürliche Zahlen sind:

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

Negative Zahlen

Darstellung durch 2-Komplement

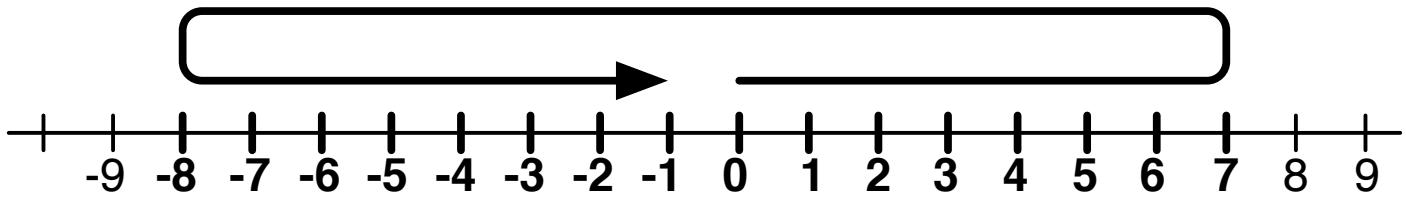
Beispiel für 4 Bits (darstellbare Zahlen: $2^4 = 16$):



Negative Zahlen

Darstellung durch 2-Komplement

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



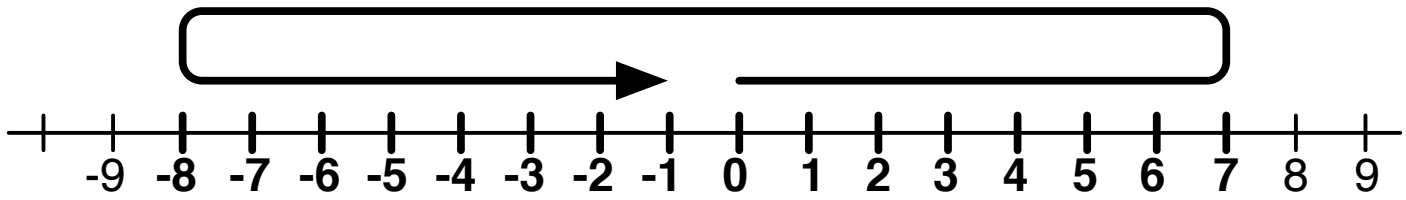
Damit erhält man:

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

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

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

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

2-Komplement Darstellung II

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

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

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

2-Komplement Darstellung II

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

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

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

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

$$(1 b_{n-1} \dots b_1)_z = \underbrace{-2^{n-1}}_{1\,0\,0\,\dots\,0 \;\; n-1 \text{ mal}} + (0 b_{n-1} \dots b_1)_2$$

$\therefore$

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

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

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-ähnlichen Sprachen

Ganze Zahlen

Die primitiven Datentypen für ganze Zahlen sind:

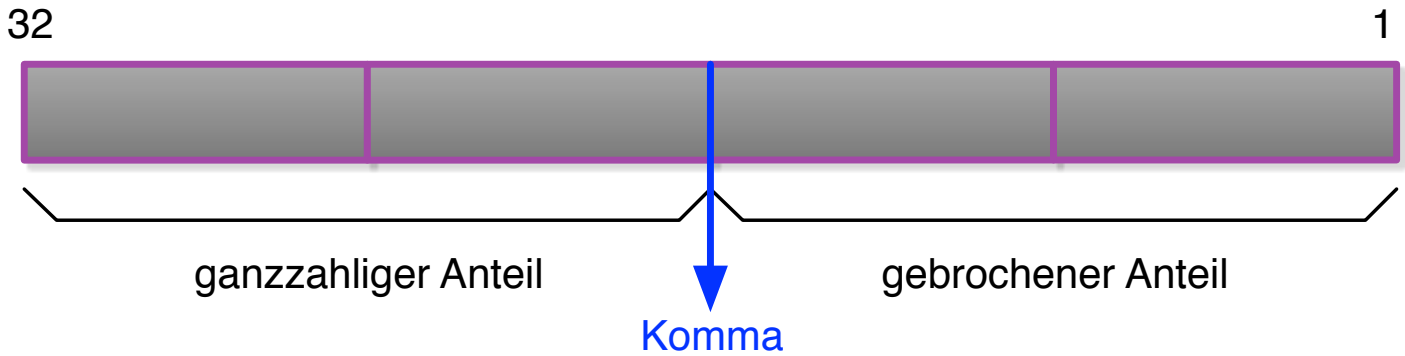
- 8 Bits: `unsigned char` $\{0, \ldots, 255\}$
  `signed char` $\{-128, \ldots, 127\}$
- 16 Bits: `unsigned short` $\{0, \ldots, 65535\}$
  `signed short` $\{-32768, \ldots, 32767\}$
- 32 Bits: `unsigned long` $\{0, \ldots, 2^{32} - 1\}$
  `signed long` $\{-2^{31}, \ldots, 2^{31} - 1\}$
- 64 Bits: `unsigned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

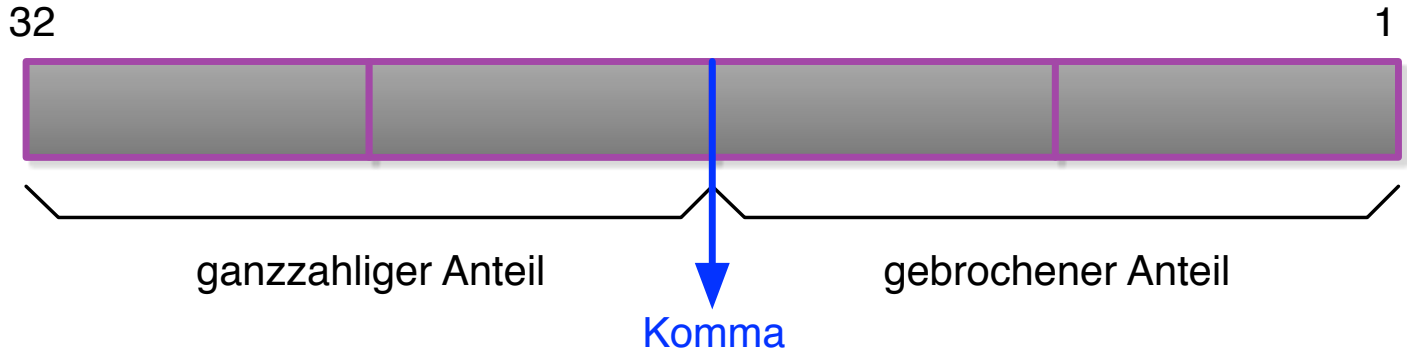
Festkomma Darstellung:

- 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 $n+1$ Vorkomma- und m Nachkomma-Ziffern

- Beispiel:

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

Floating Point Zahlen I

Wissenschaftliche Notation:

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

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
  - $1.5 \cdot 10^9$ Hz Taktfrequenz A8X Prozessor

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
  - $1.5 \cdot 10^9$ Hz Taktfrequenz A8X Prozessor

- Drei Bestandteile:
  - Vorzeichen
  - Mantisse $|a|$ (bestimmt die Genauigkeit)
  - Exponent b (bestimmt Größe des Wertebereichs)

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
  - $1.5 \cdot 10^9$ Hz Taktfrequenz A8X Prozessor

- Drei Bestandteile:
  - Vorzeichen
  - Mantisse $|a|$ (bestimmt die Genauigkeit)
  - Exponent b (bestimmt Größe des Wertebereichs)
- 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: $bias = 127$, bei 64 bit: $bias = 1023$

Floating Point Zahlen III

Übliche Floating Point Formate:

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

In 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)

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)
- Durch feste Länge der Mantisse sind ebenfalls viele Zahlen nicht darstellbar
  - Beispiel: mit 3 Ziffern Mantisse ist zwischen $1.23 \cdot 10^4 = 12300$ und $1.24 \cdot 10^4 = 12400$ keine Zahl darstellbar!

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)
- Durch feste Länge der Mantisse sind ebenfalls viele Zahlen nicht darstellbar
  - Beispiel: mit 3 Ziffern Mantisse ist zwischen $1.23 \cdot 10^4 = 12300$ und $1.24 \cdot 10^4 = 12400$ keine Zahl darstellbar!
- Kritisch sind Vergleiche von Floating Point Zahlen
  - Beispiel: `(0.1 + 0.2 == 0.3)` ist meist FALSE!

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)
- Durch feste Länge der Mantisse sind ebenfalls viele Zahlen nicht darstellbar
  - Beispiel: mit 3 Ziffern Mantisse ist zwischen $1.23 \cdot 10^4 = 12300$ und $1.24 \cdot 10^4 = 12400$ keine Zahl darstellbar!
- 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

Definition Datenstruktur

Definition Datenstruktur (nach Prof. Eckert)

Eine Datenstruktur ist eine

- logische Anordnung von Datenobjekten,
- die Informationen repräsentieren,
- den Zugriff auf die repräsentierte Information über Operationen auf Daten ermöglichen und
- die Information verwalten.

Zwei Hauptbestandteile:

- Datenobjekte
  - z.B. definiert über primitive Datentypen
- Operationen auf den Objekten
  - z.B. definiert als Funktionen

Primitive Datentypen in C

- Natürliche Zahlen, z.B. `unsigned short`, `unsigned long`
  - Wertebereich: bei n Bit von 0 bis $2^n - 1$
  - Operationen: +, -, *, /, %, <, ==, !=, >
- Ganze Zahlen, z.B. `int`, `long`
  - Wertebereich: bei n Bit von -2^{n-1} bis $2^{n-1} - 1$
  - Operationen: +, -, *, /, %, <, ==, !=, >
- Floating Point Zahlen, z.B. `double`, `float`
  - Wertebereich: abhängig von Größe
  - Operationen: +, -, *, /, <, ==, !=, >
- Logische Werte, `bool`
  - Wertebereich: `true`, `false`
  - Operationen: &&, ||, !, ==, !=

Programm heute

1. Einführung
2. Grundlagen von Algorithmen
3. Grundlagen von Datenstrukturen
   - Primitive Datentypen und Zahldarstellung
   - Felder als sequentielle Liste
   - Zeichen und Zeichenfolgen

Definition Feld

Definition Feld

Ein Feld A ist eine Folge von n Datenelementen $(d_i)_{i=1,\ldots,n}$,

$$A = d_1, d_2, \ldots, d_n$$

mit $n \in \mathbb{N}_0$.

Die Datenelemente d_i sind beliebige Datentypen (z.B. primitive).

Beispiele:

- A sind die natürlichen Zahlen von 1 bis 10, aufsteigend geordnet:
$$A = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$$
- Ist $n = 0$, so ist das Feld leer.

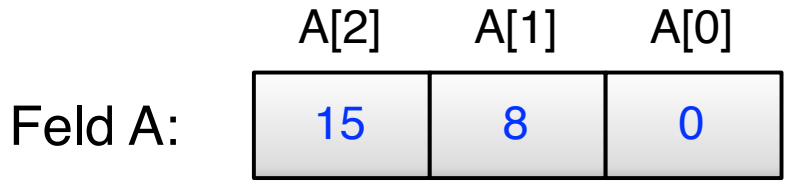
Feld als sequentielle Liste

Repräsentation von Feld A als sequentielle Liste (oder Array)

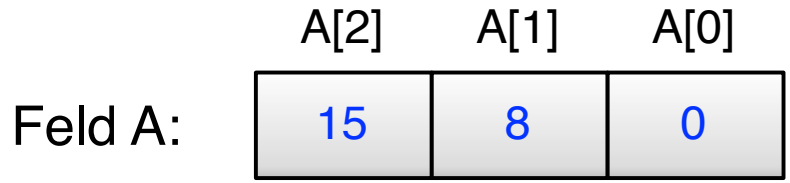
- feste Anzahl n von Datenelementen
- zusammenhängend gespeichert
- in linearer Reihenfolge mit Index
- Zugriff auf i-tes Element über Index i: A[i]

Achtung: Indizierung startet meist bei 0!

Beispiel sequentielle Liste



Beispiel sequentielle Liste



- Feld-Deklaration in C (optionales Beispiel):

```
int A[3];
```

Größe

- Zugriff auf Elemente:

```
A[0] = 0;
A[1] = 8;
A[2] = A[1] + 7; // nun: A[2] == 15
```

Eigenschaften sequentielle Liste

Feld A mit Länge n als sequentielle Liste (Array)

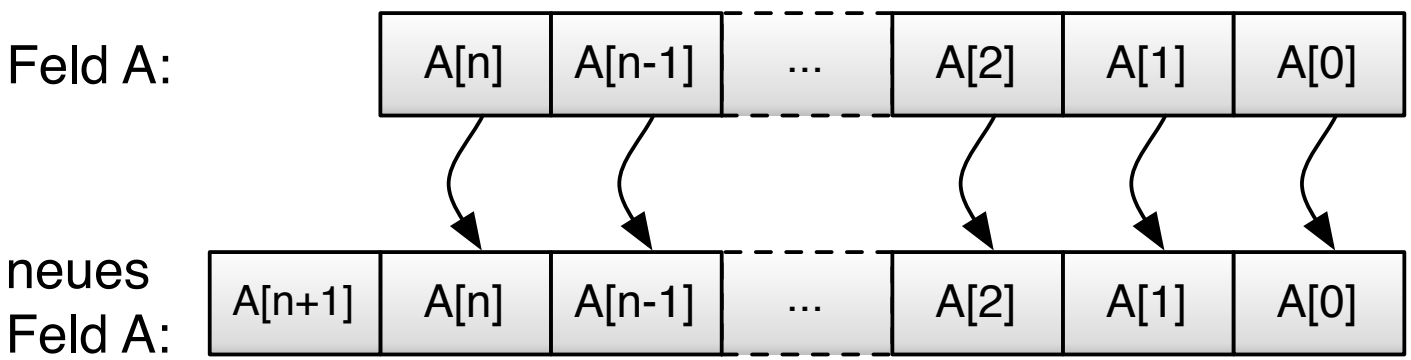
- Vorteile:
  - direkter Zugriff auf Elemente in konstanter Zeit mittels A[i]
  - sequentielles Durchlaufen sehr einfach
- Nachteile:
  - Verschwendung von Speicher falls Liste nicht voll belegt
  - Verlängern der sequentiellen Liste aufwendig
  - Hinzufügen und Löschen von Elementen aufwendig

Verlängern der sequentiellen Liste

Gegeben: Feld A, Länge n+1, als sequentielle Liste

Gewünscht: Feld A erweitert auf Länge n+2

- neuen Speicher der Größe n+2 reservieren
- alte Liste in neuen Speicher kopieren

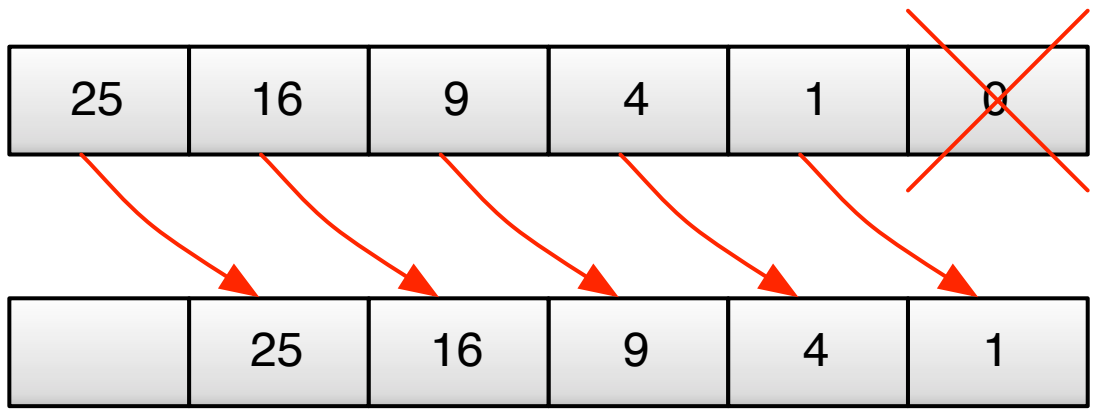


Löschen von Element aus Liste

Gegeben: Feld A, Länge n, als sequentielle Liste

Gewünscht: Element i aus Feld A löschen

- Element i entfernen
- Listenelemente nach i umkopieren

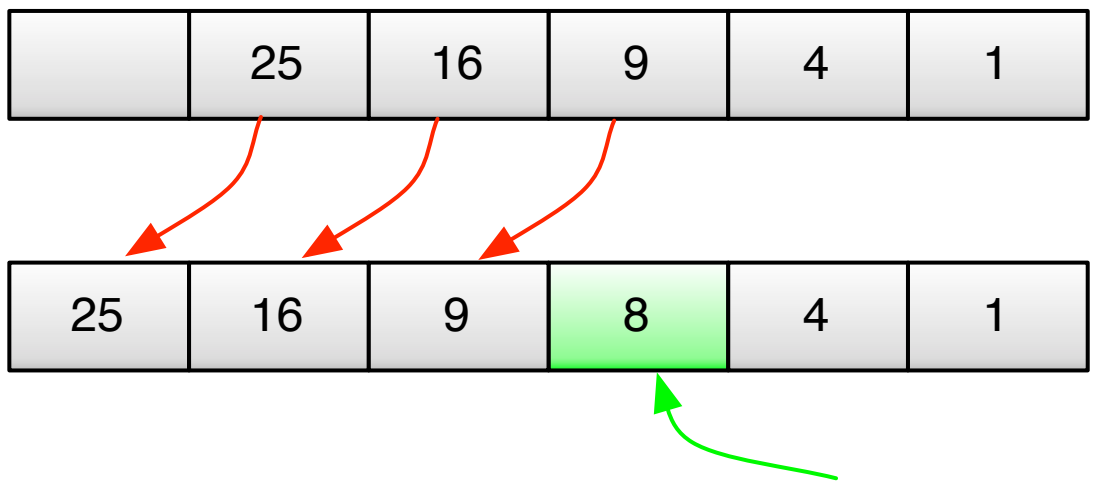


Einfügen von Element in Liste

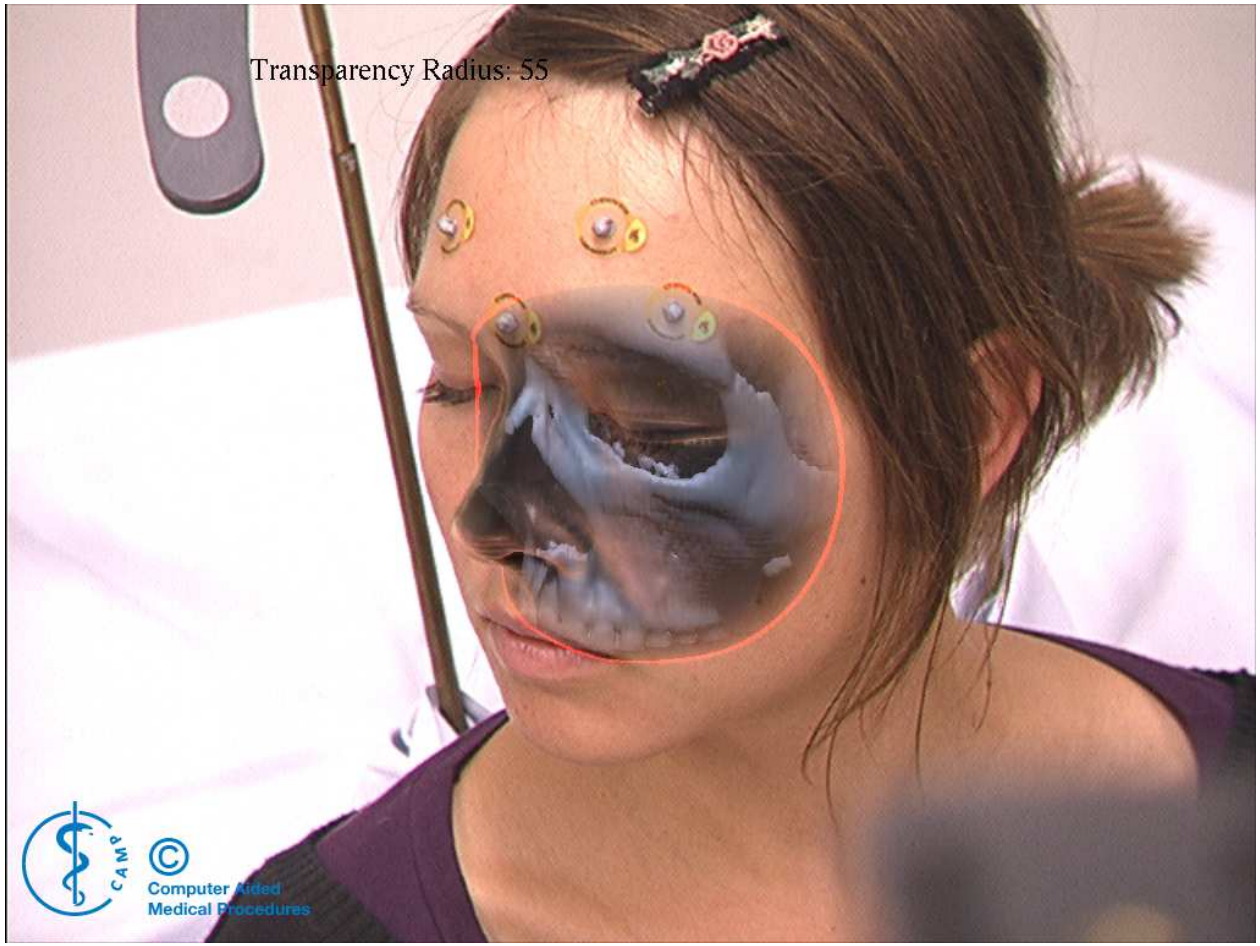
Gegeben: Feld A, Länge n, als sequentielle Liste

Gewünscht: neues Element in Feld A an Stelle i einfügen

- Listenelemente nach i umkopieren
- Element i einfügen



Ausblick: Anwendung von sequentiellen Listen



in 2D und 3D Bildern!

Programm heute

1. Einführung
2. Grundlagen von Algorithmen
3. Grundlagen von Datenstrukturen
   - Primitive Datentypen und Zahldarstellung
   - Felder als sequentielle Liste
   - Zeichen und Zeichenfolgen

Bytes und ASCII

Interpretation eines Bytes als Zeichen (anstatt Zahlen)

$\longrightarrow$ z.B. ASCII Code

7 Bit ASCII Code:

Code	..0	..1	..2	..3	..4	..5	..6	..7	..8	..9	..A	..B	..C	..D	..E	..F
0..	*nul*	*soh*	*stx*	*etx*	*eot*	*enq*	*ack*	*bel*	*bs*	*ht*	*lf*	*vt*	*ff*	*cr*	*so*	*si*
1..	*dle*	*dc1*	*dc2*	*dc3*	*dc4*	*nak*	*syn*	*etb*	*can*	*em*	*sub*	*esc*	*fs*	*gs*	*rs*	*us*
2..	*sp*	**!**	**“**	**#**	**$**	**%**	**&**	**’**	**(**	**)**	*****	**+**	**,**	**-**	**.**	**/**
3..	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**:**	**;**	**<**	**=**	**>**	**?**
4..	**@**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
5..	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**[**	\	**]**	**^**	**_**
6..	**‘**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
7..	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	**{**	**‖**	**}**	**~**	*del*

ASCII Erweiterungen, Unicode

- ASCII verwendet nur 7 Bit von einem Byte
  - enthält z.B. keine Umlaute (ä, ö, ü) oder Akzente (é, ç)

ASCII Erweiterungen, Unicode

- ASCII verwendet nur 7 Bit von einem Byte
  - enthält z.B. keine Umlaute (ä, ö, ü) oder Akzente (é, ç)
- es gibt verschiedene Erweiterungen von ASCII auf 8 Bit
  - in Europa ist ISO Latin-1 verbreitet (ISO Norm 8859-1)
  - belegt die Codes von 128-255 (bzw. 80-FF in hex)

ASCII Erweiterungen, Unicode

- ASCII verwendet nur 7 Bit von einem Byte
  - enthält z.B. keine Umlaute (ä, ö, ü) oder Akzente (é, ç)
- es gibt verschiedene Erweiterungen von ASCII auf 8 Bit
  - in Europa ist ISO Latin-1 verbreitet (ISO Norm 8859-1)
  - belegt die Codes von 128-255 (bzw. 80-FF in hex)
- Unicode wurde als 16 Bit Codierung eingeführt
  - erste 128 Zeichen stimmen mit ASCII überein
  - die nächsten 128 Zeichen mit ISO Latin-1
  - danach z.B. kyrillische, arabische, japanische Schriftzeichen

ASCII Erweiterungen, Unicode

- ASCII verwendet nur 7 Bit von einem Byte
  - enthält z.B. keine Umlaute (ä, ö, ü) oder Akzente (é, ç)
- es gibt verschiedene Erweiterungen von ASCII auf 8 Bit
  - in Europa ist ISO Latin-1 verbreitet (ISO Norm 8859-1)
  - belegt die Codes von 128-255 (bzw. 80-FF in hex)
- Unicode wurde als 16 Bit Codierung eingeführt
  - erste 128 Zeichen stimmen mit ASCII überein
  - die nächsten 128 Zeichen mit ISO Latin-1
  - danach z.B. kyrillische, arabische, japanische Schriftzeichen
- UTF-8 ist eine Mehrbyte-Codierung von Unicode (1-6 Bytes)
  - Code-Länge wird durch die ersten Bits codiert

Zeichen und Zeichenfolgen

Repräsentation eines ASCII Zeichens in C: `char`

- Zeichen-Literale in einfachen Anführungszeichen

  Beispiele: 'A', 'u', 'D'

```
char zeichen = 'A';
```

- Vorsicht bei nicht-ASCII Zeichen!

Zeichen und Zeichenfolgen

Repräsentation eines ASCII Zeichens in C: `char`

- Zeichen-Literale in einfachen Anführungszeichen

  Beispiele: 'A', 'u', 'D'

```
char zeichen = 'A';
```

- Vorsicht bei nicht-ASCII Zeichen!

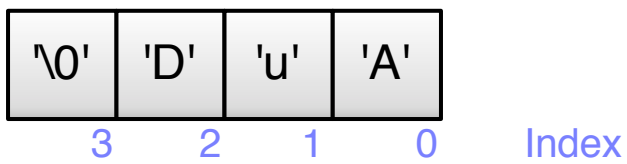
Repräsentation einer Zeichenfolge? (Englisch: String)

- String-Literale in doppelten Anführungszeichen

  Beispiel: "AuD"

```
std::string zeichenfolge = "AuD"
```

- in C gespeichert als Feld (sequentielle Liste) von Zeichen:



Zusammenfassung

1. Einführung
2. Grundlagen von Algorithmen
3. Grundlagen von Datenstrukturen
   - Primitive Datentypen und Zahldarstellung
   - Felder als sequentielle Liste
   - Zeichen und Zeichenfolgen